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 $l_i, 1 \le i \le n$. Clearly, all programs can be stored on the tape if and only if the sum of the lengths of

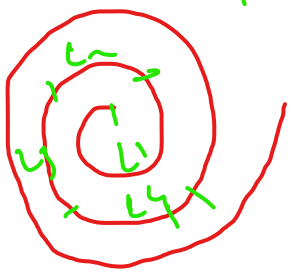
$l_1 + l_2 + l_3 \ldots + l_n \le l$.

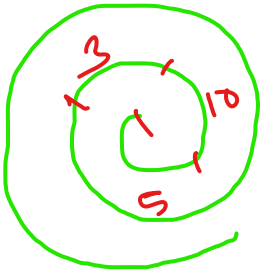
the programs is at most l. We assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front. Hence, if the programs are stored in the order $I = i_1, i_2, \ldots, i_n$, the time t_j needed to retrieve program i_j is proportional to $\sum_{1 \le k \le j} l_{i_k}$. 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

JOB SEQUENCING WITH DEADLINES

We are given a set of n jobs. Associated with job i is an integer deadline $d_i \ge 0$ and a profit $p_i > 0$. For any job i the profit p_i is earned iff the job is completed by its deadline. To complete a job, one has to process the job on a machine for one unit of time. Only one machine is available for processing jobs. A feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline. The value of a feasible solution J is the sum of the profits of the jobs in J, or $\sum_{i \in J} p_i$. An optimal solution is a feasible solution with maximum value. Here again, since the problem involves the identification of a subset, it fits the subset paradigm.

Example Let $n = 4, (p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$ and $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$. The feasible solutions and their values are:

	feasible solution	processing sequence	value
1.	(1, 2)	2, 1	110
2.	(1, 3)	1, 3 or 3, 1	115
3.	(1, 4)	4, 1	127
4.	(2, 3)	2, 3	25
5.	(3, 4)	4, 3	42
6.	(1)	1	100
7.	(2)	2	10
8.	(3)	3	15
9.	(4)	4	27

Solution 3 is optimal. In this solution only jobs 1 and 4 are processed and the value is 127. These jobs must be processed in the order job 4 followed by job 1. Thus the processing of job 4 begins at time zero and that of job 1 is completed at time 2.