CS570
Analysis of Algorithms
Spring 2016
Exam III

Name: ______________________

Student ID: __________________

Email Address:________________

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	20	
Problem 4	20	
Problem 5	10	
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.

1) 20 pts
Mark the following statements as **TRUE**, **FALSE,** or **UNKNOWN**. No need to provide any justification.

[TRUE/FALSE/UNKNOWN]
If X≤p Y, and X is NP-complete, then Y is NP-hard.

[TRUE/FALSE/UNKNOWN]
If X≤p Y, and X is NP-complete, then Y is NP-complete.

[TRUE/FALSE/UNKNOWN]
If X≤p Integer Programming, then X is NP-hard.

[TRUE/FALSE/UNKNOWN]
If X≤p Linear Programming, then X is in P.

[TRUE/FALSE/UNKNOWN]
3-SAT cannot be solved in polynomial time.

[TRUE/FALSE/UNKNOWN]
If graph G has no cycles, then the independent set problem in G can be solved in polynomial time.

[TRUE/FALSE]
Although the general Travelling Salesman Problem is NP-complete, in class, we presented a 2-approximation algorithm for it that runs in polynomial time.

[TRUE/FALSE]
Breadth first search is an example of a divide-and-conquer algorithm.

[TRUE/FALSE]
Memoization requires memory space which is linear in size with respect to the number of unique sub-problems.

[TRUE/FALSE]
The smallest element in a binary max-heap of size *n* can be found with at most *n/2* comparisons.

2) 15 pts

Ted and Marshall are taking a road trip from Somerville to Vancouver. Because they are going on a 52-hour drive, Ted and Marshall decide to switch off driving at each rest stop they visit; however, because Ted has a better sense of direction than Marshall, he should be driving both when they depart and when they arrive (to navigate the city streets). Given a route map represented as a weighted undirected graph G = (V, E, w) with positive edge weights, where vertices represent start point, end point, and all rest stops, and edges represent routes between rest stops, (edge weights representing the route distance), devise an efficient algorithm to find a route (if possible) of minimum distance between Somerville and Vancouver such that Ted and Marshall alternate edges and Ted drives the first and last edge.

Solution:

Let's make a new graph G'. For every vertex u in G, there are two vertices u_M and u_T in G' : these represent reaching the rest stop u when Marshall (for u_M) or Ted (for u_T) will drive next. For every edge (u, v) in G, there are two edges in G' : (u_M, v_T) and (u_T , v_M). Both of these edges have the same weight as the original. We run Dijkstra's algorithm on this new graph to find the shortest path from Somerville_T to Vancouver_M (since Ted drives to Vancouver, Marshall would drive next if they continued). This guarantees that we find a path where Ted and Marshall alternate, and Ted drives the first and last segment. Constructing this graph takes linear time, and running Dijkstra's algorithm on it takes O(V log V +E) time with a Fibonacci heap (it's just a constant factor worse than running Dijkstra on the original graph).

3) 20 pts

Suppose you have access to a function VALID that returns true if its input is a valid English word, and false otherwise. You are given a sentence where the punctuation has been stripped, for example, “dynamicprogrammingispowerful”. Assuming calls to VALID take constant time, give an $O(n^2)$ time algorithm to determine whether the input can be split into a sequence of valid words.

Solution:

Denote the input string by $s = s_1s_2\ldots s_n$, where s_k is the kth letter of the string.
We define VALID_SEQU(i) to be true if $s_1s_2\ldots s_i$ can be split into a sequence of valid words, and false otherwise.

As a base case, let VALID_SEQU(0) be true.

Then, VALID_SEQU(i) is true, if and only if there exists some $j < i$ such that VALID_SEQU(j) is true, and $s_{j+1}s_{j+2}\ldots s_i$ is a valid word.

