Ardalan, Alireza

DEVELOPMENT AND TESTING OF DATA STRUCTURES FOR THE CPM/MRP METHODOLOGY

The University of Arizona PH.D. 1983

DEVELOPMENT AND TESTING OF DATA STRUCTURES FOR THE CPM/MRP METHODOLOGY

by

Alireza Ardalan

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF MANAGEMENT AND POLICY

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 3

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Alireza Ardalan

entitled DEVELOPMENT AND TESTING OF DATA STRUCTURES FOR THE CPM/MRP METHODOLOGY

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Richard Bilbro	Date July 14, 1983
Wayne Thompson	Date July 13, 1983
	Date
	Date
	Date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Nicholas J. Aguilano — Dissertation Director — Date July 26, 1983

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: A. Andalon

To Roya

ACKNOWLEDGMENTS

I would like to thank my dissertation committee chairman Dr. Nicholas J. Aquilano, for his ideas and support throughout the course of this research. Without his contributions this dissertation would not exist.

I would like to specially thank Dr. Richard B. Chase for his guidance over the course of my graduate work and Dr. Mary E. Loomis for her expert advice over major parts of this dissertation. I would like to thank the other members of my committee, Dr. George W. Summers, Dr. Gustavo A. Vargas, Dr. Kalyan Singhal, Dr. Roy E. Marsten, and Dr. Jerome D. Wiest.

I would like to thank my parents whom without their support this dissertation would not start. Finally, I would like to thank my wife, Roya, who without her sacrifice and encouragement this dissertation would not have been completed.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS--Continued

LIST OF TABLES

ABSTRACT

A major purpose of this dissertation is to design and develop data structures for the Critical Path Method-Material Requirements Planning (CPM/MRP) methodology. The data structures developed consider the tradeoff between processing time required to perform the required operations on data structures and the computer capacity utilization to store data.

The CPM/MRP technique was designed to combine the capabilities of the critical path method and material requirements planning system. The critical path method is a project planning and control technique which schedules projects subject to technological sequence constraints and activity durations. When combined with material requirements planning, the methodology explicitly considers both the resources required by the activities comprising the project and the lead time to acquire the required resources. CPM/MRP contains algorithms for project scheduling subject to technological sequence and resource constraints. The early start and late start algorithms find feasible early start and late start schedules for both activity start times and resource order release times.

The major drawback of the FORTRAN IV computer program which incorporated the CPM/MRP algorithms was the tremendous computer memory capacity required. This prohibited application of the CPM/MRP to large projects. The data structures developed in this dissertation

are efficient with respect to both computer memory utilization and processing time. To design the data structures, the characteristics of storable and non-storable resources and the necessary operations within each resource category is studied.

Another purpose of this dissertation is to develop an algorithm to schedule operating rooms for surgery procedures in hospitals subject to resource constraints to increase operating suite utilization. Since the major reason for low operating suite utilization is lack of required resources when they are needed and where they are needed, the CPM/MRP concept is applied to schedule surgeries. The late start algorithm outlined in this dissertation schedules surgeries and resources required for each surgery.

The data structures and the surgery scheduling algorithm are incorporated into a FORTRAN IV computer program. The program has been tested with actual data gathered from a hospital. The results met the objectives of both computer memory utilization and low computation time.

CHAPTER 1

INTRODUCTION

One of the major problems in creating and operating computer programs is the control of data. The proper organization of data files is crucial to efficient computer program operation. This study examines data structures and presents data structures which is shown to be efficient in file space requirements and processing time. These data structures are then applied in a Critical Path Method/Material Requirements Planning (CPM/MRP) type problem.

A file is a collection of related records. The organization of the records of a file depends on the operations which will be performed on that file.

The characteristics of the file and how it will be processed significantly affect the design of file organization. The most important aspects of a file are (Nievergelt 1974):

1. Whether the file is static (the set of records is given once and for all) or dynamic (there are frequent insertions and deletions of records).
2. Whether the access probabilities for all records are known or unknown.
3. Whether the file will be kept in one-level store or two-level store.

Comparisons of file organization techniques could be according to storage utilization, successful and unsuccessful search times, ease of modification of the file, and usability of the file organization technique in both one-level and two-level stores. If the entire file is kept in random access central memory it will be called one-level store. It will be called a two-level store if only a small portion of the file is in central memory and the rest of the file is in some secondary storage.

A major objective of this study is to choose and develop efficient file organization techniques to organize the files required for the efficient operation of a combined Critical Path Method-Material Requirements Planning system. The following sections explain the components of the system briefly.

Critical Path Method

Critical path method is based on network presentation of a project. Both event and activity oriented project networks assume that all activities comprising a project have well defined start and finish characteristics. Other assumptions of CPM are (Wiest and Levy 1977):

1. The activities must be performed in a given technological sequence.
2. The activities are independent, in that they may be started and stopped independently of each other.
3. Activity times are deterministic.

4. Activities are not delayed by resource shortages; it is assumed that unlimited quantities of all resources will be available upon demand.

The CPM algorithm might be used to determine the sequence of connected activities that comprise the longest path through the project network. This path is called critical path and the activities which are on the path are critical in determining the duration of the project.

Usually the amount of resources available over the project duration are limited. This is a violation of a CPM assumption and therefore the CPM algorithm cannot be used. The problem then becomes one of how to allocate limited amount of resources to the activities of a project. This problem has been the subject of numerous studies. Resource allocation procedures are classified in three categories (Davis 1973).

1. Time-cost trade-off: the basic assumption of the procedures in this category is that activities can be expedited to some extent by allocation of more resources to those activities. Expediting an activity is usually more expensive than performing that activity in its normal time. The result of time-cost trade-off analysis is a curve that shows the relation between the duration and the lowest possible cost of the project.

2. Resource leveling: the procedures in this category consider the cost involved in changing the resource requirements from one period to the next. The allocation decisions are made for fixed

project duration by shifting the activities to smooth the resource requirements over the project duration.

3. Constrained resources: the procedures in this category assume that there is a fixed level of each resource available over the project duration. The problem here is to reduce the duration of projects with respect to constraints to the amount of resources available.

Material Requirements Planning

Material requirements planning is basically an inventory planning and control system for items which the demand for them originates from the demand for other items, usually sub-assembly, assembly or end product. MRP systems assume that uncertainty exists only at the end product level. Having determined the demand for the end product the demand for the components can be calculated from the demand for end product. A function of an MRP system is to determine the quantity and time of the demand for each component, compare this demand with the amount available in the inventory, and release orders to cover any shortages for the components. The data required by an MRP system is usually stored in the following three files:

1. Master production schedule: states time and quantity of demand for end products.
2. Bill of materials: states exactly which components and the quantity of each component that are used to manufacture an end product.
3. Inventory records: also called item master file has all necessary information about the status of all of the components and

end products for the MRP planning and control effort. The inventory records file contains data on quantity on hand, on order, and scheduled receipts for each item in the bill of material.

Critical Path Method-Material Requirments Planning (CPM/MRP)

Aquilano (1978) combined critical path method and material requirements planning to schedule projects. The CPM-MRP project scheduling technique considers the resources required for each activity explicitly. The data requirements of the CPM-MRP system and MRP system are the same. The completion date of the project is stated in the master schedule. A project bill of resources defines precedence relationships and the duration of all of the activites of the project. This bill also defines types and amounts of each resource required for each activity and the procurement lead times of the resources.

Similar to MRP systems the inventory status of all resources is stored in the inventory records file. Unlike MRP, the inventory records file in CPM-MRP includes data on both storable and non-storable resources. Non-storable resources are those resources which if not used in a period cannot be stored for use in the next period. For example, facility and personnel are non-storable resources.

CPM-MRP uses MRP logic to explode the project bill of resources to determine the quantity and time of requirements of each resource, and CPM is used to find the critical path. The advantage of CPM-MRP over CPM is that it accounts for the procurement lead time of resources required for the project. Thus, CPM-MRP does not schedule an activity unless the required resources for that activity are available or the

required resources can be ordered and received before the start time of that activity. Aquilano shows that inclusion of procurement lead time of resources might affect the duration of project and also the critical path..

Smith (1980) applied the CPM-MRP idea in project scheduling under constrained resources. The CPM-MRP algorithm under constrained resources schedules activities of a project with respect to precedence constraints and the availability of resources.

CPM-MRP explicitly considers each resource defined in the projects' bill of components. It determines the order quantity and order release time for each resource according to the requirement for these resources by activities and resource acquisition lead times. An output of the CPM-MRP is a detailed plan which shows the order release times and order quantities for all resources used for activities in a project so that sufficient quantities of the required resources are available to perform activities. The explicit scheduling of required resources is especially important in situations where the unavailability of resources, when they are required, significantly affect the operations of a productive system.

A major problem in implementing the CPM-MRP is how to store the data required by the method in a computer system. Zastera (1978) and Smith (1980) use three dimensional arrays to store most of the data on activities and resources included in a project. Their data structure uses four storage locations for each activity or resource in each period in the planning horizon. Many of these storage locations will not be used in each run and many might contain the same data. For

example, the quantity on hand for a resource will remain constant over some periods if there has been no usage or no receipt of that resource in those periods. Therefore, as Smith (1980) realized, their approach to store data is inefficient with respect to computer memory utilization.

An objective of this study is to apply the combined critical path method-material requirements planning concept in the hospital operating suit environment to find feasible surgery schedules with respect to limited resource availabilities.

Operating rooms (OR) are one of the most expensive service facilities of a hospital. The construction cost per square foot, including built-in equipment, is in the range of $125 to $150. The cost of construction has increased constantly about 8 to 10 percent per year (Schneiderman 1979). Also, the operating expenses are high because OR personnel are skilled workers with relatively high salaries. Usually the operating room is a bottleneck in the processing of surgical patients and as a result waiting lists develop. When a waiting list exists the OR influences the flow of surgical patients into the hospital. Therefore, operating room utilization should be high. Utilization is defined as the ratio of the actual total number of minutes used per day to the total potential number of minutes available, multiplied by 100 (Healy 1973). There is no universal agreement on how high utilization should be. Some authors suggest 85 to 95 percent utilization can be achieved without disruption of normal operations of the hospital (Phillips 1975).

High OR utilization reduces the idleness of OR personnel and facilities. The financial implication of high OR utilization is lower OR cost to the patient due to a decrease in the proportion of OR fixed to variable costs. The life saving implication of high OR utilization is that more patients receive the required OR services.

Surgical admission scheduling has received attention because of its important influence on operations of a hospital. Surgical admission scheduling affects the flow of patients into and out of the hospital. It also controls the workload in service areas of a hospital, especially in the operating room.

Resources of a hospital are relatively fixed at least in the short run. Surgical demand scheduling is constrained by the level of resources available. Although different surgical cases might use different types of resources they all use operating room. For this and other reasons mentioned before, the problems of surgical demand scheduling will be presented with a focus on the operating room. Other types of resources can be treated the same way.

Problems in Surgical Demand Scheduling

Pattern of Demand

Demand for surgical admission is generated from demand for elective surgeries and demand for emergencies. For elective admission the patient, doctor, and the admission department agree upon a later date for surgery. It is the responsibility of the admission scheduling department to make sure that all the necessary preoperative

services and treatments are available. These include a bed, different tests, and preparation for surgery; operating requirements such as operating room time, personnel, equipment, and tools; and postoperative services and facilities such as a bed in a special department, personnel, and other supporting services. These should be available in such a way as to make the length of stay as short as possible.

The pattern of demand for elective surgery is cyclic during each week and also during each year. Demand is consistently higher on Mondays and Tuesdays and in winter months. Demand for elective surgery is higher in vacation periods.

Most emergency cases are admitted on arrival. Resources are allocated to emergency cases according to the severity of the health condition of the patient. The emergency patient might wait in the emergency room for the next OR to be ready or another ongoing procedure might be interrupted to free resources for the emergency case, etc. The arrival of emergencies is stochastic. Hence it is neither efficient nor possible to allocate hospital resources to provide service for all emergency cases. But the number of turnaways should not be high either, because it hurts the public image of the hospital and sometimes results in loss of accreditation of the hospital.

Unlike the demand for elective surgeries there is no communication between the emergency cases and the hospital prior to occurrence of the need. Hence there is only probabilistic information about the emergency cases demanding surgeries in a period.

In addition to the demand for surgical admission there is demand for surgical procedures from patients who do not stay in the hospital before or after the procedure (outpatients). The pattern of surgical demand by outpatients is similar to the pattern of surgical demand by elective patients.

Types of Demand

Types of surgeries demanded by these three sources are not homogeneous within each source nor over all sources. Each type of surgery has its own particular characteristics. OR time, personnel, equipment, and other OR services are likely to be different from one type of surgery to another. Within each type there are also differences among cases. Procedures and techniques followed in each case might be different from other cases. The severity of case, the age, sex, physical condition of the patient, the physician, and other variables are responsible for differences in procedures.

Variations in both the total quantity of surgeries demanded and quantity of each type demanded brings about pressure on the surgical demand scheduling system to monitor effects of the schedule on workloads in most service areas of a hospital. This is a complex job presently performed manually, for the most part. Therefore, only major service areas such as bed services and occasionally nursing services are considered for the workload effect of schedules. It is the responsibility of the scheduling department to consider the capacity of each service area of the hospital and to schedule surgical demand

in a way to ensure availability of resources when they are needed. (The topic of service facilities sizing is not presented here since it does not lend itself directly to the scheduling problem. Scheduling decisions are made under the capacity constraints imposed by earlier decisions about facility sizing. Griffith (1976) has presented a good treatment of the facility planning topic in hospitals.) Unavailability of any required service, equipment, or tool produces delays in initiating procedures or results in longer procedure times. The first situation results in lower utilization of the OR and the latter results in lower efficiency of OR utilization.

Estimate of Procedure Time

The differences in cases within each type of surgery affect procedure times. An estimate of procedure time is used at the time of scheduling. The actual procedure time is most likely different from the procedure time estimated at the time of scheduling. The magnitude of the deviation depends on the quality of the estimating method used. Generally the start time and finish time of most procedures will be different from the scheduled start and finish times. A poor procedure time estimate causes large deviations between the scheduled start and completion times and the actual start and completion times. The next case is delayed if the actual procedure completion time happens to be later than the estimated time and deviations from schedule will take place. If the procedure completion time is earlier than the estimate, the OR and staff may remain idle until the next procedure.

Cancellations

A large delay beyond the scheduled start time for a procedure caused by a delay in completing earlier procedures increases the possibility of cancellation of the procedure. Other major reasons for cancellation are patients' decisions to cancel procedures and physicians' late arrivals. Because cancellations are often rescheduled, schedules already made for later periods can be disturbed. Length of stay of patients whose cases are cancelled increases. The effect on the patient is psychological stress for being in an uncertain condition and hence less satisfaction from the services.

Objectives of this Study

A major problem in implementing the CPM-MRP is how to store the data required by the method in a computer system such that the scheduling system be efficient with respect to both computer storage utilization and execution time. In this study data structures and file organization techniques are reviewed and efficient data structures are developed and tested.

As is discussed in Chapter 3, a major cause of low operating room utilization is unavailability of resources when they are needed. To increase the utilization of operating rooms, this research focuses on scheduling activities which comprise a surgery with respect to constrained resources. As is explained in Chapter 3, the Critical Path Method--Material Requirements Planning (CPM-MRP) concept is used to schedule surgeries. Objectives of this research are:

1. To design data structures for Critical Path Method-Material Requirements Planning system which are efficient in both computer time and capacity utilization.
2. To develop an algorithm based on CPM-MRP concept to find late start schedule for a surgery subject to resource limitations.
3. To write a computer program which incorporates the data structures designed to schedule surgery procedures.
4. To test the computer program on real problems.

CHAPTER 2

SURVEY OF LITERATURE

As stated in Chapter 1, comparisons of file organization techniques could be according to storage utilization, successful and unsuccessful search times, ease of modification of the file, and usability of the file organization technique in both one-level and two-level stores.

In scheduling projects using CPM-MRP technique the inventory file is continuously changing. Every time an activity is scheduled the inventory file will be updated to account for the changes in the level of resources used up by that activity. The planned order releases for those resources which are required by the scheduled activity should be updated. These and other files used in CPM-MRP are dynamic.

Different types of procedures are scheduled in each day. The start times of the same type of procedure are different in different days. Therefore, the probability that a particular procedure will be scheduled in a specific period of a day is not known. Since the time of the requirements for the resources depends on the start time of the procedures the access probabilities of the records are unknown.

Numerous studies have been performed to evaluate different characteristics of the file organization techniques in a variety of situations. For the purpose of this study that part of the literature

which deals with the dynamic file, unknown access probabilities, and both one-level and two-level stores will be reviewed.

Data Structures

A collection of related records is called a file. The data structure defines how the records relate, their domain, and what operations can be performed.

Horowitz and Sahni (1976) define data structure as: "A data structure is a set of domains D, a designated domain dD, a set of functions F, and a set of axioms A. The triple (D, F, A) denotes the data structure d and it will usually be abbreviated by writing d."

As an example, a data structure for integers has the domain (all integers), the arithmetic functions which define the input and ouptut of these functions (+, - for example), and a set of axioms which show precisely how the functions work.

Arrays

A data structure does not define the representation of data in the computer memory. The computer language's flexibility in providing data structures is in representation of data structures. Most of the computer languages provide the data structure array. The operations that can be performed on arrays are: create array, store data in a specific location in the array, and retrieve data from a specific location in the array. These operations define the data structure array (a formal definition of an array can be found in Horowitz and Sahni (1976)).

Intuitively an array is a set of pairs of indexes and values. Associated with each index is a value that has been stored in the array. For n dimensional arrays n indexes are associated with one value. For simplicity one-dimensional arrays will be discussed.

Arrays are usually represented by consecutive words in the memory. The amount of time required to store or retrieve any element is constant in this representation assuming a random access memory capability. Given a legal index the value associated with that index can be stored and later can be retrieved.

Lists

A list may be represented by an array. To find a specific element the values stored in the array should be examined one by one until that value is found. A common example is the list of telephone numbers in the telephone directory. To find the location of a number in the list the search begins from the first number. There are two possible outcomes: (1) The number in the list is equal to the number being searched. The search terminates successfully. (2) The number in the list is not equal to the number being searched. The search will continue until the number is found--a successful search--or the number does not exist in the list and it is not found--an unsuccessful search. This type of search is called sequential search.

Knuth (1973) shows that in a sequential search of a file containing N elements with uniform probability of request for elements the search terminates at most in N steps. And the average number of comparisons to find an element is $(N + 1)/2$.

When the probability of requests for keys in a file are not equal the list can be organized in a way to reduce the search time. Rivest (1976) compares two heuristic rules which are used to maintain a sequential list in approximately optimal order with respect to the average number of comparisons required to find a specific element. The move-to-front heuristic moves the element found in position A(i) to the front of the list and moves the elements in positions 1, 2, . . . (i-1) down by one position each. The transposition heuristic exchanges the element found in position A(i) by the element in position A(i-1). Hence the accessed element moves one position closer to the front of the list. Assumptions made by Rivest are:

1. The file is static.
2. Each element is searched for with a fixed probability independent of previous searches performed.
3. Probabilities of retrieving any particular element are not all equal.

Rivest compares the asymptotic search costs of the two heuristics and concludes that the transposition method outperforms the move-to-front heuristic and the transposition method might be optimal.

Tenenbaum (1978) presents the results of a simulation study on the move-to-front and transposition heuristics making the same assumptions as Rivest did. Tenenbaum shows that as the number of records in the list increases moving each accessed record toward the front of list a greater number of positions yields better results for

a fixed number of searches. Hence the transposition method is not optimal. He also shows that the move-to-front rule reaches its asymptotic cost in fewer searches than does the transposition rule.

Ordered Lists

In the telephone directory example a record is made up of the name, the number, and the address of a person. To find the number which belongs to a person the name of that person is found first and then the number is found. The name therefore is the key to the record. When the key to the record is found other data related to the key can be found easily. Therefore only keys will be considered in this paper unless the treatment of related data becomes conceptually important.

In the telephone directory example the names are listed in alphabetical order. The data structure ordered list has received considerable attention. Some of the operations that can be performed on ordered lists are:

1. Find an element.
2. Insert a new element.
3. Delete an element.

Ordered lists are usually represented sequentially. The array index is associated with the rank of the key in the ordered list. This characteristic of the sequential representation and the fact that the keys are in order has been used to devise more efficient search strategies. A three way decision is possible when a particular key (K) is sought.

1. If K < Kc: the search continues on keys in that part of file which contains keys which are less than Kc.
2. If K > Kc: the search continues on keys in that part of file which contains keys which are greater than Kc.
3. If K = Kc: the key has been found and the search is terminated.

The difference between the search strategies stems from the way they choose to "cut" the file and find Kc. Some of the strategies will be presented in the following section.

Search Methods

Binary Search. In a file containing N keys K1 < K2 . . . < KN, the cut is made in the middle of the file and the key which is located in the middle of the file (Kc) is compared to the key being searched (K). Three possible outcomes are as follows:

1. K < Kc: search continues in the part of file which contains keys less than Kc.
2. K > Kc: search continues in the part of file which contains keys greater than Kc.
3. K = Kc: search terminates successfully.

Knuth (1973) presents three algorithms for binary search and shows that in the worst case $\lfloor \log N \rfloor + 1$ comparisons are required to find a key. The average number of comparisons required to find a key, however, is logN - 1. (All logarithms are in base 2 unless otherwise stated.)

Fibonacci Search. This search strategy cuts the file in unequal segments according to Fibonacci sequence defined as: $F_i = F_{i-1} + F_{i-2}$, $i >= 2$. The sequence is 0, 1, 1, 2, 3, 5, 8, 13, . . . The search starts by assuming that the number of keys in the file is one less than some Fibonacci number, $F_k - 1$.

The following comparisons and results are possible.

1. $K < K_{F_{K-1}}$: The part of the file containing keys which are less than $K_{F_{K-1}}$ is searched next.
2. $K > K_{F_{K-1}}$: The part of the file containing keys which are greater than $K_{F_{K-1}}$ is searched next.
3. $K = K_{F_{K-1}}$: The search terminates successfully.

Fibonacci Search will make at most O(logN) comparisons to terminate a search. The function $f(N) = O(g(N))$ holds if there exist two constants c and M such that $|f(N)| =< c|g(N)|$ for all $N >= M$. This means that f(N) will at most be equal to a constant times g(N).

Interpolation Search. This search strategy uses the distribution of the keys to find where the file should be cut. Yehoshua, Itai, and Auni (1978) analyze a file in which the keys are uniformly distributed. They show that the expected location of the key is $\lceil N(P) \rceil$ and they choose the cut index c = N(P) where: N is the number of keys in the file, and $P = (K - K_0)/(K_{n+1} - K_0)$.

