

Name \_\_\_\_\_ Student No. \_\_\_\_\_

AIDS ALLOWED: One page (two sides) of handwritten notes

Answer ALL questions on test paper. Use backs of sheets for additional space. For every question, briefly justify your answer. You can use knowledge from the problem set as part of any answer.

REMINDER: You get 20% for any question or subquestion if you state “I do not know how to answer this question”. You get 10% for any question which you just leave blank.

- (10 points) We are given an edge weighted directed graph  $G = (V, E)$  with edge weights  $cap : E \rightarrow \mathbb{R}^{\geq 0}$ . (You can think of  $cap$  as the capacity of the edge.) The capacity of a path  $\pi$  is then defined as  $cap(\pi) = \min\{cap(e) | e \in \pi\}$ . Given a specified node  $s \in V$ , the goal is to compute the *maximum* capacity path from  $s$  to each node  $v \in V - \{s\}$ . Sketch an optimal greedy algorithm for this problem and indicate any key inductive property that could be used to prove the optimality of your algorithm. Hint: you may find it convenient to define the capacity of the empty path  $\pi_0$  (i.e. from  $s$  to  $s$ ) to be  $cap(\pi_0) = \infty$  just as in Dijkstra’s algorithm we set  $cost(\pi_0) = 0$ .

### SOLUTION

Adapt Dijkstra’s algorithm:

Let  $S$  be the set of explored nodes.

For each  $u \in S$  store the maximum capacity  $c(u)$  of any path from  $s$  to  $u$ .

Initially  $S = \{s\}$  and  $c(s) = \infty$ .

While  $S \neq V$  :

Select a node  $v \notin S$  with at least one edge from  $S$  for which  
 $c'(v) = \max_{e=(u,v) : u \in S} \min(c(u), cap(u, v))$  is as large as possible.  
 Add  $v$  to  $S$  and define  $c(v) = c'(v)$

The “key inductive property” is the same idea as the proof of optimality of Dijkstra’s algorithm. When node  $v$  is added to set  $S$ , the maximum capacity path from  $s$  to  $v$  has capacity  $c(v)$ . (If the maximum capacity path had some higher capacity, then the algorithm would have found it already!) This statement is true of all the vertices in  $S$ , so in particular when the algorithm finishes and  $S = V$ , it has found all the maximum-capacity paths from  $s$  to every other node.

(It is easy to reconstruct the maximum-capacity paths given the capacities  $c(v)$  for all  $v$ .)

**COMMON MISTAKES:** It was common to write something like, “Do a BFS on the nodes, recording maximum capacity as you go.” This doesn’t work! Consider the graph  $G = (V, E)$  where  $V = \{s, a, b, c\}$  and  $E = \{(s, a), (a, b), (b, c), (s, c)\}$  with capacities  $cap(s, a) = 10$ ,  $cap(a, b) = 10$ ,  $cap(b, c) = 10$ , and  $cap(s, c) = 5$ . A BFS approach will find (in the first round) that the maximum capacity path from  $s$  to  $c$  is  $s \rightarrow c$  and has capacity 5. But actually the maximum capacity path from  $s$  to  $c$  is  $s \rightarrow a \rightarrow b \rightarrow c$  and has capacity 10. This is why BFS does not work for this problem.

2. (20 points)

A University offers  $n$  courses  $\mathcal{J} = \{I_1, \dots, I_n\}$  in a certain program. A course  $I_j$  ( $1 \leq j \leq n$ ) is described by  $I_j = (s_j, f_j, \ell_j)$  where  $s_j$  denotes the start time,  $f_j$  denotes the finishing time, and  $\ell_j$  denotes the course lecturer for the course. University regulations do not allow a student to schedule two courses whose scheduled times intersect, and do not allow a student to take more than two courses from the same lecturer.

State a greedy 3-approximation algorithm  $ALG$  for maximizing the number of courses that a student can take in the program (given the above constraints). Indicate how you would prove the approximation bound; that is, how you would show that  $OPT(\mathcal{J}) \leq 3 \cdot ALG(\mathcal{J})$  for any input set  $\mathcal{J}$ .

**SOLUTION** Again this was very similar to a problem from the homework. The algorithm has some small changes:

Sort the jobs  $J$  in increasing order of finishing time:  $\mathcal{J} = \{I_1, \dots, I_n\}$ ,  $f_1 \leq f_2 \leq \dots \leq f_n$ .

The set  $S = \emptyset$  is the courses scheduled so far, the last of which ends at time  $D = 0$ .

For  $I_i \in \mathcal{J}$  in order:

If  $D \leq f_i$  and there are fewer than two classes by lecturer  $\ell_i$  in  $S$ ,  
then add  $I_i$  to  $S$  and update  $D = f_i$ .  
else disregard  $I_i$ .

Return  $S$ .

