CS570

Analysis of Algorithms

Fall 2015

Exam III

Name: ______________________

Student ID: __________________

Email Address:________________

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	12	
Problem 4	15	
Problem 5	15	
Problem 6	12	
Problem 7	11	
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.

1) 20 pts

Mark the following statements as **TRUE**, **FALSE**. No need to provide any justification.

[/FALSE]
If P = NP, then all NP-Hard problems can be solved in Polynomial time.

[/FALSE]
Dynamic Programming approach only works when used on problems with non-overlapping sub problems.

[/FALSE]
In a divide & conquer algorithm, the size of each sub-problem must be at most half the size of the original problem.

[/FALSE]
In a 0-1 knapsack problem, a solution that uses up all of the capacity of the knapsack will be optimal.

[/FALSE]
If a problem X can be reduced to a known NP-hard problem, then X must be NP-hard.

[TRUE/]
If SAT $\leq_P$ A, then A is NP-hard.

[TRUE/]
The recurrence $T(n) = 2T(n/2) + 3n$, has solution $T(n) = \theta(n \log(n^2))$.

[/FALSE]
Consider two positively weighted graphs $G_1 = (V, E, w_1)$ and $G_1 = (V, E, w_2)$ with the same vertices V and edges E such that, for any edge $e \in E$, we have $w_2(e) = (w_1(e))^2$ For any two vertices $u, v \in V$, any shortest path between u and v in G_2 is also a shortest path in G_1.

[/FALSE]
If an undirected graph G=(V,E) has a Hamiltonian Cycle, then any DFS tree in G has a depth |V| - 1.

[TRUE/]
Linear programming is at least as hard as the Max Flow problem.

2) 15 pts

A company makes three models of desks, an executive model, an office model and a student model. Building each desk takes time in the cabinet shop, the finishing shop and the crating shop as shown in the table below:

Type of desk	Cabinet shop	Finishing shop	Crating shop	Profit
Executive	2	1	1	150
Office	1	2	1	125
Student	1	1	.5	50
Available hours	16	16	10	

How many of each type should they make to maximize profit? Use linear programming to formulate your solution. Assume that real numbers are acceptable in your solution.

Solution:

Start by defining your variables:

x = number of executive desks made

y = number of office desks made

z = number of student desks made

Maximize $P=150x+125y+50z$.

Subject to:

$2x + y + z \leq 16$ cabinet hours

$x + 2y + z \leq 16$ finishing hours

$x + y + .5z \leq 10$ crating hours

$x \geq 0,\ y \geq 0,\ z \geq 0$

3) 12 pts

Given a graph $G=(V, E)$ and a positive integer $k < |V|$. The longest-simple-cycle problem is the problem of determining whether a simple cycle (no repeated vertices) of length k exists in a graph. Show that this problem is NP-complete.

Solution:

Clearly this problem is in NP. The certificate will be a cycle of the graph, and the certifier will check whether the certificate is really a cycle of length k of the given graph.

We will reduce HAM-CYCLE to this problem. Given an instance of HAM-CYCLE with graph G=(V,E), construct a new graph G'=(V', E) by adding one isolated vertex u to G. Now ask the longest-simple-cycle problem with k = |V| <|V'| for graph G'.

If there is a HAM-CYCLE in G, it will be a cycle of length k = |V| in G'.

If there is no HAM-CYCLE in G', there must not be no cycle of length |V| in G'. If there is, we know the cycle does not contain u, because it is isolated. So the cycle will contain all vertex in |V|, which is a HAM-CYCLE of the G.

The reduction is in polynomial time, so longest-simple-path is NP-Complete.

4) 15 pts

Suppose there are n steps, and one can climb either 1, 2, or 3 steps at a time. Determine how many different ways one can climb the n steps. E.g. if there are 5 steps, these are some possible ways to climb them: (1,1,1,1,1), (1,2,1,1), (3, 2), (2,3), etc. Your algorithm should run in linear time with respect to n. You need to include your complexity analysis.

Solution:

Let $T(n)$ denote the number of different ways for climbing up n stairs.

There are three choices for each first step: either 1, 2, or 3. T(n)= T(n-1)+ T(n-2)+ T(n-3)

The boundary conditions :

when there is only one step, $T(1) = 1$. If only two steps, $T(2) = 2$. T(3)= 4. ((1,1,1), (1,2), (2,1), (3))

Pseudo code as below:

```
if n is 1
  return 1
else if n is 2
  return 2
else
  T[1] = 1
  T[2] = 2
  T[3]=4
  for i = 4 to n do
    T[i] = T[i-1]+T[i-2]+ T[i-3]
  return T[n]
```

The time complexity is $\Theta(n)$.

