CS570 Fall 2019: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 4	20
Problem 2	15	Problem 5	15
Problem 3	15	Problem 6	15
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

1) 20 pts

Mark the following statements as **TRUE** or **FALSE, or UNKNOWN** (if unknown is given as a choice). No need to provide any justification.

For the next four questions consider a flow network G, increase the capacity of an edge e by an amount x>0 to obtain Network G'. Let f, f' denote the value of max flow in G, G'. Then,

a) **[TRUE/FALSE]** f' - f is either 0 or x.

b) **[TRUE/FALSE]** If there is a min-cut in G containing e and f'>f, then there's a min-cut in G' containing e.

c) **[TRUE/FALSE]** If f' = f, there is no min-cut in G' containing e

d) **[TRUE/FALSE]** If f' = f, there is no min-cut in G containing e.

[TRUE/FALSE/UNKNOWN]

Let S denote the set of problems reducible to problem X, where X is NP-hard. Then every problem in S must be in NP.

[TRUE/FALSE/UNKNOWN]

A general 3-SAT problem cannot be solved in polynomial time.

[TRUE/FALSE/UNKNOWN]

If a general integer programming optimization problem can be reduced in polynomial time to a linear programming problem, then P = NP.

[TRUE/FALSE/UNKNOWN]

If a problem is both in NP and NP-hard, then this problem is NP-complete.

[TRUE/FALSE]

A matching in a bipartite graph G=(V, E) can be tested in O(|V|+|E|) time to determine if it is a maximum size matching.

[TRUE/FALSE]

For a graph with all edges having distinct weights there is a unique MST

2) 15 pts

You are given a collection of n points U={u1,…,un} in the plane, each of which is the location of a cell-phone user. You are also given the locations of m cell-phone towers, C={c1,…,cm}. A cell-phone user can connect to a tower if it is within distance Δ of the tower. To guarantee reliability in the network each cell-phone user must be connected to at least three different towers. For each tower ci you are given the maximum number of users, mi that can connect to this tower.

Give a polynomial time algorithm, which determines whether it is possible to assign all the cell-phone users to towers, subject to these constraints. Prove its correctness. (You may assume you have a function that returns the distance between any two points in O(1) time.)

Hint: Use network flow.

Solution:

- We will do this by reduction to network flow/Max flow (It is also possible to reduce to the circulation problem.) --- 2 marks
- Creating the network flow graph --- 4 marks
  - Create an s-t network G=(V,E) where $V = U \cup C \cup \{s\} \cup \{t\}$. Create a unit capacity edge (u_i,c_j) if cell-phone user $u_i \in U$ is within distance Δ of cell phone tower $c_j \in C$. Create an edge from s to each $u_i \in U$ with capacity 3, and create an edge from $c_j \in C$ to t with capacity m_j.
- Explaining the max flow problem solution/logic/feasible schedule --- 4 marks
  - Compute the maximum (integer) flow in this network. We claim that a feasible schedule exists if and only if all the edges exiting s are saturated in the maximum flow. To prove the "only if" part, we consider the following mapping between any valid assignment and a flow in G. We first observe that if there is a valid assignment, there is a valid assignment in which each user is assigned to exactly three towers (since if it was assigned to more than three, removing the additional assignments cannot violated any of the constraints of the problem). If user u_i is assigned to tower c_j , then $dist(u_i, c_j) \leq \Delta$ and so an edge exists between them. We apply one unit of flow along the edge (u_i, c_j). Since u_i is assigned to three towers, it has three units of flow entering it, and so the incoming edge (s, u_i) is saturated. Finally, since this is a valid assignment, the number of users assigned to any tower c_i is at most m_i , and so the flow leaving each c_i is at most m_j .
- Reasoning the working(valid flow) and giving running time -- 5 marks
  - To prove the "if" part, suppose that G has a flow that saturates all the edges coming out of s. We show how to map this to a valid schedule. Because of the edge (s, u_i) is saturated, and the outgoing edges are of unit capacity, each user has three outgoing edges carrying flow. We assign u_i to these three towers. Since edges are created only when users and towers are within distance Δ, these are valid assignments. Since the flow coming out of tower j is at most m_j , tower j is assigned to more than m_j users. Thus, this is a valid assignment. The running time of the algorithm is dominated by the time for the network flow, which is $O(V^3)$, where |V | = m + n.

