CS570 Fall 2018: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 5	10
Problem 2	8	Problem 6	16
Problem 3	14	Problem 7	12
Problem 4	20		
	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**. No need to provide any justification.

[TRUE/FALSE]
Let X be a decision problem. If we prove that X is in the class NP and give a poly-time reduction from X to 3-SAT, we can conclude that X is NP-complete.

[TRUE/FALSE]
Let A be an algorithm that operates on a list of n objects, where n is a power of two. A spends $\Theta(n^2)$ time dividing its input list into two equal pieces and selecting one of the two pieces. It then calls itself recursively on that list of n/2 elements. Then A's running time on a list of n elements is O(n).

[TRUE/FALSE]
If there is a polynomial time algorithm to solve problem A then A is in NP.

[TRUE/FALSE]
A pseudo-polynomial time algorithm is always slower than a polynomial time algorithm.

[TRUE/FALSE]
In a dynamic programming formulation, the sub-problems must be non-overlapping.

[TRUE/FALSE]
A spanning tree of a given undirected, connected graph G = (V, E) can be found in O(E) time.

[TRUE/FALSE]
Ford-Fulkerson can return a zero maximum flow for flow networks with non-zero capacities.

[TRUE/FALSE]
If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $h(n) = \Theta(f(n))$

[TRUE/FALSE]
There is a polynomial-time solution for the 0/1 Knapsack problem if all items have the same weight but different values.

[TRUE/FALSE]
If there are negative cost edges in a graph but no negative cost cycles, Dijkstra's algorithm still runs correctly.

2) 8 pts

Let G = (V, E) be a simple graph with n vertices. Suppose the weight of every edge of G is one.
Note: In a simple graph there is at most one edge directly connecting any two nodes. For example, between nodes u and v there will be at most one edge uv.
(a) What is the weight of a minimum spanning tree of G? (1 pt)

(b) Suppose we change the weight of two edges of G to 1/2. What is the weight of the minimum spanning tree of G? (2 pts)
(c) Suppose we change the weight of three edges of G to 1/2. What is the minimum and maximum possible weights for the the minimum spanning tree of G? (2 pts)
(d) Suppose we change the weight of k < V edges of G to 1/2. What is the minimum and maximum possible weights for the the minimum spanning tree of G? (3 pts)

1. Any spanning tree of G has exactly V − 1 edges, since all the weights are one its total weight is V − 1.
2. To answer this question, consider running Kruskal on G, it will first pick the two lighter edges as they cannot form a cycle (the graph is simple), then V − 3 other edge each with weight one. Therefore, the total value is V − 3 + 2(1/2) = V − 2.
3. Use the similar approach as (b). Kruskal first selects two light edges. There are two possibilities for the third edge: (i) it forms a cycle with the first two edges or (ii) it does not form a cycle with the first two edges not. In case (i), Kruskal selects two edges of weight 1/2 and V − 3 edges of weight 1, therefore, the total weight is V − 3 + 2(1/2) = V − 2. In case (ii), Kruskal selects three edges of weight 1/2 and V − 4 edges of weight 1, therefore, the total weight is V − 4 + 3(1/2) = V − 5/2.
4. Similar to (c), the minimum weight happens when the k light edges do not form any cycle. In this case, MST contains k edges of weight 1/2 and n − 1 − k edges of weight 1. Therefore, its weight is k/2 + n − 1 − k = n − k/2 − 1. The maximum weight happens if the k edges span as few vertices as possible. This happens when the k edges are part of an almost *complete graph*. Let h be a variable denoting the number of nodes in this subgraph such that the number of edges in a complete graph comprised of these edges nodes exceeds k. In particular, define h to be the smallest number such that, the binomial coefficient Binomial[h,2] > k, ie., (h choose 2) > k. Accordingly, we can pack all k light edges between h vertices. Consequently, the MST has h− 1 edges of weight 1/2 and n − 1 − (h − 1) edges of weight 1. Therefore, its weight is n − h/2 − 1/2.

Rubric:
For each part (a,b,c,d), No partial marking.
This also applies to the parts where minimum and maximum is asked.
If either is wrong then there is no partial marks for that part.

Q3. 14 pts

Recall that in the discussion class we showed that the Bipartite **Undirected** Hamiltonian Cycle problem is NP-complete. Now in the Bipartite **Directed** Hamiltonian Cycle problem, we are given a bipartite directed graph and asked whether there is a simple cycle which visits every node exactly once. Note that this problem might potentially be easier than Hamiltonian Cycle because it assumes a directed bipartite graph. Prove that Bipartite Directed Hamiltonian Cycle is in fact still NP-Complete.

Solution:

i) This problem is NP. Given a candidate path, we just check the nodes along the given path one by one to see if every node in the network is visited exactly once and if each consecutive node pair along the path are joined by an edge. (3)

ii) To prove the problem is NP-complete, we reduce Directed Hamiltonian Cycle (DHC) problem to Bipartite Directed Hamiltonian Cycle (BDHC) problem. (2)

Given an arbitrary directed graph G, we split each vertex v in G into two vertex v_in and v_out. Here v_in connects all the incoming edges to v in G; v_out connects all the outgoing edges from v in G. Moreover, we connect one directed edge from v_in to v_out. After doing these operations for each node in G, we form a new graph G'. (3)

Here G' is bipartite graph, because we can color each v_in "blue" and each v_out "red" without any coloring conflict.

If there is DHC in G, then there is a BDHC in G'. We can replace each node v on the DHC in G into consecutive nodes v_in and v_out, and (v_in, v_out) is an edge in G'. Then the new path is a BDHC in G'. (3)

On the other hand, if there is BDHC in G', then there is a DHC in G. Note that if v_in is on the BDHC, v_out must be the successive node on the BDHC. Then we can merge each node pair (v_in, v_out) on the BDHC in G' into node v and form a DHC in G. (3)

In sum, G has DHC if and only if G' has a BDHC. This completes the reduction, and we confirm that the given problem is NP-complete

Alternate solution:
We can also use Undirected Hamiltonian Cycle for the reduction.
Prove that it's NP as before. (3)
Note that you are using Bipartite Undirected Hamiltonian Cycle for reduction. (2)
Given an arbitrary undirected bipartite graph G, convert G to G' so the vertices in G' stay the same, but all undirected edges are converted to directed edges in **both directions**. G' remains bipartite. (3)

If there is an undirected Hamiltonian cycle in G, you can use the corresponding edges(in either direction) to find a Directed Hamiltonian cycle in in G'. (3)
On the other hand, if there is a directed Hamiltonian cycle in G', you could convert the corresponding edges to undirected edges and find the equivalent cycle in G. (3)

Common mistakes:

Major mistake in checking for NP: -2
Checking for edges instead of vertices: -1
Missing the NP check altogether: -3

Mistake in the conversion step: -2

Missing the two-way proof: -4
Missing either side of the proof: -3
Incomplete explanation as to why the proof holds: -2

If you are proposing to reduce a certain NP-Complete problem, but are going with a solution corresponding a different NP-Complete problem, this shows misunderstanding of the concept: -7

If you are proposing to reduce an incorrect (and irrelevant) NP-complete problem: -8

4) 20 pts.
Suppose you want to sell n roses. You need to group the roses into multiple bouquets with different sizes. The value of each bouquet depends on the number of roses you used. In other words, we know that a bouquet of size i will sell for v[i] dollars. Provide an algorithm to find the maximum possible value of the roses by deciding how to group them into different-sized bouquets.
Here is an example:
Input: n = 5, v[1..n] = [2,3,7,8,10] (i.e. a bouquet of size 1 is $2, bouquet of size 2 is $3, … , and finally a single bouquet of size n=5 is $10.
Expected Output = 11 (by grouping the roses into bouquets of size 1, 1, and 3)

a) Define (in plain English) subproblems to be solved. (4 pts)
OPT(j) is the maximum value of j roses.

b) Write the recurrence relation for subproblems. (6 pts)
OPT(j) = MAX (OPT(j), {OPT(j-i) + v[i]} for all sizes 0 <= i <= j)

i.If iteration of i from 0 to j is missing => -1 points
ii.If OPT(j) is missing in the MAX term => -1 points
iii.0 points for wrong recursion

c) Using the recurrence formula in part b, write pseudocode to compute the maximum possible value of the n roses. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

```
OPT[0] = 0
OPT[1] = v[1]
For (int j=2; j < n; j ++ )
     max = 0
          For (int i=1; i < j; i++)
               If (i<=j && max < OPT[j-i]+v[i] )
                    max = OPT[j-i]+v[i]
     OPT[j]=max
```

i) Missing one loop => -1 point
ii) Missing two loops => -2 points
iii) If in b) you got x points, you get x points in the pseudo code. Apart from the case where you missed the iteration in b, where you will get x+1 points in this question
iv) 0 points for no pseudocode or any modification from b.

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

5) 10 pts
Give a tight asymptotic upper bound (O notation) on the solution to each of the following recurrences. You need not justify your answers.

$T(n) = 2T(n/8) + n$

Solution: $(n^{1/3} \lg n)$ by Case 2 of the Master Method.

$T(n) = T(n/3) + T(n/4) + 5n$

Solution: Use brute force method
$T(n) = cn + (7/12)\, cn + (7/12)^2\, cn + \ldots$
$= (n)$

Rubric:
No partial marks. 5 points for each correct answer.

6) 16 pts

There are *n* people and *m* jobs. You are given a payoff matrix, C, where C_{ij} represents the payoff for assigning person i to do job j. Each applicant has a subset of jobs that he/she is interested in. Each job opening can only accept one applicant and a job applicant can be appointed for only one job. You need to find an assignment of jobs to applicants that maximizes the total payoff of your assignment. Write an **Integer Linear Program** (with discrete variables) to solve this problem.

Solution

We are looking for a perfect matching on a bipartite graph of the minimal cost.

Let $x_{ij} = \begin{cases} 1, & if\ edge(i,j)\ is\ in\ matching \\ 0, & otherwise \end{cases}$

Objective: $\sum_{\substack{1<i<m \\ 1<j<n}} c_{ij}.x_{ij}$

Subject to:

1. $\sum_j x_{ij} \leq 1 \quad , where\ i = 1,2,\dots,n$
2. $\sum_i x_{ij} \leq 1 \quad , where\ j = 1,2,\dots,n$
3. $x_{ij} \in \{0,1\} \quad , \forall\ i,j$
4. $x_{ij} = 0$, for all jobs j that applicant i is not interested in.

Rubric:

-4 if objective function is incorrect.

-6 if you mention equal to instead of less than equal to sign for constraints 1 and 2

-2 if constraint 3. is not mentioned, but only is defined to take values 0 or 1.

-2 if discrete variable is not defined.

0 points awarded if you use a max-flow, min-cut formulation

7) 12 pts

The Maximum Acyclic Subgraph is stated as follows: Given a directed graph find an acyclic subgraph of that contains as many arcs (i.e. directed edges) as possible.

Give a 2-approximation algorithm for this problem.

Hint: Consider using an arbitrary ordering of the vertices in G.

Solution:

1. Pick an arbitrary vertex ordering This partitions the arcs into two acyclic subgraphs and . Here A1 is the set of arcs where and is the set of arcs where . Thus at least one of or contains half the arcs of G. The maximum acyclic subgraph contains at most the total number of arcs in G, so we have a factor 2-approximation algorithm (12)
2. Divide the set of nodes into two nonempty sets, A and B. Consider the set of edges from A to B and the set of edges from B to A. Throw out the edges from the smaller set and keep the edges from the larger set (break ties arbitrarily). Recursively perfomr the algorithm on A and B individually (12)

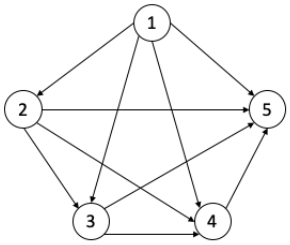
Rubric:

If you are only considering the forward or backward edges (-2)
If you start with one node and add the forward/backward edges to/from neighbors, your answer won't work for disconnected graphs (-2)
If your final answer is not 0.5 approximation (-6)
If your answer is not acyclic (-6)
If your algorithm is vague and not implementable (for example if you said, "we find max independent set", or "we find the topological order") (0)
If your algorithm has conceptual flaws (for example if you are adding an edge to the sub graph and also deleting it from the sub graph) (0 pts)

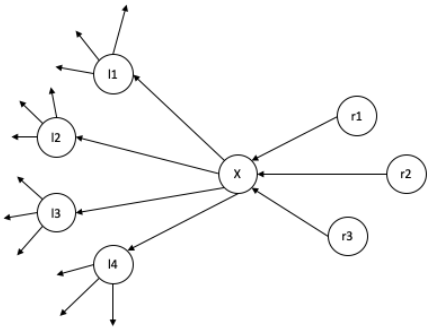
Some of the answers that do not work:

1. If your final answer is a tree (a DFS or a BFS or any other kinds of trees):

   You can imagine an acyclic complete directed graph such as below. Your algorithm should return at least half of the edges (n * (n-1) / 4) but a tree can have n-1 edges. Thus it is not 0.5 approximation (6 pts).



2. If you start adding/removing edges to/from the graph, until it is acyclic. Your algorithm can choose an edge that is repeated in many cycles and end up removing all the other edges in those cycles.
   You can try your method on the following graph (All l(i) nodes are connected to all r(j) nodes). If your algorithm chooses all the (r(j), X) and (X, l(i)) edges (7 edges), you cannot add any (l(i), r(j)) edges (12 edges). Your final answer (7) is less than half of the best answer (16/2 = 8) so it is not 0.5 approximation.



CS570 Spring 2018: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 5	15
Problem 2	15	Problem 6	10
Problem 3	15	Problem 7	10
Problem 4	15		
Total: 100 Points			

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**. No need to provide any justification.

[TRUE]
To prove that a problem X is NP-hard, it is sufficient to prove that SAT is polynomial time reducible to X.

[FALSE]
If a problem Y is polynomial time reducible to X, then a problem X is polynomial time reducible to Y.

[TRUE]
Every problem in NP can be solved in polynomial time by a nondeterministic Turing machine.

[TRUE]
Suppose that a divide and conquer algorithm reduces an instance of size n into 4 instances of size $n/5$ and spends $\Theta(\underline{n})$ time in the conquer steps. The algorithm runs in $\Theta(n)$ time.

[FALSE]
A linear program with all integer coefficients and constants must have an integer optimum solution.

[FALSE]
Let M be a spanning tree of a weighted graph G=(V, E). The path in M between any two vertices must be a shortest path in G.

[TRUE]
A linear program can have an infinite number of optimal solutions.

[TRUE]
Suppose that a Las Vegas algorithm has expected running time $\Theta(n)$ on inputs of size n. Then there may still be an input on which it runs in time $\Omega(n^2)$.

[FALSE]
The total amortized cost of a sequence of n operations gives a lower bound on the total actual cost of the sequence.

[FALSE]
The maximum flow problem can be efficiently solved by dynamic programming.

2) 15 pts

Consider a uniform hash function h that evenly distributes n integer keys {0, 1, 2, …, n-1} among m buckets, where $m < n$. Several keys may be mapped into the same bucket. The uniform distribution formally means that Pr($h(x)$=$h(y)$) =1/m for any x, y ∈ {0, 1, 2, …, n-1}. What is the expected number of total collisions? Namely how many distinct keys x and y do we have such that $h(x) = h(y)$?

Solution:

Let the indicator variable X_{ij} be 1 if $h(k_i) = h(k_j)$ and 0 otherwise for all $i \neq j$. Then let X be a random variable representing the total number of collisions. It can be calculated as the sum of all the $X_{ij}'s$: $X = \sum_{i<j} X_{ij}$

Now we take the expectation and use linearity of expectation to get:

$$E[X] = E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} P(h(k_i) = h(k_j)) = \sum_{i<j} \frac{1}{m} = \frac{n(n-1)}{2m}$$

Rubrics:

Describe choosing 2 keys from n keys: 10 pts.

Describe $\sum_{i<j} P\left(h(k_i) = h(k_j)\right)$: 10 pts.

Common mistaks:

1. n/m: The mistake is that it thinks the expected number of collisions for each key is 1/m. 7pts.
2. 2n/m: Similar to 1. 7pts.
3. (n-1)/m: Similar to 1. 7pts.
4. (m+1)/2: Basically, no clue. 3 pts.
5. n(n-1)/m: Duplicates. 13 pts.
6. n^2/m: duplicated. 13 pts
7. (1-(1-1/m)^(n-1))*n: wrong approach. 5 pts.
8. n-m: wrong approach. 3pts.
9. $\frac{1}{m}\sum_{i=0}^{n-1} i$: 15 pts
10. n(n-1)/2: The mistake is that it does not consider collision probability. 10 pts.
11. N(n+1)/2m: minor mistake. 13 pts.

3) 15 pts.

There are 4 production plants for making cars. Each plant works a little differently in terms of labor needed, materials, and pollution produced per car:

	Labor	Materials	Pollution
Plant 1	2	3	15
Plant 2	3	4	10
Plant 3	4	5	9
Plant 4	5	6	7

The goal is to maximize the number of cars produced under the following constrains:

- There are at most 3300 hours of labor
- There are at most 4000 units of material available.
- The level of pollution should not exceed 12000 units.
- Plant 3 must produce at least 400 cars.

Formulate a linear programming problem, using minimal number of variables, to solve the above task of maximizing the number of cars.

Solution:

We need four variables, to formulate the LP problem: x_1, x_2, x_3, x_4, where x_i denotes the number of cars at plant-i.

Maximize $x_1 + x_2 + x_3 + x_4$
s.t.
$x_i \geq 0$ for all i
$x_3 \geq 400$
$2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 3300$
$3x_1 + 4x_2 + 5x_3 + 6x_4 \leq 4000$
$15x_1 + 10x_2 + 9x_3 + 7x_4 \leq 12000$

Rubric:

- Identifying 4 variables (2 pts)
- Correct objective function (3 pts)
- Each condition (2 pts)

4) 15 pts.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ placed on each vertex v. You will now play the following game. You pick a vertex u that contains at least two tokens, remove two tokens from u and add one token to any one of adjacent vertices. The objective of the game is to perform a sequence of moves such that you are left with exactly one token in the whole graph. You are not allowed to pick a vertex with 0 or 1 token. Prove that the problem of finding such a sequence of moves is NP-complete by reduction from Hamiltonian Path.

Solution:

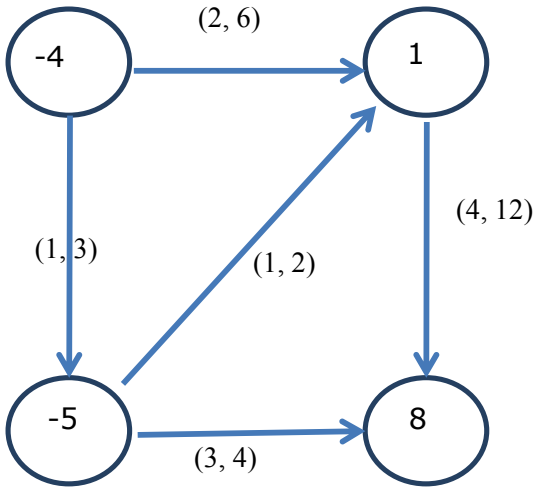
Construction: given a HP in G, we construct G' as follows. Traverse a HP in G and placed 2 tokens on the starting vertex and one token on each other vertex in the path.
Claim: G has a HP iff G' has a winning sequence.
->) by construction before the last move we will end up with a single vertex having two tokens on it. Making the last move, we will have exactly one token on the board.
<-) since there is only one vertex with 2 tokens, we will start right there playing the game. Each next move is forced. When we finish the game, we get a sequence moves which represents a HP.