K: The key being searched.

P: The probability that a random key in the file is less than or equal to K.

K_0 and K_{N+1}: are considered to be the first and the last keys in the file respectively.

In the worst case the interpolation search might terminate in N steps but as Yehoshua et al. show, the average number of comparisons to terminate the search is logN steps. And the probability that a search may require more comparisons is negligible.

Jump Search. portions of file are jumped over until the search is localized. Then either smaller jumps in that segment of the file might be taken or a sequential search is performed to locate the key. Shneiderman (1978) presents five jump searching methods. He compares their performances with respect to the number of comparisons necessary to find a key in both one-level and two-level storages. He also finds the optimum jump sizes considering the costs of: a jump, a sequential search, and an access from the secondary storage.

The search operation can be done efficiently on the ordered lists represented sequentially. The insertion and deletion on the other hand takes much time and effort. A deletion from the position i requires a movement of elements i + 1, . . ., N by one toward the front of the list. An insertion in position i requires a movement of elements i, i + 1, . . ., N by one toward the end of the list. The insertion and deletion takes O(N) time in a sequential representation of ordered lists.

Linked Lists

Ordered lists may also be represented as linked lists. In the linked lists representation the position of elements may not be the same as their position in the ordered list. Each element corresponds to a data called link which shows the position of the next element in the linked list. Figure 2.1 shows a linked representation of the ordered list D, H, M, Q, S.

The insertion and deletion operations on linked lists can be performed with real ease. A new element can be inserted into an empty position and at most two links should change. An element can be deleted by changing only one link. Figure 2.2 shows insertion/deletion of the element F into/from the above list.

Although insertion and deletion operations on a linked list are performed in constant times, the search operation should be done sequentially. The sequential search takes O(N) time, which is weak performance. The sequential and linked representation of an ordered list in a dynamic environment where frequent searches, insertions, and deletions are required are not efficient. As was shown above, the sequential representation has good search performance but is not efficient for insertions and deletions. The linked representation performs very well for insertions and deletions but has weak search performance. A data structure which maintains the search characteristic of sequential representation and performs well when frequent insertion and deletion operations are required will be described next.

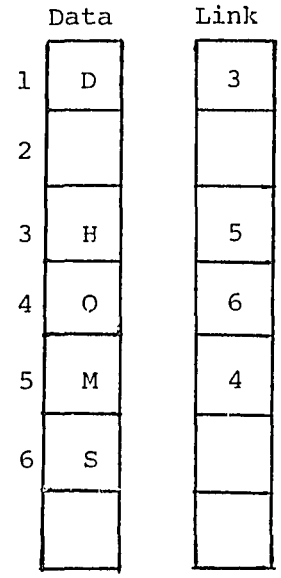


Figure 2.1 A Linked List Example

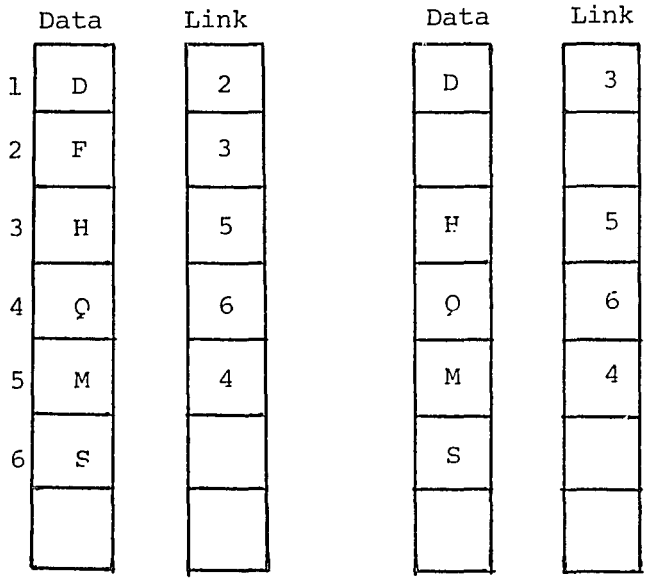


Figure 2.2 Insertion and Deletion From a Linked List

Trees

Trees are important data structures. Intuitively, a tree structure is the organization of data so that items are related by branches. A familiar example of trees is the family tree which shows parents, ancestors, and other relatives of an individual. Horowitz and Sahni (1976) define a tree as "a finite set of one or more nodes such that: (i) there is a specially designated node called the root; (ii) the remaining nodes are partitioned into n >= 0 disjoint sets T, . . ., Tn where each of these sets is a tree. Tl, . . ., Tn are called the subtrees of the root."

The second condition in the above definition prevents subtrees from connecting together. This condition implies that every node in a tree is the root of a subtree. Each node presents an item of information. The branches leaving a node present the relation of that node with other nodes. The number of branches leaving a node is the degree of the node. A node with a degree equal to zero is called a leaf or terminal node. Other nodes are called nonterminal or internal nodes.

The level of a node is equal to the number of nodes visited before that node is reached. As such, the level of the root is zero, the level of each root's children is 1 and so on. Another way of assigning levels to the nodes is to start with level one for the root, level two for the root's children, etc. The height or depth of a tree is defined as the maximum level of any node in the tree. A forest is the set of n >= 0 disjoint trees.

Different types of tree structures have been used and studied by both practitioners and researchers. Each type has its own special characteristics. In the following segments some of the particular types of trees and their characteristics will be presented.

Binary Trees: The special property of the binary tree is that every node can have a degree of two or less. Therefore, there are at most two subtrees for every node. A subtree is called the left (right) subtree depending on whether the branch leading to it is to the left (right) of the parent.

Horowitz and Sahni (1976) define binary tree as "a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree."

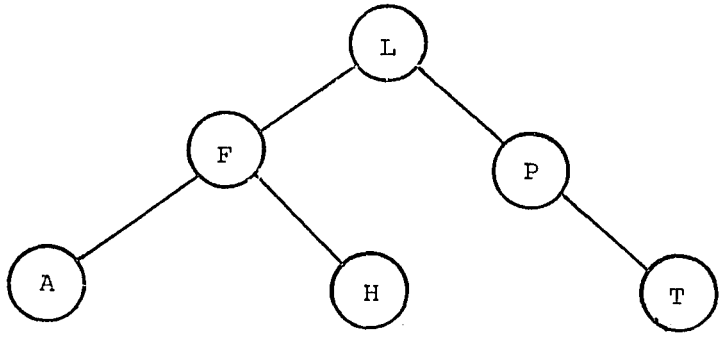
Other special properties of binary trees are summarized below. The proof of the following properties can be found in Horowitz and Sahni (1976).

1. The maximum number of nodes on level i of a binary tree is 2^i, i >= 0.
2. The maximum number of nodes in a binary tree of depth k is $2^{K+1} - 1$, K >= 0.
3. For any non-empty binary tree, T, if n_0 is the number of terminal nodes and n_2 the number of nodes of degree 2, then $n_0 = n_2 + 1$. A binary tree of height K is a full binary tree if it has $2^{K+1} - 1$ nodes. By statement (2) above $2^{K+1} - 1$ is the maximum number of nodes a binary tree with height K can have.

4. A balanced binary tree is a binary tree which has all its leaves in its two highest levels.

One operation that can be performed on binary trees is visiting each node in the tree exactly once or traversing the tree. A tree traversal produces a linear order for the data in a tree. Binary trees may be traversed in a variety of orders. For convenience, the left subtrees are visited first. This convention leaves three traversals. The three types of traversals are:

1. Inorder traversal: starts by visiting the node which is farthest down the tree and towards the left. Then it visits the parent and traverses the right child inorder.
2. Preorder traversal: visits a node (starts from the root). Visits the left child and continues until it is not possible to move left. Then moves back until it can move right. It visits the right node and continues the preorder traversal.
3. Postorder traversal: starts by visiting the node which is farthest down the tree and towards the left. It moves back until it can move to the right. Then it moves to the right child and resumes postorder traversal. If it is not possible to move down the tree, it visits the node and moves up and visits the parent. It moves back until it can move to the right and continues postorder traversal. Figure 2.3 shows the three traversals for a binary tree.



Inorder: A, F, H, L, P, T

Preorder: L, F, A, H, P, T

Postorder: A, H, F, T, P, L

Figure 2.3 Binary Tree Traversals

A binary tree can be represented sequentially. To determine the index of a node in a binary tree the nodes of a full binary tree should be numbered sequentially starting from the root and then the nodes on level one, level two and so on. The nodes on each level should be numbered from left to right. The index of a node in a binary tree is the number assigned to the corresponding node in the full binary tree.

The sequential rerpesentation of a binary tree may result in unutilized space because all binary trees are not full. Also the insertion and deletion of internal nodes require many changes because the level of other nodes is changed. Linked representation of binary tree is more appropriate because the insertion and deletion of internal nodes becomes easier. Each node will have three fields called LCHILD, DATA, and RCHILD. A node is linked to its left and right children by LCHILD and RCHILD.

A special type of binary tree can be produced by growing the tree in such a way that for each new name we compare it to the root and if the new name precedes the name at the root we move left and continue otherwise we move right and continue until we reach a leaf. At this point if the new name precedes the name at the leaf the new name becomes the left child of that leaf (becoming an internal node) otherwise the new name becomes the right child of that leaf.

Figure 2.4 shows the sequence of binary trees for growing a binary tree consisting of the M, K, T, O, L, C, Q and F names according to the above procedure. Inorder traversal of the final binary

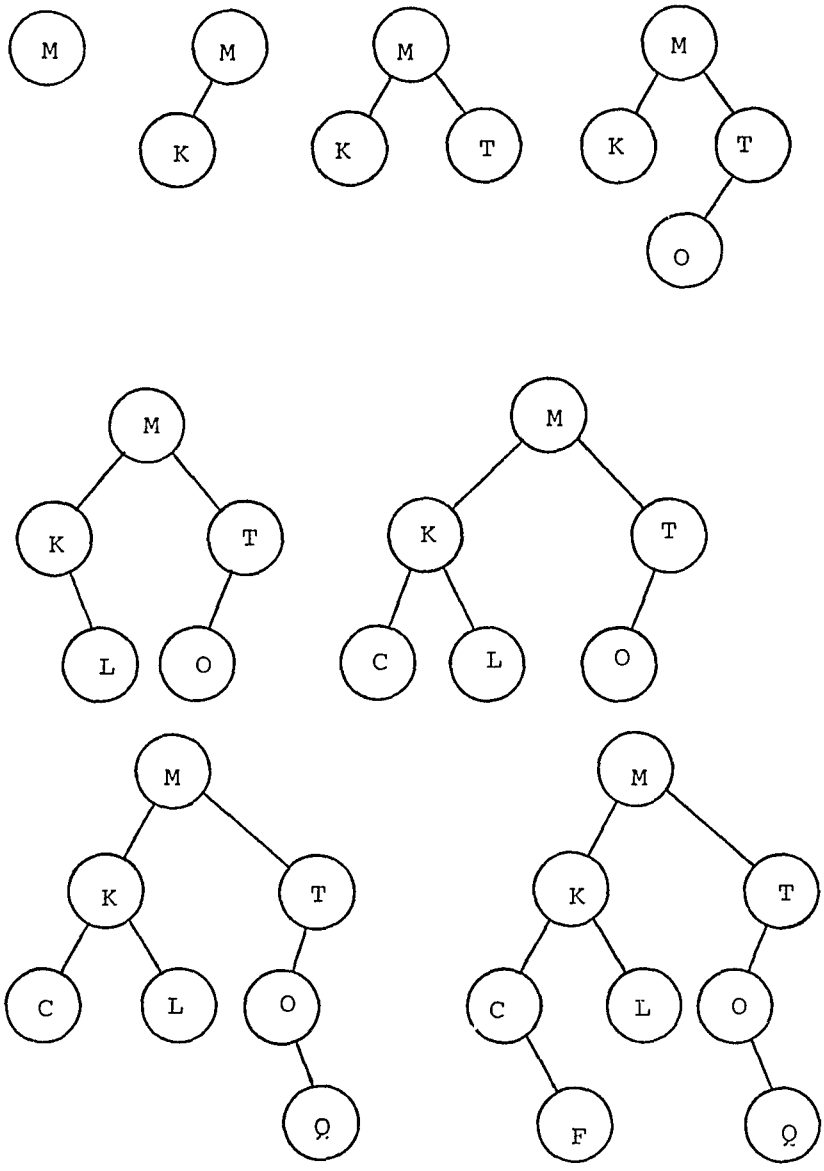


Figure 2.4 Example of Growing a Binary Tree

tree in Figure 2.4 produces the sequence C, F, K, L, M, O, Q, T. In this process the names are sorted. This is an important characteristic of the type of binary tree which can be produced by following the rules indicated above. This special type of binary tree is called binary search tree. Binary search tree and its characteristics will be presented next.

Binary Search Trees. It was noted above that binary search trees are grown based on comparisons of names. Therefore, to be able to organize a collection of data as a binary search tree every record should have a unique name (or key).

Loomis (1983) defines a binary search tree as

> the collection of n records with names, N1, N2, . . ., Nn is a binary tree each of whose nodes Ri bears one of the names Ni, for i = 1, . . ., n. The nodes of the binary tree are arranged such that for each node Ri, the following properties hold:
>
> 1. All names of nodes in the left subtree of Ri precede the name labeling Ri:
>
> If Rj = LEFT(Ri) then Nj < Ni.
>
> 2. The name labeling Ri precedes the names of all nodes in the right subtree of Ri:
>
> If Rj = RIGHT(Ri) then Ni < Nj.

There are four basic operations on binary search trees: direct search, sequential search, node insertion and node deletion. The statements 1 and 2 above imply that records in a binary search tree can be accessed directly. To find a node with name N in the binary search tree rooted at Ri:

1. If the tree is empty, then the search terminates unsuccessfully.
2. If N = Ni, then the search terminates successfully; the sought node is Ri.
3. If N < Ni, then the left subtree of Ri is searched, i.e., Ri = LEFT(Ri).
4. If N > Ni, then the right subtree of Ri is searched, i.e., Ri = RIGHT(Ri).

As noted before, an inorder traversal of a binary search tree sorts keys and as such the keys can be accessed sequentially. To insert a new node NEW with name N in the binary search tree rooted at Ri:

1. If the tree is empty, then node NEW becomes the root.
2. If N = Ni, then the insertion is not necessary the name is already in the tree.
3. If N < Ni, then the left subtree of Ri is searched until the appropriate position for NEW is found.
4. If N > Ni, then the right subtree of Ri is searched until the appropriate position for NEW is found.

The operation of deleting a node from a binary search tree is more complicated. If the node to be deleted is a leaf then it can be deleted easily. Otherwise the special characteristics of the tree and the subtrees of that node should be preserved.

A particular set of names can be organized in different binary search trees. Figure 2.5 shows some of the binary search trees which can be grown for the name set A, B, C, D, E, and F.

A measure of performance of a binary search tree is the expected number of comparisons made to terminate a search either successfully or unsuccessfully. The access probabilities of the keys affect the performance of binary search trees. For our purpose since the probabilities are not known it is assumed that the probabilities follow uniform distributions. Since the probabilities are assumed to be the same, the total path length can be used for evaluations of search performance of binary search trees.

To evaluate the performance of the binary search trees of Figure 2.5, null nodes should be added to those trees to show the unsuccessful searches. Figure 2.6 shows binary search trees of Figure 2.5 when null nodes are added. The squares represent null nodes.

The total path lengths of the binary search trees of Figure 2.4 are:

$$0 + 1 + 1 + 2 + 2 + 3 + 3 + 4 + 4 + 5 + 5 + 6 + 6 = 42$$

$$0 + 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 4 + 4 = 30$$

$$0 + 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 + 3 = 28$$

for trees a, b, and c respectively.

The binary search tree c which has the lowest total path length is called the optimal binary search tree. The optimal binary search trees have the minimum average number of comparisons for both

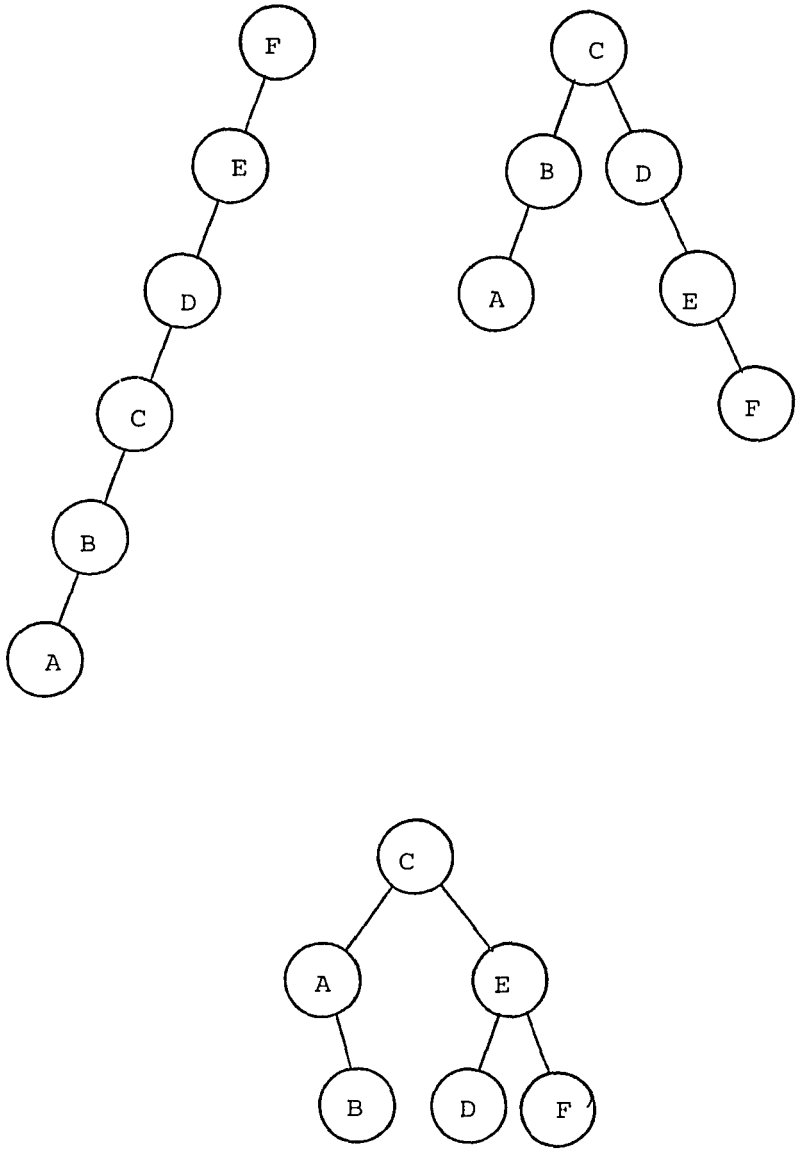


Figure 2.5 Example of Binary Search Trees for a Name Set

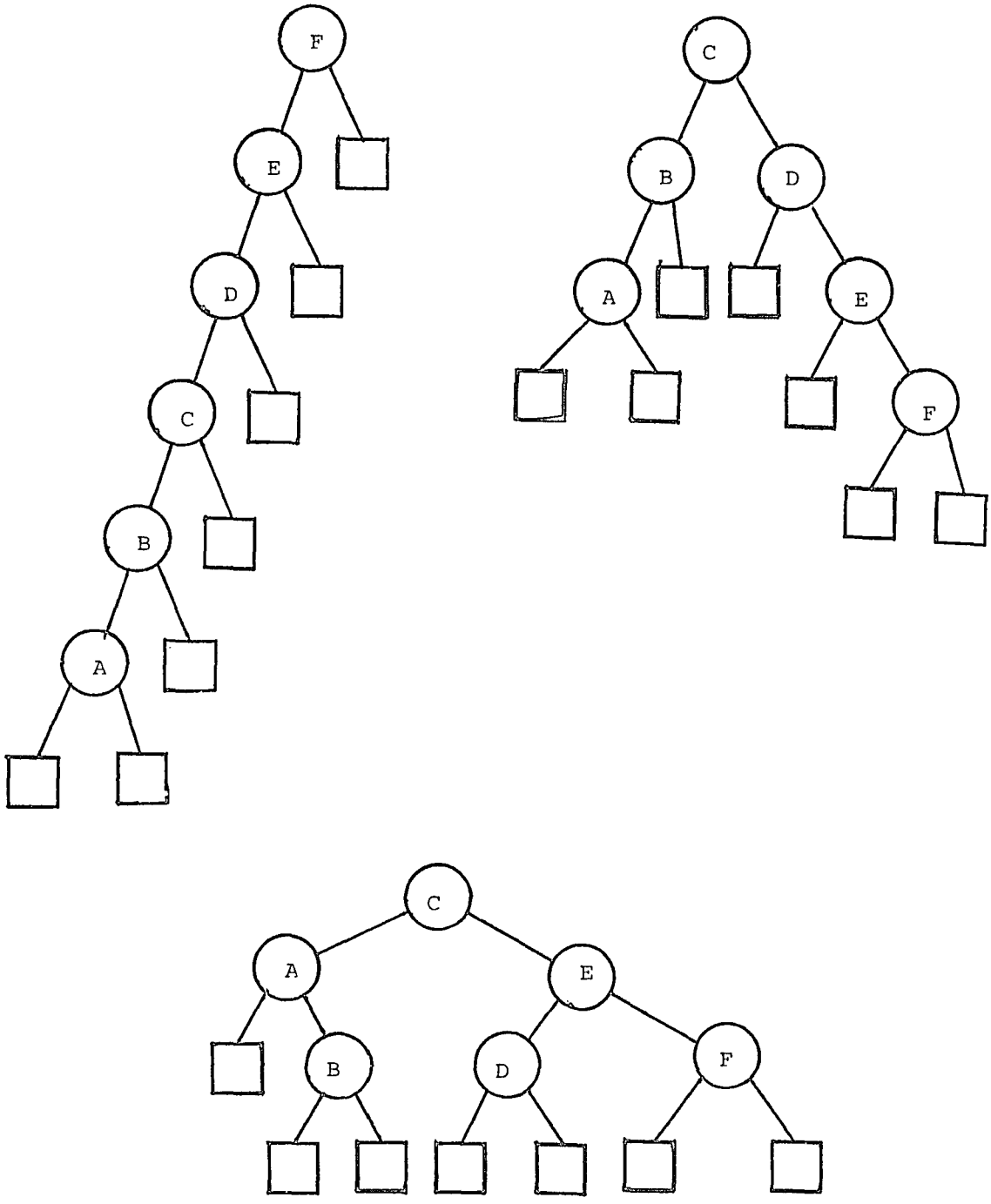


Figure 2.6 Example of Binary Search Trees with Null Nodes

successful and unsuccessful searches. Here the criterion for optimality is to keep the binary search tree as balanced as possible.