3) 15 pts

Given n (positive) integers $x_1, x_2, \ldots, x_n x_1, x_2, \ldots, x_n$. The Partition problem asks if there is a subset $S \subseteq [n] S \subseteq [n]$ such that:

$$\sum_{i \in S} x_i = \sum_{i \notin S} x_i$$

It can be proven that the Partition problem is NP-complete. Now assuming that the Partition problem is NP-complete, show that the Bin Covering problem is also NP-complete:

Bin Covering: Given a set of n items with sizes $s_1, s_2, \ldots, s_n s_1, s_2, \ldots, s_n$, a requirement R, and an integer *k*. Can the items be placed into at least *k* bins such that the total size of items in each bin is at least the requirement R?

Solution:

Bin Covering is in NP (we can easily check a proposed solution to confirm a YES instance). The reduction from Partition is as follows. Set $k = 2$ and $R = \frac{1}{2}\sum_i s_i$, where $s_i = x_i$. Clearly, there is a bin covering that uses two bins if and only if there is a partition $S \subset [n]$ such that:

$$\sum_{i \in S} x_i = \sum_{i \notin S} x_i$$

Proof from one direction:
If there is a bin covering solution for 2 bins with $R = \frac{1}{2}\sum_i s_i$, this means the sum of elements in both bins are equal and all the elements are in them, so these bins would be the solution to the partition problem. (One of them can be S)
Proof from the other direction:
If there is a solution for partition problem, it means we have two sets that cover the whole set and sum of elements in each of them is $R = \frac{1}{2}\sum_i s_i$. So this would also be a solution for bin covering problem.

Incorrect solutions:

1) Please notice that we don't know set S, so any solution which assumes we have S and based on that defines R would be incorrect. (such as copying one set, etc.)
2) Dividing sets to get to bins is not a viable solution, since we don't have the sets in advanced.
3) If k is greater than 2, the algorithm sets stronger constraints, because, now the set must be dividable to both 2 and k. Remember, you must show if there is a solution for the partition problem, it would be also a solution for bin covering with your polynomial reduction. (You may have lost some points for that)

Rubric for the given solution:
3 points for the NP part
6 points for the reduction:

- K =2 (2points)
- $R = \frac{1}{2}\sum_i s_i$ (2points) (R = $\sum_{i \in S} si$ would not work, since you don't know the set S)
- Si = xi (2points)

6 points for the proof:

- Each direction 3 points

4) 20 pts

The instructor gives lots of homework assignments, many of which are very difficult. Furthermore, he will only accept a homework assignment if it has been solved in its entirety. Each homework assignment has a value $p_i>0$ that represents its value to you (points awarded for successful completion) and an integer value $w_i>0$, indicating how exhausting it is to complete. If you choose to do homework assignment i, you will get p_i points, but you will not be able to do the next w_i homework assignment(s) (so ,if you do homework i, you will have to skip assignments i+1, i+2,... i+ w_i), and will get a score of zero for those.

Your goal is to get the highest possible total score. Given the n homework assignments' values p_i and w_i, design an efficient dynamic programming algorithm to determine the highest possible total score.

a) Define (in plain English) subproblems to be solved. (4 pts)

Solution

$OPT(j)$ is the highest possible total score one can get from assignments j to n

Rubrics:

-2 if not compatible with the formula in (b) (such as "… from assignments 1 to j")

-4 everything else that cannot be recursively solved or not a subproblem

b) Write the recurrence relation for subproblems. (6 pts)

Solution

$$OPT(j) = \max\{OPT(j+1+w_j) + p_j, OPT(j+1)\}$$

Rubrics:

-2 if "$OPT(j+w_j)$" instead of "$OPT(j+1+w_j)$"

-2 if any minor issue

-6 if the recurrence relation is wrong

c) Using the recurrence formula in part b, write pseudocode (using iteration) to compute the highest possible score. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

Solution

$OPT(n) = p_n$

$for\ j = n-1, \dots, 1$:

    $if(\ j + w_j + 1 > n\ )$:

        $OPT(j) = \max\{p_j, OPT(j+1)\}$

    $else$:

        $OPT(j) = \max\{OPT(j+1+w_j) + p_j, OPT(j+1)\}$

$Return\ OPT(1)$

Rubrics:

-2 if no initialization
-2 if missing return or do not return the optimal value
-2 if no definition for OPT[k] when k > n
-2 if any minor issue
-x if you got -x in (b)
-6 if the pseudocode is wrong

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (2 pts)

Solution
$O(n)$

Rubrics:
-2 if the time complexity is not in $O(n^2)$
-2 if incorrect complexity equation

Alternative Solution:

(a) $OPT(i)$ is the highest possible total score one can get from assignments 1 to i, if assignment i is selected