Rubrics:

- Didn't prove it is in NP: -5
- Didn't prove it is NP-hard: -10
- Assigned two tokens on one random vertex instead of the starting vertex of Hamiltonian path: -2 (Since it is a reduction from Hamiltonian path, not from Hamiltonian cycle, 2 tokens should be assigned on the starting vertex)
- Assigned wrong number of tokens on one vertex: -3
- Assigned wrong number of tokens on two vertices: -6
- Assigned wrong number of tokens on three or more vertices (considered as not a valid reduction from Hamiltonian path): -7
- Not a valid reduction from Hamiltonian path: -7

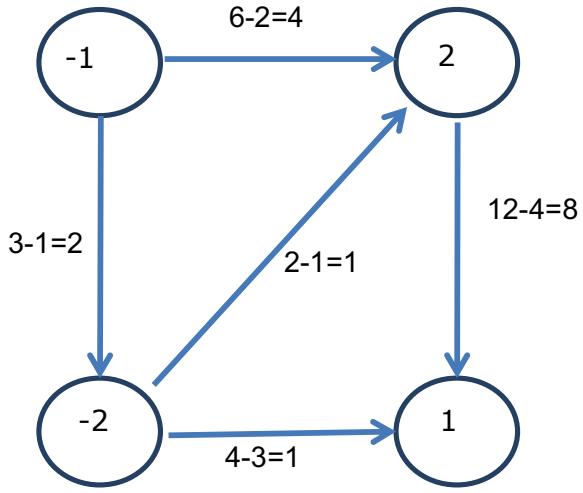
5) 15 pts.

In the network below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.



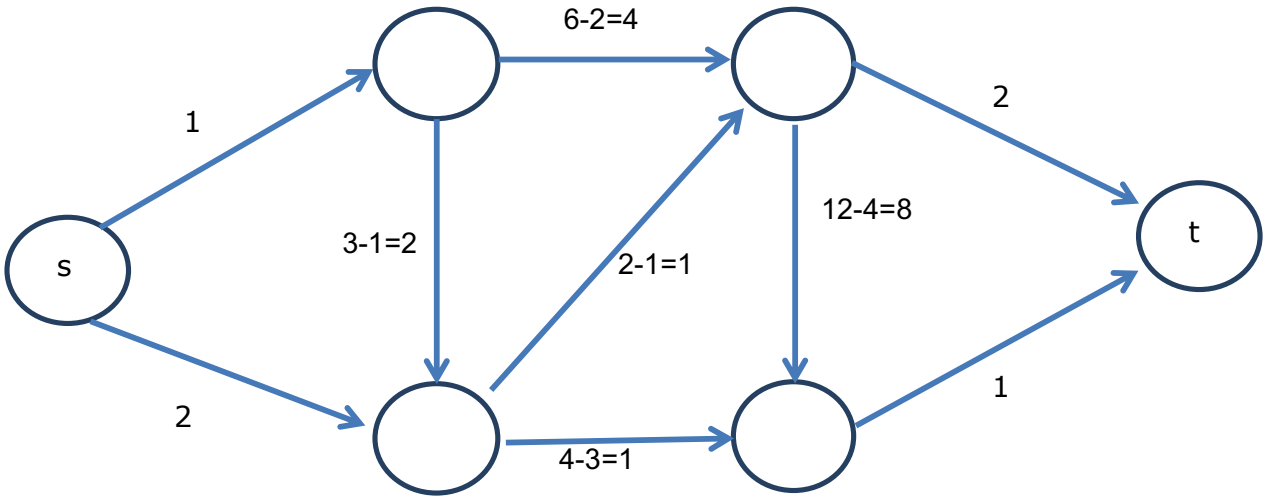
a) Turn the circulation with lower bounds problem into a circulation problem without lower bounds (6 pts)

- **Each wrong number for nodes: -1**
- **Each wrong number for edges: -0.5**



b) Turn the circulation with demands problem into the max-flow problem (5 pts)

- **s and t nodes on right places: each has 0.5 point**
- **directions of edges from s and t: each direction 0.5 point**
- **values of edges from s and t: each value 0.5 point**

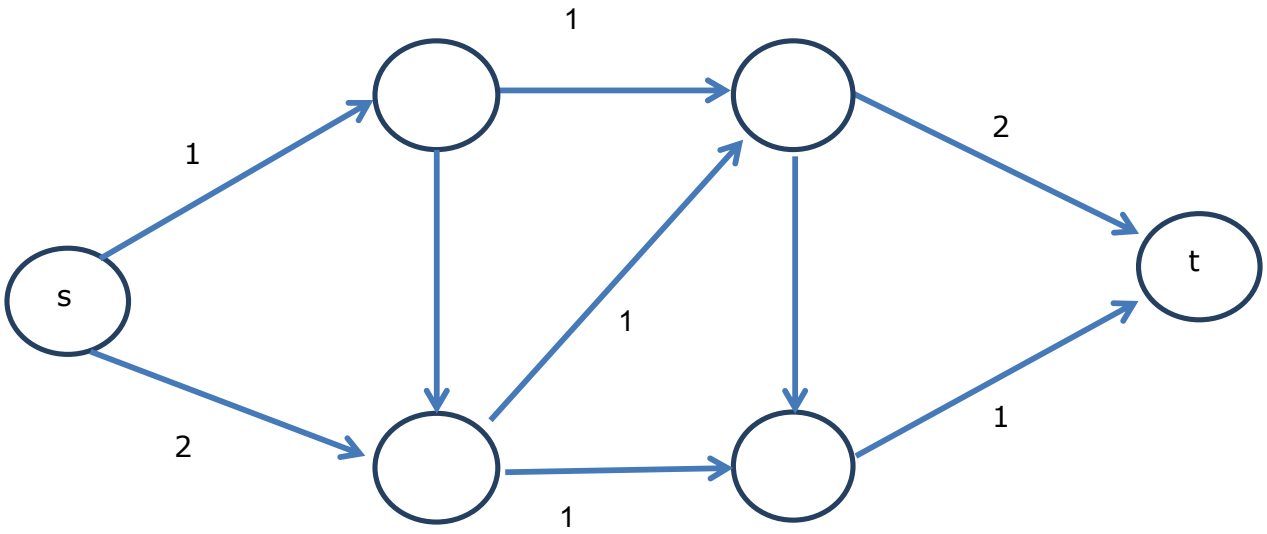


c) Does a feasible circulation exist? Explain your answer. (4 pts)

Solution.

Yes. There is a s-t flow |f|= 2+1=1+2=3. It follows, there is a feasible circulation.

- **Explanation: 2 points.**
- **Correct Graph/flow: 2 points.**



6) 10 pts.

We wish to determine the most efficient way to deliver power to a network of cities. Initially, we have n cities that are all connected. It costs $c_i \geq 0$ to open up a power plant at city i. It costs $r_{ij} \geq 0$ to build a cable between cities i and j. A city is said to have power if either it has a power plant, or it is connected by a series of cables to some other city with a power plant (in other words, the cables act as undirected edges). Devise an efficient algorithm for finding the *minimum cost* to power all the cities.

Solution

Consider a graph with the cities at the vertices and with weight r_{ij} on the (undirected) edge between city i and city j. Add a new vertex s and add an edge with weight ci from s to the ith city, for each i. The minimum spanning tree on this graph gives the best solution (power stations need to be opened at those cities *i* for which (s,i) is part of the MST.)

Common Mistakes:

- Dynamic Programming: There does not exist a predefined order in which the nodes/edges will be processed and because of this you cannot define the problem based on smaller sub-problems and hence, none of the solutions which were based on Dynamic Programming were correct.
- Max-Flow/Min Cut: It was a surprise to see many ran a Max Flow algorithm to minimize the overall cost of edges. The minimum cut in a network has nothing to do with minimizing flow!! Furthermore, in this problem we want connectivity in the graph and edges on the min cut don't guarantee any level of connectivity.
- ILP: I can't recall anybody correctly formalizing the problem as in ILP. However, even a correct formulization is going to be NP-Hard and far from efficient. A maximum of 3 points was given to a CORRECT ILP solution.
- Dijkstra's algorithm is for finding the shortest path between a single point and every other node in the graph. The resulting Tree from running Dijkstra, is not necessarily an MST so it's not the correct approach to solve this problem.

7) 10 pts.

We want to break up the graph $G = (V, E)$ into two disjoint sets of vertices $V=A \cap B$, such that the number of edges between two partitions is as large as possible. This problem is called a max-cut problem. Consider the following algorithm:

- Start with an arbitrary cut C.
- While there exists a vertex v such that moving v from A to B increases the number of edges crossing cut C, move v to B and update C.

Prove that the algorithm is a 2-approximation.
Hint: after algorithm termination the number of edges in each of two partitions A and B cannot exceed the number of edges on the cut C.

Solution

Let w(u,v) = 1, there exists an edge between u and v, and 0 otherwise. Let

$$W = \sum_{(u,v)\in E} w(u,v)$$

By construction, for any node u∈A:

$$\sum_{v\in A} w(u,v) \le \sum_{v\in B} w(u,v)$$

Here, the left hand side is all edges in partition A. The RHS – the crossing edges. If we sum up the above inequality for all u∈A, the LHS will contain the twice number of edges in A.

$$2\sum_{(u,v)\in A} w(u,v) \le \sum_{u\in A, v\in B} w(u,v) = C$$

We do the same for any node in B:

$$2\sum_{(u,v)\in B} w(u,v) \le \sum_{u\in A, v\in B} w(u,v) = C$$

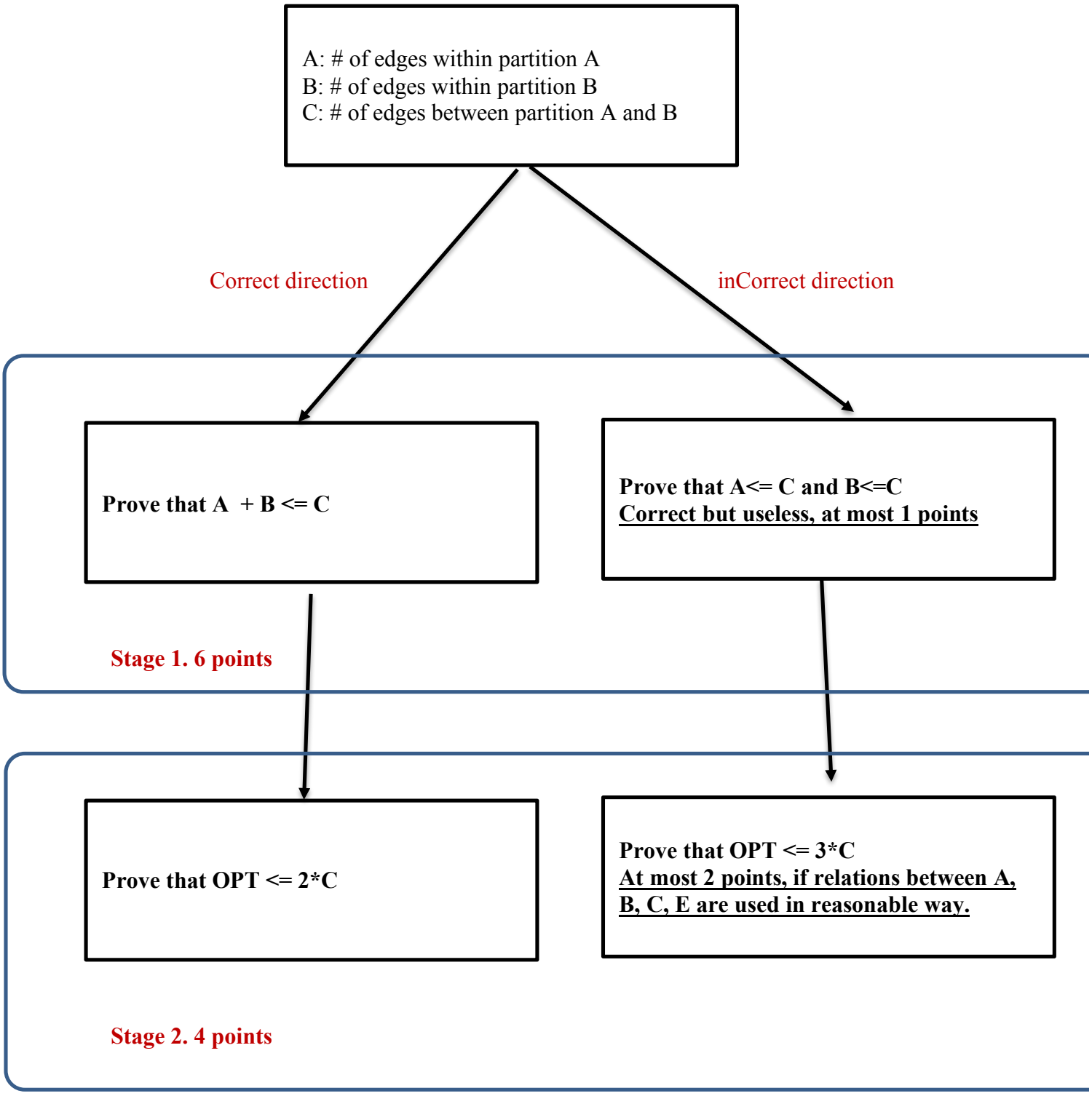
Add them up

$$\sum_{(u,v)\in A} w(u,v) + \sum_{(u,v)\in B} w(u,v) \le C$$

Next we add edges on the cut C to both sides.

$$\sum_{(u,v)\in A} w(u,v) + \sum_{(u,v)\in B} w(u,v) + \sum_{u\in A, v\in B} w(u,v) = W \le 2C$$

Clearly the optimal solution OPT≤W, thus OPT≤W ≤2C. It follows, OPT/C ≤ 2.


A: # of edges within partition A
B: # of edges within partition B
C: # of edges between partition A and B
Correct direction
inCorrect direction
Prove that A + B <= C
Prove that A<= C and B<=C
Correct but useless, at most 1 points
Stage 1. 6 points
Prove that OPT <= 2*C
Prove that OPT <= 3*C
At most 2 points, if relations between A, B, C, E are used in reasonable way.
Stage 2. 4 points

1. The solution can be divided into two parts. **Part 1** is worth 6 points, and **part 2** is worth 4 points.
    1. Proving the hint, i.e. "numbers of edges in partition A and partition B cannot exceed the number of edges on the cut C". It is most important and challenging part of this problem.
    2. Prove the algorithm is a 1/2-approximation.
2. About part 1, here are two acceptable solutions and some solutions not acceptable:
    1. If your solution is same to the official solution, using inequalities to rigorously prove the hint, it is perfect, 6 points.
    2. Here is a weaker version. A the end of algorithm, for **any** node **u** in the graph, without loss of generality assume it is in part A, **the number of edges** connecting it to nodes in part B (i.e. edges in the cut) is greater than or equal to **the number of edges** connecting it to other nodes in part A. So, the **total number of edges** in the cut is greater than or equal to the **total number of edges** within each part. This solution did not consider the important fact that both inter-partition and intra-partition edges are counted twice. 3 ~5 points according to the quality of statements.
    3. One incorrect proof: some answers declare that C is increasing and A + B is decreasing when calling steps 2 of the algorithm. This is a correct but useless statement. 0 points
    4. YOU MUST PROVE THE HINT. SIMPLY STATING OR REPHRASING IT GET 0 POINTS IN THIS STEP. Here is one example that cannot get credits: "According to the algorithm, we move one vertex from A to B if it can increase the number of edges in cut C. Then the number of edges in C is more than edges in partition A and partition B." 0 points
    5. "If move v from A to B can increase the number of edges cross cut C, it means v connect to more vertex in A than in B". It is rephrasing the algorithm, and failed to get the key point. If the above statement is changed to "If move v from A to B can increase the number of edges cross cut C, it means v connect to more vertex in different partition than in the same partition", it will get at least 3 points.
3. About part 2, the correct answer should use relation between OPT, E, A, B, C. Missing key inequalities like OPT <= E, will lead to loss of grades.
4. If the answer is incorrect, you may get some points from some steps that make sense. If the answer is on the incorrect direction in the above diagram:
    1. Many students attempted to PROVE the edges_in_partition_A <= C, and edges_in_partition_B <= C, but not proving edges_in_partition_A + edges_in_partition_B <= C. This is trivial according to step 2 of algorithm, and is not useful for proving the final statement. This will be treated as "correct but useless/irrelevant statements". AT MOST 1 point.
    2. From above inequalities, OPT<=3*C is a correct conclusion. It is different from the question, but since the logics in this stage is same, you can get at most 2 points out of 4.

CS570 Fall 2017: Analysis of Algorithms Exam III

	Points
Problem 1	20
Problem 2	15
Problem 3	10
Problem 4	15
Problem 5	15
Problem 6	10
Problem 7	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**. No need to provide any justification.

[TRUE/FALSE]
To prove that a problem X is NP-complete, it is sufficient to prove that 3SAT is polynomial time reducible to X.

[TRUE/FALSE]
Finding the minimum element in a binary max heap of n elements takes O(log n) time

[TRUE/FALSE]
We are told that in the worst case, algorithm A runs in O(n log n) and algorithm B runs in $O(n^2)$. Based on these facts, there must be some N that when n>N, algorithm A runs faster than algorithm B.

[TRUE/FALSE]
The following recurrence equation $T(n)=3T(n/3) + 0.1\ n$ has the solution: $T(n)= \Theta\ (n \log(n^2))$.

[TRUE/FALSE]
Every problem in NP can be solved in exponential time by a deterministic Turing machine

[TRUE/FALSE]
In Kruskal's MST algorithm, if we choose edges in decreasing (instead of increasing) order of cost, we will end up with a spanning tree of maximum total cost

[TRUE/FALSE]
If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[TRUE/FALSE]
If problem X can be solved using dynamic programming, then X belongs to P.

[TRUE/FALSE]
If Vertex-Cover $\in$ P then SAT $\in$ P.

[TRUE/FALSE]
Assuming P!=NP, and X is a problem belonging to class NP. There is no polynomial time algorithm for X.

2) 15 pts

We have a directed graph G=(V, E) where each edge (u,v) has a length l(u,v) that is a positive integer. Let n denote the number of vertices in G. Suppose we have previously computed a n x n matrix d[·,·], where for each pair of vertices u,v ∈ V, d[u,v] stores the length of the shortest path from u to v in G. The d[·,·] matrix is provided for you.

Now we add a single edge (a,b) to get the graph G' = (V,E'), where E' = E ∪{(a,b)}. Let l(a,b) denote the length of the new edge. Your job is to compute a new distance matrix d'[·,·], which should be filled in so that d'[u,v] holds the length of the shortest path from u to v in G', for each u,v ∈ V.

(a) Write a concise and efficient algorithm to fill in the d'[·,·] matrix. You should not need more than about 3 lines of pseudocode. (12 pts)

Solution:

For each pair of vertices u,v ∈ V:

Set d'[u,v] = min(d[u,v],d[u,a] + l(a,b) + d[b,v]).

Grading:

- Considering all pairs of vertices (6 points)
- Comparing the current shortest path with the length of the path going through edge (a,b) and choosing the smaller path (6 points)

(b) What is the asymptotic worst-case running time of your algorithm? Express your answer in O(·) notation, as a function of n and |E|. (3 pts)

Solution:

$O(n^2)$.

Grading:

- Correct Answer (3 points)
- Incorrect answer (0 points)

3) 10 pts.

Recall the 0/1 knapsack problem:

Input: n items, where item i has profit p_i and weight w_i, and knapsack size is W.
Output: A subset of the items that maximizes profit such that the total weight of the set $\leq$ W.
You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:

Sort items according to decreasing p_i/w_i (breaking ties arbitrarily).
While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio. In other words, if the value of the optimal solution is P*, prove that there is no constant ρ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

Solution:

The definition of a constant factor is that P >= ρ P*. Giving an example were P = P* and saying $\rho = 1$ or giving two different examples and showing the ratio P/P* is not always the same does not mean there does not exist a constant approximation ratio.

The correct way to show this is not possible is to provide an example with multiple (at least two) items and their respective weights and profits. Then you need to show that the greedy algorithm and the optimal algorithm will generate two different solutions (/profits) and need to show in your example the ratio P/P* can be unboundedly small.

One example can be as follows:
Consider an item with size 1 and profit 2, and another with size W and profit W. Our greedy algorithm above will take the first item, while the optimal solution is taking the second item, so this algorithm has approximation ratio 2/W which can go arbitrarily close to zero.

