

CSE 361: Complexity of Sequential and Parallel Algorithms. Fall 2005

Solutions to Exam 2.

- Sort the jobs (in non-decreasing order) based on the time that they need for completion. Select maximum possible number of jobs in the sorted order such that the sum of their running times does not exceed the global deadline D .

Complexity = Complexity of sorting + Complexity of scanning = $O(n \log n)$

Proof of optimality: Let g_1, g_2, \dots, g_m be the jobs selected in that order by the greedy strategy. Let o_1, o_2, \dots, o_k be the optimal solution. Let i be the smallest index where the greedy strategy and the optimal strategy differ. That is, g_1, g_2, \dots, g_{i-1} are the same as o_1, o_2, \dots, o_{i-1} and $g_i \neq o_i$. Then, do the following: replace o_i in the optimal solution by g_i . Since the greedy strategy has selected elements in sorted order, $T_{g_i} \leq T_{o_i}$. Hence, replacing o_i by g_i does not increase the total time taken by all the jobs in the optimal solution. By repeating the above procedure at all the indices where the jobs in the greedy solution and the optimal solution differ, we can see that the greedy solution can complete atleast as many jobs as the optimal solution. Hence, the greedy solution is optimal.

- The weight of each object is $\geq \epsilon m$. Since the knapsack capacity is m , a maximum of $1/\epsilon$ objects can be included in the solution. The total number of subsets with $1/\epsilon$ objects or less is $\sum_{i=1}^{1/\epsilon} \binom{n}{i} = O(n^{1/\epsilon})$. Compute the profit for each such subset and pick the best subset.

Complexity = $O(n^{\lceil 1/\epsilon \rceil})$.

- Let's define a function *shuffle* as below:

$shuffle(i, j) = 1$ if z_1, \dots, z_{i+j} is a shuffle of x_1, \dots, x_i and y_1, \dots, y_j .

$shuffle(i, j)$ can be calculated from $shuffle(i-1, j)$ and $shuffle(i, j-1)$ as below:

$shuffle(i, j) = 1$ if $shuffle(i-1, j) = 1$ and $z_{i+j} = x_i$ OR

$shuffle(i, j-1) = 1$ and $z_{i+j} = y_j$.

$shuffle(m, n)$ tells whether z is a shuffle of x and y .

Complexity = Complexity of calculating $shuffle(i, j)$ for $1 \leq i \leq m$ and $1 \leq j \leq n = O(mn)$.

- $S^0 = \{(0, 0)\}; S_1^1 = \{(10, 4)\}$
 $S^1 = \{(0, 0), (10, 4)\}; S_1^2 = \{(20, 6), (30, 10)\}$
 $S^2 = \{(0, 0), (10, 4), (20, 6), (30, 10)\}, S_1^3 = \{(15, 8), (25, 12), (35, 14), (45, 18)\}$
 $S^3 = \{(0, 0), (10, 4), (20, 6), (30, 10), (15, 8), (25, 12), (35, 14), (45, 18)\}$.
 Thus $f_3(13) = 30$.

- If a graph contains a square as a subgraph then there exist atleast two nodes which have two common neighbors.

Step 1: Arrange the neighbors of each node in a sorted order. This can be done by constructing tuples (i, j) for each neighbor j of a node i and sorting all the consturcted tuples of all the nodes using the radix sort. This takes $O(|V| + |E|)$ time.

Step 2: Find if there are any two nodes that have two common neighbors.

Let L_i represent the sorted list of neighbors of node i , $1 \leq i \leq n$.

for $i := 1$ to $|V|$ do

for $j := 1$ to $|V|$ do

Merge L_i and L_j and check whether i and j have two common neighbors.

Complexity of step 1 = $O(|V| + |E|)$.

Let d_i represent the length of L_i . Note that d_i is the degree of i .

Complexity of step 2 = $\sum_{1 \leq i \leq |V|, 1 \leq j \leq |V|} (d_i + d_j) = O(|V||E|)$.

Total complexity = $O(|V||E|)$.

- TRUE. We can push a flow of value c_i along P_i for $1 \leq i \leq k$. These k flows are independent.
 - FALSE. This statement need not be true always. It depends on the rest of the network.
 - FALSE. The reason is the same.
 - TRUE. Depending on the rest of the network, a flow of this value is possible.