Due: Thursday, March 5, 2020 4:59PM on MarkUs

You will receive 20% of the points for any (sub)problem for which you write "I do not know how to answer this question." You will receive 10% if you leave a question blank. If instead you submit irrelevant or erroneous answers you will receive 0 points. You may receive partial credit for the work that is clearly "on the right track." You may choose to spend your time looking for solutions on the internet and may likely succeed in doing so but you probably won't understand the concepts that way and will then not do well on the quizzes, midterm and final. So at the very least try to do the assignment initially without searching the internet. If you obtain a solution directly from the internet, you must cite the link and explain the solution in your own words to avoid plagiarizing.

1. (20 pts)

   In the inverted class tutorial, you considered the following problem:

   Input : Jobs $\{J_1, J_2, \ldots, J_n\}$ with $J_i = (p_i, v_i, d_i)$ where
   p_i is the processing time, v_i is the value, and d_i is the deadline of the ith job. Assume that all parameters are positive integers and that $d_i \leq d_k$ for all $i < k$

   A valid schedule is one where jobs complete by their deadline, and no two jobs intersect. (A job can start at exactly the same time as a previous job completes.) The objective is to maximize the total value of all jobs in the schedule. In the tutorial, we wanted a DP algorithm that runs in time $O(n \cdot \max_i d_i)$.

   Now suppose we want to compute a valid schedule that runs in time $O(n^2 \cdot \max_i v_i)$.

   (a) (10 points) Provide a semantic array A for a DP algorithm such that the optimum value can be easily found from entries of the array. (You don't have to worry about a corresponding optimum solution.)

   Solution: This is similar to the second DP for the knapsack problem. Namely, the semantic array can be $T[i, v]$ = minimum time required to obtain value v using the first i jobs. We note that any schedule (including an optimal schedule) of jobs can be rearranged so that $d_i \leq d_k$ for all $i < k$. Note also that the maximum value that can be possibly be obtained is at most $\sum_i v_i \leq n \cdot \max_i v_i$. So we consider $T[i, v]$ for $0 \leq i \leq n$ and $0 \leq v \leq n \cdot \max_i = O(n^2 \max_i v_i)$. Hence if we can compute each entry of $T[i, v]$ in constant time using previous entries we will obtain the requested time complexity.

   (b) (10 points) Provide a recursive algorithm that will compute the entries of A.

   Solution This is just like the analogous DP for the knapsack problem.

2. (20 points)

Consider the following game. The are n boxes and each box i is either empty (and has no reward) or contains a known reward of \$$v_i$. You can assume $v_1 > v_2 > \ldots > v_n$. You are also told that the ith box contains the dollar reward with some probability $p_i > 0$ (and hence with probability $1 - p_i$, the box is empty). The rules of the game are the following:

- You are only allowed to open $\ell \leq n$ boxes.
- If you open a box and it has a \$ reward (i.e., is not empty), then that is your reward and you cannot open any more boxes.

The goal is to compute a sequence $\pi(1), \pi(2), \ldots, \pi(\ell)$ that will determine which subset of boxes to open and the order in which to open these boxes so as to maximize the expectation of the reward you can receive. One way to solve this problem is to use dynamic programming. Here is an appropriate semantic array:

$V[i,t]$ = the maximum expected value that can be obtained when opening at most t boxes where the first box being opened is box i. (Recall, we are assuming $v_1 > v_2 \ldots > v_n$.)

Define a recursive algorithm that will compute $V[i,t]$ for all i ($1 \leq i \leq n$) and all t ($1 \leq t \leq \ell$). **Note:** The desired maximum expected reward is $\max_{i:1 \leq i \leq n} V[i,\ell]$. This may also be construed as a hint.

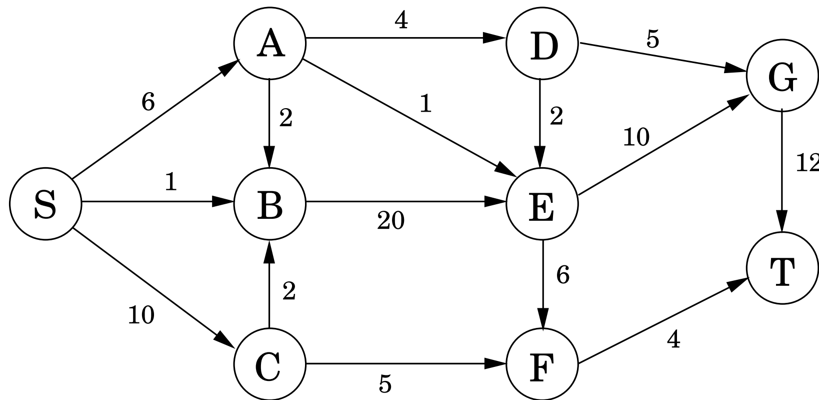
Claim: If you knew which boxes to open, then you should open them in order of decreasing values. You do not need to prove this claim.