Grading:
A correct solution has 10 points.

Common Mistakes:

- Giving two different examples and showing that the ratio of greedy to optimal is different doesn't mean there is no constant approximation ratio.
- The existence of a constant approximation factor will not imply P = NP.
- Greedy is not optimal!!

4) 15 pts.
The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color.

Prove that the five-coloring problem is NP-complete.

Hint: You can assume that graph 3-coloring is NP-complete

Solution:

Show that 5-coloring is in NP:
Certificate: a color solution for the network, i.e., each node has a color.
Certifier:

i. Check for each edge (u,v), the color of node u is different from the color of node v;
ii. Check at most 5 colors are used

This process takes O(m+n) time.

Show that 5-coloring is NP-hard: prove that 3-coloring $\leq_p$ 5-coloring

Graph construction:

Given an arbitrary graph G. Construct G' by adding 2 new nodes u and v to G. Connect u and v to all nodes that existed in G, and to each other.

G' can be colored with 5 colors iff G can be colored with 3 colors.

i. If there is valid 3-color solution for G, say using colors {1,2,3}, we want to show there is a valid 5-coloring solution to G'. We can color G' using five colors by assigning colors to G according to the 3-color solution, and then color node u and v by additional two different colors. In this case, node u and v have different colors from all the other nodes in G', and together with the 3-coloring solution in G, we use at most 5 colors to color G'.
ii. If there is a valid 5-coloring solution for G', we want to show there is a valid 3-coloring solution in G. In G', since node u and v connect to all the other nodes in G and to each other, the 5-coloring solution must assign two different colors to node u and v, say colors 4 and 5. Then the remaining three colors {1,2,3} are used to color the remaining graph G and form a valid 3-color solution.

Solving 5-coloring to G' is as hard as solving 3-color to G, then 5-coloring problem is at least as hard as 3-coloring, i.e., 3-coloring $\leq_p$ 5-coloring.

Grading Rubrics:

1. Prove the problem is NP (3 points in total)
   i) Mentioning the necessity of proving NP (1 point)
   ii) Checking two connecting nodes (u,v) having different colors (1 point)
   iii) Checking the total number of colors used is at most (1 point)
2. Prove the problem is NP-hard (12 points in total)
   i) Graph construction (4 points in total)
      a. Adding two additional nodes (1 point)
      b. Connecting to all the nodes in G (2 point)
      c. Connecting node u and v (1 point)
   ii) Proof (8 points in total)
      a. 3color -> 5 color (2 points). This is trivial even under other constructions (even incorrect constructions)
      b. 5color -> 3 color (6 points).
         - If your graph construction is incorrect, you will get 0 point in this part.
         - Under the graph construction posted in the standard solution:
            - ✓ Explaining the two additional nodes u and v must have colors different from those used in G (3 points)
            - ✓ Explaining the two additional nodes u and v must have two different colors in between (3 points)

Popular Mistakes:

1. Try to show, if G is 3-colorable, then G is 5-colorable
   – Loss 4 points for construction (actually no construction is given);
   – Lose 6 points for proof (proof for 5 color-> 3 color cannot be correct)
2. Without adding additional nodes to G, but try to "manipulate colors" during construction
   – Loss 4 points for construction (actually no construction is given);
   – Lose 6 points for proof (proof for 5 color-> 3 color cannot be correct)
3. In the graph construction, someone added additional nodes to G, but assigned additional colors to during the construction, and then in the proof, claim them "using two different colors".
   – Loss 3 points for construction (wrong construction);
   – Loss 6 points for proof
4. In the graph construction, for each node i in G, someone added two additional nodes ui and vi while connecting them to node i, and then connect or not connect ui and vi.
   – Loss 2 points for construction (at least they know connecting with somewhere in G);
   – Loss 6 points in proof
5. In the graph construction, someone added two nodes u and v to G and connect them to all the nodes in G, but did not connect them to each other
   – Loss 1 point for construction

- If they can correctly explain that, to prove G has 3-color solution given G' having 5-color solution, their construction can guarantee node u and v take different colors from all the nodes in G, they will only lose 3 points; otherwise, they will lost 6 points.

6. In the graph construction, someone added two nodes "into" each edge
   - Loss 2 points for construction
   - Loss 6 points in proof

5) 15 pts.

The price of the recently introduced cryptocurrency, Crypcoin, has been changing rapidly over the last two months. Given the hourly price chart of this currency for the last two months, you are tasked to find out the maximum profit one could have made by buying 100 Crypcoins in one transaction and subsequently selling 100 Crypcoins in one transaction within this period.

Specifically, given the price chart of Crypcoin over this period P[i] for i = 1 to n, we're looking for the maximum value of (P[k]−P[j])*100 for some indices j, k, where $1 \leq j < k \leq n$.

Examples: If the array is [2, 3, 10, 6, 4, 8, 1] then the returned value should be 8 *100 (Diff between 2 and 10, times 100 Cyrpcoins). If the array is [7, 9, 5, 6, 3, 2] then the returned value should be 2 * 100 (Diff between 7 and 9, times 100 Cyrpcoins)

Describe a divide and conquer algorithm to solve the problem in O(n log n) time. You do not need to prove that your algorithm is correct.

Solution:
CrypMAX(P[1..n]):
If $n \leq 1$, return 0.
Set max_L := CrypMAX(P[1, … , $\lfloor n/2 \rfloor$])
Set max_R := CrypMAX(P[$\lfloor n/2 + 1 \rfloor$,…,n])
Set x := min(P[1, … , $\lfloor n/2 \rfloor$]).
Set y := max(P[$\lfloor n/2 + 1 \rfloor$,…,n]).
Return max(max_L,max_R, (y−x)*100).

Complexity Analysis: There are two recursive calls on a subarray of half the size, plus a linear (Θ(n)) scan to find the smallest/largest of either half. Thus, the recurrence relation is T(n) = 2T(n/2) +Θ(n), which solves to Θ(nlogn).

Alternate solution:
We can modify the algorithm to also return the smallest and largest elements of the array (i.e. return the triple (best price, min of array, max of array)). This allows the algorithm to avoid the computation of finding the smallest/largest elements of each half of the array, as the recursive calls have already calculated them. (As an additional detail, the base case also needed a minor tweak.) Our runtime would thus be T(n) = 2T(n/2) +Θ(1), which solves to Θ(n).

Grading rubric:

1. Basic structure for an efficient divide-and-conquer algorithm (5 pts)
    a. No ambiguity, e.g., "divide it into several subproblems" or "divide it into two subproblems" will not receive the 5 pts. "… two halves" or "… two subproblems with equal size" are acceptable.
2. Correctly conquer the problem by finding the cross-segment term: max(P(mid:end)-min(P(1:mid). (5 pts)
    a. The max/min operation are required to be written explicitly, to achieve an O(n) complexity for each step. Finding the maximal different between two halves in brute force requires O(n^2).
3. Complete the entire algorithm (5 pts). Examples:
    a. -1 pts for missing base case;
    b. -1 pts for multiplying by 100 more than once inappropriately.
4. Non-divide-and-conquer algorithms:
    a. Correct O(n log(n)) algorithm receive 10 pts.
    b. Correct O(n^2) algorithm receive 5 pts.
    c. Other receive 0 pt.

Note and common mistakes:

1. You cannot buy the coin after your sell. (sell must occur no earlier than buy).
    a. You cannot simply pick the highest price to sell and the lowest price to buy.
2. If you buy on i-th day, you don't have to sell it on the i+1-th day.
3. You cannot do binary search on unsorted array.
4. **[Most common mistake]** The buy and sell prices does not have to be the lowest or highest price. For example, [10,4,5,1]. **Therefore, returning the best buy/sell dates from the subproblems does not help you with merging the two subproblems, which requires the highest/lowest price from them.**

Partially correct example solutions and grades:

1. Brute force search, O(n^2) (5pts)

```
Ret = 0;
For i in 1 to n:
        For j in i+1 to n:
                Ret = max(Ret, P(j)-P(i));
Return 100*Ret;
```

Note: the recursive version: F(i,j) -> f(i+1,j) and f(i, j-1) and P(j)-P(i) also receive 5 pts.

2. Non-divide-and-conquer O(n) (10 pts)
   Segment P into fewest monotonical increasing intervals. Calucate the max-difference within these intervals and return the largest difference * 100.
3. Non-divide-and-conquer One-pass O(n) (10 pts)

```
Min = P(1), Ret = 0;
```

```
For i in 1 to n:
    Ret = max(Ret, P(i)-Min)
    Min = min(Min, P(i))
Return Ret*100
```

4. Divide-and-conquer O(n^2) (5 pts)

```
Def Solve(P[1:m]):
k = argmax(P)
Return max(P(k)-min(P[1:k]), Solve(P[k+1:m])
```

Final solution is 100*Solve(P); Note argmax is O(n) operation on unsorted array.

6) 10 pts.

Recall the Circulation with Lower Bounds problem:

- Each directed edge e has a lower bound l_e and an upper bound c_e on flow
- Node v has demand value d_v, where d_v could be positive, negative, or zero
- Objective is to find a feasible circulation that meets capacity constraints over each edge and satisfies demand conditions for all nodes

We now add the following constraints and objective to this problem:

- For each node v with demand value $d_v = 0$, the flow value through that node should be greater than l_v and less than c_v
- Since we suspect there may be some issues at a specific node *x* in the network, we want to minimize the flow going through this node as much as possible.

Give a linear programming formulation for this problem.

Solution:

Objective: minimize $\sum_{all\ e\ into\ x} f_e$ (4 points)

Constraints:

1. $l_e \leq f_e \leq c_e \ for\ each\ edge\ e$ (2points)
2. $d_{v_i} = \sum_{all\ e\ into\ v_i} f_e - \sum_{all\ e\ out\ of\ v_i} f_e \ \ for\ each\ vertex\ v_i$ (2 points)
3. $l_{v_j} \leq \sum_{all\ e\ into\ v_j} f_e \leq c_{v_j}\ for\ each\ node\ v_j\ such\ that\ d_{v_j} = 0$ (2 points)

- 1 point may be deducted from each of your constraints where you have not specified what is the flow
- Up to 3 points may be deducted from your objective if you have not specified what is the flow in your solution. (not defining fin for instance)

7) 15 pts
Woody will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and n + 1 denote the left and right endpoints of the original log of length L. Let di denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The wood-cutting problem is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize Woody's total payment. Remember that for a single cut on a given log, Woody gets paid an amount equal to the length of that log. Give a dynamic programming algorithm to solve this problem.

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

Let $c(i, j)$ be the min cost of cutting a log with left endpoint i and right endpoint j at all its marked locations.

Grading:
Incomplete definition: -2 pts
No definition: -4pts

b) Write the recurrence relation for subproblems. (7 pts)

(Dynamic programming)
$c(i,j) = \min_{i<k<j}\{c(i,k) + c(k,j) + d_j - d_i\}$
Base case: if $j = i+1$, $c(i, j) = 0$

Grading:
Error in recursion: -2pts, if multiple errors, deduction adds up.
Wrong recurrence relation: -7pts
Base case is missing (case of $j = i+1$): -2pts
Wrong base case: -2pts

c) Compute the runtime of the algorithm in terms of n. (4 pts)

$O(n^3)$

Grading:
Used big Theta notation instead of big O notation: -2pts
Missing big O notation: -2pts
Wrong runtime complexity: -4pts

CS570
Analysis of Algorithms
Summer 2017
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	15	
Problem 5	10	
Problem 6	15	
Problem 7	10	
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** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Every problem in P can be reduced to 3-SAT in polynomial time.

[TRUE/FALSE]
If there is a polynomial-time algorithm for 2-SAT, then every problem in NP has a polynomial-time algorithm.

[TRUE/FALSE]
If all edge weights are 1, 2, or 3, the shortest path problem can be solved in linear time.

[TRUE/FALSE]
Suppose G is a graph with n vertices and $n^{1.5}$ edges, represented in adjacency list representation. Then depth-first search in G runs in $O(n^{1.5})$ time.

[TRUE/FALSE]
The weight of a minimum spanning tree in a positively weighted undirected graph is always less than the total weight of a minimum spanning path (Hamiltonian Path with lowest weight) of the graph.

[TRUE/FALSE]
If A is in NP, and B is NP-complete, and A $\leq$p B then A is NP-complete.

[TRUE/FALSE]
Given a problem B, if there exists an NP-complete problem that can be reduced to B in polynomial time, then B is NP-complete.

[TRUE/FALSE]
If an undirected connected graph has the property that between any two nodes u and v, there is exactly one path between u and v, then that graph is a tree.

[TRUE/FALSE]
Suppose that a divide and conquer algorithm reduces an instance of size n to four instances of size $n/5$ and spends $\Theta(n)$ time in the divide and combine steps. The algorithm runs in $\Theta(n)$ time.

[TRUE/FALSE]
An integer N is given in binary. An algorithm that runs in time $O(sqrt(N))$ to find the largest prime factor of N is considered to be a polynomial-time algorithm.
Ex: Prime factorization of 84: 2 x 2 x 3 x 7, so the largest prime factor of 84 is 7

2) 15 pts

We are given a set of non-equality constraints of the form $x_i \neq x_j$ over a set of Boolean variables x_1, x_2, ..., x_n. We wish to determine if there is an assignment of Boolean values 0,1 to the variables that satisfies all the constraints, and compute such a satisfying assignment if there is one. Give an algorithm that solves this problem in time $O(n+m)$, where n is the number of variables and m the number of constraints. Justify the correctness and time complexity of your algorithm.

This can be reduced to the problem of testing whether a graph is bipartite.
Given a set C of non-equalities $x_i \neq x_j$ on a set of Boolean variables x_1 , x_2 , ..., x_n we can construct a graph G which has a node for each variable and has an edge (x_i , x_j) for each constraint $x_i \neq x_j$. We claim that the graph G is bipartite if and only if the set C of constraints has a satisfying assignment.

First, suppose that the graph G is bipartite, i.e., there is a partition of the set N of nodes Into two sets subsets N_1,N_2 ($N_1 \cap N_2 = \varnothing$,$N_1 \cup N_2 = N$) so that every edge connects a node of N_1 with a node of N_2 . Then the assignment that gives value 1 to all variables corresponding to nodes in N_1 and value 0 to all variables corresponding to nodes in N_2, satisfies all the constraints: Since every edge connects a node of N_1 with a node of N_2, this assignment satisfies all the constraints.

Conversely, if there is an assignment that satisfies all the constraints, then we can form a bipartition of the node set by letting N_1 be the set of all the nodes corresponding to variables that are assigned 1 and N_2 the set of nodes corresponding to variables that are assigned 0. Since the assignment satisfies all the constraints, in every constraint $x_i \neq x_j$ one variable is assigned 1 and the other 0, hence every edge connects a node of N_1 with a node of N_2.

The graph G has n nodes and m edges. We know that we can test in $O(n+m)$ time whether a graph is bipartite using DFS or BFS.

Grading Rubrics:

1. correctly define the graph and claim that problem can be reduced into checking whether G is bipartite. 5 pts
2. Justify the correctness of the algorithm. 7 pts
3. Justify the running time of the algorithm. 3 pts

3) 15 pts

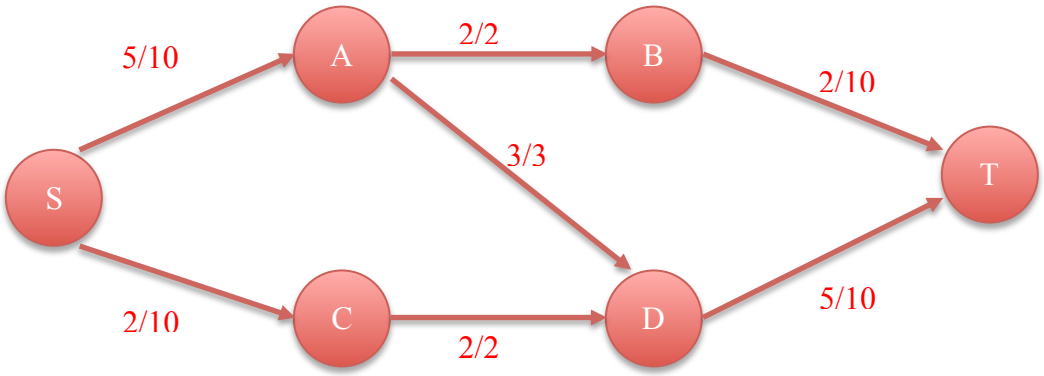
Consider the following problem: Given a flow network G with source s and sink t, and a unique $s - t$ min cut, find k edges that, when deleted, reduce the maximum $s - t$ flow in the graph by as much as possible.

Prove or disprove the following solution:

- Find the $s - t$ min cut in G.
- Sort edges on the min cut in decreasing order of weight
- Delete the first k (highest capacity) edges from the ordered list

This is False.

Counter example:



Grading:

- Saying statement is false without a correct counterexample (5 points)
- Proving the statement is True because all edges on the min-cut are saturated and hence, removing each edge from the cut will have a reduction in max flow equal to the capacity of that edge (5 points)

4) 15 pts

Given a list of non-negative numbers and a target integer *k*. Design a dynamic programming algorithm to check if the array has a continuous subarray of size at least 2 that sums up to a multiple of *k*, that is, sums up to *n*k* where n is also an integer.

Example:
Input: [23, 2, 4, 6, 7], *k*=6
Output: True
Explanation: Because [2, 4] is a continuous subarray of size 2 and sums up to 6.

Example:
Input: [23, 2, 6, 4, 7], *k*=6
Output: True
Explanation: Because [23, 2, 6, 4, 7] is a continuous subarray of size 5 and sums up to 42.

First, if k==0, we check whether these are two continuous zeros in the list. If so, return True. Otherwise, return False.

Then, for k!=0, we use the following algorithm.

Assume dp[i][j] representing the sum of sub-array from i to j in array. If there is a pair of (i,j) satisfies dp[i][j] % k == 0, the algorithm returns true. Otherwise, it returns false.

Initialize all elements in the dp array to zero.

We can calculate dp[i][j] in the follow way:
Assume n=length of list and we use nums to represent the input list
For i, we traverse from n – 1 to 0,
    Set dp[i][i] = nums[i]
    For j, we traverse from i+1 to n-1,
        dp[i][j] = dp[i+1][j] + nums[i]
        after setting the value dp[i][j], we check whether dp[i][j] % k == 0, if so, return True.

After checking all (i,j) pairs through above process, return False if there is no such (i,j) pair that meets the condition.

Grading Rubrics:

1. Correctly take care of the case when k==0. 3 pts
2. Correctly define the dp array and point out how dp array relates to the output of the algorithm. 3 pts
3. Correctly initialize the dp array. 3 pts
4. Correctly define how to fill in dp array. 6 pts

5) 10 pts
Analyze the run time complexity of the following function with respect to *n*.

```
function printalot (n: an integer power of 2)
        if n > 1 {
                printalot (n/2)
                printalot (n/2)
                printalot (n/2)
                for i=1 to n^4
                        printline ("Are we done yet?")
                endfor

                }
```

If T(n) is the number of lines printed for input n, then $T(n) = 3T(n/2) + n^4$ with a base case of $T(1) = 0$.

For an asymptotic solution, we can apply the master theorem with $n^{(\log_b a)} = n^{(\log_2 3)}$ and $f(n) = n^4$. Since f(n) polynomially dominates $n^{(\log_b a)}$, $T(n) = \Theta(f(n)) = \Theta(n^4)$ by the master theorem.

Grading:

- Used Big-O notation instead of Big-Theta notation (-1 point)
- Wrong recursion (-3 points)
- Wrong approach (-7 points)

6) 15 pts

Recall the following T/F question on exam 2:
Bellman-Ford algorithm is not guaranteed to work on all undirected graphs.
Which was a true statement.

Prove that if the Bellman-Ford algorithm were able to find a shortest simple path in an undirected graph with negative cost edges, then P must be equal to NP. In other words, prove that the problem of finding a shortest simple path in an undirected graph with negative cost edges is NP-complete. You can use ANY NP-complete problem that we have discussed in class for your reduction.