Recursive relation: (we use V(i) to represent VALID_SEQU(i))

$V(i) = (V(0)\ \&\&VALID(s_1, \ldots, s_i)\ )\ \|\ (V(1)\&\&VALID(s_2, \ldots, s_i))\ \|\ \ldots.\|\ (V(i-1)\&\&VALID(s_i))$

We can use two loops to compute VALID_SEQU(i) for i = 1…n

```
Initialize V[0] = false
For i = 1 to n
    V[i] = false
    For j = 0 to i-1
        If V[j] == true && VALID($s_{j+1}s_{j+2}\ldots s_i$) == true
            Then V[i] = true
    End
End
Return V[n].
```

According to the pseudo code above, the algorithm runs in $O(n^2)$.

4) 20 pts
Suppose we have a variation on the 3-SAT problem called Min-3-SAT, where the literals are never negated. Of course, in this case it is possible to satisfy all clauses by simply setting all literals to true. But, we are additionally given a number k, and are asked to determine whether we can satisfy all clauses while setting at most k literals to be true. Prove that Min-3-SAT is NP-Complete.

The following three sections are only the NP-Hard part of Q4, not the complete solution.

Vertex Cover <=p Min 3-SAT

Consider any instance of Vertex Cover problem, i.e., given a graph with n vertices and m edges, we want to determine if no more than k vertices can be selected to cover the m edges.

Consider the reduction from the arbitrary Vertex Cover instance to an instance of Min-3Sat in polynomial time.
We could set up one literal u for each vertex u, and we set up a clause (u|v|u) for each edge (u,v).

If there exists a set of vertices S with size no more than k that cover the m edges, we set the literals corresponding to the vertices in S to be true, while setting the rest of the literals to be false, then we set no more than k literals to be true. Since S is a vertex cover, each edge (u,v) has u or v (or both) in S, and the corresponding clause (u|v|u) is satisfied.

If there exists a set of literals S' with size no more than k such that, by setting all the literals in S' to be true, we can satisfy all the m clauses. For each literal u in S', we put the corresponding vertex u into a set S, so the size of S is no more than k. Moreover, since each clause (u|v|u) is satisfied, either u or v (or both) are in S', and correspondingly vertex u or v (or both) are in S. Then the corresponding edge (u,v) has at least one incident vertex in S. So S is a vertex cover.

Independent Set <=p Min 3-SAT

Consider any instance of Independent set problem, i.e., given a graph with n vertices and m edges, we want to determine if at least k vertices can be selected, no two of which are adjacent.

Consider the reduction from the arbitrary Independent set instance to an instance of Min-3Sat in polynomial time.

We could set up one literal u for each vertex u, and we set up a clause (u|v|u) for each edge (u,v).

If there exists a set of vertices S with size at least k, no two of which are adjacent, we set the literals corresponding to the vertices in S to be false, while setting the rest of the literals to be true, then we set no more than n-k literals to be true. Since S is an independent set, each edge (u,v) has u or v (or none) in S, and the corresponding clause (u|v|u) is satisfied.

If there exists a set of literals S' with size no more than n-k such that, by setting all the literals in S' to be true, we can satisfy all the m clauses. For each literal u not in S', we put the corresponding vertex u into a set S, so the size of S is at least k. Moreover, since each clause (u|v|u) is satisfied, either u or v (or both) are in S', and correspondingly vertex u or v (or none) are in S. Then the corresponding edge (u,v) has at most one incident vertex in S. So S is an independent set.

3-SAT <=p Min 3-SAT

Let's reduce 3-SAT to this problem in polynomial time.

For any instance of 3-SAT problem Ψ0(x1, x2, …, xn), we replace all negated xi by a new literal yi. So we can define Ψ1(x1,y1,x2,y2, ..., xn, yn) where the literals are never negated.

And we defind Ψ2(x1,y1,x2,y2, ..., xn, yn) = Ψ1(x1,y1,x2,y2, ..., xn, yn) & (x1|y1|x1) & (x2|y2|x2) & … & (xn|yn|xn) and we only allow n literals to be true.

