Due: Wed, November 3, beginning of lecture

NOTE: Each problem set only counts 5% of your mark, but it is important to do your own work (but see below). Similar questions will appear on the first term test. You may consult with others concerning the general approach for solving problems on assignments, but you must write up all solutions entirely on your own. Anything else is *plagiarism*, and is subject to the University's Code of Behavior. You will receive 1/5 points for any (non bonus) question/subquestion for which you say "I do not know how to answer this question". You will receive .5/5 points if you just leave the question blank.

1. (20 points)

   Consider the following clustering problem. The input is a sorted set X of n distinct real numbers $x_1 < x_2 \ldots < x_n$. A clustering of X is a partitioning of the points into k sets of consecutive inputs for some $1 \leq k \leq n$. $X_1, \ldots X_k$ for some $1 \leq k \leq n$. That is, each cluster X_i is a consecutive sequence of inputs $x_r, x_{r+1} \ldots, x_s$ for some $1 \leq r \leq s \leq n$. The *diameter* d_i of $X_i = max[x \in X_i] - min[x \in X_i]$. Let $\tau > 0$ and define the cost of a clustering $\{X_1, \ldots, X_k\}$ to be $k \cdot \tau + \sum_{i=1}^{k} d_i$. That is, we charge τ and the diameter of X_i for each cluster X_i.

   Provide a dynamic programming algorithm for computing the cost of an optimal clustering; that is, the algorithm must determine the optimal k and the clustering. Provide the appropriate semantic and computational arrays, including base case(s). What is the time complexity of your algorithm?

2. (30 points)

   Consider the following one machine scheduling problem. We are given n jobs $J_1, \ldots, J_n$ with $J_i = (d_i, t_i, v_i)$ where d_i is the deadline for job J_i, t_i is its processing time, and v_i is its profit. Assume all input parameters are positive integers. A schedule is a function $\sigma : \{1, \ldots, n\} \rightarrow \{0, 1, 2, \ldots\} \cup \{\infty\}$ where $\sigma(i) = \infty$ means that job J_i is not scheduled and $\sigma(i) = k$ means that job J_i begins executing at time k. A schedule is feasible if
   (1) for all $i \neq j$ if $\sigma(i) \neq \infty$ and $\sigma(j) \neq \infty$ then $[\sigma(i), \sigma(i)+p_i) \cap [\sigma(j), \sigma(j)+p_j) = \emptyset$
   (2) for all i, if $\sigma(i) \neq \infty$ then $\sigma(i) + p_i \leq d_i$.
   That is, no two scheduled jobs will overlap and every scheduled job finishes before its deadline. The optimization problem is to find a feasible schedule σ that maximizes $\sum_{\sigma(i) \neq \infty} v_i$; That is, to maximize the profit of scheduled jobs in a feasible schedule.

   (a) (5 points)

   Using an exchange argument show that every feasible schedule can be rearranged so that $\sigma(i) < \sigma(j) \neq \infty$ implies that $d_i \leq d_j$; that is, jobs in a feasible schedule can be scheduled in order of their deadlines. [5 points]

(b) (10 points)

Now consider the case that all processing times are not too large, say $p_i \leq n^2$ for all i. Describe a polynomial time dynamic programming algorithm for computing the value of an optimal solution. In particular, specify appropriate semantic and computational arrays, briefly justify that your algorithm is correct, and estimate the time complexity of your algorithm. Hint: First sort the jobs so that $d_1 \leq d_2 \ldots \leq d_n$. Observe that this problem is a generalization of the knapsack problem. The knapsack problem is the special case where all $d_i = W$ where W is the weight bound for the knapsack and the knapsack item weights are $w_i = p_i$.

(c) (10 points)

Now consider the case that all job values are not too large, say $v_i \leq n^2$ for all i. Describe a polynomial time dynamic programming algorithm for computing the value of an optimal solution. Again, specify appropriate semantic and computational arrays, briefly justify that your algorithm is correct, and estimate the time complexity of your algorithm.

(d) (5 points)

Briefly discuss whether or not there is an FPTAS for this problem.

3. (20 points)

Consider the triangulation of a convex polygon in the plane. The polygon is represented by its vertices $v_0, v_2, \ldots, v_{n-1}$ in clockwise order. Any convex polygon with n vertices is triangulated using $n-3$ chords (lines connecting non adjacent vertices) resulting in $n-2$ triangles $\Delta_1, \ldots, \Delta_{n-2}$. Consider an arbitrary weight function w on triangles; for example, we could have $w(\Delta) =$ perimeter of Δ.

Construct a dynamic program for computing the (weight of a) triangulation for an n vertex convex polygon that minimizes $\sum_{i=1}^{n-2} w(\Delta_i)$. That is, define semantic and computational arrays for your dynamic program. What is the time complexity of your algorithm?

Hint: Think of a triangulation as a parse tree.

4. (20 points)

Consider the following local search algorithm for finding a maximum weight independent set in a $k+1$ claw-free graph $G = (V, E)$ with weight function $w : V \rightarrow \Re$. Let $S \subseteq V$ be an independent set and define its neighbourhood $Nbhd(S) = \{S' : S' = S + v - N(v)$ for some $v \notin S\}$ where $N(v)$ is the neighbourhood of vertex v. Let $w(S) = \sum_{v \in S} w(v)$.

$S :=$ any independent set
While $\exists S' \in Nbhd(S)$ such that $w(S)') > w(S)$
  $S := S'$
End While

Show that the locality gap is k.

5. (20 points)

Suppose we have a maximum integral flow f in a flow network $\mathcal{F} = (G, s, t, c)$ with integral capacities.

(a) (5 points) Does there always exist an edge e such that by decreasing the capacity of e by one unit to $c(e) - 1$, the value of the maximum flow is decreased by exactly one unit? Justify your answer.

(b) (5 points)

Does there always exist an edge e such that by increasing the capacity of e by one unit to $c(e) + 1$, the value of the maximum flow is increased by exactly one unit? Justify your answer.

(c) (10 points) Suppose that there is an edge e such that increasing its capacity by one unit will result in the increase of the maximum flow value. Show how to find such an edge in time that is determined by the computation of Dijkstra's least cost paths algorithm.