Step #1. Prove X (the given problem) is in NP
Certificate: ordered list of vertices on the shortest simple path
Certifier: Check if all vertices are visited. There has to be an edge between every pair of adjacent vertices in the order. Check if the given certificate is the shortest path.

Step #2. Choose a problem Y that is known to be NP-complete
Y can be HAM-PATH or HAM-Cycle.

Step #3. Prove that Y $\leq$p X
Reduction from HAM-PATH

- Given undirected graph $G = (V, E)$
- Transform to graph G' with -1 weight for every edge.
- Claim: G has a HAM-PATH if and only if SP w/ negative edge has a path of length $-|V| + 1$ between two arbitrary nodes. (Need to call the blackbox n times each time using a different starting point and finding the shortest path to all other nodes)

Reduction from HAM-Cycle

- Given undirected graph $G = (V, E)$
- Transform to graph G' with -1 weight for every edge. Remove an arbitrary edge (uv) in G', add two new nodes v' and u', and add edges uu' and vv' with weight -1.
- Claim: G has a HAM-Cycle if and only if SP w/ negative edge has a path of length $-|V| - 1$ between any two points v'u'. (Need to call the blackbox m times, each time removing a different edge)

Grading:

- Didn’t mention X is in NP (-5 points)
- Mentioned X is in NP, but didn’t provide certificate/certifier (-3 points)
- Didn’t mention a problem Y is known to be NP-complete (-3 points)
- Missing step #2 and #3 (-10 points)
- Missing/wrong proof for step #3 (-5 points)
- Wrong notation (-1 point)
- Wrong approach without mentioning three steps (-15 points)

7) 10 pts

720 Students have pre-enrolled for the "Analysis of Algorithms" class in Fall. Each student must attend one of the 16 discussion sections, and each discussion section i has capacity for D_i students. The happiness level of a student assigned to a discussion section i is proportionate to $\alpha_i (D_i - S_i)$, where α_i is a parameter reflecting how well the air-conditioning system works for the room used for section i (the higher the better), and S_i is the actual number of students assigned to that section. We want to find out how many students to assign to each section in order to maximize total student happiness.

Express the problem as a linear programming problem.

Our variables will be the S_i 's.

Objective function:

maximize $\sum_{i=1}^{16} \alpha_i (D_i - S_i)$

subject to: $D_i - S_i \geq 0$ for $0 < i \leq 16$

$S_i \geq 0$ for $0 < i \leq 16$

$\sum_{i=1}^{16} S_i = 720$

Grading:

- Objective function (4 points)
- Each constraint (2 points)

CS570
Analysis of Algorithms
Spring 2017
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
Total		

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

[TRUE /]
Given a minimum cut, we could find the maximum flow value in O(E) time.

[/ FALSE]
Any NP-hard problem can be solved in time O(2^poly(n)), where n is the input size and poly(n) is a polynomial.

[TRUE /]
Any NP problem can be solved in time O(2^poly(n)), where n is the input size and poly(n) is a polynomial.

[TRUE /]
If 3-SAT $\leq_p$ 2-SAT, then P = NP.

[/ FALSE]
Assuming P ≠ NP, there can exist a polynomial-time approximation algorithm for the general Traveling Salesman Problem.

[/ FALSE]
Let (S,V−S) be a minimum (s,t)-cut in the network flow graph G. Let (u,v) be an edge that crosses the cut in the forward direction, i.e., u ∈ S and v ∈ V−S. Then increasing the capacity of the edge (u, v) necessarily increases the maximum flow of G.

[/ FALSE]
If problem X can be solved using dynamic programming, then X belongs to P.

[/ FALSE]
All instances of linear programming have exactly one optimal solution.

[/ FALSE]
Let Y $\leq_p$ X and there exists a 2-approximation for X, then there must exist a 2-approximation for Y.

[TRUE /]
There is no known polynomial-time algorithm to solve an integer linear programming.

2) 16 pts

A set of n space stations need your help in building a radar system to track spaceships traveling between them. The i^{th} space station is located in 3D space at coordinates (x_i, y_i, z_i). The space stations never move. Each space station i will have a radar with power r_i, where r_i is to be determined. You want to figure how powerful to make each space station's radar transmitter, so that whenever any spaceship travels in a straight line from one space station to another, it will always be in radar range of either the first space station (its origin) or the second space station (its destination). A radar with power r is capable of tracking space ships anywhere in the sphere with radius r centered at itself. Thus, a space ship is within radar range through its trip from space station i to space station j if every point along the line from (x_i, y_i, z_i) to (x_j, y_j, z_j) falls within either the sphere of radius r_i centered at (x_i, y_i, z_i) or the sphere of radius r_j centered at (x_j, y_j, z_j). The cost of each radar transmitter is proportional to its power, and you want to minimize the total cost of all of the radar transmitters. You are given all of the (x_1, y_1, z_1), ... ,(x_n, y_n, z_n) values, and your job is to choose values for r_1,...,r_n. Express this problem as a linear program.

(a) Describe your variables for the linear program. (3 pts)

Solution: r_i = the power of the i^{th} radar transmitter., i=1,2,….,n (3 pts)

(b) Write out the objective function. (5 pts)

Solution: Minimize $r_1 + r_2 + \cdots + r_n$.
Defining the objective function without mentioning r_i : -3 pts

(c) Describe the set of constraints for LP. You need to specify the number of constraints needed and describe what each constraint represents. (8 pts)

Solution: $r_i + r_j \geq \text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2)$. Or, $r_i + r_j \geq d_{i,j}$ for each pair of stations i and j, where $d_{i,j}$ is the distance from station i to station j (6 pts)

we need Sigma_(i=1 to n-1) i = $(n^2-n)/2$ constrains of inequality (2 pts)

3) 16 pts
Given a row of n houses that can each be painted red, green, or blue with a cost P(i, c) for painting house i with color c. Devise a dynamic programming algorithm to find a minimum cost coloring of the entire row of houses such that no two adjacent houses are the same color.

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

Solution: Let C(i, c) be the minimum possible cost of a valid coloring of the first i houses in which the last house gets color c.

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

Solution: The recurrence is $C(i, c) = \min_{c` \neq c} \{C(i - 1, c`) + P(i, c)\}$.
The base case: C(1, c) = P(1, c), for all colors c.

c) Describe (using pseudocode) how the value of an optimal solution is obtained using iteration. (4 pts)

```
Set r[1] = P(1,r),g[1] = P(1,g), b[1] = P(1,b)
for i=2 to n
  r[i] = P(i,r) + min(g[i-1],b[i-1])
  g[i] = P(i,g) + min(r[i-1],b[i-1])
  b[i] = P(i,b) + min(r[i-1],g[i-1])
endfor
return min{r[n], g[n], b[n]}
```

d) Compute the runtime of the algorithm. (2 pts)

Solution: O(n)
The runtime is O(n) since there are n subproblems each of which takes O(1) time.

Grading rubric:

A. Part (a):
   a. Missing color in subproblem definition -2
   b. Unclear description -2 ~ -3
B. Part (b):
   a. Missing color in recurrence -4

b. Totally wrong recurrence -6
c. Fail to consider no same color in adjacent choice -2
d. Wrong symbols or typos in the recurrence -1 ~ -2

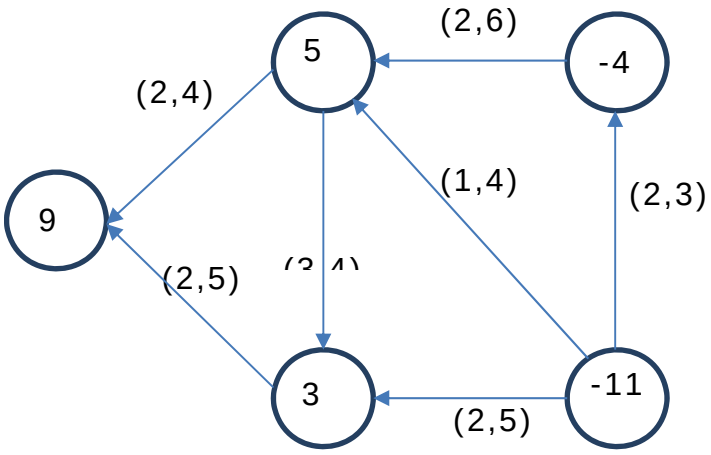
C. Part (c):
a. If wrong answer in part (b) -2
b. If no pseudo code -2
c. If the answer is not according to wrong answer in part (b) -4
d. If wrong order of for loop, color before house id -2

D. Part (d):
a. The question is graded according to recurrence and pseudo on part (b) and (c)
b. If there is additional constant in big O notation -1

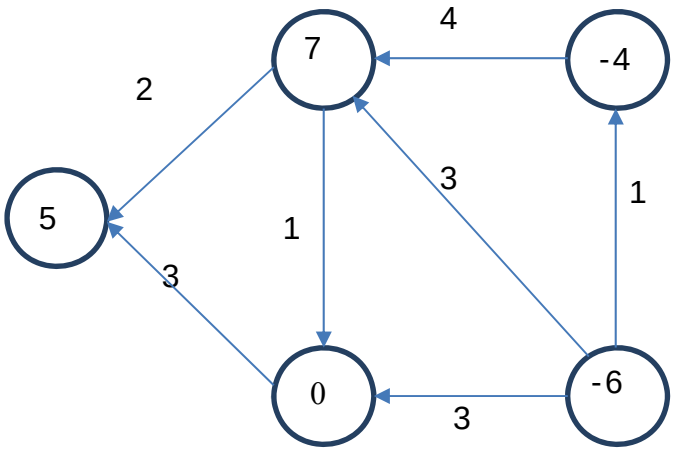
Common errors:

1. Failed to include the color of house in the definition of subproblem.
   a. Score if there are no other mistakes: part (a) -2, part (b) -4, part (c) -2 = 8
2. Each house must be painted into one of the three colors.
   a. Score if there are no other mistakes: part (a) -4, part (b) -4, part (c) -2 = 6
3. Use interval (2d state + 2 colors for the ends)
   a. Score if there are no other mistakes: part (a) -1, part (b) -1, part (c) -1 = 13

4) 16 pts
In the network below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.



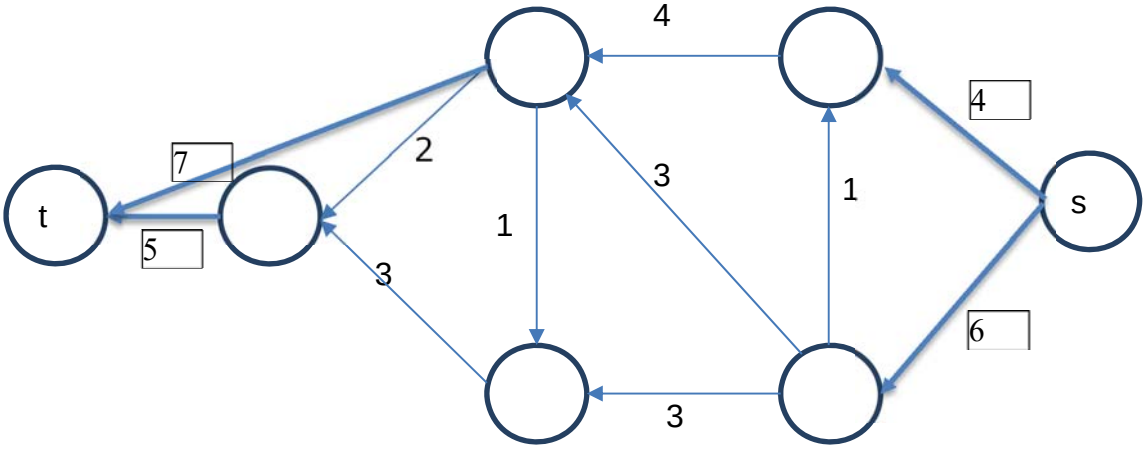
a) Turn the circulation with lower bounds problem into a circulation problem without lower bounds (6 pts)



Grading rubric

1) a) There are total 12 values in the network-flow graph that needs to be written, 5 values (demands) on vertices and 7 values on edges (modified capacities). For each correct value 0.5 marks will be awarded.
b) 2 marks will be deducted for adding a source and sink vertices and edges.

b) Turn the circulation with demands problem into the maximum flow problem (6 pts)



Grading rubric:

a) 2 marks for adding two super nodes, 1 for each node.
b) 2 marks for connecting negative demand vertices to supply nodes and positive demands to sink nodes, 0 if they are connected in opposite way.
c) 2 marks for placing the capacities on the newly added edges between super nodes and the vertices, will lose 0.5 marks for each edge capacity mistake (maximum deduction here is 2 marks).
d) will loose 0.5 marks for any incorrect edge capacity value on the edges in the original network flow.

c) Does a feasible circulation exist? Explain your answer. (4 pts)

No, a feasible circulation doesn't exist. In the given original network, total demand values on the vertices doesn't match with the total supply value on all the vertices.

Grading rubric:

a) 2 marks for answering no, 2 marks for explanation.
If you have answered yes and obtained a max-flow value of 10 and show that edges from source are saturated then 1.5 marks are awarded.

5) 16 pts

We want to become celebrity chefs by creating a new dish. There are n ingredients and we'd like to use as many of them as possible. However, some ingredients don't go so well with others: there is $n \times n$ matrix D giving *discord* between any two ingredients, i.e., $D[i,j]$ is a real value between 0 and 1: 0 means i and j go perfectly well together and there is no discord and 1 means they go very badly together. Any dish prepared with these ingredients incurs a *penalty* which is the sum of the discords between all pairs of ingredients in the dish. We would like the total penalty to be small. Consider the decision problem EXPERIMENTAL CUISINE: can we prepare a dish with at least k ingredients and with the total penalty at most p?

a) Show that Experimental Cuisine is in NP. (4 pts)

Certificate: a dish with at least k ingredients with total penalty of at most p.
Verifier: Go through ingredients and calculate the penalty and the total number of ingredients which takes at most quadratic time.

Grading rubric

-2 points for not mentioning about how to compute the penalty(e.g. add, search, compare, etc.)

b) Show that EXPERIMENTAL CUISINE is NP-complete by giving a reduction from INDEPENDENT SET. (12 pts)

Given an instance of IS, (G,k), where G is a graph, and k is a parameter. Construct the following: Set D to be an incident matrix of G, i.e., D[i,j]=1, iff vertices i and j are connected by an edge, otherwise D[i,j]=0.
Set p=0. (D,k,p) is an instance of EXPERIMENTAL CUISINE.

Claime: (D,p,k) returnes YES for EXPERIMENTAL CUISINE iff (G,k) returns YES for INDEPENDENT SET problem. Proof: Take instances, observe that a collection of vertices form an IS iff corresponding p = 0. k parameter stands for size of solution in both cases.

Grading rubric

Construction(4 points)

Since the problem is stated to reduce FROM IS, it should start with Independent Set to construct a discord matrix for EXPERIMENTAL CUISINE.

Claim (4 points)

Correct claim should be made for case of reduction: (D,k,p) returns YES for EXPERIMENTAL CUISINE iff (G,k) returns YES for INDEPENDENT SET problem.

Proof of reduction case (4 points)

The proof should be made in both directions when p=0.

6) 16 pts

There are *n* positive integers. Your goal is to concatenate them to form a single integer which is as large as possible. For example, for the following three integers 111, 222, and 3, the largest integer is 3222111.

a) Develop a greedy algorithm to solve the problem for arbitrary *n* (10 pts)

1. Sort the n integers using the following rule: If a is larger than b, then put a in front of b. (i) The comparison for number a and b is conducted by scanning through each digit. Need to consider several different cases: a) if the current digit is the same b) if the length of one number is shorter than another. (ii) or the comparison can be easily conduct by concatenating two numbers i..e concat(a,b) > concat(b,a)
2. Concatenate all the sorted integers together.

b) Provide proof of correctness. (6 pts)

Exchange argument, induction should be the correct way to do it.

Grading Rubric and common mistakes:

For problem (a),

- Your solution is incorrect, but you explain your own solution in a meaningful way, you will get ~4 points.
- the idea is generally correct, but if doesn't consider complete cases (including how to handle different length of numbers? i.e. 67 vs 676, what about the current comparing digit is the same? and etc), will deduct 2-4 points.
- some students write the solution like this: “comparing integers.. “ this is not correct since the comparison should be conducted digit by digit, thus not a right solution
- Some students answered - “appending zero”, this is not correct. i.e 67 vs 676, after appending zero, 676 > 67, so we get 67667 but the larger one should be 67676

For problem (b),

Common mistakes:

- the idea is in general right, but the proof itself is not a rigorous, get ~2 points deducted
- some students give proof by example, this is not correct. Showing by example is similar to writing a unit-test for programming, which is not a proof at all. deduct ~ 4 points
- some students follow the induction/exchange argument procedure, but does not show any valid evidence among the proof. i.e. use the proof framework and directly get the conclusion without showing any valid evidence. Deduct ~ 2-4 points

Additional Space

CS570
Analysis of Algorithms
Fall 2016
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	18	
Problem 3	18	
Problem 4	18	
Problem 5	18	
Problem 6	8	
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** or **FALSE**. No need to provide any justification.

[TRUE] This is the definition of NP.
Every problem in NP has a polynomial time certifier.

[FALSE] Decision problems can be in P.
Every decision problem is in NP-complete.

[TRUE] If 3-SAT reduces to it, it's NP-Hard and we now it's NP so it's NPC.
An NP problem is NP-complete if 3-SAT reduces to it in polynomial time.

[FALSE] It will be O(m^2 + mn) and not linear.
If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[FALSE] consider a graph with edges (a,b) = 4, (b,c) = 3 and (a, c) = 5. The shortest path from a to c is 5 but (a, c) is not in the MST.
Let T be a minimum spanning tree of G. Then, for any pair of vertices s and t, the shortest s-t path in G is the path in T.

[TRUE] Master's Theorem
If the running time of a divide-and-conquer algorithm satisfies the recurrence $T(n) = 3\,T(n/2) + \Theta(n^2)$, then $T(n) = \Theta(n^2)$.

[FALSE] Consider the problem in the previous item.
In a divide and conquer solution, the sub-problems are disjoint and are of the same size.

[FALSE] Many of them are NP-Complete.
All Integer linear programming problems can be solved in polynomial time.

[TRUE]
If the linear program is feasible and bounded, then there exists an optimal solution.

[FALSE] Each specific operation can be larger than O(log n).
Suppose we have a data structure where the amortized running time of Insert and Delete is O(lg n). Then in any sequence of 2n calls to Insert and Delete, the worst-case running time for the nth call is O(lg n).

2) 18 pts

We are given an infinite array where the first n elements contain integers in sorted order and the rest of the elements are filled with ∞. We are not given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in O(lg n) time.

Solution:

- First determine the smallest power of 2 larger than n (8 points)

```
N=1
while A[i] ≠ ∞
    N = N*2
endwhile
```

N will be the smallest power of 2 larger than n

- Use binary search on A[1..N] to find x (8 points)

- Complexity: both components run in O(lg n) time

Common mistakes:

1. Without finding N, directly do binary search on A[1..n]
Note that n is not given. Cases like this get 4 points.

2. Without finding N, directly do binary search on A[1..2x+1]
x could be a negative integer; in this case, 2x+1 is also negative.
Cases like this get 8 points.

3. Time complexity is larger than O(log n) time. (e.g., linearly scan the array, or build a min heap first)
Cases like this get 0 points

3) 18 pts

For bit strings A = $a_1,..., a_m$, B = $b_1,..., b_n$ and C = $c_1,..., c_{m+n}$, we say that C is an interleaving of A and B if it can be obtained by interleaving the bits in A and B in a way that maintains the left-to-right order of the bits in A and B . For example if A = 101 and B = 01 then $a_1a_2b_1a_3b_2$ = 10011 is an interleaving of A and B, whereas 11010 is not. Give the most efficient algorithm you can to determine if C is an interleaving of A and B. Analyze the time complexity of your solution as a function of *m* and *n*.