Knuth (1971) shows that when some names are known to be much more likely to occur than others, the best possible binary search tree will not necessarily be balanced. He presents a systematic method for finding optimum binary search trees when probabilities are known. Hu and Tan (1972) give the least upper bound on the weighted path length of optimum binary search trees proposed by Knuth (1971).

The average number of comparisons for searching is not the only measure of performance. Nievergelt (1974) indicates that keeping binary search trees as balanced as possible requires a lot of work and it is time consuming. Especially when a binary search tree is large the benefits gained by keeping the tree balanced are more than offset by expenses for keeping it balanced. Nievergelt surveys the binary search trees which are nearly optimal. The nearly optimal binary search trees have a search performance which is close to the optimal search performance and tree maintenance after node insertions and deletions can be done with much less effort and time than for completely balanced binary search trees.

Height-balanced Trees. A height-balanced tree (AVL tree named after its creators) is a binary search tree and for every node the height of left and right subtrees differ by at most one. It is relatively easy to maintain a tree height-balanced. Figure 2.7

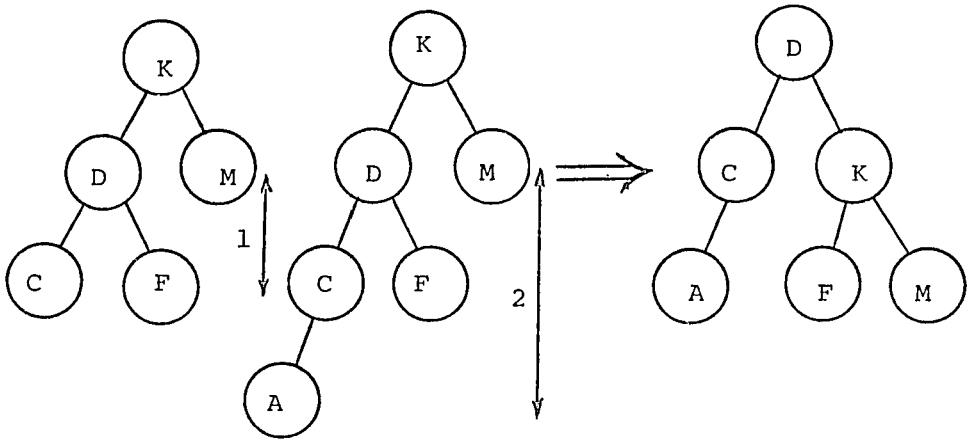


Figure 2.7 A Single Rotation Example

shows an example of a process to preserve the tree as height-balanced after the new node A is inserted.

After insertion the height constraint has been violated at node K where the height of its left subtree is two levels more than the height of its right subtree. A single right rotation of the tree returns the tree back to height-balanced. Other more complicated situations may require a double rotation. Since two types of rotations (left or right) exist a double rotation may consist of the following rotations: a left and a right, a right and a left, both left, and both right. Figure 2.8 shows an example of a left then right double rotation.

Nievergelt (1974) indicates that insertion in height-balanced trees require at most a single or a double rotation. Deletions may require at most h/2 transformations (where h is the height of the tree), but on the average the number of transformations required is a small constant independent of tree size.

Karlton, Fuller, Scroggs, and Kaehler (1976) performed a simulation study of the behavior of height-balanced binary search trees. For both insertion and deletion operations they tabulated the mean and variance for:

1. The number of comparisons made to locate where a new node should be inserted or to locate the node which should be deleted.
2. The percentage of insertions or deletions that caused a rotation to be performed.

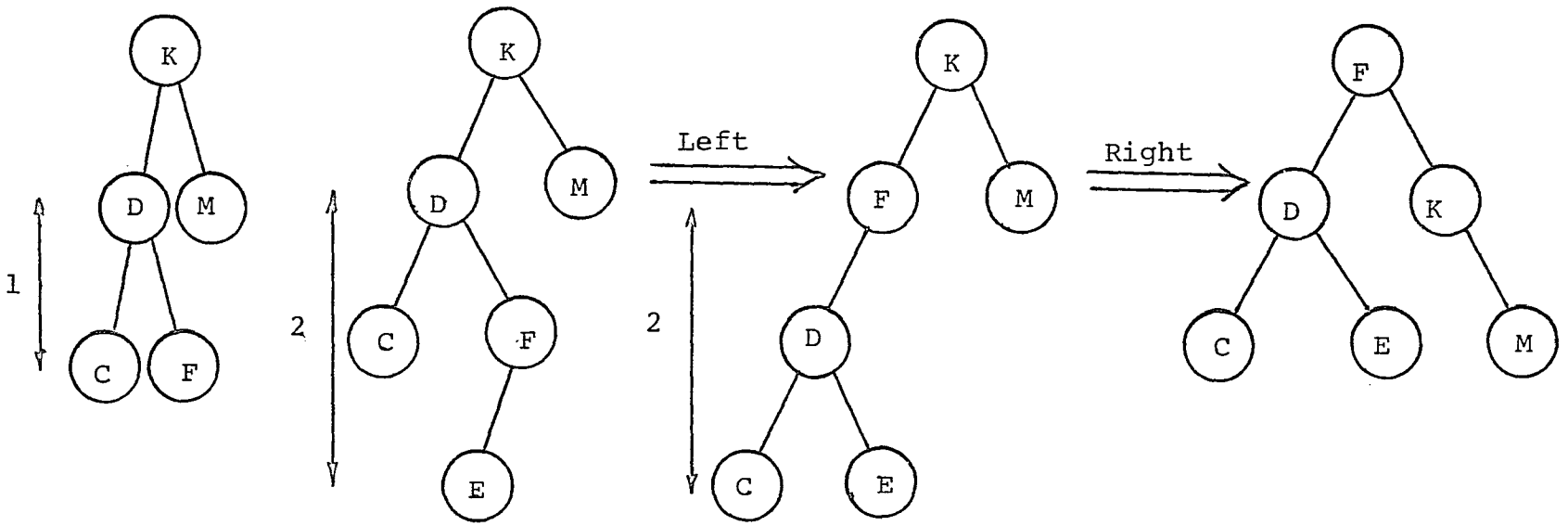


Figure 2.8 A Double Rotation Example

3. The number of nodes visited during the traceback procedure. After an insertion or a deletion the traceback procedure is followed to update nodes' balance factors. Restructurings are performed if any balance factor indicates violation of the tree property.

The result of their simulation study showed that the average number of comparisons made is a logarithmic function of the number of nodes in the tree (the size of the tree). All other statistics reached their asymptotic values for trees which had about 30 nodes or more.

Karlton et al. performed the same experiment on height-balanced k trees. An HB[K] tree is a binary search tree and for every node in the tree the height of the left and right subtrees differ by at most K. The values used for K were 2, 3, 4, 5, 8, and 16. The larger the value of K the smaller is the number of restructures. Also as the experiment results showed, the number of comparisons made to locate a node increases as the balance factor(K) increases. For all HB[K] trees considered, the average number of rotations was independent of the tree size for trees of size 30 or over. As they indicate, the trade-off between the search time and restructure time depends on the value of the balance factor. Hence, the value of the balance factor should be chosen based on the types of operations expected to be performed on the tree.

The sequential processing of records in height-balanced binary search trees is not efficient. The sequential processing requires inorder traversal of tree with each traverse requiring a search. A height-balanced binary search tree has n + 1 null links and 2n links. Perlis and Thorntorn (1960) propose to use null links to point to other nodes in the tree. These pointers, called threads, point to the node which immediately succeed or precede the pointing node in-order (Figure 2.9). The sequential processing of threaded height-balanced binary search trees is efficient since an inorder traversal of the tree does not require search for the successor nodes.

One-sided Height-balanced Trees. Knuth (1973) imposed the following question: "Explore the idea of using the restricted class of balanced trees whose nodes all have balance factors of 0, or +1. (Then the length of the B field can be reduced to one bit.) Is there a reasonably efficient insertion procedure for such trees?"

The B field is the field in which the balance factor (-1, 0, and +1) is stored. In this type of trees the height of the right subtree of every node should always be equal to or at most be one level more than the height of the left subtree.

Hirshberg (1976) calls these trees one-sided height-balanced trees and he presents an algorithm to insert a node in them. Using Hirshberg's algorithm, when a node is being inserted O(logN) levels are involved. A restructuring at a level may disturb the balance of nodes which are one level higher. Each level may require O(logN)

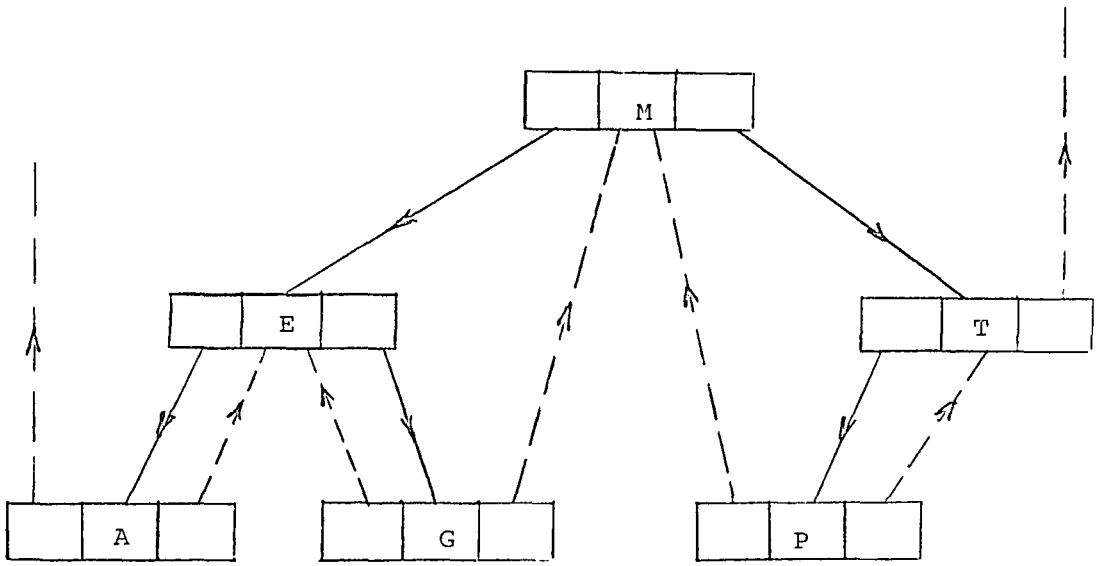


Figure 2.9 A Threaded Height-Balanced Binary Search Tree

restructurings. Therefore, in the worst case $O((\log N)^2)$ restructurings are required for an insertion.

Kosaraju (1978) presents algorithms to insert and delete nodes in one-sided height-balanced trees. For both insertion and deletion, the path from the node (which is a leaf) to the root is traced up to ensure that the tree's property is preserved. In case restructuring is necessary at any level a specific type of rotation is performed. There are O(logN) levels and the number of steps in each level is O(logN). Therefore Kosaraju's algorithm operates in $O((\log N)^2)$ steps.

Ottmann and Wood (1978) and Zweben and McDonald (1978) independently proposed essentially the same solution for deletion in one-sided height-balanced trees. They show that deletion of a leaf will take O(logN) operations since there are O(logN) levels and each level requires an O(1) operation. A property of one-sided height-balanced trees is that the predecessor of a node is the rightmost leaf of that node's left subtree. Using this property they change the deletion of an internal node to the deletion of a leaf which needs O(logN) operations. Hence their deletion algorithm requires O(logN) operations and is independent of the location of the node to be deleted.

Raiha and Zweben (1979) present a solution for insertion to a one-sided height-balanced tree. An arbitrary insertion may change the balance factors of some of the nodes on the insertion path so that restructuring operations should be performed. They use two algorithms; one algorithm is used to keep the tree balanced, (called

Down). (This balancing algorithm was presented by Ottman and Wood (1979)). The other algorithm is used to update the balance factors at each node, (called Up).

Both algorithms have O(logN) complexity. By detailed analysis of the subtree which requires both Down and Up algorithms (the worse case) they propose a solution which at most requires three calls of Down for each call of Up and as such they avoid an $O(\log N)^2$) complexity. Hence, their solution terminates in O(logN) steps.

Raiha and Zweben (1979) indicate that in practice the saving of one bit resulting from using one-sided height-balanced trees instead of height-balanced trees is negligible. Also the algorithms used for deletion and insertion in one-sided height-balanced trees are much more complicated. However, from a theoretical point of view this data structure has been interesting. At present it has been shown that the find, insertion, and deletion operations in both height-balanced and one-sided height-balanced trees can be performed in O(logN) steps.

Bounded-balanced Trees. Or weight-balanced trees are another class of nearly optimal binary search trees. The special characteristic of the bounded-balanced trees is the constraint on the relative size of the left subtree to the size of the tree as a whole. The constraining parameter limits the deviation of bounded-balanced trees from optimality. Hence, the parameter controls the trade-off between the search time and the frequency of tree restructuring required to maintain the property of the tree.

Nievergelt and Reingold (1973) introduce the bounded-balanced trees and give the following definitions:

> The root-balance $\rho(Tn)$ of a binary tree $Tn = (TL, v, TR)$ of $n >= 1$ nodes is $\rho(Tn) = (L + 1)/(n + 1)$.
>
> A binary tree Tn is said to be of bounded balance, or in the set $BB[\alpha]$, for $0 =< \alpha < 1/2$, if and only if either $n = 1$ or, for $n > 1$ and $Tn = (TL, v, TR)$, the following holds:
>
> 1. $\alpha =< \rho(T\eta) = < 1 - \alpha$, and
>
> 2. both TL and TR are of bounded balance α.

In the above definitions "L" represents the number of nodes in the left subtree and "n" represents the number of nodes in the tree. Nievergelt and Reingold also show that:

Internal path length; $|Tn| =< (1/H(\alpha))(n + 1)\log(n + 1) - 2n$,

where

$H(\alpha) = -\alpha\log\alpha - (1 - \alpha)\log(1 - \alpha)$, and

maximum height of $Tn = (\log(n + 1) - 1)/\log(1/(1 - \alpha))$.

The internal path length divided into the number of nodes in the tree determines the average search time. The maximum height of a tree presents a bound on the worst case search time. As such, the above formulas provide bounds on both the average and the worst case search times for $BB[\alpha]$, for any α.

After an insertion or a deletion of a node the balance of the tree might be disturbed relative to the balance factor α. The single and double rotation algorithms which were presented for height-balanced binary search trees can be used to restore the balance.

Nievergelt and Reingold tabulated the performance of random, height-balanced, BB[1 - $\sqrt{2}/2$], and completely balanced trees of n nodes (Table 1). Table 1 shows that the performance of weight-balanced trees is the same as the performance of height-balanced trees.

M-Way Trees. As it was shown before, the length of the worst case search is equal to the height of the tree. To minimize the height of a tree the outdegree (the number of branches leaving a node) should be increased. In this way the structure becomes more bushy and shorter. Loomis (1983) defines a multiway (M-way) search tree as follows:

> An m-way search tree is a tree in which each node has outdegree <= m. When an m-way search tree is not empty, it has the following properties.
>
> 1. Each node of the tree has the following structure: (Figure 2.10)
>
> where the $P_0, P_1, \ldots, P_n$ are pointers to the node's subtrees and the $K_0, K_1, \ldots, K_{n-1}$ are key values. The requirement that each node have outdegree <= m forces $n <= m - 1$.
>
> 2. The key values in a node are in ascending order: $K_i < K_{i+1}$ for $i = 0, 1, \ldots, n$.
>
> 3. All key values in nodes of the subtree pointed to by P_i are less than the key value K_i: $i = 0, \ldots, (n - 1)$.
>
> 4. All key values in nodes of the subtree pointed to by P_n are greater than the key value K_{n-1}.
>
> 5. The subtrees pointed to by the P_i, $i = 0, 1, \ldots, n$ are also m-way search trees.

Figure 2.11 illustrates a 3-way search tree.

Table 2.1 Comparison of Balanced Trees

	Number of comparisons needed to search the tree in the worst possible cases	Expected number of comparisons needed to search an average tree	Time required to return an unbalanced tree to its class
Random trees of n nodes BB[0]	n	1.39 log(n + 1)	0
AVL trees of n nodes	1.44 log(n + 1)	log(n + 1) + 0.25	0(log n)
BB[1 - $\frac{\sqrt{2}}{2}$] trees of n nodes	2 log(n + 1)	1.05 log(n + 1)	0(log n)
completely balanced trees of n nodes (BB[1/2], ignoring semileaves)	log(n + 1)	log(n + 1)	0(n)

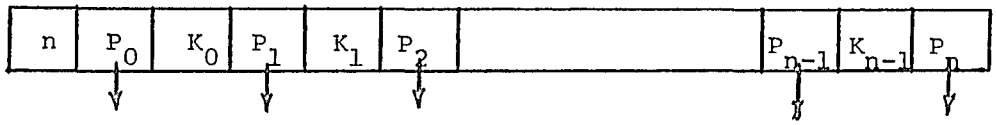


Figure 2.10 Example of an M-Way Search Tree Node Structure

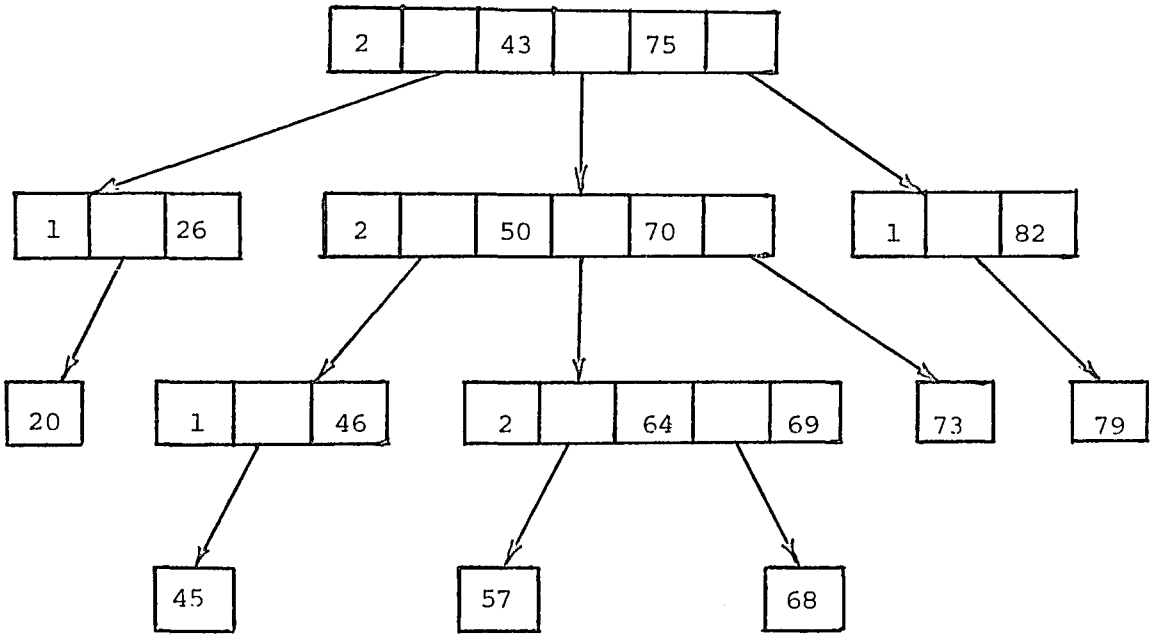


Figure 2.11 A 3-Way Search Tree Example

It can be shown that the minimum height of an m-way tree of n keys is $\log_m(n + 1)$. Therefore the search length of an optimal m-way tree is $O(\log_m n)$. The shorter search length of the optimal m-way trees is a desirable characterisitc when secondary storage is being used. The contents of the nodes on the search path should be accessed from secondary storage. This process is expensive. Since the height of the m-way trees are shorter than their equivalent binary search trees, the number of accesses will also be less.

In dynamic situations where frequent insertions and deletions are made, a great deal of effort is required to maintain an optimal m-way search trees. Like binary search trees there are types of m-way trees which require less maintenance effort than the optimal m-way trees and have search performance which is sufficiently close to optimal m-way search tree's search time. The discussion of the types of m-way trees follows.

B-Trees. Loomis (1983) gives the following definition for a B-tree.

> A B-tree of order m is an m-way search tree with the following properties:
>
> 1. Each node of the tree, except for the root and the leaves, has at least (1/2)m subtrees (and no more than m subtrees).
>
> 2. The root of the tree has at least 2 subtrees, unless it is itself a leaf.
>
> 3. All leaves of the tree are on the same level.

Figure 2.12 illustres a 3-way B-tree.

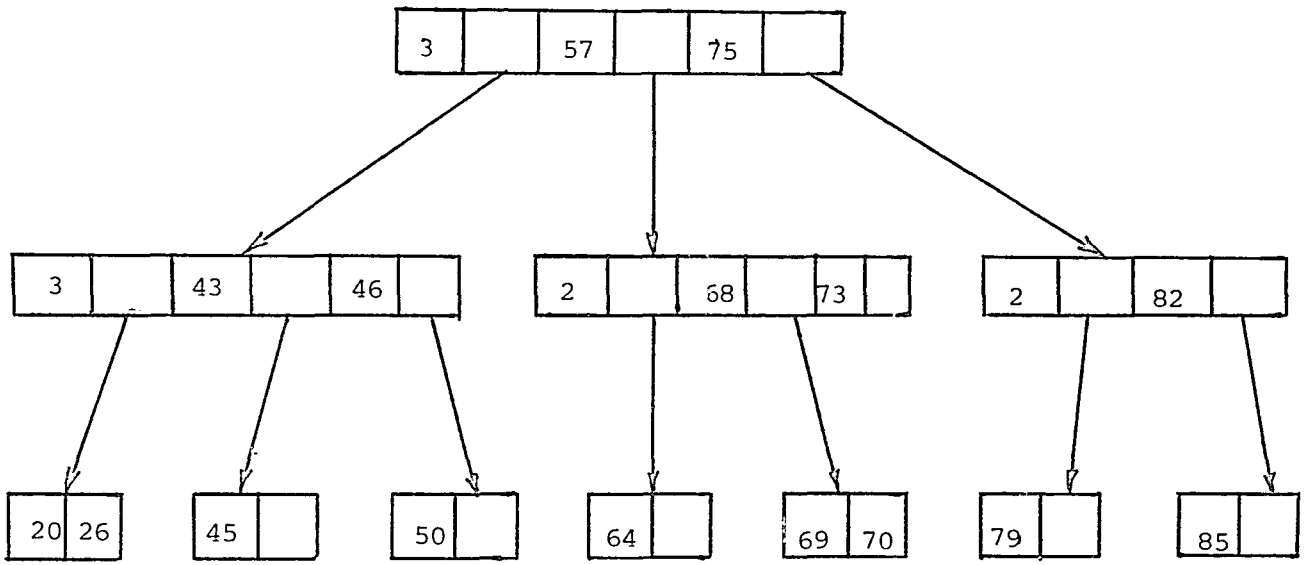


Figure 2.12 A 3-Way B-Tree Example

Bayer and McCreight (1972) present algorithms for finding, deletion, and insertion into B-trees and the costs involved in these operations. Since the leaves of a B-tree should always be at the same level (by definition) the deletion and insertion methods proposed by Bayer and McCreight maintain the B-trees balanced. The find, deletion, and insertion operations will be presented below.

