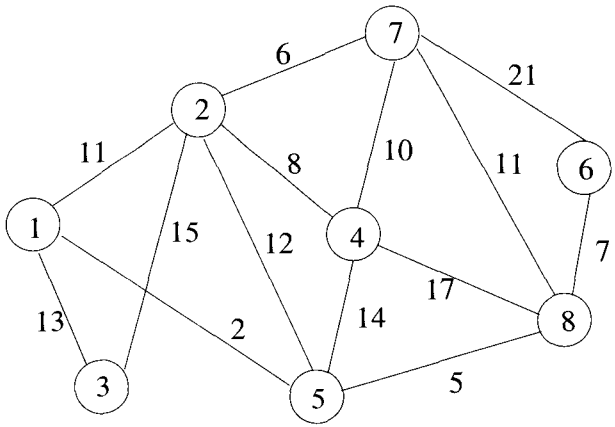

Figure 4.10 Graph for Exercise 1

(b) Show that if any of the edges on this unique cycle is deleted from $E(t) \cup \{q\}$, then the remaining edges form a spanning tree of G.

6. In Figure 4.9, find a minimum-cost spanning tree for the graph of part (c) and extend the tree to obtain a minimum cost spanning tree for the graph of part (a). Verify the correctness of your answer by applying either Prim's algorithm or Kruskal's algorithm on the graph of part (a).

7. Let $G(V, E)$ be any weighted connected graph.

   (a) If C is any cycle of G, then show that the heaviest edge of C cannot belong to a minimum-cost spanning tree of G.

   (b) Assume that F is a forest that is a subgraph of G. Show that any F-heavy edge of G cannot belong to a minimum-cost spanning tree of G.

8. By considering the complete graph with n vertices, show that the number of spanning trees in an n vertex graph can be greater than $2^{n-1}-2$.

4.6 OPTIMAL STORAGE ON TAPES

There are n programs that are to be stored on a computer tape of length l. Associated with each program i is a length $l_i, 1 \leq i \leq n$. Clearly, all programs can be stored on the tape if and only if the sum of the lengths of

the programs is at most l. We assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front. Hence, if the programs are stored in the order $I = i_1, i_2, \ldots, i_n$, the time t_j needed to retrieve program i_j is proportional to $\sum_{1 \le k \le j} l_{i_k}$. If all programs are retrieved equally often, then the expected or *mean retrieval time* (MRT) is $(1/n) \sum_{1 \le j \le n} t_j$. In the optimal storage on tape problem, we are required to find a permutation for the n programs so that when they are stored on the tape in this order the MRT is minimized. This problem fits the ordering paradigm. Minimizing the MRT is equivalent to minimizing $d(I) = \sum_{1 \le j \le n} \sum_{1 \le k \le j} l_{i_k}$.

Example 4.8 Let $n = 3$ and $(l_1, l_2, l_3) = (5, 10, 3)$. There are $n! = 6$ possible orderings. These orderings and their respective d values are:

ordering I	$d(I)$		
$1, 2, 3$	$5 + 5 + 10 + 5 + 10 + 3$	$=$	38
$1, 3, 2$	$5 + 5 + 3 + 5 + 3 + 10$	$=$	31
$2, 1, 3$	$10 + 10 + 5 + 10 + 5 + 3$	$=$	43
$2, 3, 1$	$10 + 10 + 3 + 10 + 3 + 5$	$=$	41
$3, 1, 2$	$3 + 3 + 5 + 3 + 5 + 10$	$=$	29
$3, 2, 1$	$3 + 3 + 10 + 3 + 10 + 5$	$=$	34

The optimal ordering is $3, 1, 2$. □

A greedy approach to building the required permutation would choose the next program on the basis of some optimization measure. One possible measure would be the d value of the permutation constructed so far. The next program to be stored on the tape would be one that minimizes the increase in d. If we have already constructed the permutation $i_1, i_2, \ldots, i_r$, then appending program j gives the permutation $i_1, i_2, \ldots, i_r, i_{r+1} = j$. This increases the d value by $\sum_{1 \le k \le r} l_{i_k} + l_j$. Since $\sum_{1 \le k \le r} l_{i_k}$ is fixed and independent of j, we trivially observe that the increase in d is minimized if the next program chosen is the one with the least length from among the remaining programs.

The greedy algorithm resulting from the above discussion is so simple that we won't bother to write it out. The greedy method simply requires us to store the programs in nondecreasing order of their lengths. This ordering can be carried out in $O(n \log n)$ time using an efficient sorting algorithm (e.g., heap sort from Chapter 2). For the programs of Example 4.8, note that the permutation that yields an optimal solution is the one in which the programs are in nondecreasing order of their lengths. Theorem 4.8 shows that the MRT is minimized when programs are stored in this order.

Theorem 4.8 If $l_1 \leq l_2 \leq \cdots \leq l_n$, then the ordering $i_j = j, 1 \leq j \leq n$, minimizes

$$\sum_{k=1}^{n} \sum_{j=1}^{k} l_{i_j}$$

over all possible permutations of the i_j.