Solution: The general form of the subproblem we solve will be: determine if $c_1,..., c_{i+j}$ is an interleaving of $a_1,..., a_i$ and $b_1,...,b_j$ for $0 \le i \le |A|$ and $0 \le j \le |B|$. Let c[i, j] be true if and only if $c_1,...,c_{i+j}$ is an interleaving of $a_1,..., a_i$ and $b_1,..., b_j$. We use the convention that if i = 0 then $a_i = \Phi$ (the empty string) and if j = 0 then $b_j = \Phi$. The subproblem c[i, j] can be recursively defined as shown (where c[|A|, |B|] gives the answer to the optimal problem):

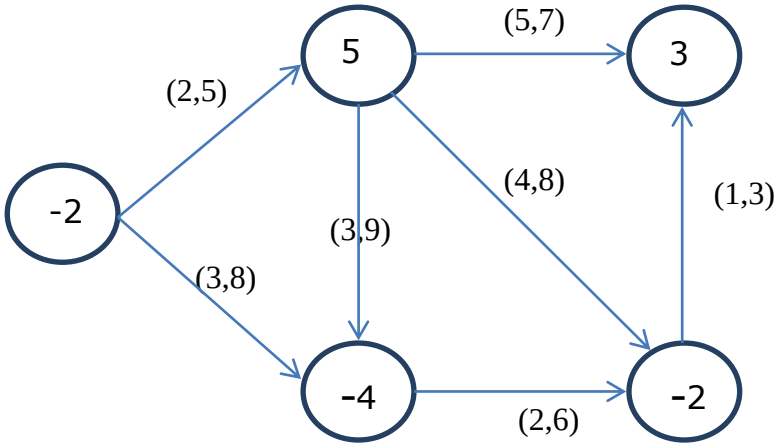
$$c[i,j] = \begin{cases} true, & if\ i = j = 0 \\ false, & if\ a_i \neq c_{i+j}\ and\ b_j \neq c_{i+j} \\ c[i-1,j], & if\ a_i = c_{i+j}\ and\ b_j \neq c_{i+j} \\ c[i,j-1], & if\ a_i \neq c_{i+j}\ and\ b_j = c_{i+j} \\ c[i-1,j]c[i,j-1], & if\ a_i = b_j = c_{i+j} \end{cases}$$

The time complexity is clearly O(|A||B|) since there are |A|x|B| subproblems each of which is solved in constant time. Finally, the c[i, j] matrix can be computed in row major order.

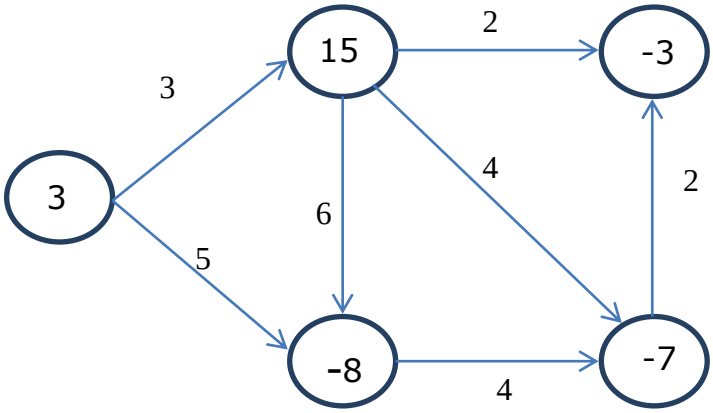
Proof of recurrence not asked for in the problem statement: *We now argue this recursive definition is correct. First the case where i = j = 0 is when both A and B are empty and then by definition C (which is also empty) is a valid interleaving of A and B. If $a_i \neq c_{i+j}$ and $b_j = c_{i+j}$ then there could only be a valid interleaving in which a_i appears last in the interleaving, and hence c[i, j] is true exactly when $c_1,..., c_{i+j-1}$ is a valid interleaving of $a_1,..., a_{i-1}$ and $b_1,..., b_j$ which is given by c[i-1, j]. Similarly, when $a_i \neq c_{i+j}$ and $b_j = c_{i+j}$ then c[i, j] = c[i-1, j].*
Finally, consider when $a_i = b_j = c_{i+j}$. In this case the interleaving (if it exists) must either end with b_i (in which case c[i-1, j] is true) or must end with b_i (in which case c[i, j-1] is true). Thus returning c[i-1, j] ∨ c[i, j-1] gives the correct answer.
Finally, since in all cases the value of c[i, j] comes directly from the answer to one of the subproblems, we have the optimal substructure property.

- *Recurrence and definition of OPT (12 pts)*
- *Algorithm (5 pts)*
- *Complexity (3 points)*

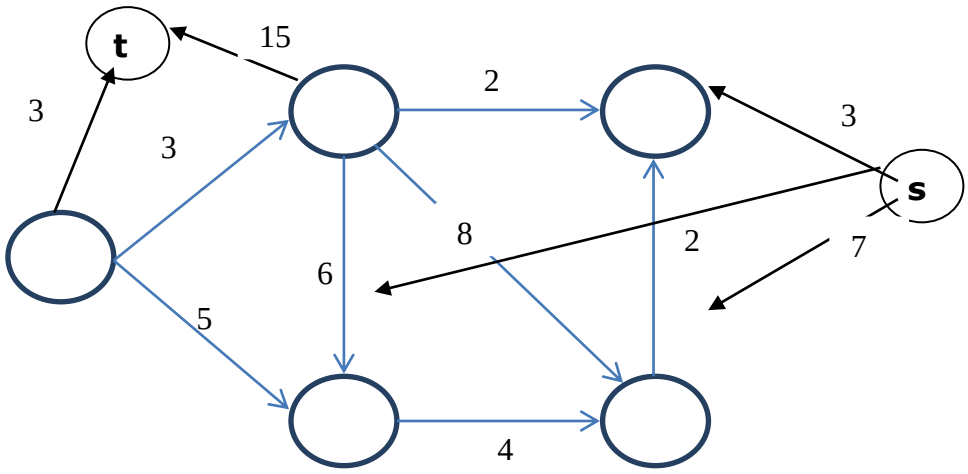
4) 18 pts
In the network below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.



Solution.
Remove lower bounds by changing demands on each vertex



Reduce it to max-flow by adding s and t vertices.



Check if there is a s-t flow |f|= 18.
There is no such flow. It follows, no circulation.

6 points - for eliminating lower bounds and converting to problem with only demands
6 points - for eliminating demands and converting to max-flow problem
6 points - Checking if there exists a max-flow of value D

5) 18 pts

The Metric Traveling Salesman Problem (metric-TSP) is a restriction of the Traveling Salesman Problem, defined as follows. Given n cities $c_1, \ldots, c_n$ with inter-city distances $d(c_i, c_j)$ that form a metric:

$d(c_i,c_i) = 0$

$d(c_i, c_j) > 0$ for $i \neq j$

$d(c_i, c_j) + d(c_j, c_k) \geq d(c_i, c_k)$ for i, j, k in $\{1, \ldots, n\}$

Is there a closed tour that visits each city exactly once and covers a total distance at most C? Prove that metric-TSP is in NP-complete.

Solution.

1. It's easy to see that is in NP
2. Reduce form HC.

Given a graph G=(V,E) on which we want to solve the HAM-CYCLE problem. We will construct a complete G' with metric d as follows

d(u,v) = 0, if u = v
d(u,v) = 1, if (u,v) is in E
d(u,v) = 2, ow

Since triangle inequalities hold in G' (edges of any triangle will have sizes of 1 or 2), then $d(c_i, c_j) + d(c_j, c_k) \geq d(c_i, c_k)$ for i, j, k in $\{1, \ldots, n\}$

Let the bound C = |V|.
Claim: metric-TSP on G' is solvable if and only if the HC problem on G is solvable.
->) If there is a metric-TST of length |V|, then every distance between successive cities must be 1. Therefore these define as HC on G.
<-) If G has a hamiltonian cycle, then it defines a metric-TSP of size |V|.

6) 8 pts
Convert the following linear program

maximize ($3 x_1 + 8 x_2$)

Subject to
$x_1 + 4 x_2 \leq 20$
$x_1 + x_2 \geq 7$
$x_1 \geq -1$
$x_2 \leq 5$

to the standard form. You need to show all your steps.

x1 + 1 = z1 >= 0
x2 – 5 <= 0 so 5 – x2 = z2 >= 0

x1 + 4.x2 <= 0 → (z1 – 1) + 4(5 – z2) <= 20 → z1 – 4.z2 <= 1
x1 + x2 >= 7 → (z1 – 1) + (5 – z2) >= 7 → z2 – z1 <= -3

finally we get:

maximize 3.z1 – 8.z2 + 37
subject to:
z1 – 4.z2 <= 1
z2 – z2 <= -3
z1 >=0
z2 >= 0