(b) $OPT(i) = \max\left\{p_i, \max_{j+w_j<i}\{\mathrm{OPT}(j) + p_i\}\right\}$

Rubrics:
-4 if "$j + w_j + 1 = i$" or "$argmax_j(j + w_j + 1 \le i)$" (i.e. the largest index) instead of iterating through all possible j
-4 if only use a function to represent the compatible set $\{j : j + w_j < i\}$ without explaining how to calculate the function

(c)

$OPT(0) = 0, ans = 0$
$for\ i = 1, \dots, n$:
    $OPT(i) = p_i$
    $for\ j = 1, \dots, i-1$:
        $if\ (j + w_j < i)$:
            $OPT(i) = \max\{OPT(i), OPT(j) + p_i\}$
    $ans = \max\{ans, OPT(i)\}$
$Return\ ans$

(d) $O(n^2)$

5) 15 pts
A politician knows three ways to campaign. She estimates that each hour of public speech will generate 20 votes for her, each hour spent on the telephone will generate 15 votes for her, and each hour spent talking with people in shopping areas will generate 35 votes for her. The candidate wants to spend at most 5.5 hours in shopping areas and is required to perform at least eight one-hour speeches. Our politician thinks she can win if she generates a total of at least 1000 votes. How should the politician split her time in order to win while spending the least amount of time campaigning? Present a linear programming solution. You do not need to solve the problem numerically.

a) Describe your variables

P: Hours spent on public speeches
T: Hours spent on the phone
S: Hours spent in shopping areas

b) Present an objective function

Minimize(P+T+S)

c) Present your constraints

P >= 8 2pts
0<=S <= 5.5 2pts
T>=0 1pt
20*P+15*T+35*S >= 1000 2pts

6) 15 pts

Once again, Tom is looking to solve the problem of buying smartphones. He has the information on upcoming phone releases. Each phone *i* releases to the public at some time t_i and is given software support for some number of years s_i and **has a cost of c_i.** Tom wants to spend as little as possible over the rest of his (unfortunately finite) lifetime. Assuming that we know the date of Tom's demise and all the phone release data until that time, design an efficient algorithm to minimize the total cost of Tom's phone purchases for the rest of his life while ensuring that he never goes without a supported phone.

Solution:

Let Ts represents current time and Td represents Tom's demise time. Construct a graph as follows:

- Represent a phone as 2 vertices u_i and v_i with a directed edge u_i->v_i of weight c_i.
- Add outgoing edge of weight 0 from v_i to u_j whenever phone j releases before i's support ends, i.e., whenever s_j is in (s_i, $s_i + t_i$).
- Finally add start and end vertices u,v and add edges u->u_i of weight 0 for all i s.t. Ts is in (s_i,t_i) and edges v_i->v of weight 0 for all i s.t. Td is in (s_i,t_i).

Then, just compute the shortest u->v path in this graph.

Alternative solution (rephrase the above one in DP's language)

Let Ts represents current time and Td represents Tom's demise time. Sort all phones in non-decreasing order by their release date, we get phone 1, phone 2, …, phone i, ...

Subproblem: Let OPT(i) denote minimal money spent to ensure that Tom has a valid phone plan from Ts to t_i + s_i.

Recurrence: OPT(i) = min_j(c_i + OPT(j)) , for all "phone j" such that t_j + s_j >= t_i

Initialization: OPT(0) = 0

Remarks about incorrect DP formulation

Notice that in this problem, t_i may not be integer (though s_i will be an integer). So the following DP solution won't work and is not efficient (pseudo-polynomial).

DP solution (wrong when t_i can be floats)

Let Ts represents current time and Td represents Tom's demise time.

Subproblem: Let OPT(t) denote minimal money spent to ensure that Tom has a valid phone plan from t to Td.

Recurrence: OPT(t) = min_i(c_i + OPT(t_i + s_i)) , for all "phone i" that are available at t, i.e., t_i >= t

Initialization: OPT(t) = 0, for all t >= Td

Rubrics

- max 15 pts if one writes one of the two solutions

- max 12 pts if one writes the incorrect DP formulation as above
- max 9 pts if one uses greedy algorithm
- max 6 pts if one uses brute-force algorithm (to list all possible combinations and select the max)

Extra penalty terms

- -5pts if the optimality is not discussed
- -4pts if the written algorithm is not clearly stated
- -2pts if the algorithm doesn't specify which phones to by
- -1pts if it's not clearly shown that the running time is polynomial
-

Additional Space

Additional Space