College of Computer & Information Science
Northeastern University
CS7800: Advanced Algorithms

Fall 2014
Handout 10
30 September 2014

Sample Solution to Problem Set 3

1. (10 points) Matroids

Exercise 16.4-4 of text.

Answer: For the hereditary property, note that if $A \in I$ and $B \subseteq A$, then $|B \cap S_i| \leq 1$ whenever $|A \cap S_i| \leq 1$, for all i, implying that B is also in I.

For the interchange property, suppose we have $A, B \in I$ with $|A| < |B|$. Then, there exists an S_i such that $e \in S_i$ and $e \in B$, yet $S_i \cap A = \emptyset$. Thus, $A \cup \{e\} \in I$.

2. (5 + 5 = 10 points) Minimizing average completion time

Problem 16-2 of text.

Answer: This is a sketch, omitting details.

(a) The algorithm is to simply schedule the jobs in increasing order of their completion times. To see why this works, suppose there exists an optimal solution that executed job task a_1 and a_2 one after another such that $p_1 > p_2$; so a_1 completes at time $T + p_1$ and a_2 at time $T + p_1 + p_2$ for some T. Then, switching the two changes their completion times to $T + p_2$ and $T + p_1 + p_2$, respectively, this reducing the total completion time by $p_1 - p_2$.

(b) Since preemption is allowed, at any instant we would have jobs that would have completed partially. If no more jobs came, then we should execute them in increasing order of their *remaining processing times*.

So the general algorithm is to execute, at any instant, that job (among the ones that have been released) which has the smallest remaining processing time. An exchange argument similar to the one in part (a) can be used to show that this algorithm is optimal.

3. (10 points) Minimum-length encoding

This is a continuation of Problem 4 of PS2. Recall that Alice and Bob are using an encoding scheme based on a set S of m *code words* that they both share. Alice wants to send a data string D of length n to Bob. In this exercise, we assume that S is an *arbitrary* set of m code words.

Design a polynomial-time algorithm to determine a minimum-length encoding of a given string D using code words from S. If no such encoding exists, then the algorithm must indicate so. Analyze the worst-case running time of your algorithm. Make your algorithm as efficient as you can, in terms of its worst-case running time.

Answer: We use dynamic programming. The optimal substructure property holds as before. The algorithm recursively computes an optimal solution for $D[1..i]$, for $1 \leq i \leq n$. We represent the optimal solution for $D[1..i]$ by the last code word used in the encoding – this can be specified by an

index $L[i]$ so that $D[L[i]..i]$ is the last code word. Let $C(i)$ denote the optimal cost. The algorithm determines $C(i)$ and $L(i)$ by solving the following recurrence (here k is the maximum length of a code word):

$$\begin{aligned} C(0) &= 0 \\ L(0) &= \varepsilon \\ C(i) &= 1 + \min_{0 \le j < k : D[i-j..i] \in S} C(i-j-1) \\ L(i) &= \text{argmin}_{0 \le j < k : D[i-j..i] \in S} C(i-j-1). \end{aligned}$$

The above recurrence relation calculates $C(i)$ as follows. For each j in $[0, k)$, we identify all j such that $D[i-j..i]$ is a codeword; and among these j, we select the one that has the smallest value of $C(i-j-1)$.

We can solve the recurrence by going over each codeword and checking which of these match a suffix of $D[1..i]$. This takes time $O(km)$. One can make this more efficient (in time $O(k)$) by storing the codewords in a tree structure and then doing the comparison one character by character. We also keep track of the j for which $D[i-j..i]$ has a match with a codeword. Finding the minimum among the $C(i-j)$ takes time $O(k)$. So the total time for calculating a $C(i)$ is $O(k)$. Time for calculating $C(n)$ is $O(kn)$. Since k can be bounded by $\min\{m, n\}$, we have a running time of $O(n \cdot \min\{m, n\})$.