The find operation follows the logic of the find operation in binary search trees. For example, to find the key 45 in the B-tree of Figure 2.10, the search starts at the root. Since 45 is less than 57 the pointer located to the left of 57 is followed to the next node. In that node the pointer between the keys 43 and 46 is followed because 45 is greater than 43 but less than 46. This process continues if other nodes were present. In Figure 2.10 the pointer located between 43 and 46 points to the node containing 45. In case a search is not successful the pointer points to the leaf which should contain that key.

To insert a key into a B-tree the leaf which should contain that key is found first. Two cases are possible. First, if the node has enough space for a key then the key is inserted while preserving the correct sequence of the keys in the node (by definition). Second, if the node is full a split occurs.

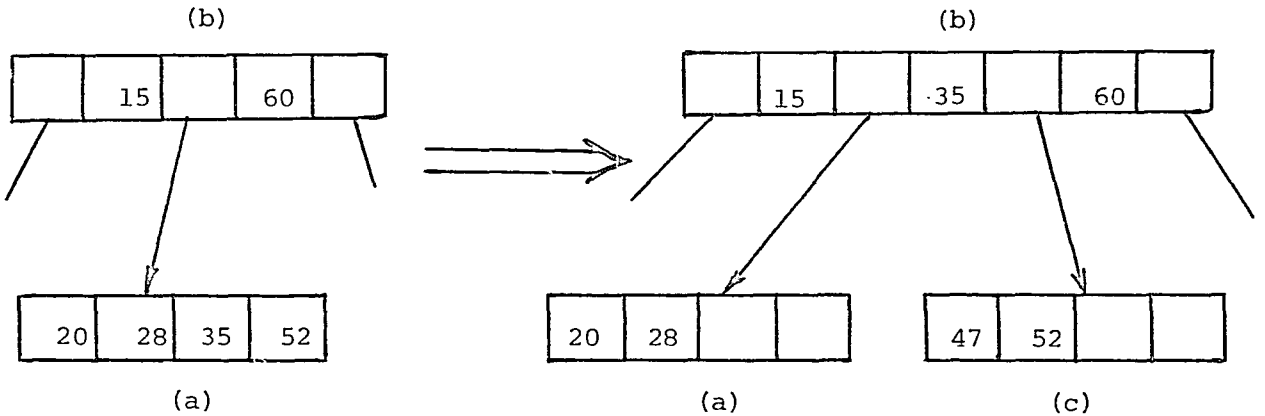
Figure 2.13 shows how a split occurs when a key is inserted to a full node. To insert the key 47 in the full leaf (a) the smaller key values are placed in one node and the larger key values in the other. The remaining key is promoted to the father node (b) as the

separator of the two nodes (a and c). In case the parent node is full another split occurs at that node. The split may continue to the root of the tree and when the root splits the height of the tree increases by one.

To delete a key from a B-tree the location of that key should be determined through a find operation. The following two cases may occur. First, the key resides in a leaf. The key is removed easily but the node should be checked to ensure that at least $\lceil (1/2)m \rceil$ keys reside in that node. If less than $\lceil (1/2)m \rceil$ keys are present a key may be obtained from the neighboring node. To lower the costs in the long run instead of borrowing only one key a redistribution of more than one key might take place if the neighboring node has a few keys more than $\lceil (1/2)m \rceil$ in that node. Figure 2.14 shows how redistribution is performed when the key 62 is deleted.

This redistribution process requires that at least m keys reside in two nodes involved in the redistribution process. If less than m keys are present a concatenation occurs. Concatenation is the opposite of split. The two nodes involved are combined to make a single node. Figure 2.15 shows the concatenation operation. The concatenation may continue to the root and only at that time the height of the B-tree is lowered by one.

Second, if the key resides in an internal node then the successor key should replace that key. The successor key is located at the leftmost position of the leftmost leaf of the key's right subtree. Figure 2.16 shows how the successor of 40 which is 42 can be found.



2.13 B-Tree Node Split

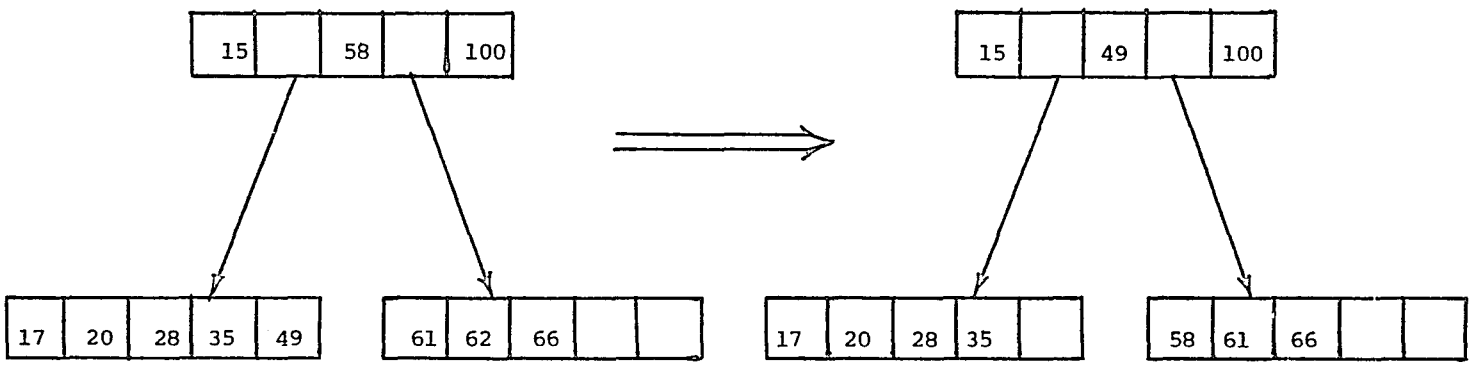


Figure 2.14 Redistribution among Nodes of a B-Tree

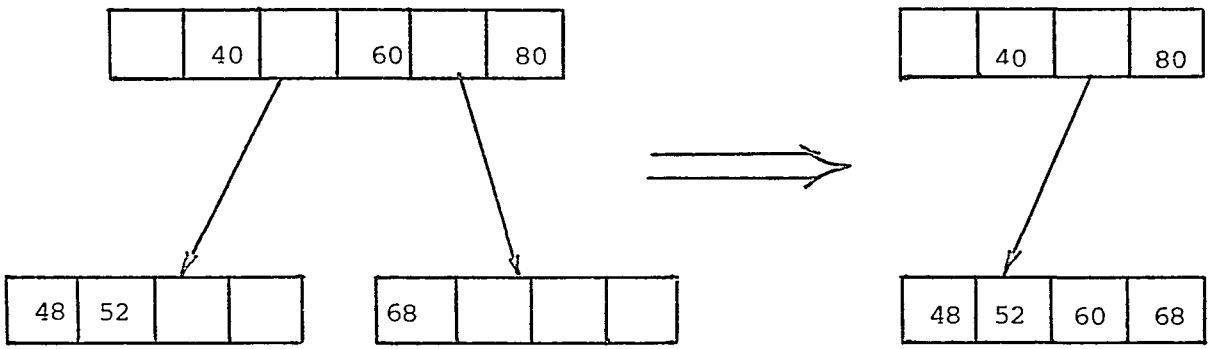


Figure 2.15 Concatenation of a Node of a B-Tree

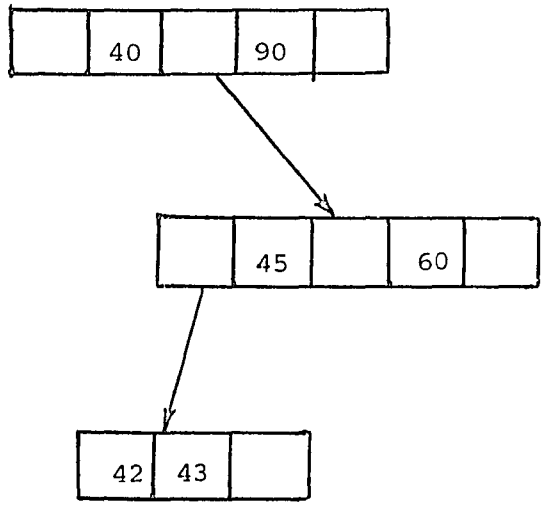


Figure 2.16 Search for Successor In a B-Tree

The leaf should then be checked to ensure that there are at least $\lceil (1/2)m \rceil$ keys present in it. If there are not enough keys in that leaf a redistribution or a concatenation (as discussed above) may be necessary.

Comer (1979) surveys the literature on the B-tree and its variants. Comer shows that the height of a B-tree is $n <= \log_{1/2m}((n + 1)/2)$ and hence the complexity of the find operation is $O(\log_{1/2m}((n + 1)/2))$. He argues that the complexities of insertion and deletion algorithms are $O(\log_{1/2m} n)$. As m (the order of tree) increases the quantities describing find, insertion, and deletion complexities decreases.

B*-Tree. A variant of B-tree is B*-Tree. The nodes of a B*-tree are at least 2/3 full instead of at least 1/2 full. This node property reduces the frequency of node splitting. Insertion into a full node requires a redistribution of keys with a sibling node. If the sibling node is also full then the two nodes will split into three nodes each at least 2/3 full. The restriction on the node size reduces the height of the tree and hence improves the search performance of the B-tree.

The sequential processing of a B-tree requires tree traversal. Since there are a number of keys in each node a node should be accessed many times during the tree traversal. The node accesses from the secondary storage are expensive. Therefore B-tree's sequential processing is not efficient. B^+-tree, which is another variant

of B-trees, is designed to improve the sequential processing performance of B-trees.

In a B^+-tree all keys are located in the leaves in sequence. Leaf nodes are linked sequentially to other leaves. B^+-trees are composed of two parts. 1) The sequence set or the sequence of leaves. 2) The Index part which is composed of the internal nodes. The index part contains keys which are used to direct searches to the proper leaves.

An insertion to a B^+-tree is almost the same as an insertion into a B-tree. The new key is inserted into the proper leaf. Insertion in the leaf is made according to the order of the keys which are already in the leaf. If a split occurs a copy of the rightmost key in low-ordered node is promoted to the parent node as a separator. A deletion from a B^+-tree is easier than a deletion from a B-tree. The key to be deleted needs only be deleted from the proper leaves containing that key. It is not required to delete that key from the index part of the tree.

The search for a key starts from the root of the B^+-tree. The logic of the search is similar to that of search in B-trees except that when the match is found in the index part the proper pointers should be followed to reach the leaf which contains that key. Figure 2.17 shows a B^+-tree.

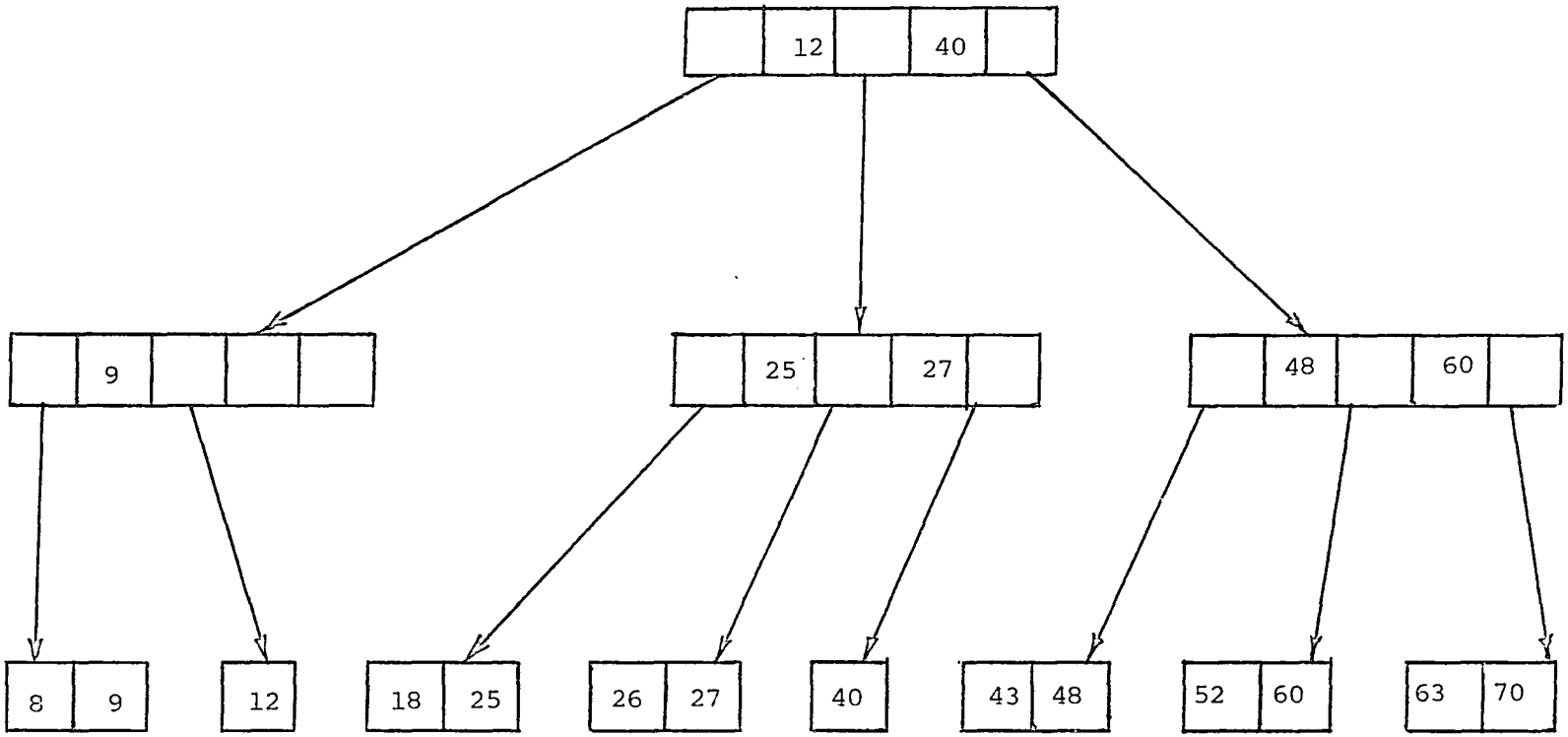


Figure 2.17 B^+-Tree Example

Conclusion

A major goal in the study of data structures is to find a way to represent data structures that is computationally efficient. The types of operations to be performed on a data structure has an important effect of the way a data structure should be represented. In this chapter some of the common data structures and the alternative ways of representing them were presented. Usually a representation resulted in higher performance of one particular operation and lower performance of other operations. The trade-offs between these performances should be evaluated carefully in the process of a selection of a file organization technique.

CHAPTER 3

SURGICAL DEMAND SCHEDULING

A review of literature indicates that surgical demand scheduling is done in two steps; advance scheduling, and allocating scheduling. Advance scheduling refers to scheduling surgeries for a later date. Allocation scheduling refers to sequencing the procedures which should be performed on a particular day assuming that all the patients are in the hospital and ready for surgery.

Advance Scheduling

Advance scheduling procedures are categorized based on the number of constraints considered in scheduling. Single constraint models which are the most common type of advance scheduling models consider total OR time available as the only constraint. Multiple constraint models which are more complex than single constraint models consider total capacity of some other hospital resources such as personnel and bed services available as constraints on scheduling.

Advance Scheduling Under Single Constraint

The single constraint models are categorized based on the allocation of available OR time to physicians. In blocked systems blocks of OR time are reserved for each physician. Procedures for each physician are scheduled within physician's predetermined block.

In nonblocked systems OR time is allocated if available based on some rule; usually first-come, first-served (FCFS).

Nonblocked Systems. The majority of hospitals use nonblocked scheduling systems for advance scheduling. Nonblocked systems schedule surgeries based on FCFS:

1. Until a preset maximum number of cases is reached.
2. Until the OR capacity, as measured by the estimated procedure time is filled.
3. Without limit.

A representative of many nonblocked systems is described by Magerlein (1978) as follows:

> When the surgeon determines that surgery is required, a call is made to the OR and the case is placed on schedule for the specified day. No starting time or room assignment is made when the request is placed since information concerning the entire surgical load for the day in question is incomplete. Given that the schedule is not full, surgeons can continue to call in cases until 2:00 p.m. on the day preceding surgery, at which time the cases are taken in the order received and scheduled for the earliest available time in any available room. (The formation of this tentative schedule is an allocation decision and is considered in more detail below.) The scheduling process continues until all cases are scheduled, with the scheduler estimating the expected procedure time. Once this initial schedule is completed, surgeons may continue to book cases on an FCFS basis until 5:00 p.m. when the schedule is closed, except for emergencies.

This scheduling procedure yields highly variable utilization rates. Other authors (Magerlein 1978) have also observed the high variability of utilization rates and have hypothesized that high variance occurs because an unlimited number of patients can be added to the schedule on any day. The staffing and OR availability is

considered for maximum number of patients in each period. Hence, utilization varies significantly.

Davis (1964) studied the decrease in variation of OR utilization by reducing the variation in the number of operations per week and the range of work load occuring in various days of a week. He used forecasts for the number of operations per week (which was multiplied by the average procedure time to determine required OR time) and workload for each day in that week to schedule staff and OR. The result was reduced variation in OR utilization. Davis then hypothesized that scheduling patients on the basis of expected OR time reduces the variance of OR utilization. Other authors (Magerlein 1978) propose to set a limit on the number of surgeries to be scheduled each day or schedule a specific portion of final schedule by advance schedules and fill the remainder of the final schedule by scheduling patients who are in the hospital but have not been scheduled yet.

Some authors (London 1964), (Williams 1971), (Shelter 1972), and (Phillips 1975) have investigated the reasons for low utilization of OR in hospitals under their study and recommended certain actions to increase the OR utilization. The reasons stated by most of these authors for low OR utilization are: (1) Patients sometimes arrived in the suite too late for surgery to start on time. The main reason given for lateness was that the patient was improperly prepped, not prepared, or not premedicated. Other factors included: floor unaware of time patient was scheduled for surgery, preoperative orders not written; escort staffing problems; name or room number of patient

incorrectly listed. (2) Surgeons were late. (3) Problems of staffing by anesthesiologists and OR nurses. Reasons included: lateness of anesthesia or nursing personnel and shortness of staff. (4) Time lost by last minute cancellations. (5) Other sources of delay were: necessary equipment or medication was not on hand; poor coordination with the x-ray department in scheduling arteriograms; lunch breaks; and arrival of emergency cases.

These studies propose to attack the problem areas individually and based on the severity of the problem. Generally they propose to study the variation in the surgery work load, expansion or contraction of OR facilities, transportation times from various nursing units to surgical suite, and communication between the surgical suite and floor units and admission department.

A major drawback of nonblocked systems is high degree of competition among physicians for OR time. Physicians schedule cases far in advance. Long waiting lists develop and possibility of cancellations increase. The competition is more tense for morning OR schedules. The start time of morning schedules is more reliable than start time of afternoon schedules because of variations in procedure time. The preference for morning OR schedules over afternoon schedule results in lower OR utilization in afternoons. Another disadvantage of high competition for OR time, as stated by Stewart (1971), is that highly elective cases (need for surgery is known far before the time of surgery) are scheduled far in advance leaving small

amount of OR time for less elective cases such as orthopedic, thoracic, and neuro-surgery to be scheduled. To overcome the drawback of competition for OR time blocked booking systems have been used.

Blocked Booking. In blocked booking systems a block of OR time is assigned to each physician on a periodic basis. The duration of a block is determined on the basis of past experience according to the amount of time used for elective and emergency cases and the OR time available. The assigned block is reserved for the physician's use until a cut-off time after which the unused portion of block will be assigned to other physicians who need OR time.

The scheduling system described by Morgan (1972, 1973) typify blocked booking systems--although details might be different. The office of each physician would schedule cases two months in advance. Scheduling would be accepted by the booking clerk at the hospital until 48 hours of the actual day of surgery. If free time of the total time allotted to the surgeon was available at that time, this would be used by physicians needing additional hours to that allotted to them.

Morgan (1972, 1973) reports major findings from changing nonblocked scheduling to blocked scheduling in the hospital under study. Under nonblocked system OR utilization was low and a long waiting list was present. Also anesthesiologists and OR staff were idle for long periods during each day. After the blocked system was

implemented, the OR utilization was higher, waiting list was much shorter, the number of surgeries was higher, and anesthesiologists and OR staff were busy most of the time. This was done without the need for additional OR nurses, but some OR equipment had to be acquired because of excess demand for them.

To increase the effectiveness of blocked scheduling systems in OR utilization, Barnoon and Wolfe (1968) propose to group blocks and assign a longer block to each physician instead of assignment of shorter blocks which are scattered over a week. However, if a block is too long the physician gets tired, a condition which is undesirable.

Blocked scheduling systems have some advantages over non-blocked systems. OR utilization is high in a blocked system. Part of the increase in OR utilization is the result of better OR utilization in the afternoons. A block in the afternoon gives physicians a reliable start time of procedure. In this respect a morning schedule would not be preferable to afternoon schedule anymore. Some physicians might be willing to perform surgeries in the afternoon and keep office hours in the morning. This shift of preference from a morning to an afternoon schedule results in higher OR utilization in the afternoon.

Another result of the shift from morning to afternoon is the decrease of competition for morning schedules. Physicians would not schedule their cases far in advance and therefore the number of cancellations would be less. The decrease in number of cancellations is responsible for some part of the increase in utilization of OR.

Also responsible for the increase in OR utilization is better use of the OR in each block. Proponents of blocked scheduling systems have hypothesized these advantages which have not been validated by any empirical evidence.

There are some possible disadvantages associated with blocked systems (Magerlein 1978). First, if the cut-off time is too near the date of surgery, the unfilled portion of block may remain unscheduled. Second, surgeons might keep the unused portion of the block for the more urgent cases that might arrive. Surgeons might keep patients in the hospital who have not been scheduled for surgery to fill the unused portion of their blocks. This increases the length of stay of these patients. Finally, some urgent cases might have to be delayed until the patient's surgeon is scheduled to operate.