(a) (5 points) What is the expected value in $V[i,1]$; that is, you are allowed to open only one box. This is the base case.

Solution: $Vi,1] = \mathbb{E}[p_i \cdot v_i]$.

(b) (15 points) What is the recursive definition of $V[i,t]$ for $t > 1$? Justify briefly

Solution: For $\ell > 1$, $V[i,\ell] = \mathbb{E}[p_i v_i + (1 - p_i) \max_{j<i} V[j, \ell - 1]]$.



3. (20 points)

   Consider the flow network above with integral capacities as depicted.

   (a) (10 points) Compute a maximum flow in the network using the Ford Fulkerson method. Show each iteration. That is, show an augmenting path at the start of each iteration and then the new flow at the end of the iteration.

   (b) (5 points) Identify a min cut. Explain how you found this min cut.

   Solution: You can identify a min cut by doing a breadth first search from the source s in the residual graph of an optimal flow. If we define the cut as a partition $(S, V \setminus S)$ then S is the set of nodes reachable from s.

   (c) (5 points) Identify a minimum set of edges $E' \subset E$ such by raising the capacity of each edge in $e \in E'$, the maximum flow increases by 1 unit. Explain how you found this set E'.

   Solution: Let $G_f = (V, E_f)$ be the residual graph of an optimal solution. We need to create a path from s to t by changing some edges in G_f from having zero residual capacity to capacity 1. So we create an edge wighted cost graph $G' = (V, E)$ where for each edge e in G_f (i.,e., $c_f(e) > 0$) we set $c(e) =$. All other edges have cost 1. Then we find a least cost path in G'. This path identifies the edges whose capacity can be increased by 1 to create a new augmenting path.

4. (20 points)

   An orientation of an undirected graph creates a directed graph $G' = (V, E')$ by giving a direction to each undirected edge $e = (u, v)$; that is, either (u, v) becomes an edge $\langle u, v\rangle \in E'$ from u to v or it becomes an edge $\langle v, u\rangle \in E'$ from v to u.

   Consider the following graph orientation problem:
   Given: An unweighted undirected graph $G = (V, E)$.
   Output: An orientation of G so as to minimize the maximum in-degree of any node. That is, we want to minimize $\max_{v \in V}\{\sum_{u:\langle u,v\rangle \in E'}\}$.

   (a) (15 points) Describe a method to optimally solve this problem in polynomial time.
   Hint: Construct a bipartite graph H whose vertices are the disjoint union of V_1 and V_2 where V_1 are the edges of G and V_2 are the vertices of G. Then convert H to a number of flow networks.

   Solution: We create a flow network (as in the algorithm for computing a maximum matching in a bipartite graph) with a distinguished source node s connected to nodes in V_1, all edges in $V_1 \times V_2$, and edges from V_2 to a distinguished target node t. All edges have capacity 1 except for the edges in $V_2 \times \{t\}$. For each $d = 1, 2, \cdots |E|$, we set the capacity of edges in $V_2 \times \{t\}$ to d and see if we can achieve flow equal to m. The minimum such d will be the min in-degree possible.

   (b) 5 (points) What is the time complexity of your method?

5. (10 points)

   Show that the 4-colourability problem is NP complete. That is, given a graph G, the problem is to decide whether or not G has a valid colouring using at most 4 colours.

   Solution: It is easy to check correctness of each certificate in poly-time, thus, 4-colourability is in NP. Given a graph $G = (V, E)$, we create a new graph $G' = (V', E')$ where $V' = V \cup \{t\}$ where $t \notin V$, and $E' = E \cup \{V \times \{t\}\}$. Then G has a 3 colouring iff G' has a 4 colouring.

6. (20 points)

   Consider the following decision problem which we will call $SATMOST3$:
   Given: A CNF formula F such that every propositional variable x occurs at most 3 times in F. That is, each x occurs (that is, either as x or its complement $\neg x$) is in at most 3 of the clauses in F.
   Decision: Decide if F has a satisfying assignment.

   Show SATMOST3 is NP hard by showing $SAT \leq_p SATMOST3$
   Hint: Consider the formula $(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge \ldots \wedge (\bar{x}_{t-1} \vee x_t) \wedge (\bar{x}_t \vee x_1)$.

   Solution: It is enough to consider only when at least one variable occurs in more than 3 clause. The idea is that we will create a new variable x_j for the jth occurrence each of these variables, x. So if say x occurs for the jth time in some clause C , it is replaced by x_j. Lets

say that x occured r times. It remains to add clauses to that all the x_j are equivalent. That is, we add clauses to force $x_1 \equiv x_2 \ldots \equiv x_r$. This is quivalent to $x_1 \rightarrow x_2 \ldots x_r \rightarrow x_1$ which in turn is the conjection of clauses where we replacce $y \rightarrow z$ by $(\neg y \vee z)$.