Proof: Let $I = i_1, i_2, \ldots, i_n$ be any permutation of the index set $\{1, 2, \ldots, n\}$. Then

$$d(I) = \sum_{k=1}^{n} \sum_{j=1}^{k} l_{i_j} = \sum_{k=1}^{n} (n - k + 1) l_{i_k}$$

If there exist a and b such that $a < b$ and $l_{i_a} > l_{i_b}$, then interchanging i_a and i_b results in a permutation I' with

$$d(I') = \left[\sum_{\substack{k \\ k \neq a \\ k \neq b}} (n - k + 1) l_{i_k} \right] + (n - a + 1) l_{i_b} + (n - b + 1) l_{i_a}$$

Subtracting $d(I')$ from $d(I)$, we obtain

$$\begin{aligned} d(I) - d(I') &= (n - a + 1)(l_{i_a} - l_{i_b}) + (n - b + 1)(l_{i_b} - l_{i_a}) \\ &= (b - a)(l_{i_a} - l_{i_b}) \\ &> 0 \end{aligned}$$

Hence, no permutation that is not in nondecreasing order of the l_i's can have minimum d. It is easy to see that all permutations in nondecreasing order of the l_i's have the same d value. Hence, the ordering defined by $i_j = j, 1 \leq j \leq n$, minimizes the d value. □

The tape storage problem can be extended to several tapes. If there are $m > 1$ tapes, $T_0, \ldots, T_{m-1}$, then the programs are to be distributed over these tapes. For each tape a storage permutation is to be provided. If I_j is the storage permutation for the subset of programs on tape j, then $d(I_j)$ is as defined earlier. The *total retrieval time* (TD) is $\sum_{0 \leq j \leq m-1} d(I_j)$. The objective is to store the programs in such a way as to minimize TD.

The obvious generalization of the solution for the one-tape case is to consider the programs in nondecreasing order of l_i's. The program currently

```
1   Algorithm Store(n, m)
2   // n is the number of programs and m the number of tapes.
3   {
4       j := 0; // Next tape to store on
5       for i := 1 to n do
6       {
7           write ("append program", i,
8               "to permutation for tape", j);
9           j := (j + 1) mod m;
10      }
11  }
```

Algorithm 4.12 Assigning programs to tapes

being considered is placed on the tape that results in the minimum increase in TD. This tape will be the one with the least amount of tape used so far. If there is more than one tape with this property, then the one with the smallest index can be used. If the jobs are initially ordered so that $l_1 \leq l_2 \leq \cdots \leq l_n$, then the first m programs are assigned to tapes $T_0, \ldots, T_{m-1}$ respectively. The next m programs will be assigned to tapes $T_0, \ldots, T_{m-1}$ respectively. The general rule is that program i is stored on tape $T_{i \bmod m}$. On any given tape the programs are stored in nondecreasing order of their lengths. Algorithm 4.12 presents this rule in pseudocode. It assumes that the programs are ordered as above. It has a computing time of $\Theta(n)$ and does not need to know the program lengths. Theorem 4.9 proves that the resulting storage pattern is optimal.

Theorem 4.9 If $l_1 \leq l_2 \leq \cdots \leq l_n$, then Algorithm 4.12 generates an optimal storage pattern for m tapes.

Proof: In any storage pattern for m tapes, let r_i be one greater than the number of programs following program i on its tape. Then the total retrieval time TD is given by

$$TD = \sum_{i=1}^{n} r_i l_i$$

In any given storage pattern, for any given n, there can be at most m programs for which $r_i = j$. From Theorem 4.8 it follows that TD is minimized if the m longest programs have $r_i = 1$, the next m longest programs have

$r_i = 2$, and so on. When programs are ordered by length, that is, $l_1 \leq l_2 \leq \cdots \leq l_n$, then this minimization criteria is satisfied if $r_i = \lceil (n-i+1)/m \rceil$. Observe that Algorithm 4.12 results in a storage pattern with these r_i's. □

The proof of Theorem 4.9 shows that there are many storage patterns that minimize TD. If we compute $r_i = \lceil (n-i+1)/m \rceil$ for each program i, then so long as all programs with the same r_i are stored on different tapes and have $r_i - 1$ programs following them, TD is the same. If n is a multiple of m, then there are at least $(m!)^{n/m}$ storage patterns that minimize TD. Algorithm 4.12 produces one of these.

EXERCISES

1. Find an optimal placement for 13 programs on three tapes T_0, T_1, and T_2, where the programs are of lengths $12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10$, and 6.

2. Show that replacing the code of Algorithm 4.12 by

```
for i := 1 to n do
    write ("append program", i, "to permutation for
        tape", (i - 1) mod m);
```

   does not affect the output.

3. Let $P_1, P_2, \ldots, P_n$ be a set of n programs that are to be stored on a tape of length l. Program P_i requires a_i amount of tape. If $\sum a_i \leq l$, then clearly all the programs can be stored on the tape. So, assume $\sum a_i > l$. The problem is to select a maximum subset Q of the programs for storage on the tape. (A maximum subset is one with the maximum number of programs in it). A greedy algorithm for this problem would build the subset Q by including programs in nondecreasing order of a_i.

   (a) Assume the P_i are ordered such that $a_1 \leq a_2 \leq \cdots \leq a_n$. Write a function for the above strategy. Your function should output an array $s[1:n]$ such that $s[i] = 1$ if P_i is in Q and $s[i] = 0$ otherwise.

   (b) Show that this strategy always finds a maximum subset Q such that $\sum_{P_i \in Q} a_i \leq l$.

   (c) Let Q be the subset obtained using the above greedy strategy. How small can the tape utilization ratio $(\sum_{P_i \in Q} a_i)/l$ get?

   (d) Suppose the objective now is to determine a subset of programs that maximizes the tape utilization ratio. A greedy approach