Zimmerman (1963) studied the distribution of unused OR time by time of day and day of week and correlated the unused segments to the assigned allotments of specific surgeons and services in a hospital. He also analysed the data for the reasons for unused OR time. Unscheduled OR time was a major portion of unused time. The range of unscheduled OR time was 40 minutes to an entire day. This factor alone accounted for about 85 percent of unused OR time. Another reason was found to be internal delays. Although these occured with high frequency their duration were about 15 minutes or less. Other reasons for unused OR times were cancellations and late physician arrivals.

The studies concerned with advance scheduling with OR time as the only constraint are valuable because they point up the potential problems with operations of operating rooms. These studies are limited in some respects: First, the measures of performance used in these studies are limited to utilization of OR, number of surgeries performed over a period, and the amount of delays between cases. These measures concentrate on the OR alone and do not consider areas outside OR -bed and nursing services. Second, the scheduling systems presented above do not consider availability of hospital resources other than OR. Advance scheduling systems with multiple constraints consider other resources of a hospital.

Advance Scheduling Under Multiple Constraints

An alternative to scheduling surgical patients on the basis of OR time is to schedule both medical and surgical patients through an admission scheduling system which considers other resources such as occupancy level and nursing services' availability. This type of schedule improves the overall performance of a hospital. Some of the admission scheduling systems discussed in the literature do not consider surgical patients. For example, Young (1965) presents a model based on queueing theory that helps anticipate inpatient bed occupancy. This information permits a more effective allocation of hospital bed services.

Average daily census is roughly a function of average length of stay and admission rate: $C = \gamma.\bar{t}$

c = the average daily census

γ = admission rate

$\bar{t}$ = the average length of stay

Young states that although census can be controlled to some extent through the control of average length of stay by means of discharge policies, efforts to date have concentrated on admission procedures. In Young's model the decision as to how many patients can be admitted (S) is based on the decision level (B) and the state of the census (N); S = B - N. Young shows how one might investigate the effects of various B levels on the average occupancy, the expected overflow, the effects of delays in scheduled arrivals, and the size of the call list or backlog to be maintained to make sure sufficient patients are on hand to be able to schedule to the preset B level. However, Young's model does not consider the interaction of census to other areas of a hospital.

Offensend (1972) presents a model based on Markov processes to compare scheduling systems based on patient census and nursing work load. The following policies were compared:

Policy I - delay the admission of the nonurgent patients whenever the census exceeds a certain level.

Policy II - delay the admission of the nonurgent patients whenever the work load of the patients already in the ward is above a certain level.

His examples show that great savings are realized when the nursing work load policy is used. The improved performance of the system is because nursing work load policy considers patient differences as to the amount of nursing services required. In scheduling systems based on census the work load on nursing services varies in different periods because of differences in patients' demands for nursing services. Since the amount of nursing resources is determined roughly by the magnitude of peak demand on nursing services and since personnel costs comprise 70 percent of a hospital's budget the census policy is more expensive than nursing policy. Offensend concludes that hospital planning should focus on the work demands and not solely on patient census.

The admission scheduling system proposed by Robinson, Wing, and Davis (1968) considers surgical patients implicitly by assignment of special attributes to them. Using their simulation model they investigated six scheduling systems for one set of cost ratios for empty bed, overflow, and turnaways. The scheduling systems investigated were: (1) Constant number of admissions for each day (filled-page). (2) Admission based on the expected length of stay (ELOS) where a patient is scheduled for admission on the earliest requested day in which his presence in the hospital will not cause the expected census to exceed some previously defined limit. There were four degrees of estimate precision within this system. (3) Admission based on expected length of stay with probability table (ELOS-PT). This

includes information about the conditional probability distribution of actual length of stay, given the expected length of stay. Instead of increasing the expected census by one each day of the ELOS, the expected census is increased by the probability that the patient will still be in the hospital on that specific day. The values of the parameters for the lowest cost for each system was found. The parameters were the size of the page for the filled-page system, and size of the hospital for the ELOS and ELOS-PT systems. Their simulation model can be used for sets of cost ratios which describe costs in other institutions.

The model proposed by Connors (1970) is based on the assumption that patients' inconveniences and inefficiency in hospital census are quantifiable and a cost figure can be determined from these measures. Patient inconvenience is related to the difference between the patient's scheduled admission day and the set of admission days he/she prefers. The operating efficiency of the hospital is assumed to be related to the deviation of actual occupancy from a desired occupancy specified by the hospital administration.

The scheduling process proposed by Connors is highly sophisticated. Based on a probability density function of length of stay for a patient, for each day in a planning period, the probability of exceeding the limits of the desired hospital census and service census if the admission is made on that date is calculated. For each day, to satisfy the constraints on census and required services the rooms whose most probable co-occupants have attributes compatible with the

patient are identified. For each feasible room the probability of exceeding the room's capacity if the proposed admission is made is calculated. Then the costs of the feasible admission dates are determined which are then screened to find the lowest cost. The algorithm used in this model does not affect the operating room schedule.

Models of Robinson et al. and Connors are mathematically complicated and therefore have not been implemented. Lack of validation in a real environment limits the expansion of these approaches into the surgical admission scheduling area.

The admission scheduling and control system presented by Hancock et al. is relatively simple. It has been implemented in some hospitals. The desirable results of the implementations make it a potential system for extension into the surgical scheduling in more detail.

The objective of this scheduling system is to minimize cost by maintaining maximum use of beds subject to the following constraints:

1. Cancellations do not exceed an acceptable number.
2. Turnaways do not exceed an acceptable number.
3. Weekend census does not exceed policy.

The system consists of a set of policy rules called allowances such as medical allowance and surgical allowance which are the maximum number of patients in each category that can be scheduled in each day. Using simulation they determined allowances that met the

constraints. The result in their study hospital was 94.1 percent average census with no emergency turnaways, cancellation rate of less than one per week, and all elective surgery was scheduled.

Except Offensend's nursing work load policy, other systems that were presented for advance scheduling under multiple constraints use census as the scheduling criteria. Scheduling a number of surgical patients with no account for their demand on other services results in large fluctuations in work load in other service areas.

Allocating Scheduling

Allocating scheduling models assign start time and specific rooms to procedures in a day to which patients have been assigned by advance scheduling systems. Allocating scheduling is important since the way OR and start times are assigned to procedures significantly affects the utilization of OR. The allocating scheduling systems presented have not been implemented, therefore, to compare them and to analyze them researchers have used simulation models.

Goldman, Knappenberger, and Moore (1969) performed a simulation study to compare three priority rules for allocation of OR to surgical procedures. Also two levels of expediting was considered. The priority rules selected were first-come, first-served (FCFS), longest-cases-first (LPT), and shortest-cases-first (SPT). FCFS was chosen because it is used by most hospitals. LPT was chosen to avoid overtime. If a case scheduled at the beginning of a day runs much longer than expected, there is a good chance that the following

shorter cases can be more easily fitted into available time slots in other rooms. SPT was chosen because it results in early process of some of the procedures and therefore the load on recovery room is spread more uniformly over the day.

Some of the important assumptions made by their model are:

1. There will be sufficient operating room staff, ancillary facilities, and hospital beds so as not to restrict the flow of patients through the operating suite.

2. All cases scheduled for a day will be performed that day unless cancelled.

3. Emergency cases will be placed in the next available room.

4. All patients seeking surgery remain in the system until surgery is completed unless cancelled for medical reasons.

5. All cases can be performed in any operating room.

6. The historical relationship between the surgeon's estimate of the procedure time and the actual procedure time will remain constant.

Several criteria were used to judge the effectiveness of the policies. These were: percent utilization, the average elapsed time per room per day, the total overtime per day, the percent of unused time between cases, the average number of cases expedited per day, and the average number of patients waiting.

A statistical analysis on each of the eight criteria was carried out. The analysis showed that scheduling policies had a

significant effect upon percent utilization, average total overtime per day, and the average number of delayed cases transferred per day. The LPT scheduling policy resulted in the highest utilization, the lowest total overtime per day, and the largest number of delayed cases transferred. The opposite effects were observed for the SPT. The FCFS policy produced intermediate results in all three variables. They conclude that although it may be difficult to implement a scheduling procedure which always schedules the longest cases first, the result of this study suggest that an effort should be made to schedule the longest cases first.

Kwak, Kuzdrall, and Schmitz (1976) conducted a simulation study to compare the following five scheduling policies.

1. Random input to surgery: This is the policy that is the baseline for comparison of alternative policies.

2. Preemptive priority: Patients requiring recovery room facilities are given a priority; that is, they are operated on first without regard to the length of the surgery to be performed.

3. Inverted lineup: Patients requiring the longest surgery are served first. Those not requiring the use of recovery room are operated on last, again with descending length of surgery time setting the criterion for the scheduling.

4. Patients requiring the longest surgery are served first, but no queue discipline is applied to those not requiring recovery room facilities.

5. Surgeries with duration more than two hours are assigned priority on the basis of the longest surgery first. Those not requiring recovery room facilities are served last, with the remainder surgeries served after the first group.

The measures of performance were the utilization rates of each facility, the time at which the last patient left each facility, the maximum contents of the recovery facility, and any queueing that might have taken place prior to patient's entering the recovery facility.

The analysis of the simulation output showed that there is a highly significant difference between the baseline policy and the alternate policies used. Improvements in operating and recovery room utilization rates, increased patient load in the recovery room during a shorter day, and more timely completion of the surgical and recovery schedule were demonstrated by employing each of the alternative policies tested. Longest-cases-first based policies yield higher OR utilization rates. Policy four performed best in utilization of operating and recovery rooms. They conclude that using one of the LPT policies the manpower requirements for recovery room could be reduced and more surgeries could be performed with the same number of recovery room personnel.

In both simulation studies by Goldman et al. and Kwak et al. the longest-cases-first policies were shown to perform best in increasing the operating room utilization. The implementation of this

policy might face resistance especially from physicians who have specific preferences about the timing of their procedures.

Other issues which are important in scheduling surgeries have been studied by researchers. Schmitz and Kwak (1972), and Kuzdrall, Kwak, and Schmitz (1974) developed a simulation model to analyze the effect of increase in number of beds on the operating room and recovery room times required. They assumed that as the number of beds increases the demand for operating room and recovery room by different types of cases will increase proportionately. Based on the empirical data on the length of stay of different types of surgeries in the operating room of the hospital under their study they tested and found that length of stay in an operating room is a negative exponential distribution. Their simulation model is flexible. It might be used to determine the number of operating and recovery rooms necessary to satisfy the present surgical demand on a hospital.

Also Barnoon and Wolfe (1968), Fetter and Thompson (1965), and Goldman and Knappenberger (1968) developed simulation models for the operating room to investigate facility sizing. Barnoon and Wolfe studied OR and staff requirements in a blocked booking system which was also constrained by the number of beds available in the hospital. Fetter and Thompson's study was part of an effort to prepare a collection of simulation models to analyze all important areas of a hospital. Goldman and Knappenberger studied how to determine the number of operating rooms required for different levels of demand.

Whitston (1965) used a queueing model to analyze several methods for staffing the surgical suite.

A major drawback of present surgical demand scheduling methods is that the start time of procedures is determined in two steps as mentioned before. In the first step a preset number of patients are scheduled for each day without regard to their demand on different service areas of the hospital. In the second step a major resource such as operating room time is allocated to the scheduled patients. There is no guarantee that all the scheduled patients will be assigned a procedure start time by the allocation scheduling because number of patients is a crude measure of resource requirements.

Demand for operating room time and other resources is usually different for one patient from demand for resources for another patient due to differences in types of cases and also differences among cases even within each type. Hence, it is sometimes not feasible to schedule surgeries for all patients that are admitted to the hospital in the same day. This results in an increase in the length of stay. In some days the patients' demand for operating room time might be low such that a major portion of the capacity of the operating room remain unused. This results in low utilization of the operating room.

The review of literature indicates that most of the present surgical demand scheduling systems use operating room capacity as the only criterion for scheduling demand for surgery. An implicit assumption in these systems is that all other resources of a hospital

are available when they are needed and that the only binding constraint is the capacity of operating room.

Capacity of hospital resources are fixed at least in the short run. Demand for hospital resources fluctuates in different periods. A waiting line develops for those services for which the demand for them in a period exceeds their capacity in that period. In effect those activities in the waiting line and their successors are delayed. The severity of the delays depends on their effect on the costs associated with the delay. For example, suppose all necessary material, personnel, facility and equipment for a procedure are present except a special equipment which is not available. The procedure is delayed until the special equipment becomes available. This results in low operating room utilization and high costs to the patient.

Another reason for delays is lack of consideration of patient preparation activities' lead times and resources' procurement lead times. None of the present surgical demand scheduling methods consider these lead times when scheduling decisions are made.

Delays caused by ineffective scheduling procedures affect other control variables and operations of a hospital. Delays result in increasing the length of stay of patients in the hospital. The unnecessary increase in the average length of stay reduces the capacity of some hospital resources such as beds. This in turn results in the delay of admission of other patients and hence reduction of

the number of surgeries over a period. The decrease in demand for surgeries results in low operating room utilization.

Delays are also a major reason for the cancellation of procedures. When a procedure is cancelled the operating room would not be utilized. This reduces the utilization of operating room.

To start the procedure on time, all of the required resources and the patient should be in the operating room and ready for the operation. This can happen only when the predecessor activities such as preparation of the patient for the operation had been started so that their finish times coincide with the start time of the procedure. Also, orders for all the resources such as personnel and equipment should have been issued with respect to their acquisition lead times and their availability so that their acquisition times coincide with the start time of the procedure. Hence, scheduling demand for surgery can be viewed as project scheduling under constrained resources.

The only project scheduling technique which explicitly considers resource limitations and the resource acquisition lead time is the Critical Path Method - Material Requirements Planning technique.

CHAPTER 4

DEVELOPMENT AND DESIGN OF FILE ORGANIZATION METHODS FOR A CPM/MRP TYPE PROBLEM

Similar to the Critical Path Method-Material Requirements Planning method, the Operating Room Scheduling System presented in this study is composed of three major elements.

1. Surgery projects' bill of components file which specifies the components and the number of units of each component required to accomplish each surgery included in the file. The term component is used here to include activities and resources such as labor, in addition to materials.

2. Inventory file which contains information concerning the availability of resources over the planning horizon. Inventory file also contains information on gross requirements in each period, order release times and order quantities for activities and resources.

3. The operating room scheduling algorithms which is a set of procedures followed to find a feasible start time for any surgery procedure.

Surgery Projects' Bill of Components File

The single-level format is used to represent the parent-child relationships in the project bill of components (Table 4.1). In this

Table 4.1. Project Bill of Components for the Example Surgery

ACTIVITY #	DURATION	COMPONENTS	QUANTITY	KIND	TYPE	DESCRIPTION
1	20					OPEN HEART
		203	1	L	1	
		205	3	L	1	
		204	4	L	1	
		201	1	F	0	
		202	1	F	0	
		101	1	M	1	
		102	1	M	1	
		103	1	M	1	
		2	1	A		
		4	1	A		
		5	1	A		
2	1					PATIENT TO OR
		204	1	L	1	
		3	1	A		
3	2					PREPARE PATIENT
		204	2	L	1	
		202	1	F	0	
		6	1	A		
4	2					PREPARE PHYS 1
		204	1	L	1	
		6	1	A		
5	2					PREPARE PHYS 2
		204	1	L	1	
		6	1	A		
6	0					START

research, parents can be activities only. Parents are listed without indentation. Their duration and description accompany activity numbers on the same row. For example, activity one in Table 4.1 is the open-heart surgery and it takes 20 periods to perform.

Children of each parent are indented and follow the parent. Children can be activities, storable resources, non-storable resources, or a combination of these categories. Information about each child such as the child's number, amount required, and whether it can be acquired from outside sources if there is none on hand, follow the child on the same row. In Table 4.1, for example, activity number one requires one unit of labor number 203. The one in the last column indicates the item can be acquired from outside sources if there is not enough quantity of labor 203 when it is required. A zero in that column indicates the material cannot be acquired from outside sources.

The data in the projects' bill of components file may be used to prepare the project structure tree (Figure 4.1). The project structure tree is a schematic diagram of the project bill of components. The structure tree shows parent-child relationships and the quantity of each child required by its parent. Figure 4.1 shows the project structure tree for the example project of Table 4.1. An activity may occur in different parts of the project structure tree. Repetition of an activity does not mean that it will be scheduled more than once. When scheduling the project, an activity is scheduled only once.

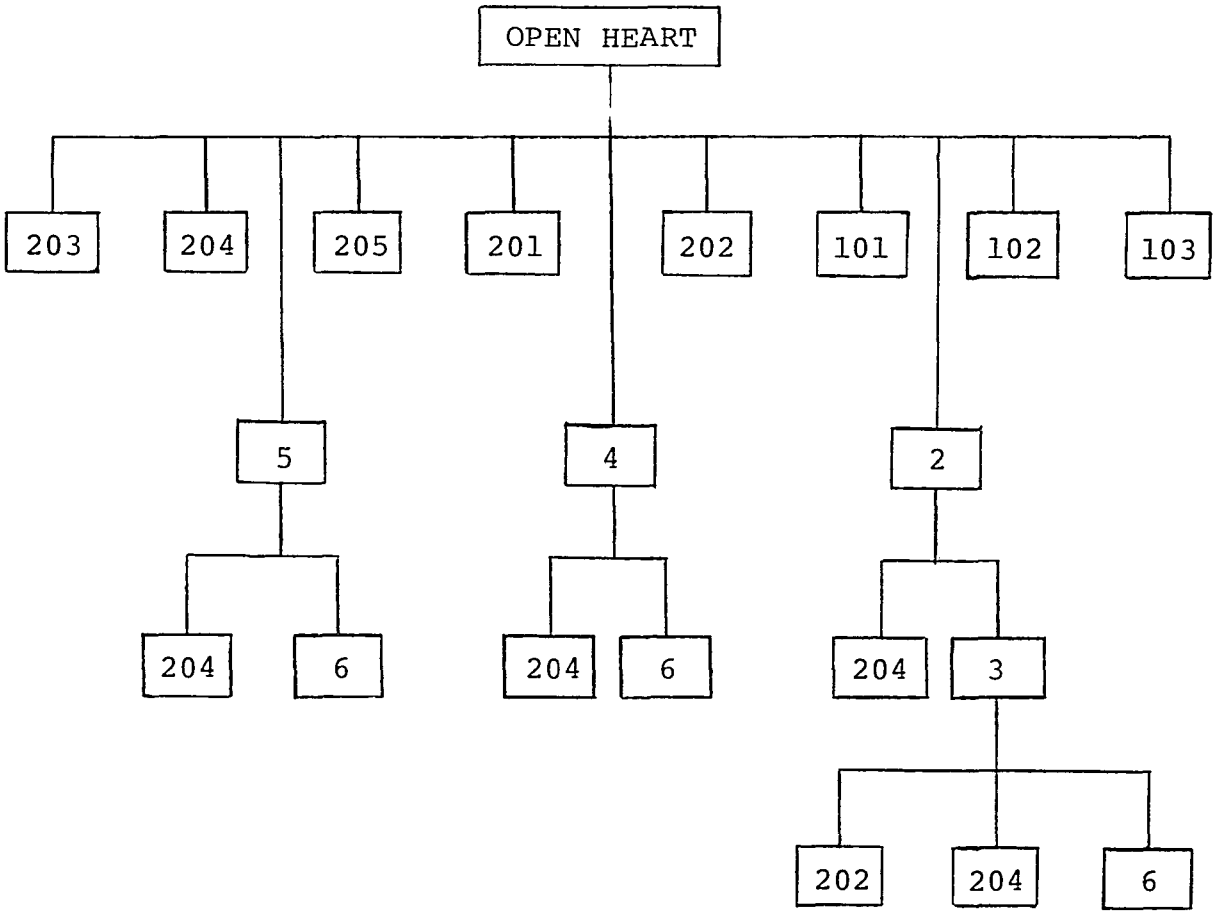


Figure 4.1 Surgery Structure Tree for the Example Surgery

The numbering system in both the project bill of components and project structure tree assigns the lowest number to the highest level--the last activity in the project. The largest number is assigned to the lowest level in the project--the start activity. An activity number, therefore, is a crude measure of an activity's level compared to other activities in a project.

Numbers assigned to activities are always smaller than numbers assigned to resources. The last activity is always activity number one. Predecessors of the last activity are numbered two, three, four, etc. The first activity's number is equal to the number of activities in the project. Numbers assigned to resources are larger than the number assigned to the first activity. In this research, numbers one through 99 are reserved for activities. Resource numbers may be any number larger than 99.

Inventory File

The inventory file contains information about the availability of resources, gross requirements, and the order release quantities over the relevant time horizon. The length of the time horizon depends on how far in advance surgeries are scheduled. In this research, the time horizon is assumed to be one month.

Two categories of resources are involved in surgical projects. The distinction between the two categories of resources is important. The storable resources may be stored in one period and be used in later periods. For example, operating room material, such as bandages,

can be stored for future use and each unit of it can only be used once.

Non-storable resources cannot be stored for future use. If they are not utilized in one period, their utility is wasted for that period but they may be utilized in future periods. For example, if an equipment is available for use in one period, but it is not used in that period, its utility for that period is wasted, but it is still available for use in future periods.

The distinction between the storable and non-storable resources implies that they should be treated differently with respect to scheduling and organization of data. As far as the resources are concerned in scheduling an activity, all of the storable and non-storable resources required by the activity should be available. The major concern for storable resources is that the total amount of each resource required by the activity should be available at the start time of the activity. The non-storable resources, on the other hand, should be available over the duration of the activity. For example, suppose the duration of an activity is three periods and it requires five units of a storable resource and an equipment; to schedule this activity, the five units of the storable resource should be available at the start time of the activity, but one equipment should be available in all of the three periods. At the finish time of the activity, the storable resource is used up but the equipment is freed and can be utilized in other activities.

As shown by the example above, besides the difference in treatment in scheduling process, the storable and non-storable resources differ in the way they influence the level of resource availability in other periods. The allocation of a non-storable resource in a period does not affect its availability in any other period. The allocation of a storable resource, on the other hand, might affect its availability in other periods in the planning horizon. The difference in the properties of the two categories of resources is an important factor in organization of data which will be discussed next.

Non-storable Resources

The allocation of these resources in a period does not affect their availability in other periods. Figure 4.2 shows the availability of a non-storable resource over 15 periods. Suppose one unit of this resource drops by one unit in period four, but the level of the resource does not change in other periods. Figure 4.3 shows the level of the resource after one unit of that resource is allocated in period four.