5) 15 pts

We'd like to select frequencies for FM radio stations so that no two are too close in frequency (creating interference). Suppose there are n candidate frequencies {f1,…fn}. Our goal is to pick as many frequencies as possible such that no two selected frequencies fi, fj have |fi-fj|<e (for a given input variable e). Design a greedy algorithm to solve the problem. Prove the optimality of the algorithm and analyze the running time.

Solution: Our greedy algorithm:

1) Let F be the sorted list of the given frequencies in non-decreasing order.
2) Select the minimum frequency, say f, from the list F and output it in the result set.
3) Remove all the frequencies that are not compatible (i.e. |fi-fj| < e) with the above selected minimum frequency from the list F.
4) Repeat step 2 and 3 until the list F is empty.

Complexity of the algorithm: Sorting n frequencies takes O(nlogn), steps 2 and 3 can be performed in O(n) time, so complexity of the algorithm is O(nlogn).

We prove optimality of our algorithm by using mathematical induction.

Let $\{f_1^o, f_2^o, \ldots, f_m^o\}$ denote the frequencies selected by the optimal set in sorted order. Similarly $\{f_1^g, f_2^g, \ldots, f_p^g\}$ for our greedy algorithm.

We claim that $f_i^g \le f_i^o$

Initial step: If $f_1^g = f_1^o$, our proof is complete. If it isn't, we know that $f_1^g < f_1^o$; as we are selecting the minimum frequency first.

Inductive step: Assume that $f_i^g \le f_i^o$, need to prove for $f_{i+1}^g \le f_{i+1}^o$.

As f_i^o is compatible with f_{i+1}^o , $f_{i+1}^o \geq f_i^o + e$,

induction hypothesis – $f_i^o \geq f_i^g$

from above equations $f_{i+1}^o \geq f_i^g + e$. So f_{i+1}^o is compatible with f_i^g . Our greedy algorithm at i+1 step, selects the minimum frequency among all the compatible frequencies. So $f_{i+1}^g \leq f_{i+1}^o$. Hence our claim is proved.

If size of the optimal set (m) is greater than the size our greedy frequency set(p), then $f_p^g + e \leq f_p^o + e \leq f_{p+1}^o$. So f_{p+1}^o is compatible with f_p^g and can be added to our greedy frequency set, which contradicts the fact that our greedy algorithm cannot select any frequency that is greater than f_p^g . So size of our greedy frequency set is equal to the size of our optimal set, hence optimality of our algorithm is proved.

6) 12 pts

Let S be an NP-complete problem, and Q and R be two problems whose classification is unknown (i.e. we don't know whether they are in NP, or NP-hard, etc.). We do know that Q is polynomial time reducible to S and S is polynomial time reducible to R. Mark the following statements True or False **based only on the given**

information, and explain why.

(i) Q is NP-complete

False. Because Q <p S, Q is at most as hard as S. Because S is in NPC, Q is not necessary to be in NPC, e.g., it could be in P, or could be in NP but not in NPC.

(ii) Q is NP-hard

False. Because Q <p S, Q is at most as hard as S. Then Q is not necessary to be in NP-hard, e.g., it could be in P.

(iii) R is NP-complete

False. Because S <p R, R is at least as hard as S. Then R is not necessary to be NPC. It is possible to be in NP-hard but not in NPC.

(iv) R is NP-hard

True. Because S <p R, R is at least as hard as S. Then R in NP-hard.

7) 11 pts

Consider there are n students and n rooms. A student can only be assigned to one room. Each room has capacity to hold either one or two students. Each student has a

subset of rooms as their possible choice. We also need to make sure that there is at least one student assigned to each room.
Give a polynomial time algorithm that determines whether a feasible assignment of students to rooms is possible that meets all of the above constraints. If there is a feasible assignment, describe how your solution can identify which student is assigned to which room.

Solution:

Construct a flow network as follows,
For each student i create a vertex a_i, for each room j create a vertex b_j, if room j is one of student i's possible choices, add an edge from a_i to b_j with upper bound 1.
Create a super source s, connect s to all a_i with an edge of lower bound 0 and upper bound 1.
Create a super sink t, connect all b_j to t with an edge of lower bound 1 and upper bound 2.
Connect t to s with an edge of lower bound and upper bound n.
Find an integral circulation of the graph, because all edges capacity and lower bound are integer, this can be done in polynomial time.
If there is one, for each a_i and b_j, check whether the flow from a_i to b_j is 1, if yes, assign student i to room j.

Additional Space

Additional Space