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 $s = s_1 s_2 \ldots s_n$, where s_k is the kth letter of the string.
We define VALID_SEQU(i) to be true if $s_1 s_2 \ldots s_i$ 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 $i = 1 \ldots n$

```
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}...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, $p_1, p_2, \ldots, p_n$ 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 $p_1, p_2, \ldots, p_n$ corresponding to the n people, k vertices $s_1, s_2, \ldots, s_k$ corresponding to the k sets, and two special vertices s and t.

For $i = 1, \ldots, n$, 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 $j = 1, \ldots, k$, 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 $p_1, p_2, \ldots, p_n$; 2pt for $s_1, s_2, \ldots, s_k$; 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.

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.

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

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

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

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

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

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

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

[TRUE/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.

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

[TRUE/FALSE]
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.

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.

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.

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.

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

(ii) Q is NP-hard

(iii) R is NP-complete

(iv) R is 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.

CS570
Analysis of Algorithms
Summer 2015
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
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** or **FALSE**. No need to provide any justification.

[TRUE/]
If all edge capacities in a flow network are integer multiples of 7, then the maximum value of flow is a multiple of 7.

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

[/FALSE]
Let T be a complete binary tree with n nodes. Finding a path from the root of T to a given vertex $v \in T$ using breadth-first search takes O(log n) time.

[TRUE/]
Halting Problem is an NP-Hard problem.

[TRUE/]
Every decision problem in P has a polynomial time certifier.

[/FALSE]
In a flow network, if we increase the capacity of an edge that happens to be on a minimum cut, this will increase the max flow in the network.

[/FALSE]
If the capacity of every arc is odd, then there is a maximum flow in which the flow on each arc is odd.

[TRUE/]
If the edge weights of a weighted graph are doubled, then the number of minimum spanning trees of the graph remains unchanged

[/FALSE]
The linear programming solution to the shortest path problem discussed in class can fail in the presence of negative cost edges.

[/FALSE]
In a divide and conquer solution, the sub-problems are disjoint and are of the same size.

2) 16 pts

You are given n jobs of known duration $t_1, t_2, \ldots, t_n$ for execution on a single processor. All the jobs are given to you at the start and they can be executed in any order, one job at a time. We want to find a schedule that minimizes **the total time spent by all the jobs** in this system. The **time spent** by one job is the **sum of the time spent on waiting plus the time spent on its execution**. In other words, the total time spent by all jobs is the total sum of their finish times. Give an efficient solution for this problem. Analyze the complexity of your solution and prove that it is optimal.

Solution: We can use a greedy approach to solve this problem. Sort the jobs in increasing order of duration and schedule the jobs based on the sorted order, first job in the sorted order is scheduled first. Complexity of the approach is O(nlogn) algorithm. To prove that the algorithm is optimal, we can use proof by exchange technique.

Assume that there is optimal way to schedule jobs other than our greedy approach. Assign jobs a rank which is the index in the increasingly sorted array. If the array is not increasingly sorted then we can find adjacent jobs, j and j+1 such that rank of j is greater than rank of j+1. Let S be the starting time of job j, then $S + t_j$ is the finishing time of job j and $S + t_j + t_{\{j+1\}}$ is the finishing time of job j+1. When the jobs j and j+1 are only swapped, we can observe that the finishing times of other jobs are not affected. Also the total time spent by the jobs after swapping is not worse than the before as $S + t_{j+1} + S + t_{\{j+1\}} + t_j \leq S + t_j + S + t_j + t_{\{j+1\}}$, since $t_j \geq t_{\{j+1\}}$.

We can keep on swapping such jobs until we reach our greedy solution. So our greedy solution is no worse than the optimal solution, so our greedy solution is also optimal.

3) 16 pts

For bit strings $X = x_1 \ldots x_m$, $Y = y_1 \ldots y_n$ and $Z = z_1 \ldots z_{m+n}$, we say that Z is an interleaving of X and Y if it can be obtained by interleaving the bits in X and Y in a way that maintains the left-to-right order of the bits in X and Y. For example if X = 101 and Y = 01 then $x_1x_2y_1x_3y_2 = 10011$ is an interleaving of X and Y, whereas 11010 is not. Give the most efficient algorithm you can to determine if Z is an interleaving of X and Y. Prove your algorithm is correct and analyze its time complexity as a function m = |X| and n = |Y|.

Solution

The general form of the subproblem we solve will be: determine if $Z = z_1 \ldots z_{i+j}$ is an interleaving of $x_1 \ldots x_i$ and $y_1 \ldots y_j$ for $0 \le i \le m$ and $0 \le j \le n$. Let c[i; j] be true if and only if $z_1 \ldots z_{i+j}$ is an interleaving of $x_1 \ldots x_i$ and $y_1 \ldots y_j$. We use the convention that if i = 0 then $x_i = \lambda$ (the empty string) and if j = 0 then $y_j = \lambda$. The subproblem c[i, j] can be recursively defined as shown (where c[m, n] gives the answer to the original problem):

$$c[i; j] = \begin{cases} \text{true} & \text{if } i = j = 0 \\ \text{false} & \text{if } x_i \neq z_{i+j} \text{ and } y_j \neq z_{i+j} \\ c[i - 1, j] & \text{if } x_i = z_{i+j} \text{ and } yj \neq z_{i+j} \\ c[i, j - 1] & \text{if } x_i \neq z_{i+j} \text{ and } y_j = z_{i+j} \\ c[i - 1, j] \text{ OR } c[i, j - 1] & \text{if } xi = y_j = z_{i+j} \end{cases}$$

We now argue this recursive definition is correct. First the case where i = j = 0 is when both X and Y are empty and then by definition Z (which is also empty) is a valid interleaving of X and Y . If $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then there could only be a valid interleaving in which xi appears last in the interleaving, and hence c[i; j] is true exactly when $z_1 \ldots z_{i+j-1}$ is a valid interleaving of $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_j$ which is given by c[i – 1, j]. Similarly, when $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then c[i, j] = c[i - 1; j]. Finally, consider when $x_i = y_j = z_{i+j}$. In this case the interleaving (if it exists) must either end with xi (in which case c[i - 1, j] is true) or must end with y_i (in which case c[i, j - 1] is true). Thus returning c[i – 1, j] OR c[i, j - 1] gives the correct answer. Finally, since in all cases the value of c[i, j] comes directly from the answer to one of the subproblems, we have the optimal substructure property.

The time complexity is clearly O(nm) since there are n.m subproblems each of which is solved in constant time. Finally, the c[i, j] matrix can be computed in row major order.

4) 16 pts

You are given two decision problems L1 and L2 in NP. You are given that L1 $\leq_p$ L2. For each of the following statements, state whether it is true, false or an open question. Justify your answers.

(a) If L1 $\in$ P, then L2 $\in$ P.

(b) If L1 $\in$ NP-complete, then L2 $\in$ NP-complete.

(c) If L2 $\in$ P, then L1 $\in$ P.

(d) Suppose L2 is solvable in O(n). Then, L1 is also solvable in O(n).

Solution:

(a) **Open question**. If P = NP, then L2 $\in$ P. Otherwise, L2 can be P, NP or NP-complete.

(b) **True**. L2 is NP-complete, since we can convert any problem in NP to problem regarding L1 in polynomial time. Therefore, if we can reduce problems of L1 to problems in L2 in polynomial time, we can reduce all problems in NP to problems in L2. This is the technique we use to prove NP-completeness – by reducing an unknown problem to a known NP-complete problem.

(c) **True**. We can solve problems in L1 by first reducing it to problems in L2 in polynomial time, and then use the polynomial time algorithm for L2 solve the reduce problem. This gives a polynomial time algorithm for L1.

(d) **False**. The reduction L1 $\leq_p$ L2 may take more than linear time, therefore, we are not guaranteed a linear time algorithm for L1. We are guaranteed that L1 $\in$ P can be solved in polynomial time – time to reduce L1 to L2 plus the time to solve problem in L2.

5) 16 pts

USC Admissions Center needs your help in planning paths for Campus tours given to prospective students or interested groups. Let USC campus be modeled as a weighted, directed graph G containing locations V connected by one-way roads E. On a busy day, let *k* be the number of campus tours that have to be done at the same time. It is required that the paths of campus tours do not use the same roads. Let the tour have k starting locations A = {a1, a2, . . . , ak} ⊂ V. From the starting locations the groups are taken by a guide on a path through G to some ending location in B = {b1, b2, . . . , bk} ⊂ V. Your goal is to find a path for each group *i* from the starting location, ai , to any ending location bj such that no two paths share any edges, and no two groups end in the same location bj .

(a) Formulate this as a flow problem. Find an algorithm (if any) to find *k* paths ai -> bj that start and end at different vertices and that share no edges, and briefly justify the correctness of your algorithm. (8 pts)

(b) It is still possible that some paths share a vertex. Modify your algorithm to find *k* paths ai -> bj, that start and end in different locations and that share neither vertices nor edges. (8 pts)

Solution:

(a) The complete algorithm is as follows:

1. Create a flow network G' containing all vertices in V, all directed edges in E with capacity 1, and additionally a source vertex s and a sink vertex t. Connect the source to each starting location with a directed edge (s, ai) and each ending location to the sink with a directed edge (bi, t), all with capacity 1.

2. Run Ford-Fulkerson on this network to get a maximum flow f on this network. If |f| = k, then there is a solution; if |f| < k, then there is no solution, so we return FALSE. We will later show that it is always the case that |f| <= k.

3. To extract the paths from ai to bj (as well as which starting location ultimately connects to which ending location), run a depth-first search on the returned max flow f starting from s, tracing a path to t. Remove these edges and repeat k times until we have the k disjoint paths.

To **justify** correctness, any argument conveying that there is a flow of size k if and only if there are k disjoint paths is correct.

(b) Duplicate each vertex *v* into two vertices *vin* and *vout*, with a directed edge between them. All edges (*u*, *v*) now become (*u*, *vin*); all edges (*v*,*w*) now become (*vout*,*w*). Assign the edge (*vin*, *vout*) capacity 1. With this transformation, we now have a graph in which there is a single edge corresponding to each vertex, and thus any paths that formerly shared vertices would be forced to share this edge. Now, we can use the same algorithm as in part (a) on the modified graph to find k disjoint paths sharing neither edges nor vertices, if they exist.

6) 16 pts

A company makes three products and has 4 available manufacturing plants. The production time (in minutes) per unit produced varies from plant to plant as shown below:

		Manufacturing Plant			
		1	2	3	4
Product	1	5	7	4	10
	2	6	12	8	15
	3	13	14	9	17

Similarly the profit ($) contribution per unit varies from plant to plant as below:

		Manufacturing Plant			
		1	2	3	4
Product	1	10	8	6	9
	2	18	20	15	17
	3	15	16	13	17

If, one week, there are 35 working hours available at each manufacturing plant how much of each product should be produced given that we need at least 100 units of product 1, 150 units of product 2 and 100 units of product 3. Formulate this problem as a linear program. You do not have to solve the resulting LP.

Solution

Variables:

At first sight we are trying to decide how much of each product to make. However on closer inspection it is clear that we need to decide how much of each product to make at each plant. Hence let

xij = amount of product i (i=1,2,3) made at plant j (j=1,2,3,4) per week.

Although (strictly) all the xij variables should be integer they are likely to be quite large and so we let them take fractional values and ignore any fractional parts in the numerical solution. Note too that the question explicitly asks us to formulate the problem as an LP rather than as an IP.

Constraints:

We first formulate each constraint in words and then in a mathematical way.

- Limit on the number of minutes available each week for each workstation

$5x11 + 6x21 + 13x31 <= 35(60)$

$7x12 + 12x22 + 14x32 <= 35(60)$

$4x13 + 8x23 + 9x33 <= 35(60)$

$10x14 + 15x24 + 17x34 <= 35(60)$

- Lower limit on the total amount of each product produced

$x11 + x12 + x13 + x14 >= 100$

$x21 + x22 + x23 + x24 >= 150$

$x31 + x32 + x33 + x34 >= 100$

All variables are greater than equal to zero.

Objective: Maximize total profit.

Maximize

$10x11 + 8x12 + 6x13 + 9x14 + 18x21 + 20x22 + 15x23 + 17x24 + 15x31 + 16x32 + 13x33 + 17x34$

Additional Space

CS570
Analysis of Algorithms
Spring 2015
Exam III

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
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** or **FALSE**. No need to provide any justification.

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

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

[TRUE/FALSE]
If P equals NP, then NP equals NP-complete.

[TRUE/FALSE]
Let X be a decision problem. If we prove that X is in the class NP and give a poly-time reduction from X to Hamiltonian Cycle, we can conclude that X is NP-complete.

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

[TRUE/FALSE]
On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra.

[TRUE/FALSE]
Linear programming is at least as hard as the Max Flow problem in a flow network.

[TRUE/FALSE]
If you are given a maximum s-t flow in a graph then you can find a minimum s-t cut in time O(m) where m is the number of the edges in the graph.

[TRUE/FALSE]
Fibonacci heaps can be used to make Dijkstra's algorithm run in O(|E| + |V| log|V|) time on a graph G=(V,E)

[TRUE/FALSE]
A graph with non-unique edge weights will have at least two minimum spanning trees

2) 16 pts
Given a graph G=(V, E) with an even number of vertices as the input, the HALF-IS problem is to decide if G has an independent set of size |V|/2. Prove that HALF-IS is in NP-Complete.

3) 16 pts

A variant on the decision version of the subset sum problem is as follows: Given a set of n integer numbers $A = \{a_1, a_2, \ldots, a_n\}$ and a target number *t*. Determine if there is a subset of numbers in A whose product is precisely *t*. That is, the output is *yes* or *no*. Describe an algorithm (and provide pseudo-code) to solve this problem, and analyze its complexity.

4) 16 pts

We've been put in charge of a phone hotline. We need to make sure that it's staffed by at least one volunteer at all times. Suppose we need to design a schedule that makes sure the hotline is staffed in the time interval $[0, h]$. Each volunteer i gives us an interval $[s_i, f_i]$ during which he or she is willing to work. We'd like to design an algorithm which determines the minimum number of volunteers needed to keep the hotline running. Design an efficient greedy algorithm for this problem that runs in time $O(n \log n)$ if there are n student volunteers. Prove that your algorithm is correct.

You may assume that any time instance has at least one student who is willing to work for that time.

5) 16 pts

A software house has to handle 3 projects, P_1, P_2, P_3, over the next 4 months. P_1 can only begin after month 1, and must be completed within month 3. P_2 and P_3 can begin at month 1, and must be completed, respectively, within month 4 and 2. The projects require, respectively, 8, 10, and 12 man-months. For each month, 8 engineers are available. Due to the internal structure of the company, at most 6 engineers can be working, at the same time, on the same project. Determine whether it is possible to complete the projects within the time constraints. Describe how to reduce this problem to the problem of finding a maximum flow in a flow network and justify your reduction.

6) 16 pts

There are n people and n jobs. You are given a cost matrix, C, where C[i][j] represents the cost of assigning person i to do job j. You want to assign all the jobs to people and also only one job to a person. You also need to minimize the total cost of your assignment. Can this problem be formulated as a linear program? If yes, give the linear programming formulation. If no, describe why it cannot be formulated as an LP and show how it can be reduced to an integer program.

Additional Space

CS570
Analysis of Algorithms
Fall 2014
Exam III

Name: _____

Student ID: _____

Email:_____

Wednesday Evening Section

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
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** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
All the NP-hard problems are in NP.

[TRUE/FALSE]
Given a weighted graph and two nodes, it is possible to list all shortest paths between these two nodes in polynomial time.

[TRUE/FALSE]
In the memory efficient implementation of Bellman-Ford, the number of iterations it takes to converge can vary depending on the order of nodes updated within an iteration

[TRUE/FALSE]
There is a feasible circulation with demands $\{d_v\}$ if $\sum_v d_v = 0$.

[TRUE/FALSE]
Not every decision problem in P has a polynomial time certifier.

[TRUE/FALSE]
If a problem can be reduced to linear programming in polynomial time then that problem is in P.

[TRUE/FALSE]
If we can prove that P $\neq$ NP, then a problem A $\in$ P does not belong to NP.

[TRUE/FALSE]
If all capacities in a flow network are integers, then every maximum flow in the network is such that flow value on each edge is an integer.

[TRUE/FALSE]
In a dynamic programming formulation, the sub-problems must be mutually independent.

[TRUE/FALSE]
In the final residual graph constructed during the execution of the Ford–Fulkerson Algorithm, there's no path from sink to source.

2) 16 pts
In the Bipartite Directed Hamiltonian Cycle problem, we are given a bipartite directed graph $G = (V; E)$ and asked whether there is a simple cycle which visits every node exactly once. Note that this problem might potentially be easier than Directed Hamiltonian Cycle because it assumes a bipartite graph. Prove that Bipartite Directed Hamiltonian Cycle is in fact still NP-Complete.

3) 16 pts

A tourism company is providing boat tours on a river with n consecutive segments. According to previous experience, the profit they can make by providing boat tours on segment i is known as a_i. Here a_i could be positive (they earn money), negative (they lose money), or zero. Because of the administration convenience, the local community of the river requires that the tourism company should do their boat tour business on a contiguous sequence of the river segments, i.e, if the company chooses segment i as the starting segment and segment j as the ending segment, all the segments in between should also be covered by the tour service, no matter whether the company will earn or lose money. The company's goal is to determine the starting segment and ending segment of boat tours along the river, such that their total profit can be maximized. Design an efficient algorithm to achieve this goal, and analyze its run time (Note that brute-force algorithm achieves $\Theta(n^2)$, so your algorithm must do better.)

4) 16 pts

Consider the following matching problem. There are m students $s_1, s_2, \ldots, s_m$ and a set of n companies $C = \{c_1, c_2, \ldots, c_n\}$. Each student can work for only one company, whereas company c_j can hire up to b_j students. Student s_i has a preferred set of companies $\Lambda_i \subseteq C$ at which he/she is willing to work. Your task is to find an assignment of students to companies such that all of the above constraints are satisfied and each student is assigned. Formulate this as a network flow problem and describe any subsequent steps necessary to arrive at the solution. Prove correctness.

5) 16 pts

Consider a directed, weighted graph G where all edge weights are positive. You have one Star, which lets you change the weight of any one edge to zero. In other words, you may change the weight of any one edge to zero. Give an efficient algorithm using Dijkstra's algorithm to find a lowest-cost path between two vertices s and t, given that you may set one edge weight to zero. Note: you will receive 10 pts if your algorithm is efficient. You will receive full points (16 pts) if your algorithm has the same run time complexity as Dijkstra's algorithm.

6) 16 pts

You are given n rods; they are of length $l_1, l_2, \ldots, l_n$, respectively. Our goal is to connect all the rods and form a single rod. The length after connecting two rods and the cost of connecting them are both equal to the sum of their lengths. Give an algorithm to minimize the cost of connecting them to form a single rod. State the complexity of your algorithm and prove that your algorithm is optimal.

Additional Space

CS570
Analysis of Algorithms
Fall 2013
Exam III

Name: _____

Student ID: _____

____Tuesday/Thursday Session ____Wednesday Session ____DEN

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	20	
Problem 5	20	
Total	100	

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

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

[TRUE/FALSE]
Let A and B be decision problems. If A is polynomial time reducible to B and B is in NP-Complete, then A is in NP.

[TRUE/FALSE]
In a network with source s and sink t where each edge capacity is a positive integer, there is always a max s-t flow where the flow assigned to each edge is an integer.

[TRUE/FALSE]
Let ODD denote the problem of deciding if a given integer is odd. Then ODD is polynomial time reducible to 3-SAT.

[TRUE/FALSE]
Not every decision problem in P has a polynomial time certifier.

[TRUE/FALSE]
The set of all vertices in a graph is a vertex cover.

[TRUE/FALSE]
A minimum spanning tree of a connected undirected graph remains being a minimum spanning tree even if each edge weight is doubled.

[TRUE/FALSE]
A minimum spanning tree of a bipartite graph is not necessarily a bipartite graph.

[TRUE/FALSE]
Dijkstra's algorithm can always find the shortest path between two nodes in a graph as long as there is no negative cost cycle in the graph.

[TRUE/FALSE]
Given a binary max heap of size n, the complexity of finding the smallest number in the heap is O(log n).

[TRUE/FALSE]
Given a graph G=(V,E) and an approximation algorithm that solves the vertex cover problem in G with an approximation ratio r, then this algorithm can also provide a solution to the independent set of G with the same approximation ratio r.

2) 20 pts

At a dinner party, there are n families $\{a_1,a_2,...,a_n\}$ and m tables $\{b_1,b_2,...,b_m\}$. The i^{th} family a_i has g_i members and the j^{th} table b_j has h_j seats. Everyone is interested in making new friends and the dinner party planner wants to seat people such that no two members of the same family are seated in the same table. Design an algorithm that decides if there exists a seating assignment such that everyone is seated and no two members of the same family are seated at the same table.

Construct the following network G=(V,E). For every family introduce a vertex and for every table introduce a vertex. Let a_i denote the vertex corresponding to the ith family and let b_j denote the vertex corresponding to the jth table. From every family vertex a_i to every table vertex b_j, add an edge (a_i,b_j) of capacity 1. Add two more vertices s and t. To every family vertex a_i add an edge (s,a_i) of capacity g_i. From every table vertex b_j add an edge (b_j,t) of capacity h_j.

Claim: There exists a valid seating if and only if the value of max flow from s to t in the above network equals g_1+g_2+...+g_n.

Proof of Claim: Assume there exists a valid seating, that is a seating where every one is seated and no two members in a family are seated at a table. We construct a flow f to the network as follows. If a member of the ith family is seated at the jth table in the seating assignment, then assign a flow of 1 to the edge (a_i,b_j). Else assign a flow of 0 to the edge (a_i,b_j). The edge (s,a_i) is assigned a flow equaling the number of members in the ith family that are seated (which since the seating is valid equals g_i). Likewise the edge (b_j,t) is assigned a flow equaling the number of seats taken in the table b_j (which since the seating is valid is at most h_j). Clearly the assignment is valid since by construction the capacity and conservation constrains are satisfied. Further, the value of the flow equals g_1+g_2+...+g_n.

Conversely, assume that the value of the max s-t flow equals g_1+g_2+...+g_n. Since the capacities are integers, by the correctness of the Ford-Fulkerson algorithm, there exists a maxflow (call f) such that the flow assigned to every edge is an integer. In particular, every edge between the family vertices and table vertices has a flow of either 0 or 1 (since these edges are of capacity 1). Construct a seating assignment as follows, seat a person of the ith family at the jth table if and only if f(a_i,b_j) is 1. By construction at most one member of a family is seated at a table. Since the value of f equals the capacity of the cut ({s},V-{s}), every edge out of s is saturated. Thus by flow conservation at a_i, for every a_i the number of edges out of a_i with a flow of 1 is g_i. Thus in the seating assignment, every one is seated. Further, since the flow f(b_j,t) out of b_j is at most h_j, at most h_j persons are seated at table b_j. Thus we have a valid seating.

Remark. You can solve the problem using "circulation" as well.

One way is to modify the above network by adding (for every a_i) a lower bound of g_i to the edge (s,a_i) and adding an edge (t,s) of infinite capacity. Then the modified network has a valid circulation if and only if there is a valid seating.

Another way is to remove s and its incident edge from the above network and add at t a demand of g_1+g_2+..+g_n and at each a_i a demand of -g_j. The modified network has a circulation if and only if there is a valid seating.

3) 20 pts

A decision version of the subset sum problem is as follows: Given a set of n integer numbers A = $\{a_1, a_2, \ldots, a_n\}$ and a target number *t*. Determine if there is a subset of numbers in A that add up precisely to *t*? That is, the output is *yes* or *no*. Describe an algorithm (and provide pseudo-code) to solve this problem, and analyze its complexity.

Solution: Note that this is not to show the subset-sum is an NP-complete problem. In this question, students are asked to provide a specific solution to the problem without restriction on the complexity.

Use dynamic programming:
Let S[i,j] shows if there exists a subset of $\{ a_1, a_2, \ldots, a_i \}$ that add up to j, where $0 \leq j \leq t$. S[i,j] is true or false.
Initialize: S[0,0] = true; S[0,j] = false if $j \neq 0$
Recurrence formula:
S[i,j] = S[i-1,j] OR S[i-1, j-a_i]
Output is S[n,t]
Complexity is O(t.n) – pseudo-polynomial

4) 20 pts

Given a binary search tree T, its root node *r*, and two random nodes *x* and *y* in the tree. Find the lowest common ancestry of the two nodes *x* and *y*. Note that a parent node *p* has pointers to its children *p.leftChild()* and *p.rightChild()*, a child node does **not** have a pointer to its parent node. The complexity must be $O(\log n)$ or better, where *n* is the number of nodes in the tree.
Recall that in a binary search tree, for a given node *p*, its right child *r*, and its left child *l*, $r.value() \geq p.value() \geq l.value()$. Hint: use divide and conquer

Solution: Use divide and conquer:

```
Node  LCA(r, x, y){
        If (x.value() < r.value && y.value() > r.value())
                Return r;
        Else If(x.value() < r.value && y.value() < r.value())
                Return LCA(r.leftChild() x, y);
        Else
                Return LCA(r.rightChild() x, y);
}
```

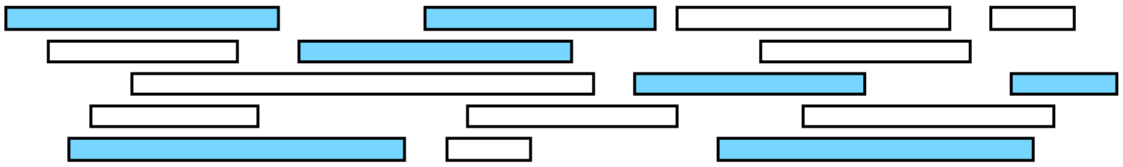
Complexity O(log n)

Note: Using the property of the binary search tree to decide if two nodes belong to the same branch or different branches of a parent node is crucial for maintaining a complexity of O(log n). Simply using the binary search to answer this question will make the overall complexity of the algorithm to be $O(\log^2 n)$.

Note that the property of a classic binary search tree is that no duplicates are allowed, so students need not care about the fact that the elements in the tree can be equal. Some few modern implementations allow duplicates in the binary search tree, but that is specifically implemented according to the requirements of each application and is not considered as a pure binary search tree.

5) 20 pts

Let X be a set of n intervals on the real line. A subset of intervals Y⊂ X is called a tiling path if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The size of a tiling cover is just the number of intervals.



A set of intervals. The seven shaded intervals form a tiling path

The Tiling problem can be posed as follows: Given a set of intervals with their start times and end times, find the smallest tiling path of X.

Decide if

a) This problem can be solved in polynomial time. If so, provide a solution and analyze its complexity, or
b) State the decision version of the problem and prove that it is NP-complete

It can be solved in polynomial time using the following greedy approach:
First sort all intervals based on their start times in ascending order. If the start time are identical, sort them based on the length of the intervals in descending order.
At each step, pick up the first interval ϑ from X and add it to Y, deletes all intervals included completely in that interval, and change the start time of the interval whose start time is in ϑ but they finish after ϑ to the finish time of . Do the sorting based on the above explanation and then continue until we do not have any interval left in X.

CS570
Analysis of Algorithms
Summer 2013
Exam III

Name: _____

Student ID: _____

____On Campus ____DEN

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	20	
Problem 5	10	
Problem 6	10	
Total	100	

2 hr exam
Close book and notes

If a description to an algorithm is required please limit your description to within 150 words, anything beyond 150 words will not be considered.

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

[TRUE/FALSE] True
Assume P !=NP. Let A and B be decision problems. If A is in NP-Complete and $A \leq_P B$, then B is not in P.

[TRUE/FALSE] True
There exists a decision problem X such that for all Y in NP, Y is polynomial time reducible to X.
It is in fact the Cook-Levin theorem that proves that there are problems that are NP-complete. Picking X to be 3-SAT (or any other in problem in NP-complete) ensures that ensures that every Y in NP is reducible to X.

[TRUE/FALSE] True
If P equals NP, then NP equals NP-complete.
True. A problem X is NP-hard iff any problem in NP can be reduced in polynomial time to X. If P equals NP, then we can reduce any problem in NP to any other problem by just solving the original problem.

[TRUE/FALSE] False
The running time of a dynamic programming algorithm is always theta(P) where P is the number of sub-problems.
Solution: False. The running time of a dynamic program is the number of subproblems times the time per subproblem. This would only be true if the time per subproblem is O(1).

[TRUE/FALSE] True
A spanning tree of a given undirected, connected graph G=(V,E) can be found in O(| E|) time. You can just walk on the graph.

[TRUE/FALSE] True
To find the minimum element in a max heap of n elements, it takes O(n) time

[TRUE/FALSE] True
Kruskal's algorithm for finding the MST works with positive and negative edge weights.

[TRUE/FALSE] False
If a problem is not in P, then it must be in NP.

[TRUE/FALSE] False
If an NP-complete problem can be solved in linear time, then all NP-complete problems can be solved in linear time.

[TRUE/FALSE] True

Linear programming problems can be solved in polynomial time.

2) 20 pts
Consider the following heuristic to compute a vertex cover of a connected undirected graph G. Pick an arbitrary vertex as the root and perform depth first search. Output the set of non-leaf vertices in the resulting depth first search tree.

(i) Show that the output is a vertex cover for G.
(ii) How good an approximation to a minimum vertex cover does this heuristic assure? That is, upper bound on the ratio of the number of vertices in the output to the number of vertices in a minimum vertex cover of G.

2) Let G=(V,E) denote the graph, T=(V,E') the resulting depth first search tree, L the set of leaf vertices in the depth first search tree and N (=V - L) the set of non leaf vertices.

(i) Assume there is an edge e=(u,v) in E that is not covered by N. This implies that both u and v are in L. Without loss of generality, assume that DFS explored u first. At this stage since e was available to DFS to leave u, the DFS would have left u to explore a new vertex thereby making u a non leaf. Hence our assumption is incorrect and N does indeed cover every edge in E.

(ii) If a vertex cover of G contains a vertex u in L, then u can be replaced with its parent in the DFS tree while still covering every edge in E without increasing the number of vertices. Hence there exists a min vertex cover (call A) of G that does not contain a vertex in L. In particular, A is also a vertex cover of the DFS tree T.

We next create a matching M as follows. Recall that a matching is a set of edges such that no two distinct edges in the set share a vertex.

For every vertex u in N, pick one edge that connects u to one of its descendents and call it e_u. We call the set of odd level non leaf vertices in the DFS tree ODD and the set of even level non leaf vertices as EVEN. If the set ODD is bigger or equal to the set EVEN, set BIG:=ODD. Else set BIG:=EVEN.

Since the total number of non leaf vertices in the tree T is |N|, BIG has size at least |N|/2. Now the edge set M = {u_e | u is in BIG} is a matching of size at least |N|/2.

Since M is a matching, to cover every edge in the matching the optimal vertex cover A has to contain at least |M| vertices. (This is because in a matching no two distinct edges share a vertex). Thus A has to contain at least |N|/2 vertices while our solution has |N| vertices. Hence our solution is at worst a 2-approximation.

It can be shown that our solution can be indeed twice as bad by considering E={(a,b),(b,c)} with DFS rooted at a.

3) 20 pts

There is a precious diamond that is on display in a museum at m disjoint time intervals. There are n security guards who can be deployed to protect the precious diamond. Each guard has a list of intervals for which he/she is available to be deployed. Each guard can be deployed to at most A time slots and has to be deployed to at least B time slots. Design an algorithm that decides if there is a deployment of guards to intervals such that each interval has either exactly one or exactly two guards deployed.

3.) We create a circulation network as follows. For the ith guard introduce a vertex g_i and for the jth time interval introduce a vertex t_j. If the ith guard is available for the jth interval, then introduce an edge from g_i to t_j of capacity 1. Add a source s and a sink t. To every guard vertex add an edge from s of capacity A and lower bound B. From every interval vertex add an edge to t of capacity 2 and lower bound 1. Add an edge from s to t of infinite capacity. We claim that there exists a valid deployment if and only if the above network has a valid circulation. The proof of the claim is similar to the survey design problem in the text. The algorithm proceeds by determining if the network has a circulation (by reducing it to a flow problem and then applying Ford-Fulkerson) and answers "yes" if and only if there is a circulation.

4) 20 pts

Your input is a string S which is a sequence of n characters from the English alphabet. The string S is believed to be a corrupted version of a text where the spacing between the words have been erased. You have access to a program Dictionary(), which takes a string w as input and returns ``yes'' if w is a valid word and ``no'' otherwise. Design an algorithm that decides if S can be partitioned (by inserting spaces) into a sequence of valid words. The running time should be polynomial in n assuming that each call to Dictionary() takes polynomial time.

For example, if S = ``sortingiseasy'', then your algorithm should output ``yes'', since ``sorting",``is",``easy'' is a sequence of valid words.

Let the length of your input string of size N.

Let b(n) be a boolean: true if the document can be split into words starting from position n in the string.

b(N) is true (since the empty string can be split into 0 words). Given b(N), b(N - 1), ... b(N - k), you can construct b(N - k - 1) by considering all words that start at character N - k - 1. If there's any such word, w, with b(N - k - 1 + len(w)) set, then set b(N - k - 1) to true. If there's no such word, then set b(N - k - 1) to false.

Eventually, you compute b(0) which tells you if the entire document can be split into words.

In pseudo-code:

```
def try_to_split(string):
  N = len(string)
  b = [False] * (N + 1)
  b[N] = True
  for i in range(N - 1, -1, -1):
    for word starting at position i:
      if b[i + len(word)]:
        b[i] = True
        break
  return b
```

There's some tricks you can do to get 'word starting at position i' efficient, but you're asked for an O(N^2) algorithm, so you can just look up every string starting at i in the dictionary.

To generate the words, you can either modify the above algorithm to store the good words, or just generate it like this:

```
def generate_words(doc, b, idx=0):
  length = 1
  while true:
    assert b(idx)
    if idx == len(string): return
    word = doc[idx: idx + length]
    if word in dictionary and b(idx + length):
```

output(word)
idx += length
length = 1

Here b is the boolean array generated from the first part of the algorithm.

5) 10 pts
Show that the following problem is NP-complete:
For an undirected graph G=(V,E), does G have a spanning tree with at most 2 leaf vertices?

5.) Call the decision problem in question ST2L.

Given the graph G (instance) and a set of edges that forms a spanning tree (certificate), we can verify in polynomial time if the set of edges indeed forms a spanning tree and that spanning tree has at most two leaves. Thus the ST2L problem is indeed in NP.

All that is left is to prove that ST2L is NP-Hard. Recall that a leaf vertex in a tree is a vertex of degree 1. Thus a spanning tree has to have at least 2 leaves. A spanning tree with exactly 2 leaves is in one to one correspondence with a Hamiltonian path. Hence our problem is merely a restatement of the Hamiltonian Path problem. Hence you can use the reduction in the last HW to reduce Hamiltonian cycle to Hamiltonian Path (and then Hamiltonian Path to ST2L.)

(Note: If you assumed that the spanning tree is rooted and that the root is not a leaf even if it has degree one, then we need to slightly modify the argument that reduces Hamiltonian path to ST2L. Given a graph G as an instance of Ham-Path, if G has exactly 2 vertices and one edge output "yes", Else call the blackbox that solves ST2L with input G and return the result.)

6) 10 pts

The following are a few of the design strategies we learned in class to solve problems.

1. Dynamic programming.
2. Greedy strategy.
3. Divide-and-conquer.

For each of the following problems, mention which of the above design strategies (or combinations of strategies) we used in class to solve these problems:

1. Determining if a graph has a negative cycle
   Dynamic programming

2. Minimum spanning tree
   Greedy strategy

3. Closest pair of points on a plane
   Divide and conquer

4. Memory efficient sequence alignment
   Dynamic programming + divide and conquer

5. Stable matching
   Greedy Strategy

CS570
Analysis of Algorithms
Summer 2011
Final Exam

Name: _____

Student ID: _____

____Check if DEN student

	Maximum	Received
Problem 1	20	
Problem 2	13	
Problem 3	14	
Problem 4	13	
Problem 5	20	
Problem 6	20	
Total	100	

2 hr exam
Close book and notes

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification except for the question at the bottom of the page.

[TRUE/FALSE]
In a flow network, if all edge capacities are distinct, then the max flow of this network is unique.

[TRUE/FALSE]
To find the minimum element in a max heap of n elements, it takes O(n) time.

[TRUE/FALSE]
Let T be a spanning tree of graph G(V, E), let k be the number of edges in T, then k=O(V)

[TRUE/FALSE]
Linear programming problems can be solved in polynomial time.

[TRUE/FALSE]
Consider problem A: given a flow network, find the maximum flow from a node s to a node t. problem A is in NP.

[TRUE/FALSE]
Given n numbers, it takes O(n) time to construct a binary min heap.

[TRUE/FALSE]
Kruskal's algorithm for finding the MST works with positive and negative edge weights.

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

[TRUE/FALSE]
If a problem is not in P, then it must be in NP.

[TRUE/FALSE]
L1 can be reduced to L2 in Polynomial time and L1 is in NP, then L2 is in NP

2) 13 pts

Imagine that you constructed an approximation algorithm for the Traveling Salesman Problem that could always calculate a solution that is correct within a factor of $1/k$ of the optimal tour in $O(n^{2k})$ time. Would you be able to use this approximation algorithm to obtain a "good" solution to all other NP-Complete problems? Explain why or why not.

Yes. You can use it.

In the Traveling Salesperson Problem, we are given an undirected graph G = (V,E) and cost c(e) > 0 for each edge e 2 E. Our goal is to find a Hamiltonian cycle with minimum cost. A cycle is said to be Hamiltonian if it visits every vertex in V exactly once.
TSP is known to be NP-complete, and so we cannot expect to exactly solve TSP in polynomial time. What is worse, there is no good approximation algorithm for TSP unless P = NP. This is because if one can give a good approximation solution to TSP in polynomial time, then we can exactly solve the NP-Complete Hamiltonian cycle problem (HAM) in polynomial time, which is impossible unless P = NP. Recall that HAM is the decision problem of deciding whether the given graph G = (V,E) has a Hamiltonian cycle or not.
Theorem 1 ([2]) The Traveling Salesperson Problem cannot be approximated within any factor, unless P = NP.
Proof: For the sake of contradiction, suppose we have an approximation algorithm A on TSP with an approximation ratio C. We show a contradiction by showing that using A, we can exactly solve HAM in polynomial time. Let G = (V,E) be the given instance of HAM. We create a new graph H = (V,E0) with cost c(e) for each e 2 E0 such that c(e) = 1 if e 2 E, otherwise c(e) = B, where B = nC + 1 and n = |V |.
We observe that if G has a Hamiltonian cycle, OPT = n, otherwise OPT § n−1+B = n(C+1).
(Here, OPT denotes the cost of an optimal TSP solution in H.) Note that there is a "gap" between when G has a Hamiltonian cycle and when it does not. Thus, if A has an approximation ratio of C, we can tell whether G has a Hamiltonian cycle or not: Simply run A on the graph H; if G has a Hamiltonian cycle, A returns a TSP tour in H of cost at most COPT = Cn. Otherwise, H has no TSP tour of cost less than n(C + 1), and so A must return a tour of at least this cost.

3) 14 pts

Prove the following statement or give a counterexample: "For any weighted graph G there is node v in G such that a Shortest Path Tree (tree found by running Dijkstra's) with v as source is identical to a Minimum Spanning Tree of G."

False. Note that the MST will always avoid the largest weight edge in a cycle (assuming there is a unique largest), whereas the shortest path tree might use it. Consider the following graph where the diagonal edges

have weight 3 and the other edges have weight 2. A MST consists of three of the weight 2 edges, whereas a shortest path tree will use one of the diagonal edges.

4) 13 pts

Define the capacity of a node as the maximum incoming flow it can have. Briefly show how to reduce the problem of finding the maxflow in a network with capacity limits on the nodes to the usual maxflow problem. Answer in one or two sentences.

Replace the node v with two nodes v' and v". Add an edge from v' to v" with the desired node capacity. Replace all of the edges (u, v) that enter v to (u, v') so that they enter v'. Replace all of the edges (v, w) that leave v to (v", w) so that they leave v".

5) 20 pts

Let G = (V,E) be an undirected graph in which the vertices represent small towns and the edges represent roads between those towns. Each edge e has a positive integer weight d(e) associated with it, indicating the length of that road. The distance between two vertices (towns) in a graph is defined to be the length of the shortest weighted path between those two vertices.

Each vertex v also has a positive integer c(v) associated with it, indicating the cost to build a fire station in that town.

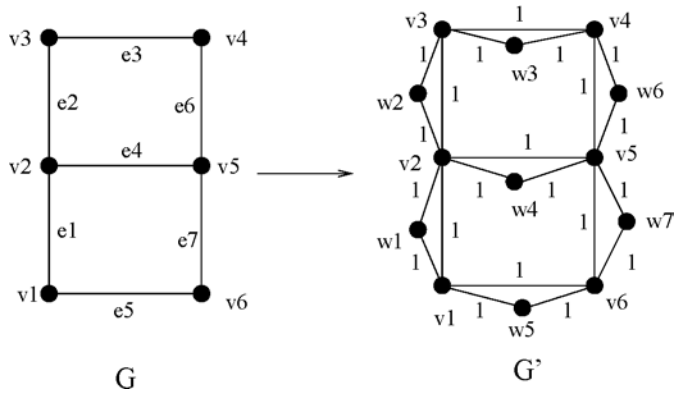
In addition, we are given two positive integer parameters D and C. Our objective is to determine whether there is a way to build fire stations such that the total cost of building the fire stations does not exceed C and the distance from any town to a fire station does not exceed D. This problem is known as the Rural Fire Station (RFS) Problem.

Your company has been hired by the American League of Rural Fire Departments to study this problem. After spending months trying unsuccessfully to find an efficient algorithm for the problem, your boss has a hunch that the problem is NP-complete. Prove that RFS is NP-complete.

We prove that FIRE-STATION-PLACEMENT is NP-complete. To see that it is NP we use the set of vertices for placing the fire stations as a certificate. Clearly, there is a polynomial time verification algorithm.

We now prove that VERTEX-COVER $\leqslant_p$ FIRE-STATION-PLACEMENT. Let (G, k) be the input for VERTEX-COVER. Without loss of generality, we assume that no vertices in G have degree 0. (If G had any vertices of degree 0, then remove them. Notice that this won't affect the size of a minimum vertex cover of G.)

The graph for the fire-station-placement problem $G' = (V',E')$ is obtained from $G = (V,E)$ as follows. Let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We will create a new vertex w_i for each edge in E. Let $W = \{w_1, \ldots, w_m\}$. Then $V' = V \cup W$ and for each edge $e = (u, v) \in E$ (with associated vertex w) we place the following 3 edges in E': (u, v), (u,w), (v,w). All edges have a weight of 1. So $|V'| = |V| + |E|$ and $|E'| = 3|E|$. Here is an example of this transformation:



Clearly G' can be constructed in polynomial time. The input for FIRE-STATION-PLACEMENT is (G', C = k, D = 1) and the cost of building each fire station is 1.
We now prove that G has a vertex cover of size k if and only if G' has a placement of k fire stations so that each vertex has distance of at most 1 to its nearest fire station. Suppose that G has a vertex cover C of size k. We show that by placing a fire station in G' at each vertex in C all vertices are directly connected to a fire station (or have a fire station). Suppose not. Let u be such a vertex greater than distance 1 from its nearest fire station. First suppose that $u \in V$ (i.e. an original vertex from G). Since all vertices in G have at least one edge, u must be connected in G' to some vertex $v \in V$ where v is also not in C. This contradicts that C is a vertex cover. Next suppose that $u \in W$ (it is one of the added vertices), then u is connected to two vertices v_1 and v_2 where $(v_1, v_2) \in E$ and there is not a fire station at v_1 or v_2 thus contradicting that C was a vertex cover.
We first prove that if F is the set of vertices to place fire stations so that $|C| = k$ and all vertices have distance at most 1 to some fire station, then there is a vertex cover of size k in G.

We first transform F into a fire station placement solution F_0 in which $|F'| = |F|$ and in which F' is a subset of V (i.e. F' only includes original vertices). We build F' from F by doing the following: for each vertex $w \in W \setminus F$ (in any order). Suppose w is the vertex added corresponding to edge (u_1, u_2). In this case, we can simply move the fire station at vertex w to u_1 (or anywhere else if u_1 already has a fire station). Notice that the fire station at w could only be a nearest fire station for w, u_1 and u_2 and hence if there is a fire station at u_1 then each of w, $u_1$and u_2 can use u_1 as a nearest fire station. Hence we maintain the invariant (started with F) that our solution at each step (and hence F') satisfies the requirements that each vertex in $V' = V \cup W$ is within distance 1 from a fire station.
We now prove that F' is a vertex cover in G. Suppose not. Then there is an edge $(u, v) \in G$ such that neither of u or v belong to F'. Consider the vertex w corresponding to the edge (u, v) (i.e the triangle u,v,w). Notice that w is connected only to u and v and thus w must have a distance of greater than 1 to its nearest fire station thus contradicting the requirements for a proper fire station placement in G'.

6) 20 pts
You are given a array of n positive numbers A[1]…A[n]. You are not allowed to re-order them. You need to find two indexes i and j, where 1<=i<j<=n, such that A[j]-A[i] is maximized. Solve this problem in O(n) time using a dynamic programming method.

Let OPT[i] be the max difference of the first i numbers.

OPT[i+1]=max{OPT[i], A[i+1]-min[i]}
Where min[i] is the minimum of the first i numbers.
min[i+1]=min{A[i+1], min[i]}.
To find the indices, we can record the choice OPT[] and min[] of each step.
Calculate this takes O(n) because each element of the array is visited once.

CS570
Analysis of Algorithms
Summer 2010
Final Exam

Name: _____

Student ID: _____

____Check if DEN student

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	20	
Problem 4	15	
Problem 5	15	
Problem 6	10	
Total	100	

2 hr exam
Close book and notes

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

[TRUE/FALSE]
Given a network *G(V, E)* and flow *f*, and the residual graph G_f(V', E'), then |V|=|V'| and 2|E|>=|E'|.

[TRUE/FALSE]
The Ford-Fulkerson Algorithm terminates when the source *s* is not reachable from the sink *t* in the residual graph.

[TRUE/FALSE/UNKOWN]
NP is the class of problems that are not solvable in polynomial time.

[TRUE/FALSE/UNKOWN]
If problem A is NP complete, and problem B can be reduced to problem A in quadratic time. Then problem B is also NP complete

[TRUE/FALSE/UNKOWN]
If X can be reduced in polynomial time to Y and Z can be reduced in polynomial time to Y, then X can be reduced in polynomial time to Z.

[TRUE/FALSE]
Let G(V,E) be a weighted graph and let T be a minimum spanning tree of G obtained using Prim's algorithm. The path in T between *s* (the root of the MST) and any other node in the tree must be a shortest path in G.

[TRUE/FALSE]
DFS can be used to find the shortest path between any two nodes in a non-weighted graph.

[TRUE/FALSE]
The Bellman-Ford algorithm cannot be parallelized if there are negative cost edges in the network.

[TRUE/FALSE]
A perfect matching in a bipartite graph can be found using a maximum-flow algorithm.

[TRUE/FALSE]
Max flow in a flow network with integer capacities can be found exactly using linear programming.

2.We know that max-flow problems can be solved in polynomial time. However, consider a modified max-flow problem in which every edge must either have zero flow or be completely saturated. In other words, if there is any flow through an edge at all, the flow through the edge is the capacity of the edge. Let's call a flow a **saturated** flow if it satisfies this additional constraint, as well as the usual constraints of flow problems. The modified problem asks us to find the maximum saturated flow.

a.(2 point) Formulate this problem as a decision (yes/no) problem. (We'll call it MAX-SATURATED-FLOW.) Write your answer in the form of a question.

ANSWER.

Is there a saturated flow of value *k* or greater? (Alternative: is there a saturated flow of value k?) Here, *k* is an arbitrary number.

b. (5 points) Show that MAX-SATURATED-FLOW is in NP.

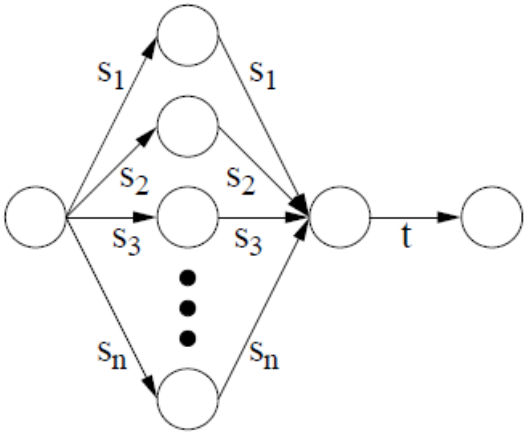
ANSWER.

For a certificate, we use a list of all the saturated edges. Given such a list, we can check in linear time the flow out of the source node, and whether the conservation condition for each node (exceptthe source and sink) is satisfied.

c. (13 points) Show that MAX-SATURATED-FLOW is NP-hard. Hint: use subset-sum problem.

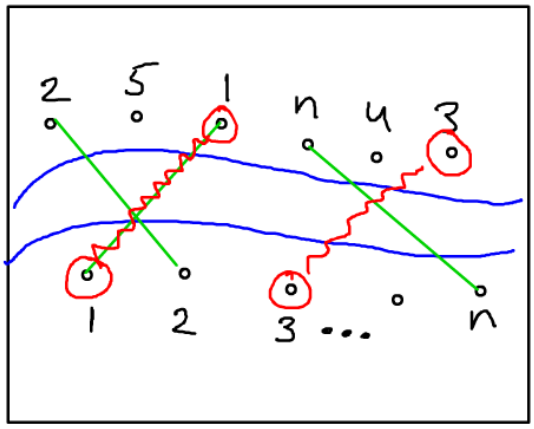
ANSWER.

It's relatively easy to reduce SUBSET-SUM to MAX-SATURATED-FLOW. Consider the NP-hard problem of finding a subset of $\{ s_1, s_2 \ldots s_n\}$that sums to *t*. This problem is solved if we find a saturated flow of value *t* in the following graph (or alternatively, the amaximum saturated flow of this graph).



3.Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates a(1) ... a(n) and n cities on the northern bank with x-coordinates b(1) ... b(n). Note that the x-coordinates are NOT in sorted order. You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. But when connecting cities, you can only connect city i on the northern bank to the corresponding city i on the southern bank. Design an efficient algorithm for that.

Answer:
Let's us first interpret the question correctly. We want to build as many none crossing bridge as possible. As shown in the following image,



In the lower level of river, we use 1…n to denote all the cities in the south, in the upper level of the river, we use the same number to denote the corresponding city in the north. Our problem is to connect as many of them as possible.

Let X(i) to be the index of corresponding city on the northern bank (e.g., X(1) = 3 in the above figure). To compute each X(i), we need O(n) time
Therefore, the problem to reduce to "Find the longest increasing subsequence in X(1)…X(n)". This problem is solved in Homework 4, please refer to the solution of problem 3 in Homework 4.

The complexity of overall algorithm is $O(n^2)$

Grading Strategy:
Correct Solution: 12 points
Complexity : 4 points.
Correctness Proof: 4 points

Any ***CORRECT*** solution within time complexity $O(n^3)$ (including $O(n^3)$) will receive full credits for the solution part, otherwise, will minus one or two points.

4.15 pts

Consider the following vertex cover algorithm for an undirected graph G = (V, E).
0. Initialize C = Null set.
1. Pick any edge e = (u, v) ϵ E, add u and v to C. Delete all edges incident on either u or v.
2. If E is empty, Output C and exit. Otherwise, go to step 1.
Show that C is a vertex cover and that the size of C is at most twice as big as the minimum vertex cover. (Thus we have a 2-approximation)

Answer:

Every edge $e \in E$ was either picked in step 1 or deleted in step 1. If it was picked, then both its edges were added to C and is hence covered. If it was deleted, then it already had at least one incident vertex contained in C and is hence covered. Thus C is indeed a vertex cover.

Let M be the set of edges picked in step 1. Let C_{opt} be an minimal vertex cover. Observe that no two edges in M share a vertex (When we pick an e, the deletion step ensures that no other edge that shares an edge with e is ever picked). Thus for each edge $e \in M$, C_{opt} has to contain at least one of its incident vertices (Moreover these vertices are distinct as reasoned in the previous line). Hence

$$|C_{opt}| \geq |M|$$

From the algorithm, we have that $|C| = 2|M|$. Thus

$$|C| \leq 2|C_{opt}|$$

5.15 pts

a- Show that an undirected graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge (i.e., a unique edge of smallest cost) crossing the cut.

b- Show that the converse is not true by giving a counterexample.

Answer:

a) We will use the following facts proven in class

For every cut, if there are no ties, the lightest edge has to be a part of every MST.

For any cut, replacing the edge of a spanning tree crossing that cut with a lighter edge reduces the weight of the spanning tree

Consider any cut $(S, V \setminus S)$. The two facts imply that, to connect S and $V \setminus S$ we have to include the lightest edge in the MST and have no alternatives. Thus there is only one MST.

b) Let a, b, c, d be the set of vertices. There are 3 edges, $(a, b), (a, c)$ and (c, d) with weights 3,4 and 3 respectively. The graph is it self a tree and hence has a unique spanning tree. However, the cut $(\{a, c\}, \{b, d\})$ has two lightest edges crossing. Thus "unique MST" need not imply that every cut has a unique lightest edge.

Proof details (not expected to write these down)

Let $G = (V, E)$ be the graph. Define a subgraph $T = (V, F)$ where $F \subset E$ is an edge set that contains $e \in E$ if and only if e is the lightest edge of some cut. Here, T is in fact a tree because for every cut there is excatly one edge crossing the cut (unique lightest edge). And T spans the graph as every cut has at least one edge crossing it. Thus T is a spanning tree.

We claim that $w(T) < w(T')$ for every other tree T'.

Assume otherwise. Let T' be a minimal counterexample to our claim. Let e' be an edge in $T\prime$ that is not in T. Take any cut that has e' crossing it. As e' is not in E, we know that by replacing e' by the lightest edge of the cut, we contradict the minimality of T'.

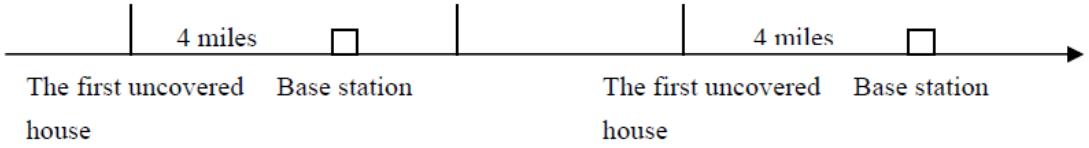
Thus there is a unique MST(which is T).

6. 10 pts

Consider a long country road with houses scattered very sparsely along it. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal, using as few base stations as possible.

Answer:

The algorithm to locate the least number of base stations is given as follows:
Line 1 Start from the beginning of the road
Line 2 Find the first uncovered house on the road
Line 3 If there is no such a house, terminate this algorithm; otherwise, go to next line
Line 4 Locate a base station at 4 miles away after you find this house along the road
Line 5 Go to Line 2



This algorithm is **O(n).**
Now we prove its correctness. We want to show that this algorithm (denoted by S) does provide the smallest number of base stations. Let some other optimal solution be S'. Let DS(i) denote the number of homes covered by first i base stations in solution S. We now show DS(i)>DS'(i) for all i.
First, we can see DS(1)>=DS'(1) because S tries to put the first base station as faraway to the first home as possible.
Now given DS(i)>=DS'(i), we show that DS(i+1)>=DS'(i+1). This is true unless in S' the (i+1)th base station cover at least one more home than the (i+1)th base station in S further down the road. This is not possible because according to our greedy approach we have covered as far as possible in S. Thus this base station in S' much covered more than 8 miles. This is contradiction.
Therefore we proved the correctness of our algorithm.

CS570
Analysis of Algorithms
Summer 2009
Final Exam

Name: _____

Student ID: _____

	Maximum	Received
Problem 1	10	
Problem 2	20	
Problem 3	15	
Problem 4	20	
Problem 5		
Problem 6		
Total	100	

2hr exam, closed books and notes.

1) 10 pts

For each of the following sentences, state whether the sentence is known to be TRUE, known to be FALSE, or whether its truth value is still UNKNOWN.

(a) If a problem is in P, it must also be in NP.

TRUE.

(b) If a problem is in NP, it must also be in P.

UNKNOWN.

(c) If a problem is NP-complete, it must also be in NP.

TRUE.

(d) If a problem is NP-complete, it must not be in P.

UNKNOWN.

(e) If a problem is not in P, it must be NP-complete.

FALSE.

If a problem is NP-complete, it must also be NP-hard.

TRUE.

If a problem is in NP, it must also be NP-hard.

FALSE.

If we find an efficient algorithm to solve the Vertex Cover problem we have proven that P=NP

TRUE.

If we find an efficient algorithm to solve the Vertex Cover problem with an approximation factor $\rho \geq 1$ (a single constant) then we have proven that P=NP

FALSE.

If we find an efficient algorithm that takes as input an approximation factor $\rho \geq 1$ and solves the Vertex Cover problem with that approximation factor, we have proven that P=NP.

TRUE.

2) 20 pts

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time.

Consider running Dijkstra's algorithm on a graph, where the weight function is $w : E \to \{1, \ldots W - 1\}$. To solve this efficiently, implement the priority queue by an array A of length $WV + 1$. Any node with shortests path estimate d is kept in a linked list at A[d]. A[$WV + 1$] contains the nodes with ∞ as estimate.

EXTRACT-MIN is implemented by searching from the previous minimun shortest path estimate until a new is found. DECREASE-KEY simply moves vertices in the array. The EXTRACT-MIN operations takes a total of $O(VW)$ and the DECREASE-KEY operations take $O(E)$ time in total. Hence the running time of the modified algorithm will be $O(VW + E)$.

3) 15 pts

At a dance, we have n men and n women. The men have height g(1),…,g(n), and women h(1),…,h(n). For the dance, we want to match up men with women of roughly the same height. Here are the precise rules:

a. Each man i is matched up with exactly one woman w_i, and each woman with exactly one man.
b. For each couple (i, w_i), the mismatch is height difference $|g(i)-h(w_i)|$.
c. Our goal is to find a matching minimizing the maximum mismatch, $MAX_i |g(i)-h(w_i)|$.

Give an algorithm that runs in O(n logn) and achieves the desired matching. Provide proof of correctness.

We use an exchange argument. Let w_i denote the optimal solution. If there is any pair i, j such that $i < j$ in our ordering, but $w_i > w_j$ (also with respect to our ordering), then we evaluate the effect of switching to $w'_i := w_j, w'_j := w_i$. The mismatch of no other couple is affected. The couple including man i now has mismatch $|g(i) - h(w'_i)| = |g(i) - h(w_j)|$. Similarly, the other switched couple now has mismatch $|g(j) - h(w_i)|$.

We first look at man i, and distinguish two cases: if $h(w_j) \leq g(i)$ (i.e., man i is at least as tall as his new partner), then the sorting implies that $|g(i)-h(w_j)| = g(i)-h(w_j) \leq g(j)-h(w_j) = |g(j)-h(w_j)|$. On the other hand, if $h(w_j) > g(i)$, then $|g(i) - h(w_j)| = h(w_j) - g(i) \leq h(w_i) - g(i) = |h(w_i) - g(i)|$. In both cases, the mismatch between man i and his new partner is at most the previous maximum mismatch.

We use a similar argument for man j. If $h(w_i) \leq g(j)$, then $|g(j)-h(w_i)| = g(j)-h(w_i) \leq g(j)-h(w_j) = |g(j) - h(w_j)|$. On the other hand, if $h(w_i) > g(j)$, then $|g(j) - h(w_i)| = h(w_i) - g(j) \leq h(w_i) - g(i) = |h(w_i) - g(i)|$. Hence, the mismatch between j and his partner is also no larger than the previous maximum mismatch.

Hence, in all cases, we have that both of the new mismatches are bounded by the larger of the original mismatches. In particular, the maximum mismatch did not increase by the swap. By making such swaps while there are inversions, we gradually transform the optimum solution into ours. This proves that the solution found by the greedy algorithm is in fact optimal.

4) 20 pts

You are given integers p0, p1, …, pn and matrices A1, A2, …, An where matrix Ai has dimension (pi-1) * pi

(a) Let m(i, j) denote the minimum number of scalar multiplications needed to evaluate the matrix product Ai Ai+1 … Aj. Write down a recursive algorithm to compute m(i, j), 1 <= i <= j <= n, that runs in O(n^3) time.
Hint: You need to consider the order in which you multiply the matrices together and find the optimal order of operations.