It is possible to store the quantity on hand for each period, but, the resource level may remain the same for several periods. To reduce the redundancy, it is proposed to store the beginning and ending period numbers of those intervals over which the level of resource is constant. For example, it is sufficient to have three pieces of data including one, four, and three to show that there are three units available over periods one through four. The keys to a record include

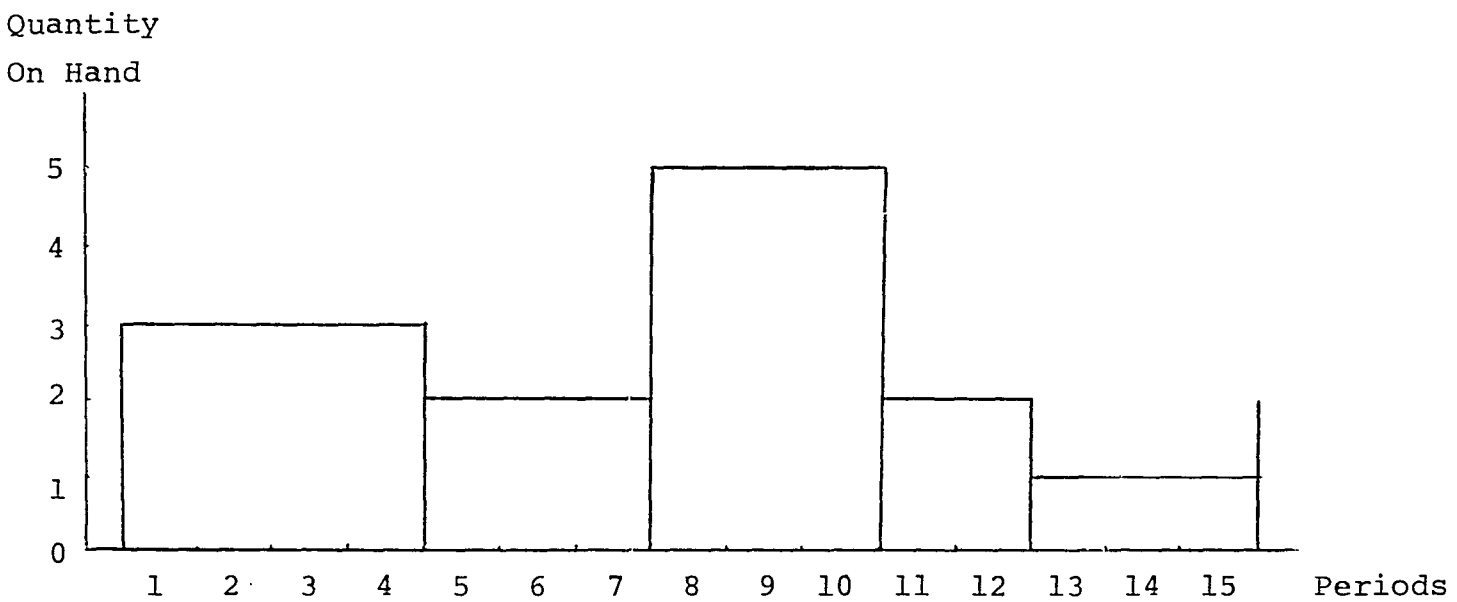


Figure 4.2 Availability of a Non-Storable Resource

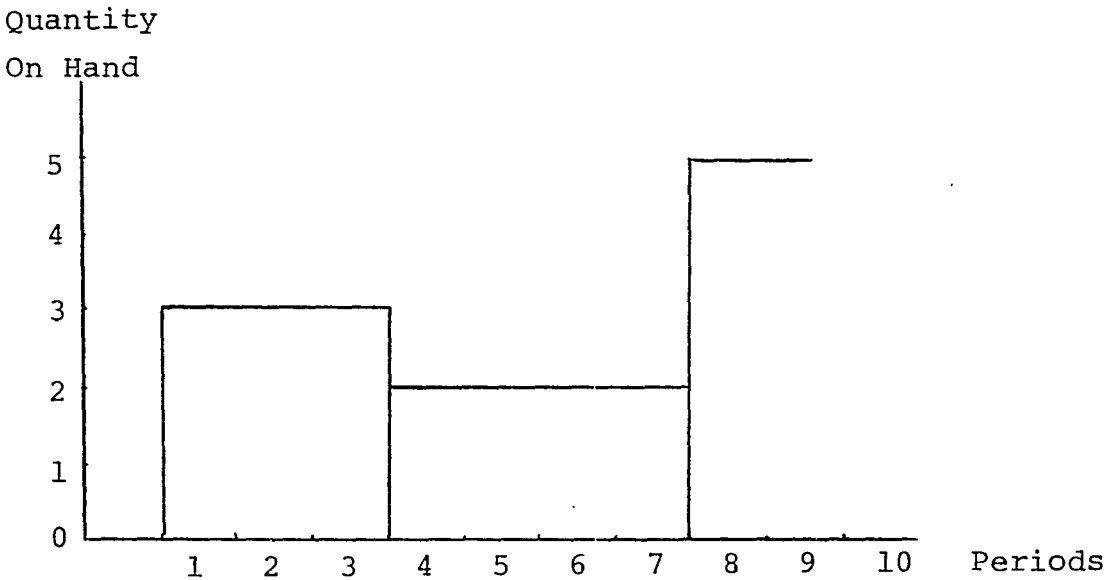


Figure 4.3 Updated Non-Storable Resource Availability

two data elements - one and four in this example. The smaller key is called the left key (LKEY) and the larger key is called the right key (RKEY).

New records are created in a variety of situations such as: when the utilization takes place in a period between the two keys (beginning and ending periods) of a record; the record should be segmented to three records. For example, consider the non-storable resource whose availability is represented in Figure 4.2. Allocation of one unit of the resource in period two results in conversion of the record 1-4-3 to the records 1-1-3, 2-2-2, and 3-4-3 as shown in Figure 4.4.

To compare the resource requirements for an activity with the quantity on hand of those resources, the period numbers included in the duration of activity are searched for. Required comparisons are made, and new records are generated. Both direct and sequential search are performed. Direct search is performed to find the key corresponding to the start time of an activity. Having found the keys to a record which includes the start time of the activity, sequential search is performed to find the keys to a record which includes the finish time of the activity. The sequential search is required, since the resource availability check should be performed for all periods included in the duration of the activity. When a new record is created, it should be inserted into the file. Hence, the operations performed on the file are: direct and sequential search, sequential processing, and insertion.

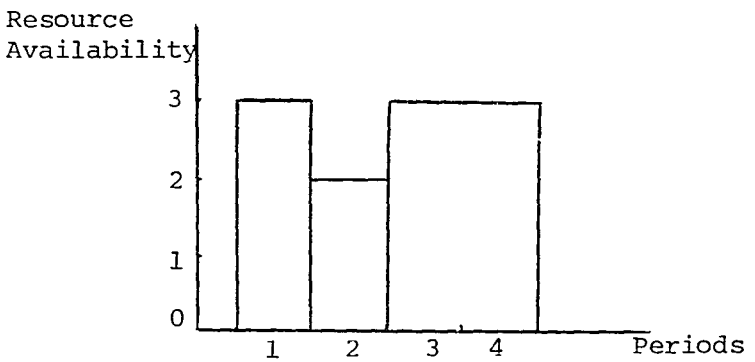


Figure 4.4 Record Changes Resulted from an Allocation

To efficiently perform the operations presented above, the keys to the records (period numbers) can be considered as an ordered list. The operations to be performed on this list are both direct and sequential search, and insertion of keys and their related information. As was presented before, the data structure threaded height-balanced binary search trees have satisfactory performance regarding these operations. This study proposes the use of this data structure to store non-storable resources records.

The linked representation is used to represent this tree structure. Figure 4.5 shows the threaded height-balanced binary search tree equivalent of the data in Figure 4.2. There are two keys present in each node. Most often the start and the finish time of an activity happen in different periods. Node(s) containing the start time and the finish time of the activity should be found. To compare the availability of a resource with its quantity required for an activity, and/or, to allocate some units of the resource to an activity, the nodes containing the periods over which the activity is performed should be accessed.

Search and Insertion

The search starts from the root and continues as follows: (Most of the notations used to present procedures for threaded height-balanced binary search trees search, insertion, and balancing are the notations used by Knuth (1973) in "algorithm A (Balanced tree search and insertion)").

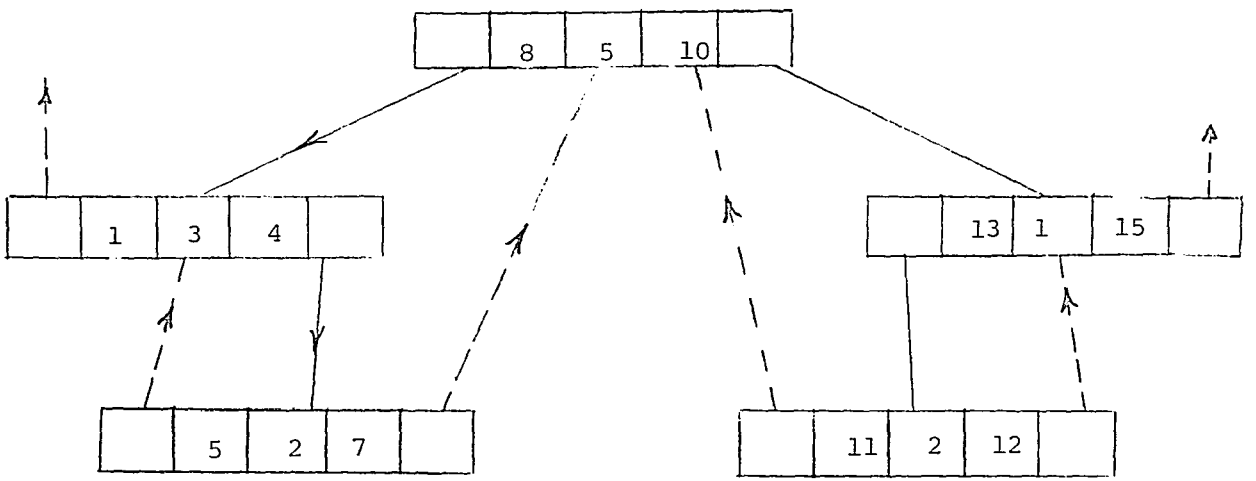


Figure 4.5 Threaded Height-Balanced Binary Search Tree Representation of the Example Non-Storable Resource

A1. [Initialize]. Set T= HEAD, S= P= RLINK (HEAD). The pointer P moves in the tree; S points to the node where rebalancing may be necessary and T points to the parent of S. RLINK (j) is the pointer to the right child of node j, HEAD is the header node with its right link pointing to the root of the tree.

A2. [Search]. If LF(a)<= RKEY(P), go to A3.

ELSE IF LS(a)> RKEY(P), go to A4.

ELSE IF LS(a)< LKEY(P), the late start time of the activity is less than the left key of the node, and the late finish time of the activity is larger than the right key of the node. The search for the node containing the late start time is performed first. The immediate predecessor of node P is examined and the process of checking the predecessor nodes continues until the node containing the late start time is found. Procedure A7 is followed to find predecessor of node P. Then the search for the node containing the late finish time of the activity is performed. The immediate successor of node P is examined and the process of checking the successor nodes continues until the node containing the late finish time of the activity is found. Procedure A6 is followed to find the successor node of node P.

ELSE IF LS(a)= LKEY(P) the late start time of the activity is equal to the left key of this node. The late finish time of the activity is larger than the right key of this node. The search for the node containing the late finish time of the activity continues. The immediate successor of node P is examined first and the process of checking the immediate successors continues until the node containing

the late finish time of the activity is found. To find the successor node of node P, procedure A6 is followed.

ELSE the late start time of activity is between the node's two keys and the late finish time of activity is larger than the right key of the node. The search for the node containing the late finish time of the activity continues. The immediate successor of node P is examined first and the process of checking the immediate successors continues until the node containing the late finish time of the activity is found. To find the successor node of node P procedure A6 is followed.

A3. [Search continues]. IF LF(a)< LKEY(P) nodes containing the keys are located in the left subtree of this node. The left subtree of this node should be searched. Procedure A5 is followed to move to left.

ELSE IF LF(a) = RKEY(P) the node containing the late finish time is found. The search for the late start time of the activity continues in this node and the predecessor nodes.

ELSE IF LS(a)= LKEY(P) the node contains the late start time of the activity too. The search is terminated.

ELSE IF LS(a)< LKEY(P) the search for the late start time of the activity continues in the predecessor nodes. Procedure A7 is followed to find predecessor nodes until the node containing the late start time is found.

ELSE the late start and the late finish times of the activity are in the interval closed by the left and right keys of node P.

A4. [Move right]. The right child of node P is accessed and pointers are adjusted if necessary. Set Q= RLINK(P).

If B(Q) = 0, set P= Q

ELSE, set T= P, S= Q, and P=Q.

A5. [Move left]. The left child of node P is accessed and pointers are adjusted if necessary. Set Q= LLINK(P).

If B(Q) = 0, set P= Q

ELSE, set T=P, S= Q, and P=Q.

A6. [Find successor]. This routine is the mirror image of routine "Find predecessor." To find the immediate predecessor of node P;

If right subtree of node P is empty, set I = RLINK(P), otherwise set Q = RLINK(P) and repeat setting Q = LLINK(Q) until a node Q which has empty right subtree is found. Set I = Q. Node I is the immediate successor of node P.

A7. [Find predecessor]. If the left subtree of node P is empty then the left link of node P is a thread pointing to the immediate predecessor of node P. If the left subtree of Node P is not empty, the node with the largest key in the left subtree of P is the immediate predecessor of node P. In a threaded binary search tree, the right link of the immediate predecessor of node P is a thread pointing to node P. Therefore, to find the immediate predecessor of node P;

If the left subtree of node P is empty, set I = LLINK(P), otherwise set Q = LLINK(P) and repeat setting Q = RLINK(Q) until a

node Q which has empty right subtree is found. Set I = Q. Node I is the immediate predecessor of node P.

The algorithms described above find node(s) containing the late start time and the late finish time of an activity. The search procedure may be performed for two purposes.

1. During the non-storable resources availability check procedure, where the intention is to make sure there is sufficient quantity of a particular resource on hand over the duration of an activity. In this case, the level of resource on hand in relevant node(s) is compared to the required quantity and decisions are made as to whether it is possible to schedule the activity or not. The relevant node(s) are those nodes which contain the late start time and/or late finish time of the activity and all nodes with the left and right keys smaller than the late finish time of the activity and larger than the late start time of the activity.

2. During the update procedure, where the intention is to update the level of a non-storable resource to account for the allocation of that resource to the activity being scheduled. In this case the level of non-storable resource allocated to the activity being scheduled should be reduced by the amount required by the activity. The level of the resource is reduced for those periods over which the activity is performed. The reduction may result in a discrepancy in the resource level over the interval represented by a node. Depending on the late start time and the late finish time of the scheduled

activity, as related to the keys of the node(s) which include(s) them, the following cases may result.

1. The late start time and the late finish time of the activity coincide with the left key and the right key of a node. The resource availability is reduced for all periods represented by the node and no discrepancy results.

2. The late start time of the activity coincides with the left key of a node and the late finish time of the activity is less than the right key of the same node. The reduction of the resource level over the earlier periods represented by the node creates a discrepancy in the resource level. The node should be segmented into two new nodes. One to represent those periods over which the reduction takes place and the other one to represent the rest of the periods in that node.

3. The late finish time of the activity coincides with the right key of a node and the late start time of the activity is larger than the left key of the same node. This case is the same as the second case described above except that the reduction of the resource level takes place over the later periods represented by the node. In this case also the node is segmented into two new nodes.

4. The late start time and the late finish time of the activity are larger than the left key of a node and smaller than the right key of the same node. The allocation of resource creates discrepancies in the resource level over the periods represented by the node. The node is divided into three new nodes. The resource

level in two of these nodes remains the same as the original node. One of the two nodes represents the time interval from the left key of the original node to the start time of the activity. The other node represents the time interval from the late finish time of the activity to the right key of the original node. The node which has a different resource level than the original node, because of resource reduction, represents the time interval corresponding to the duration of the activity.

5. The late start time of the activity coincides with the left key of a node and the late finish time of the activity coincides with the right key of a successor node. The level of resource in the relevant nodes are updated but no new nodes are created. The relevant nodes are those nodes which contain the late start time and/or late finish time of the activity and all nodes with the left and right keys smaller than the late finish time of the activity and larger than the late start time of the activity.

6. The late start time of the activity coincides with the left key of a node and the late finish time of the activity is between the left and the right keys of a successor node. The resource level of the nodes including the node containing the late start time of the activity and all successor nodes up to the node containing the late finish time of the activity are updated. The node containing the late finish time of the activity is segmented into two new nodes. The resource level of the new node which represents the time interval included in the duration of the activity is updated. The resource level of the other new node remains the same.

7. The late finish time of the activity coincides with the right key of a node and the late start time of the activity is between the left and the right keys of a predecessor node. The resource level of the nodes including the node containing the late finish time of the activity and all predecessor nodes up to the node containing the late start time of the activity are updated. The node containing the late start time of the activity is segmented into two new nodes. The resource level of the new node which represents the time interval included in the duration of the activity is updated. The resource level of the other new node remains the same.

8. The late start time of the activity is between the left and the right keys of a node and the late finish time of the activity is between the left and the right key of a successor node. The two nodes containing the late start time and the late finish time of the activity are divided into two segments each. The resource levels of the two new nodes which represent the time intervals included in the duration of the activity are updated, while the resource levels of the other two nodes remain the same. Also nodes which lie between the two nodes containing the late start and the late finish times of activity in an inorder tree traversal are updated.

When new nodes are created, they should be inserted in the right location in the tree. The new nodes are created only as a result of node segmentation. The new nodes, therefore, can be inserted as the immediate successor to the segmented node. To insert a node as the immediate successor of a node two cases are possible.

1. The right subtree of the node is empty. The new node is inserted as the right child of the node. For a particular non-storable resource the following changes in the tree takes place. Set RLINK (AVAIL) = RLINK(P), RLINK(P) = AVAIL, LLINK(AVAIL)= P. The new node is a leaf and its left and right pointers are threads. AVAIL is the row index for the next available empty row in the table. Information about a new node is stored in that row.

2. The right subtree of the node is not empty. The immediate successor of the node is the node containing the smallest key in the right subtree of the node. The left subtree of the immediate successor node is always empty. The new node is inserted as the left child of the immediate successor node as follows:

Set LLINK(AVAIL)= LLINK(Q), LLINK(Q)= AVAIL, RLINK(AVAIL)= Q. Q is the row index of the immediate successor node. To find the successor node of a node "find successor" routine which was described above is followed.

A new node is inserted. The length of some subtrees are affected by this insertion. The balance factors of the nodes which lie between nodes S and the new node should be updated as follows: If RKEY(AVAIL)<LKEY(S), set R= P= LLINK(S), otherwise set R= P= RLINK(S). Then repeat the following operation until P= AVAIL: if RKEY(AVAIL)<LKEY(P), set B(P)= -1 and P= LLINK(P); if RKEY(AVAIL) >LKEY(P), set B(P)= +1 and P= RLINK(P).

The insertion of a node may create an unbalanced tree. The balancing procedure is performed to keep the tree height-balanced.

If RKEY(AVAIL) < LKEY(S) set a = -1, otherwise set a = +1. The following cases are possible.

1. If B(S) = 0, set B(S) = a. No balancing is required.
2. If B(S) = -a, set B(S) = 0. No balancing is required.
3. If B(S) = a, if B(R) = a go to step A8, if B(R) = -a go to step A9.

A8. [Single rotation]. In the single rotation, node R becomes the new root and node S becomes a child of node R. Set P = R.

If LINK(-a, R) is a thread to node S, set LINK(a, S) = R as a thread, LINK(-a, R) = S as a child.

ELSE, set LINK(a, S) = LINK(-a, R), LINK(-a, R) = S as a child. The balance factors for both nodes S and R are adjusted as B(S) = B(R) = 0. Go to A10.

A9. [Double rotation]. In double rotation, node P becomes the root of the subtree where rebalancing occurs. Nodes S and R become the children of node P. The original children of node P, if any, become children of nodes S and R. Links of nodes S and P may need to be changed from regular pointers to threads or vice versa.

Set P = LINK(-a, R).

If LINK(a, P) is a thread to node R, set LINK(-a, R) = P as a thread, LINK(a, P) = R as a child; otherwise, set LINK(-a, R) = LINK(a, P), LINK(a, P) = R.

If LINK(-a, P) is a thread to node S, set LINK(a, S) = P as a thread, LINK(-a, P) = S as a child; otherwise, set LINK(a, S) = LINK(-a, P), LINK(-a, P) = S as a child.

The balance factors at nodes S, R, and P should be adjusted as follows:

If B(P) = a, set B(S) = -a and B(R) = 0

If B(P) = 0, set B(S) = 0 and B(R) = 0

If B(P) = -a, set B(S) = 0 and B(R) = a

then set B(P) = 0 and go to A10.

A10. [Head adjustment]. The pointer to the root is adjusted as follows:

If S = RLINK(T) then set RLINK(T) = P; otherwise, set LLINK(T) = P.

As an example, suppose one unit of the resource whose availability is represented in Figure 4.5 is utilized in period two. New nodes are created because of the discrepancy in the level of resource in period two and its adjacent periods. Figure 4.6 shows the restructuring of the tree to preserve its balance. At first one new node is inserted and a rotation is performed. Then the other new node is inserted. After each insertion restructurings are performed if tree properites are violated.