To prove that this algorithm is a 3-approximation, we need to show that:

$OPT(\mathcal{J}) \leq 3 \cdot ALG(\mathcal{J}) = 3 \cdot |S|$ . This can be done with a charging argument, by defining function  $f : OPT(\mathcal{J}) \rightarrow \mathcal{ALG}(\mathcal{J})$  which is *at most* 3-to-1.

Two jobs  $I_i$  and  $I_j \in \mathcal{J}$  **conflict** if they overlap in time (so either  $s_i \leq s_j \leq f_i$  or  $s_i \leq f_j \leq f_i$ ) or they have the same lecturer ( $\ell_i = \ell_j$ ).

For  $I \in OPT(\mathcal{J})$ , define  $f(I)$  to be the earliest-finishing job in  $ALG(\mathcal{J})$  that conflicts with  $I$ .

Function  $f$  is at most 3-to-1 if every job  $J \in \mathcal{ALG}(\mathcal{J})$  has at most three jobs  $I, K, L \in OPT(\mathcal{J})$  that map to it:  $f(I) = J = f(K) = f(L)$ . At most two jobs from  $OPT$  can map to  $J$  because they conflict in lecturer. At most one job can map to  $J$  because it overlaps in time. (The proof is similar to the homework problem, and the EFT proof from class.) Therefore at most 3 jobs from  $OPT(\mathcal{J})$  can map to one job from  $ALG(\mathcal{J})$ .

3. (20 points) Consider the following problem. The input is a set  $X$  of  $n$  points  $x_1, \dots, x_n$  on the real line and without loss of generality let us say  $x_1 < x_2 < \dots < x_n$ . We are also given an integer parameter  $k$ ,  $1 \leq k \leq n$ . The goal is to select  $k$  points  $Y \subseteq X$  so as to minimize  $\max_{i=1}^n \min_{y \in Y} |x_i - y|$ . Provide a dynamic programming (DP) algorithm for this problem.

- (10 points) Provide a semantic array definition for computing the cost of an optimal solution.

**SOLUTION** We will assume that  $j \geq 1$  since otherwise (unless  $i = 0$ ) the cost is infinite and the problem is not really well defined. We will use the semantic array  $C[i, j] =$  cost of an optimal solution of the points  $x_1, \dots, x_i$  that selects  $j$  points. That is, we choosing a set  $Y$  with  $|Y| = j$  so as to minimize  $\max_{i=1}^i \min_{y \in Y} |x_i - y|$  and  $C[i, j]$  represents the cost of such a solution. As in the problem set it is best to consider the problem as clustering the points since one we have the optimal clustering we can then easily choose the point  $Y$ . And as in other DP algorithms, we concentrate on finding the value of an optimal solution and then modify this so as to compute an actual solution.

- (10 points) Provide a recursive (computational) definition for computing values of this array and briefly justify why your computational definition is equivalent to the semantic definition.

**SOLUTION**

$C'[i, 1] =$  the radius of  $x_1, \dots, x_i$ .

$C'[i, j] = \min_{\ell < i} \max\{C'[\ell, j - 1], \text{radius}(x_{\ell+1}, \dots, x_i)\}$

The “radius” of a set of points  $x_{\ell+1}, \dots, x_i$  on the line is the  $\min_{m: \ell+1 \leq m \leq i} \max\{|x_m - x_{\ell+1}|, |x_i - x_m|\}$ .

Although it would be sufficient and somewhat more efficient to limit ourselves to  $j < i$  (since otherwise the opt cost = 0), the base case and the fact that any cluster containing a single point has radius = 0 will guarantee that  $C'[i, j]$  will be zero whenever  $j \geq i$ .

Clearly  $C[i, 1] = C'[i, 1]$  for all  $i \geq 1$ . (In particular,  $C[0, 1] = C[1, 1] = 0$ .)

By induction on  $j$ , we can prove that  $C[i, j] = C'[i, j]$  for all  $i \geq 0$ . So suppose this is true for some  $j$ , show that it is true for  $j + 1$ . Given that we want to cluster the points  $x_1, \dots, x_i$  into  $j + 1$  clusters, we consider what point will be included in the rightmost cluster. That rightmost cluster will have some leftmost point  $x_{\ell+1}$  for  $\ell + 1 \leq i$ . Then this rightmost cluster has the points  $x_{\ell+1} < \dots < x_i$  and the “cost of this cluster” is the “radius” of this cluster. Once the points in this cluster have been removed, we have one less cluster to cluster the points  $x_1 \dots x_\ell$ . By induction  $C'[i, j]$  is equal to the optimal clustering cost of  $x_1 \dots x_\ell$  using  $j$  clusters.

**NOTE:** A frequent mistake was to use the median  $x_m \in \{x_1, \dots, x_i\}$  as the best point to take, and say that the cost of  $\{x_1, \dots, x_i\}$  is  $\max(|x_1 - x_m|, |x_i - x_m|)$ . But this is not true! As an example, consider  $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 100\}$ . The *median* point is  $x_3$ , which is a distance of 97 from  $x_5$ . However, the *radius* of this set is 96, obtained by selecting  $x_4$ .