If Ψ2 can be satisfied while setting at most n literals to be true, in any pair of xi and yi, xi and yi can't be both false. Also, xi and yi can't be both true for only n literals can be true. So we can gaurantee that yi = !xi and Ψ0(x1, x2, …, xn) = true for such assignment.

If Ψ0 can be satisfied, we just assign yi = !xi so that Ψ2 is satisfied and only n literals are true among 2n literals.

5) 10 pts

A manufacturer produces two products, X and Y, with two machines, A and B.

- The cost of producing each unit of X is:
  a) for machine A: 50 minutes,
  b) for machine B: 30 minutes.
- The cost of producing each unit of Y is:
  a) for machine A: 24 minutes,
  b) for machine B: 33 minutes.
- Working plans for a particular week are:
  a) 40 hours of work on machine A,
  b) 35 hours of work on machine B.
- The week starts with:
  a) A stock of 30 units of X and 90 of Y, (in other words, at the start of the week we already have 30 units of X and 90 units of Y)
  b) A demand of 75 units of X and 95 of Y. (in other words, this many units of X and Y need to delivered to clients at the end of the week)

Plan the production (i.e. how many units of X and Y to produce), in order to end the week with the maximum combined stock of X and Y, while satisfying the above time constraints and demand constraints. Formulate the problem using Linear programming

Solution:

Define
xa = units of X to be produced by A
xb = units of X to be produced by B
ya = units of Y to be produced by A
yb = units of Y to be produced by B

Then the formulation is as follows:
Maximize $(xa+xb + 30 - 75) + (ya+yb + 90 - 95)$
Subject to:
$50xa + 24ya \leq 40 \times 60$
$30xb + 33yb \leq 35 \times 60$
$xa+xb \geq 75 - 30$
$ya+yb \geq 95 - 90$
xa>=0; xb>=0;
ya>=0,yb>=0;

6) 15 pts

There are n people, $p_1, p_2, \dots, p_n$ and k sets. Each set consists of several people and a person can be in more than one set. We need to select one person from each set and the maximum times a person is selected should be less than m. Give a polynomial time algorithm to find such a selection if there is one, or to indicate that such a selection is not possible. Prove that your solution is correct.

Solution:

We reduce this problem to maximum flow problem as follows:

Construct a graph with $n + k + 2$ vertices: n vertices $p_1, p_2, \dots, p_n$ corresponding to the n people, k vertices $s_1, s_2, \dots, s_k$ corresponding to the k sets, and two special vertices s and t.

For $i = 1, \dots, n$, put an edge from s to p_i with capacity $m – 1$. Put an edge from p_i to s_j with capacity 1 if person p_i is in the s_j. For $j = 1, \dots, k$, put an edge from s_j to t with capacity 1.

We now find a maximum flow in this graph. If the flow has value k, then there is a way to select one person from each set such that each person is not selected more than $m – 1$ times. The flow has one incoming edge for each set; if for set s_j this edge comes from person p_i then p_i is the person selected for set s_j.

To prove that the solution is correct is to prove that the reduction is correct: the graph has a maximum flow of value k only if there is a way to select one person from each set such that each person is not selected more than $m – 1$ times.

Since the graph has a flow of value k, those edges from s_j to t are all saturated. The conservation constraint for each s_j implies that some person in s_j is selected. The capacities of those edges from s to p_i being $m – 1$ ensure that each person is selected less than m times.

Grading Guideline:

4pts for vertices: 2pt for $p_1, p_2, \dots, p_n$; 2pt for $s_1, s_2, \dots, s_k$; 0pt for s and t.
6pts for edges: 2pts for edges from s to p_i; 2pts for edges from p_i to s_j; 2pts for edges from s_j to t.

2pts for how to a find a selection using the maximum flow of the constructed graph.
3pts for proof of correctness.