In the scheduling process, whenever a check on the on hand level of a resource in a specific period is necessary the on hand data should be accessed. Necessary comparisons are made and if some units of that resource are allocated, the on hand data should be updated. In the example of Figure 4.2, period four is found first. Then, since there were more than one unit of that resource available

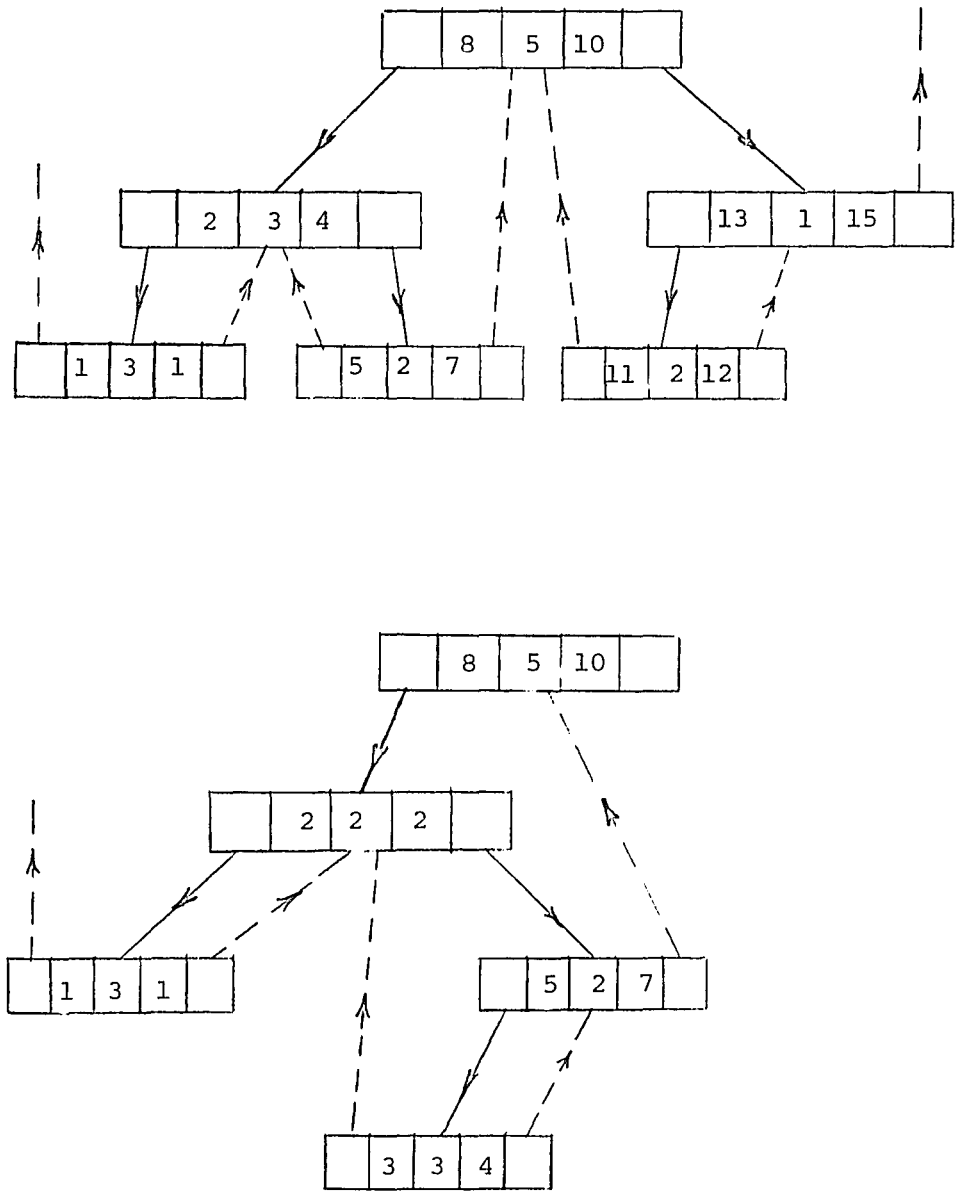


Figure 4.6 Restructuring of the Example Tree

in period four, one unit is allocated and the resource level is updated to account for the allocation.

Since the allocation of a non-storable resource in a period does not affect the resource availability in other periods, it is quite possible that the resource level in a period can be lower than the resource level in an earlier period in the planning horizon. In the scheduling process, when sufficient quantity of a non-storable resource is not available in a period, it might be available in an earlier period in the planning horizon. Therefore, it might be feasible to shift the start time of the activity which requires the resource to an earlier period.

The non-storable resources required by an activity should be available over the duration of that activity. The amount of a non-storable resource required by activities of a project in a number of time intervals over the project duration may be constant. Similar to the non-storable resource availability file, the beginning and ending period numbers of each interval and the resource requirements over that interval is stored in the non-storable resource requirement file.

When a requirement arises, new records may be inserted to indicate the requirement over a particular interval. The operations required to accomplish this are: "search" and "insertion." The requirements for each non-storable resource by all activities of a project are stored in a linked list. The search operation on a linked list is not efficient, but, since there are only a few records

in a list, the search time is short. The insertion into a linked list is very efficient.

Figure 4.7 shows the linked list representation of the resource requirements for a non-storable resource. Suppose requirements arise for 10 units of the resource in periods 14 and 15. The linked list is updated as shown in Figure 4.8. The interval represented by period numbers 10 and 18 is segmented to three sections. Two new records are inserted into the list and the links are updated to preserve the period number order.

Storable Resources

The quantity on hand of storable resources in each period is determined by adding the quantity on hand at the beginning of that period (at the end of last period) to the amount receipt in that period. The quantity on hand over some periods remains the same if there is no receipt over those periods. It is sufficient to have data on those periods in which receipts occur.

Figure 4.9 shows the level of a storable resource in periods one through ten. Receipts occur in periods three and seven. The data on quantity on hand in each period can be calculated using the data on the quantity on hand in period one and quantity receipts in periods three and six. Hence, there is no need for the redundant data of periods two, four, five, six, and so on.

It is sufficient to store the periods in which the receipts occur and also the quantity on hand after the receipts occur. The

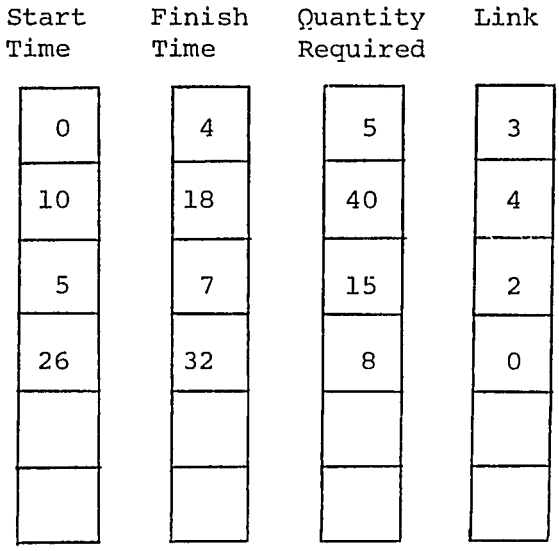


Figure 4.7 Linked List Representation of Resource Requirements for a Non-Storable Resource

Start Time	Finish Time	Quantity Required	Link
0	4	5	3
10	13	40	5
5	7	15	2
26	32	8	
14	15	50	6
16	18	40	4

Figure 4.8 Updated Linked List for the Example Non-Storable Resource Requirements

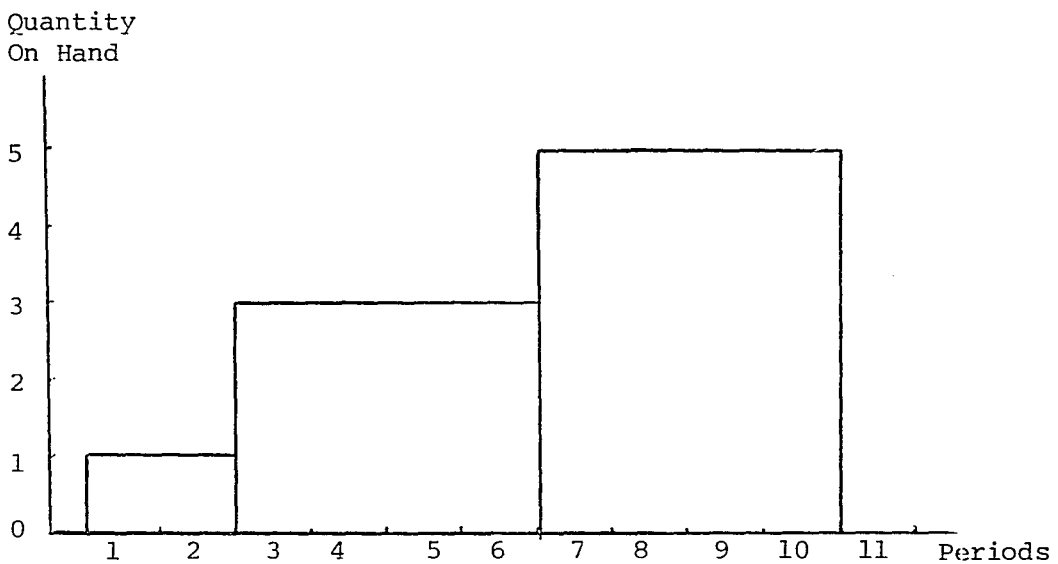


Figure 4.9 Availability of a Storable Resource

number of periods in which receipts occur is not usually large, therefore, records for only a few periods will be stored.

The keys to these records are period numbers. These keys constitute an ordered list. The operations that are performed on this ordered list are search and sequential processing. Sequential search is performed when the key to the record including data concerning the availability of the resource at the start time of an activity is needed. The sequential processing on this ordered list is required because the allocation of a number of units of a storable resource results in the reduction of the same number of units from the future periods in the planning horizon. Therefore, the list should be sequentially updated each time a transaction occurs. The allocation might also influence the level of resource in earlier periods which also requires sequential processing.

An ordered list is used to store the on hand quantity of storable resources. The sequential processing of ordered lists is very efficient. The sequential search operation on ordered lists is not efficient, but, since the number of elements in the list is not large, the search time will not be long. Figure 4.10 is an ordered list representation of the data in Figure 4.10. Two adjacent data in the time list show the beginning period and the period up to which the level of resource is constant. Suppose one unit of that resource is used in period five. Period five is searched for first, and the list is updated then. To update the resource availabilities, one is subtracted from quantities on hand in all future periods. That one

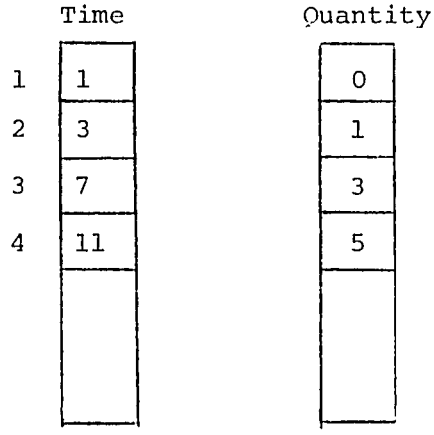


Figure 4.10 Ordered List Representation of the Example Storable Resource

allocated unit will not be available in periods three and four. Figure 4.11 shows the updated list after necessary changes are done.

The quantity on hand of a storable resource in each period is determined by adding the quantity on hand at the beginning of that period (at the end of last period) to the amount receipt in that period. The quantity on hand of a storable resource in a period is always greater than or equal to the quantity on hand of that resource in an earlier period. The units of a storable resource allocated in any period will not be available for use in any other period. The allocation of a storable resource in a period always reduces the quantity on hand of that resource in all future periods. The quantity on hand in the earlier periods might be reduced because of the allocation.

The number of earlier periods affected depends on the quantities and times of the receipts. Generally, the periods included in the interval from the last period in the planning horizon back to the last receipt period for which the quantity allocated is less than or equal to the sum of receipts within allocation period and that period will be affected by an allocation.

Figure 4.12 shows the level of a storable resource in periods one through ten. Receipts occur in periods three and seven. The quantity on hand is two in periods one and two, five in periods three through six, and seven in periods seven through ten. Allocation of one unit of this resource in period seven reduces the quantity on hand in periods seven through ten but it will not affect any earlier periods (Figure 4.13a). Allocation of one unit in period eight

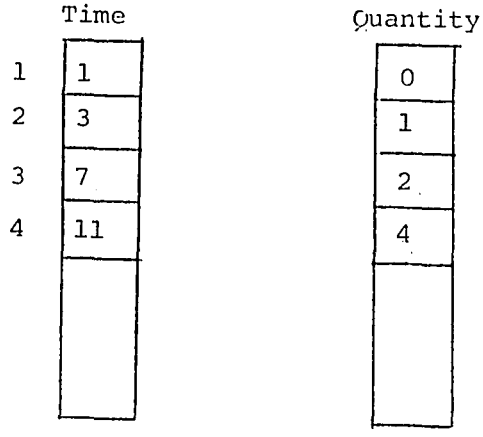


Figure 4.11 Updated Ordered List for the Example Storable Resource

Figure 4.12 Availability of a Storable Resource

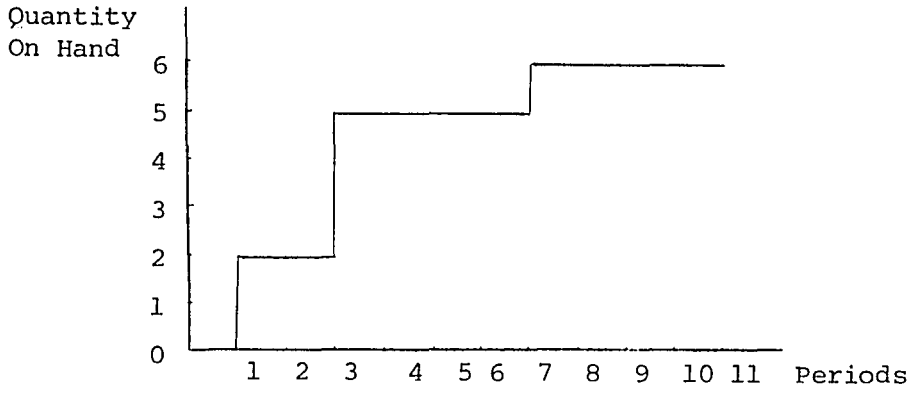


Figure 4.13a Updated Availabilities of the Storable Resource Example

reduces the quantity on hand in periods eight through 10 as well as period seven which is one period earlier (Figure 4.13b). Finally, allocation of six units in period eight reduces the quantity on hand in all periods in the planning horizon (Figure 4.13c).

The requirement for a storable resource may arise in only a few periods during a surgery project. When a requirement arises, the record indicating the requirement may be inserted into the requirements file. The operations required to accomplish this are; search and insertion. The requirements for each storable resource by all activities of a project are stored in linked lists. The search operation on a linked list is not efficient, but, since there are only a few records in a list, the search time is short. The insertion operation on a linked list is very efficient. Figure 4.14 shows the linked list representation of the resource requirements for a storable resource.

To insert the requirements for five units of the same resource in period 12, since period 12 is between periods 10 and 15, the links are updated to represent order of the period numbers. As shown in Figure 4.15, period 10 is linked to period 12 and period 12 is linked to period 15.

In its present format, the inventory file contains data concerning the resource availabilities, the list of scheduled surgeries and resources allocated to them for 31 days. To cope with the random nature of the type of surgery and the proposed start time for the

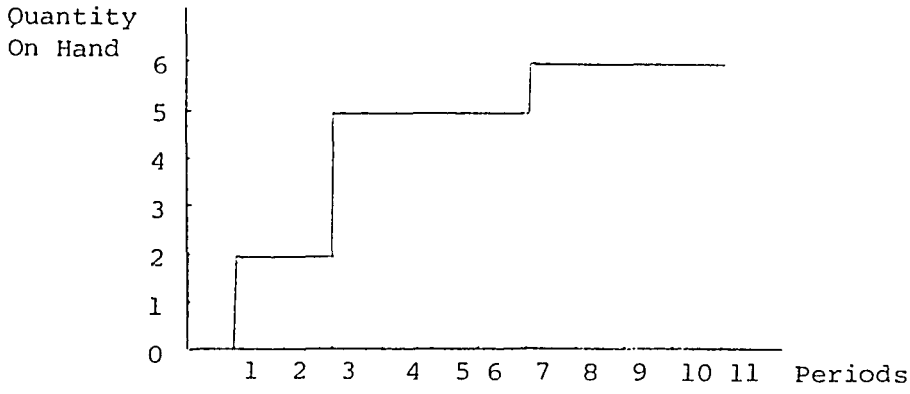


Figure 4.13b Updated Availabiltiies of the Storable Resource Example

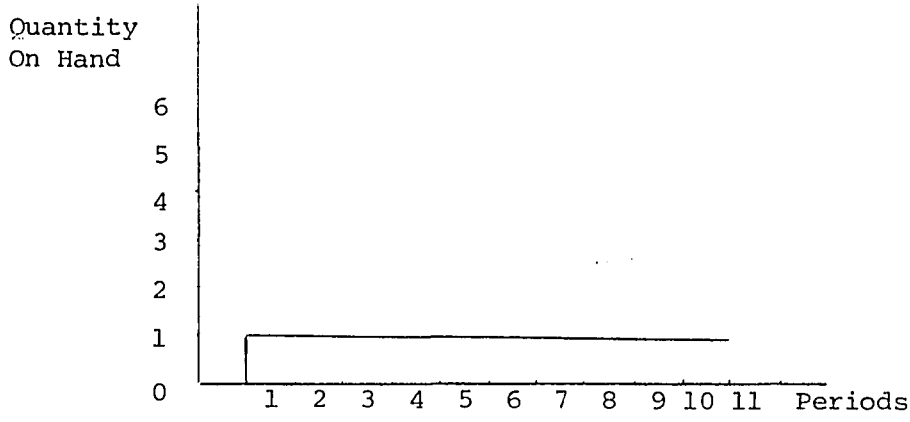


Figure 4.13c Updated Availabilities of the Storable Resource Example

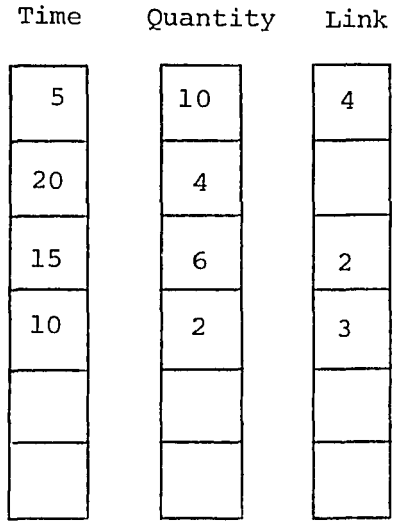


Figure 4.14 Linked List Representation of Resource Requirements for a Storable Resource

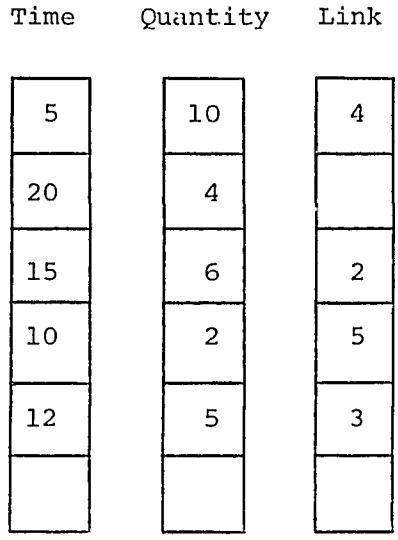


Figure 4.15 Updated Linked List for the Example Storable Resource Requirements

procedure, a direct access storage device is used to store inventory file.

The organization of this file is relative. Relative organization is an efficient method when data is accessed randomly. The day number in the planning horizon is the key to data concerning that day and the location of a resource in the list of resources is the key to the quantity on hand of that resource in a specified day. Therefore, given the location of the resource on the list and the day of surgery, the data pertaining to that resource can be accessed.

Both storable and non-storable resources are of two types:

1. The resource can be acquired from outside sources if there is not enough of it on hand.
2. The resource can not be acquired from outside sources if there is not enough of it on hand.

The above classification is time dependent. The quantity of some resources is fixed at least in the short run. For example, if there is a need for an operating room and none is available it is not possible to order for it and receive it from outside sources. It is possible to change the level of some of resources in the short run by transfers and overtime or undertime. For example, if there is a need for a nurse in a period and there is none available it may be possible to transfer a nurse from another department to the surgery department or ask a nurse for overtime.

Surgery Scheduling

A project is usually composed of three categories of components with different characteristics. The properties of each category dictates the way they should be treated in the scheduling process. To distinguish each category the following notations are used in the projects' bill of components file.

A = activity

M = a storable resource

L = a human element in the project such as a physician, nurse, or a technician

F = a non-storable resource except any human elements

The scheduling process starts with a proposed start time (T) for a surgery in a given day. The surgery's bill of components and the inventory records for the components of that surgery are accessed. The algorithm explodes the project's bill of components. The algorithm trys to find a feasible late start schedule. The following notations are used to formulate the late start schedule algorithm.

For activity a:

LF(a) = Late finish time for a

LS(a) = Late start time for a

For resource r:

AVAIL(r, t) = on hand quantity of resource r in period t

REQ(a, r, t) = requirements for resource r at time t for activity a

The process starts with the last activity in the project.

To schedule the last activity the storable resources are checked for their availability at the start time of the activity. The non-storable resources required by the activity are checked for their availability over the duration of the activity. The following cases may result.

1. All resources are available when they are needed.

   The activity is scheduled.

2. One or more storable resources are not available.

When a storable resource is not available in a period, it is not available in previous periods. This is a characteristic of storable resources. Future periods are searched to find a period in which enough of the storable resources are available.

The activity is postponed to the period in which enough of the required storable resources are available. This period is the earliest period in which the activity can be started. A sign is posted to prevent efforts to schedule the activity in an earlier period. Surgery scheduling starts all over again.

3. One or more non-storable resources are not available.

When a non-storable resource is not available over the duration of the activity, it may be available in either earlier or future periods. The earlier periods are tried first to find an interval equal to the activity's duration over which enough of that resource is available. If the search for the interval is successful, the proposed start time of the activity is moved to the earlier period found by the search procedure.

If there is not enough of the resource in previous periods, future periods are scanned to find an interval equal to the duration of the activity over which enough of the required resource is available. If the search is successful, the proposed start time of the activity is postponed and a sign is posted to warn that this proposed start time is the earliest possible start time because of resource limitations.

Every time there is a change in the start of the activity all of the resources are checked for their availability when they are needed. The recheck procedure makes sure the activity start time is a feasible start time with respect to all resources required.

When the last activity is scheduled the inventory file is updated to allocate the resources required by the activity.

For storable resources,

AVAIL(r, t) = AVAIL(r, LS(a)) - REQ(a, r, LS(a))

Where t starts at the earliest time at which the requirements can be satisfied from receipts;

AVAIL(r, LS(a)) - REQ(a, r, LS(a)) > AVAIL(r, t)

and it finishes at the time of the first receipt after the late start time of activity a;

AVAIL(r, t) > AVAIL(r, LS(a))

For the rest of the periods in the planning horizon;

AVAIL(r, t) = AVAIL(r, t) - REQ(a, r, LS(a))

For non-storable resources;

AVAIL(r, t) = AVAIL(r, t) - REQ(a, r, t)

where t; LS(a) through LF(a)

Resources required by the scheduled activity are time phased and planned order releases are prepared.

The late finish time of the immediate predecessors of the scheduled activity is set with respect to the late start time of the last activity.

The scheduled activity is deleted from the list of schedulable activities and all activities which all of their successors are scheduled are included in the list of schedulable activities.

