

CSE 259: Algorithms and Complexity

Fall 2006, Exam II; Help Sheet

SELECTION. Given a sequence of n keys and an integer i with $1 \leq i \leq n$, the problem of selection is to identify the i th smallest element from out of the n keys. The BFPRT algorithm solves this problem in $O(n)$ time.

BUCKET and RADIX SORTING. If X is a sequence of n keys where each key is an integer in the range $[1, m]$ then X can be sorted in $O(m + n)$ time. If $m = n^c$ for any constant c , then radix sort can be used to sort X in $O(n)$ time.

MATRIX MULTIPLICATION. Strassen's algorithm multiplies two $n \times n$ matrices in $O(n^{\log_2 7})$ time.

GREEDY ALGORITHMS. This technique is used when we are interested in finding a subset of n given objects that satisfies a set of constraints and optimizes a given objective function. The general idea is to start with the empty set; select the next object O to be examined using a selection criterion; if the inclusion of O into the solution S will still keep it feasible we add O into S , otherwise we discard O ; proceed in a similar fashion until all the objects have been examined; and finally output the solution S .

We were able to solve the fractional knapsack problem in $O(n \log n)$ time using the greedy approach. The idea is to process the objects in nonincreasing order of their profit densities.

We also showed that the minimum weight spanning tree problem can be solved in $O((|V| + |E|) \log |V|)$ time on any weighted undirected graph $G(V, E)$ employing the greedy technique. Prim's algorithm has only one tree at any time. It looks at all the outgoing edges from the tree and includes the edge with the minimum weight. Kruskal's algorithm starts with a forest of n trees and inserts one edge at a time into the forest (if the edge does not cause a cycle). The edges are sorted in nondecreasing order of the edge weights to begin with.

Dijkstra's algorithm for the single source shortest path problem runs in time $O((|V| + |E|) \log |V|)$ time. This algorithm assumes that the input graph does not have any edges with negative weights.

DYNAMIC PROGRAMMING. The general solution technique here typically involves the following steps: 1) define a suitable function such that the outputs of interest are specific values of this function; 2) write a recurrence relation for this function; and 3) solve the recurrence relation to get the values of interest – the base cases for the function are usually the inputs.

For the zero-one knapsack problem, we define $f_i(y)$ to be the optimal profit obtainable from the objects 1 through i when the capacity constraint is y . A recurrence relation for $f_i(y)$ takes the form:

$$f_i(y) = \max\{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}.$$

When the weights are integers, the above recurrence relation can be used to solve the problem in $O(mn)$ time.

In the all-pairs shortest paths problem, the input is a directed graph $G(V, E)$. The goal is to find the shortest path from the node i to node j for every pair of nodes i and j in V . We define the function $A^k(i, j)$ to be the shortest path length from i to j from among all paths (from i to j) whose intermediate nodes are $\leq k$. We are interested in the values $A^n(i, j)$ (for every i and j in V), where $n = |V|$. A recurrence relation for $A^k(i, j)$ can be written as follows:

$$A^k(i, j) = \min\{A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j)\}.$$

We start with A^0 and compute A^1, A^2, \dots, A^n . The total run time is $O(n^3)$.