For all activities in the schedulable activities list, resource availabilities are checked. The checking process is the same as the checking process for the last activity presented before, except that, when the required resources are not available in the previous periods and a feasible start time is found in later periods, the start time of the activity can not be postponed because it makes the schedule infeasible. The start time of the last activity is postponed by an amount equal to the difference between the proposed start time of the activity found by the late start schedule algorithm and the feasible start time of the activity found by resource availability check procedure.

In this case all the changes made to the inventory file for the activities scheduled are undone and the algorithm starts with the new proposed start time for the last activity.

When all of the activities in the list are checked for their resources and start times are determined for each activity, the

following priority rules are applied to choose the activity to be scheduled.

1. Longest activity
2. Smallest activity start time
3. Smallest activity number

When an activity is scheduled, inventory files are updated to account for the resources allocated to that activity. The activity is deleted from the schedulable activities list and all of its immediate predecessors are checked to determine whether all of their successors are scheduled. Those activities which have all of their successors scheduled are added to the candidate list.

If the list is empty the project is scheduled and the scheduling process stops. If the list is not empty all of the activities in the list are examined for resource availabilities and the same process is repeated until either a feasible solution is found or the project can not be scheduled in the proposed date. Figure 4.16 shows the flow-chart of the operating room scheduling algorithms discussed above.

The output of the process includes the following:

1. The schedule including the late start and late finish times of activities, the resources required by each activity, the time to order enough of each resource, the time to receive the resource, and the time to release the resource if the resource is a non-storable resource.

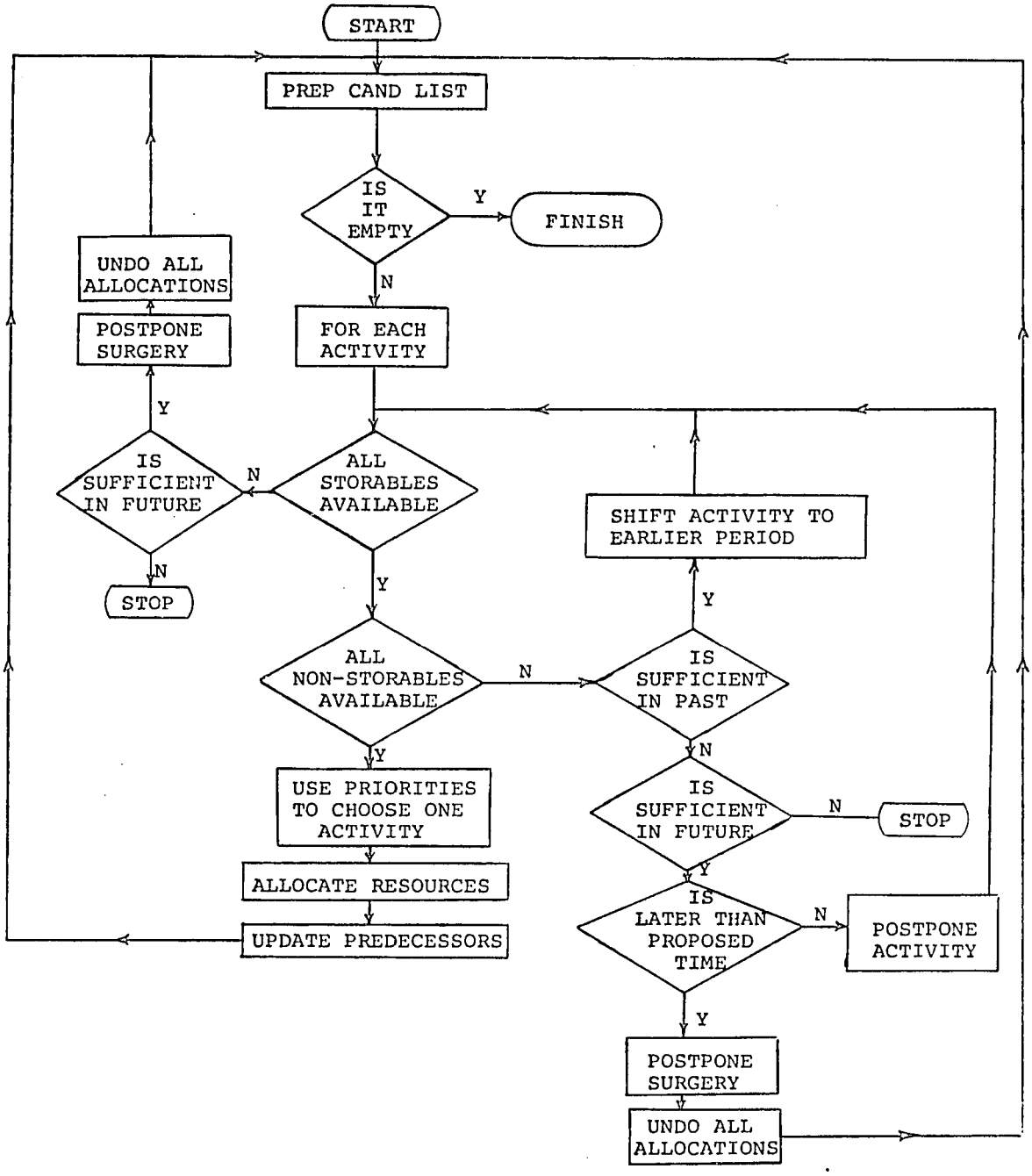


Figure 4.16 Flow Chart of the Operating Room Scheduling Algorithm

2. The resource usage profiles for both storable and non-storable resources. The resource usage profile reports show the quantity of each resource allocated to the surgery in all periods over the duration of the surgery.

3. Exception report for the resources which are not available but may be acquired from other sources. These reports include the time to order the resource, the quantity to be ordered, and the planned receipt time for each resource on this report. The time to order these resources is determined according to the acquisition lead time, from outside sources, which is usually longer than the stock withdrawal time of these resources.

4. List of the surgeries scheduled for each day in the planning horizon and the resources allocated to each of these surgeries.

As an example, the algorithm is used to schedule a sample surgery to start at period 50. The planning horizon is assumed to be 100 periods long. The appropriate project network is shown in Figure 4.17.

Table 4.2 shows the schedule for activities and required resources for each activity in the sample surgery project. The surgery is scheduled to start in period 50 and finish in period 70. The storable resources should be in the operating room in period 50. The non-storable resources should be in the operating room from period 50 through period 70. The order release time of the resources and their planned receipts are also determined.

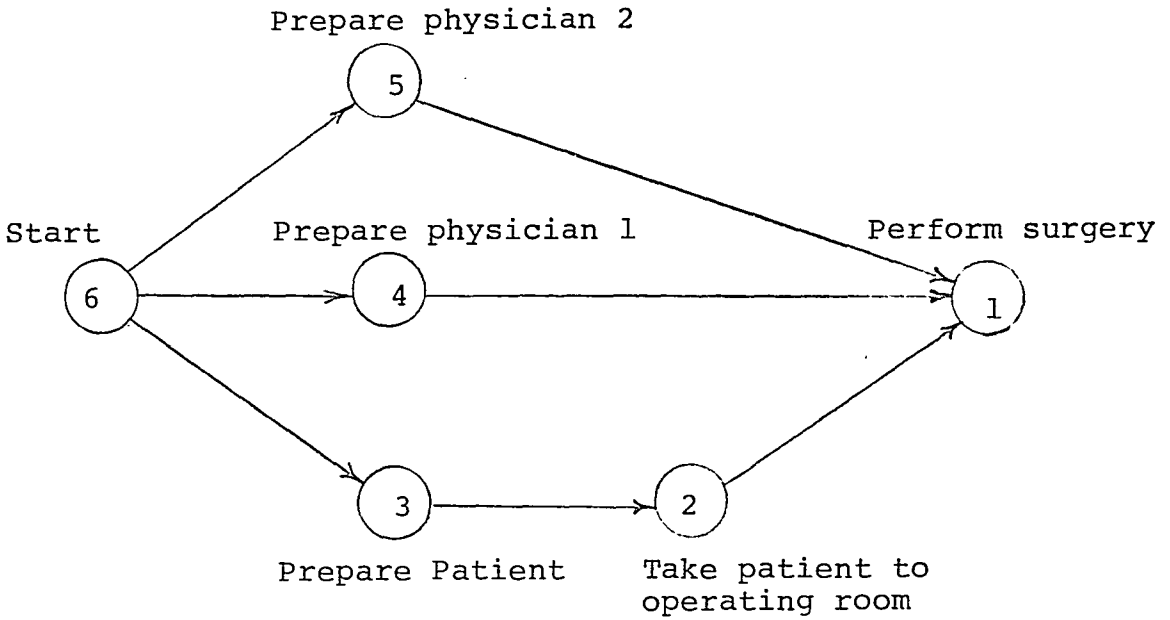


Figure 4.17 Activity on Node Diagram for the Example Surgery

Table 4.2. Scheduled Surgery Prepared by the Computer Program

ACTIVITIES	START TIME	FINISH TIME	RESOURCES	QUANTITY	TIME TO ORDER	TIME TO RECEIVE	TIME TO RELEASE
OPEN HEART	50	70					
			ANEST	1	42	50	70
			SCRUB NURS	3	46	50	70
			NURSE	4	48	50	70
			OPER. ROOM	1	46	50	70
			MACH. 2	1	48	50	70
			COTTON	1	40	50	
			BANDAGE	1	40	50	
			TRAY	1	38	50	
PAT TO OR.	49	50					
			NURSE	1	47	49	50
PREP PATIE	47	49					
			NURSE	2	45	47	49
			MACH. 2	1	45	47	49
PREP PH1	48	50					
			NURSE	1	46	48	50
PREP PH2	48	50					
			NURSE	1	46	48	50
START	47	47					

Tables 4.3 and 4.4 show the resource usage profiles for the storable and non-storable resources allocated to the surgery. The exception report for the tray is shown in Table 4.5. In this example the only resource which was not available and had to be ordered from other sources is tray. The lead time to order a tray when it is not available is 40 periods. The order release time for the tray is therefore period 10.

Table 4.3 Storable Resource Usage Profile

RESOURCE	PERIOD	QUANTITY
COTTON	50	1
BANDAGE	50	1
TRAY	50	1

Table 4.4 Non-Storable Resource Usage Profile

RESOURCE	FROM PERIOD	TO PERIOD	QUANTITY
ANEST.	50	70	1
SCRUB NURS	50	70	3
NURSE	47	48	2
	48	49	4
	49	50	3
	50	70	4
OPER. ROOM	50	70	1
MACH. 2	47	49	1
	50	70	1

Table 4.5 Exception Report for a Storable Resource

RESOURCE	TIME TO ORDER	TIME TO RECEIVE	QUANTITY
TRAY	10	50	1

CHAPTER 5

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

The objectives of this research have been accomplished. Data structures were designed and developed for efficient operations of the CMP-MRP system applied in this study. The data structures were developed to maintain a balance between the computer capacity limitations and the cost of performing the required operations on data structures. A late start scheduling algorithm based on the CPM-MRP concept was presented which was used to find a late start schedule for surgery cases. A computer program which incorporated the chosen and developed data structures and the surgery scheduling algorithm was written and tested on real problems. The results satisfied the objectives of this research. This chapter presents the conclusions drawn from different phases of this study and proposes some areas for future research.

Data Organization and File Organization

A major part of this study was concerned with the way the required files by the CPM-MRP method were organized. A major drawback of the original file organization techniques used in CMP-MRP was their huge memory utilization which prohibited the application of the technique to large problems.

This study used the distinguishing characteristics of activities, storable resources and non-storable resources to choose and develop data structures which would be appropriate for each category. For example, as Smith (1980) suggested, now only two memory locations are reserved to store start time and finish time of an activity as opposed to four times the number of periods in the planning horizon which was reserved before. The lower memory utilization allows the use of CPM-MRP to schedule larger projects and/or increasing the number of periods in the planning horizon.

A complex data structure used in this study is a modified version of threaded height-balanced binary search tree. The algorithms present in the literature on search and insertion operations on height-balanced binary search trees were modified to suit the new version developed in this study. An additional key in each mode resulted in more complicated search and insertion strategies.

In some cases a node is segmented into three new nodes. One of the new nodes remains in the same location in the tree. The other two nodes are inserted as the successors of that node. The insertions take place in two steps. First, the node with smaller keys is inserted. The balance factors are checked and the tree is restructured if the tree properties are violated. Then the balance factors are adjusted.

The restructuring of the tree and changes in the balance factors results in the loss of required information to check the violation of tree properties when the last node is inserted. Therefore, in

the second step the right place to insert the new node, already known, is searched for by starting from the root. At present this is the only way possible to acquire the required information to check the violation of tree property when a new node is inserted. Since this type of insertion is frequent, efficient insertion strategies reduce the computer time spent on this operation. Therefore, this study suggests more research on search and insertion operations.

The data structures employed should also be used in the CPM-MRP computer program developed by Zastera (1978) and Smith (1980). Data structures designed for storable resources may be used in MRP systems. Since only those periods in which a transaction occurs are stored in files designed in this research, the MRP software should be modified to generate the regular MRP output which includes data on all periods in the planning horizon.

The data structures designed in this research may be employed in capacity requirements planning systems. The resources may be categorized to storable resources. The planned capacity requirements for each category may be efficiently stored in proper files.

Hospital Operating Room Scheduling

The hospital operating room scheduling algorithm prepares a feasible late start schedule subject to technological sequence and resource constraints. Given the proposed start time for a surgery procedure, one or more iterations may be required to either find a feasible start time for the surgery procedure or terminate the process

unsuccessfully. The feasible start time determined by the algorithm may be different from the time proposed by physician's office. This is then communicated with the physician's office for physician's approval.

This research shows that all critical resources used in a surgery procedure should be considered in the operating room scheduling process to avoid delays and cancellations of surgery procedures. This research also shows that the CPM-MRP concept can successfully be applied in operating suit environment. The MRP logic provides information concerning the resource requirements for each activity and timings of those requirements and the heuristics which are borrowed from project scheduling techniques are used to choose the next activity to be scheduled from a set of schedulable activities.

The hospital operating room scheduling algorithm can be extended to schedule patient admissions into a hospital. Given the capacity of all of the relevant operating departments in a hospital and a complete list of patient's preparations and lab jobs and their lead times, the algorithm prepares a thorough schedule for all activities and resources required for the patient from patient's arrival until the individual leaves the hospital.

As far as the physicians are concerned, the most promising application of this algorithm is where the patient's admission scheduling for some hospitals is controlled by a common center. Since there is more than one hospital in the system, the possibility of scheduling surgery procedures for the physician's requested time is higher than when there is only one hospital in the system. Physicians who usually

prefer to have their cases scheduled when they request it will therefore be more satisfied from the system.

The hospital operating room scheduling algorithm is deterministic. It assumes that estimated procedure times and resource acquisition lead times are the same as the actual procedure time and lead times respectively. Because of the inherent stochasticity of operations, the estimated times are usually different from the actual times. The magnitude of the difference depends on the method used for estimation.

The difference between estimated and actual times creates discrepancies between the schedules and actual operations. The algorithms should expand to consider the probabilistic nature of the operations. An output may be generated to provide management with probabilities for occurrence of events in specific ranges.

The algorithm can be extended to help in resource planning. The storable and non-storable resource usage profiles for all surgeries in a day may be aggregated to prepare a resource profile for the day. This profile may then be used to schedule nurses and other resources as opposed to planning resources for the peak resource demand.

The algorithm can also be extended to the billing operations. The cost of supplies and resources utilized for each surgery may be accessed from inventory or accounting files. Using these cost figures the operating room expenses for each procedure may be computed automatically.

REFERENCES

Aquilano, N. J. "Project Scheduling with MRP." The University of Arizona, College of Business and Public Administration, working paper, 1978.

Barnoon, S. and H. Wolfe. "Scheduling a Multiple Operating Room System: A Simulation Approach," Health Services Research, (Winter 1968).

Bayer, R. and C. McCreight, "Organization and Maintenance of Large Ordered Indexes," Acta Informatica (1972).

Berry, R. E. "Cost and Efficiency in the Production of Hospital Services," Health and Society, 52 (Summer 1974).

Connors, M. M. "A Stochastic Elective Admission Scheduling System," Health Services Research, 5 (Winter 1970).

Davis, E. W. "Project Scheduling Under Resource Constraints--Historical Review and Categorization of Procedures," AIEE Transactions, 5 (December 1973).

Davis, J. G. and R. Reed. "Variability Control is the Key to Maximum Operating Room Utilization," Modern Hospital, (April 1964).

Fetter, R. B. and J.D. Thompson. "The Simulation of Hospital Systems," Operations Research, (September-October 1965).

Goldman, J. and H. A. Knappenberger. "How to Determine the Optimum Number of Operating Rooms," Modern Hospital, (September 1968).

Goldman, J., H.A. Knappenberger, and E. W. Moore, "An Evaluation of Operating Room Scheduling Policies," Hospital Management, (April 1969).

Griffith, J. R., W. M. Hancock, and F. C. Munson. Cost Control in Hospitals, Health Administration Press, Ann Arbor, MI. (1976)

Hancock, W. M., D. M. Warner, S. Heda, and P. Fuhs. "Admission Scheduling and Control Systems," Cost Control in Hospitals, pp. 150-185. Health Administration Press, Ann Arbor, MI, (1976).

Healy, T. M. "Operating Room Utilization," Connecticut Medicine, 37 (March 1973).

Horowitz, E. and S. Sahni. Fundamentals of Data Structures, Computer Science Press Inc., Potomac, Maryland, (1976).

Hu, T. C. and K. C. Tan. "Least Upper Bound on the Cost of Optimum Binary Search Trees," Acta Informatica (1972).

Jonas, S. Health Care Delivery in the United States, Springer, New York, NY, (1977).

Karlton, P. L., S. H. Fuller, R. E. Scroggs, and E. B. Kaehler "Performance of Height-balanced Trees," CACM, (January 1976).

Knuth, D. E. "Optimum Binary Search Trees," Acta Informatica, (1971).

Knuth, D. E. The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass., (1973).

Kosaraju, S. Rao. "Insertions and Deletions in One-sided Height-balanced Trees," CACM, (March 1978).

Kuzdrall, P. J., N. K. Kwak, and H. H. Schmitz. "The Monte Carlo Simulations of Operating-Room and Recovery-Room Usage, Operations Research, 22 (March-April 1974).

Kwak, N. K., P. J. Kuzdrall, and H. H. Schmitz. "The GPSS Simulation of Scheduling Policies for Surgical Patients," Management Science, 22 (May 1976).

London, M. "Know your O. R. Vacancy Rate To Improve Surgical Scheduling," Modern Hospital, 103 (October 1964).

Loomis, M. E. Principles of Data Management and File Processing, to be published by Prentice-Hall, (1983).

Magerlein, J. M. and J. B. Martin, "Surgical Demand Scheduling: A Review," Health Services Research, (Winter 1978).

Milsum, H. H., E. Turban, and I. Vertinsky. "Hospital Admission Systems: Their Evaluation and Management," Management Science, 19 (February 1973).

Morgan, D. M. "Block Booking in the O.R.--A Solution to Multitude of Sins," Canadian Hospital, 49 (February 1972).

Morgan, D. M. "Improved Scheduling Through Block Booking," Canadian Hospital, 50 (February 1973).

Nievergelt, J. "Binary Search Trees and File Organization," Computing Surverys, (September 1974).

Nievergelt, J. and E. M. Reingold. "Binary Search Trees of Bounded Balance," SIAM, (March 1973).

Offensend, F. L. "A Hospital Admission System Based On Nursing Work Load," Management Science, 19 (October 1972).

Orlicky, J. Material Requirements Planning, McGraw-Hill, New York, NY, (1975).

Ottmann, T. H. and D. Wood. "Deletion in One-sided Height Balanced Search Trees," International Journal of Computer Mathematics, (1978).

Perl, Yehoshua, Alan Itai, and Haim Auni. "Interpolation Search--A Log Log N Search," CACM, (October 1978).

Perlis, A. and C. Thornton. "Symbol Manipulation by Threaded Lists," CACM, Vol. 3, No. 4, pp. 195-204 (April 1960).

Phillips, K. T. "Operating Room Utilization," Hospital Topics, 153 (March-April 1975).

Raiha, Kari-Jouko and Stuart H. Zweben. "An Optimal Insertion Algorithm for One-Sided Height-balanced Binary Search Trees," CACM, (Sept. 1979).

Rivest, Ronald. "On Self-Organizing Sequential Search Heuristics," CACM, (Feb. 1976).

Robinson, G. H., P. Wing, and L. E. Davis. "Computer Simulation of Hospital Patient Scheduling Systems," Health Services Research, 3 (Summer 1968).

Schmiz, H. H. and N. K. Kwak. "Monte Carlo Simulation of the Operating-Room," Operations Research, 20 (November-December 1972).

Schneiderman, H. B. "Building a New Surgical Suite," AORN Journal, 30 (July 1979).

Shelter, M. G. "Utilization of the Surgical Suite," AORN Journal, 15 (January 1972).

Shneiderman, Ben. "Jump Searching: A Fast Sequential Search Technique," CACM, (Oct. 1978).

Smith, D. E. "A Combined Critical Path Method-Material Requirements Planning Model for Project Scheduling Subject to Resource Constraints," The University of Arizona, Ph.D. Dissertation, (1980).

Stevens, B. J. "Systems Approach to Managing O.R.," AORN Journal, 27 (June 1978).

Stewart, J. T. "Surgical Specialties Affect Scheduling," Hospitals, 45 (September 1, 1971).

Tenenbaum, Aaron. "Simulation of Dynamic Sequential Search Algorithms," CACM, (Sept. 1978).

Vraciu, R. A. "Programming, Budgeting, and Control in Health Care Organizations: The State of the Art," Health Services Research, (Summer 1979).

Whitston, C. W. "An Analysis of the Problems of Scheduling Surgery-I," Hospital Management, 99 (April 1965).

Whitston, C. W. "An Analysis of the Problems of Scheduling Surgery-II," Hospital Management, 99 (May 1965).

Wiest, J. D. and F. K. Levy. A Management Guide to PERT/CPM, Prentice-Hall, Englewood Cliffs, N.J. (1977).

Williams, W. L. "Improved Utilization of the Surgical Suite," Hospitals, 45 (March 1, 1971).

Young, J. P. "Stabilization of Inpatient Bed Occupancy through Control of Admissions," Hospitals, 39 (October 1, 1965).

Zimmerman, O. C. "Utilizing Operating Room Time," Hospital Topics, 41 (February 1963).

Zweben, S. H. and M. A. McDonald. "An Optimal Insertion Algorithm for One-Sided Height-Balanced Binary Search Trees," CACM (Sept. 1979).