```
Initialize m[i, j] = -1  1 <= i <= j <= n

Output mcr(1, n)

mcr(i, j) {
   if m[i, j] >= 0 return m[i, j]
   else if I == j m[i, j] = 0
   else m[i, j] = min (i <= k < j) { mcr(i, k) + mcr(k+1, j) +Pi-1*Pk*Pk}

   return m[i, j]
}
```

(b) Use this algorithm to compute m(1,4) for p0=2, p1=5, p2=3, p3=6, p4=4

m[i, j]

i\j	2	3	4
1	30	66	114
2		90	132
3			72

m[1, 4] = 114

5) 20 pts

In a certain town, there are many clubs, and every adult belongs to at least one club. The townspeople would like to simplify their social life by disbanding as many clubs as possible, but they want to make sure that afterwards everyone will still belong to at least one club.
Prove that the Redundant Clubs problem is NP-complete.

First, we must show that Redundant Clubs is in NP, but this is easy: if we are given a set of K clubs, it is straightforward to check in polynomial time whether each person is a member of another
club outside this set.
Next, we reduce from a known NP-complete problem, Set Cover. We translate inputs of Set Cover to inputs of Redundant Clubs, so we need to specify how each Redundant Clubs input element
is formed from the Set Cover instance. We use the Set Cover's elements as our translated list of people,
and make a list of clubs, one for each member of the Set Cover family. The members of each club are just the elements of the corresponding family. To finish specifying the Redundant Clubs input,
we need to say what K is: we let $K = F - K_{SC}$ where F is the number of families in the Set Cover instance and K_{SC} is the value K from the set cover instance. This translation can clearly be done in polynomial time (it just involves copying some lists and a single subtraction).
Finally, we need to show that the translation preserves truth values. If we have a yes-instance of Set Cover, that is, an instance with a cover consisting of K_{SC} subsets, the other K subsets form a solution to the translated Redundant Clubs problem, because each person belongs to a club in the
cover. Conversely, if we have K redundant clubs, the remaining K_{SC} clubs form a cover. So the answer
to the Set Cover instance is yes if and only if the answer to the translated Redundant Clubs instance
is yes.

6) 15 pts

A company makes two products (X and Y) using two machines (A and B). Each unit of X that is produced requires 50 minutes processing time on machine A and 30 minutes processing time on machine B. Each unit of Y that is produced requires 24 minutes processing time on machine A and 33 minutes processing time on machine B.

At the start of the current week there are 30 units of X and 90 units of Y in stock. Available processing time on machine A is forecast to be 40 hours and on machine B is forecast to be 35 hours.

The demand for X in the current week is forecast to be 75 units and for Y is forecast to be 95 units—these demands must be met. In addition, company policy is to maximize the combined sum of the units of X and the units of Y in stock at the end of the week.

a. Formulate the problem of deciding how much of each product to make in the current week as a linear program.

Solution

Let

- x be the number of units of X produced in the current week
- y be the number of units of Y produced in the current week

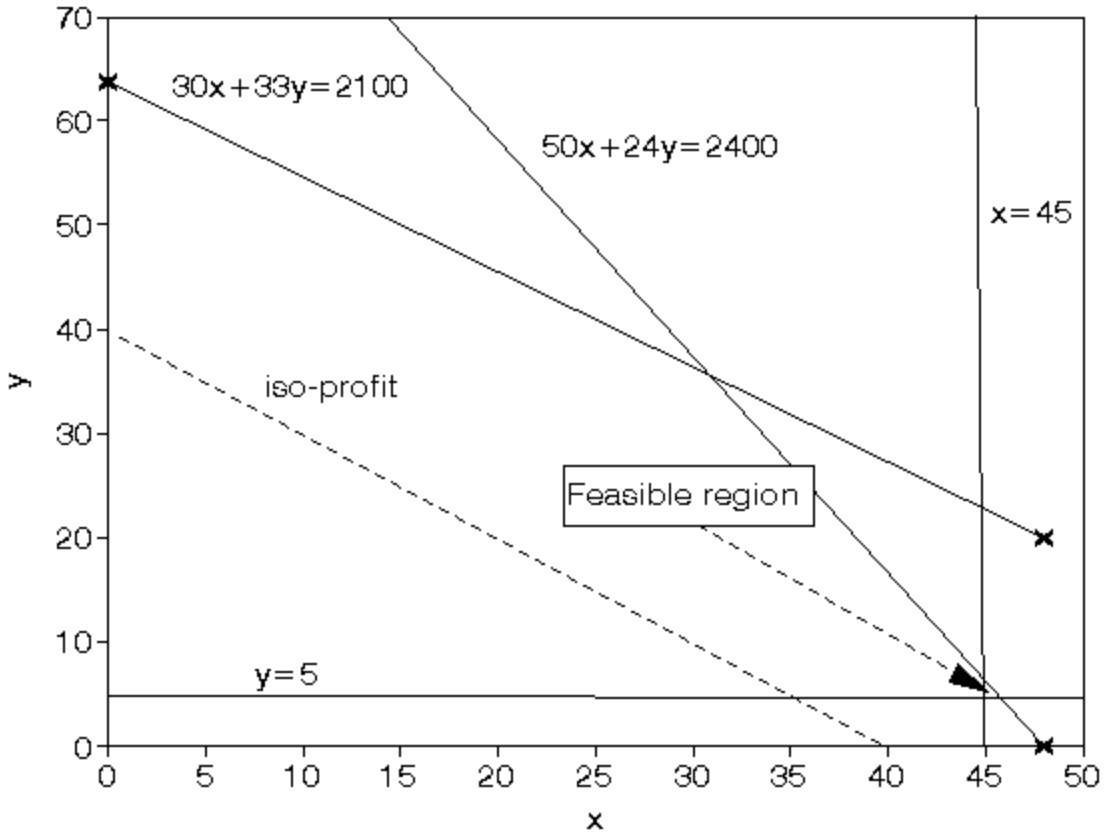
then the constraints are:

- 50x + 24y <= 40(60) machine A time
- 30x + 33y <= 35(60) machine B time
- x >= 75 - 30
- i.e. x >= 45 so production of X >= demand (75) - initial stock (30), which ensures we meet demand
- y >= 95 - 90
- i.e. y >= 5 so production of Y >= demand (95) - initial stock (90), which ensures we meet demand

The objective is: maximise (x+30-75) + (y+90-95) = (x+y-50)
i.e. to maximise the number of units left in stock at the end of the week

b. Solve this linear program graphically.

It can be seen in diagram below that the maximum occurs at the intersection of x=45 and 50x + 24y = 2400



Solving simultaneously, rather than by reading values off the graph, we have that x=45 and y=6.25 with the value of the objective function being 1.25

CS570
Analysis of Algorithms
Fall 2008
Final Exam

Name: _____

Student ID: _____

____Monday Section ____Wednesday Section ____Friday Section

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	20	
Problem 5	20	
Problem 6	10	
Problem 7	10	
Total	100	

2 hr exam
Close book and notes

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

[TRUE]
If NP = P, then all problems in NP are NP hard

[FALSE]
L1 can be reduced to L2 in Polynomial time and L2 is in NP, then L1 is in NP

[FALSE]
The simplex method solves Linear Programming in polynomial time.

[FALSE]
Integer Programming is in P.

[FALSE]
If a linear time algorithm is found for the traveling salesman problem, then every problem in NP can be solved in linear time.

[TRUE]
If there exists a polynomial time 5-approximation algorithm for the general traveling salesman problem then 3-SAT can be solved in polynomial time.

[FALSE]
Consider an undirected graph G=(V, E). Suppose all edge weights are different. Then the longest edge cannot be in the minimum spanning tree.

[FALSE]
Given a set of demands D = *{dv}* on a directed graph *G(V,E)*, if the total demand over *V* is zero, then *G* has a feasible circulation with respect to *D.*

[TRUE]
For a connected graph G, the BFS tree, DFS tree, and MST all have the same number of edges.

[FALSE]
Dynamic programming sub-problems can overlap but divide and conquer sub-problems do not overlap, therefore these techniques cannot be combined in a single algorithm.

Grading Criteria: Pretty clear, each has two point. These T/F are designed and answered by Professor Shamsian

2) 10 pts
Demonstrate that an algorithm that consists of a polynomial number of calls to a polynomial time subroutine could run in exponential time.

Suggested Solution:
Suppose X takes an input size of n and returns an output size of n^2. You call X a polynomial number of times say n. If the size of the original input is n the size of the output will be n^{2n} which is exponential WRT n.

Grading Criteria: Rephrasing the question doesn't get that much. If you show you at least understood polynomial / exponential time, you get some credit.

3) 10 pts

Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Suggested solution :

```
DIJKSTRA(G,w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← 0
3 Q ← V[G]
4 while Q ≠ 0
5       do u ← EXTRACT-MIN(Q)
6       S ← S ∪ {u}
7       for each vertex v ∈ Adj[u]
8             do RELAX(u, v,w)

RELAX(u, v,w)
1 if d[v] > d[u] + w(u, v)
2 then d[v] ← d[u] + w(u, v)
3 π[v]← u
```

We have the algorithm as shown above.

We show that in each iteration of Dijkstra's algorithm, d[u]=δ(s,u) when u is added to S (the rest follows from the upper-bound property). Let $N^-(s)$ be the set of vertices leaving s, which have negative weights. We divide the proof to vertices in $N^-(s)$ and all the rest. Since there are no negative loops, the shortest path between s and all $u \in N^-(s)$, is through the edge connecting them to s, hence δ(s,u)=w(s,u), and it follows that after the first time the loop in lines 7,8 is executed d[u]= w(s,u)= δ(s,u) for all $u \in N^-(s)$ and by the upper bound property d[u]= δ(s,u) when u is added to S. Moreover it follows that S= $N^-(s)$ after $|N^-(s)|$ steps of the while loop in lines 4-8.

For the rest of the vertices we argue that the proof of Theorem 24-6 (***Theorem 24.6 - Correctness of Dijkstra's algorithm, Introduction to Algorithem by Cormen)*** holds since every

shortest path from s includes at most one negative edge (and if does it has to be the first one). To see why this is true assume otherwise, which mean that the path contains a loop, contradicting the property that shortest paths do not contain loops.

Grading Criteria : You need to argue, so bringing just one example shouldn't get you the whole credit , but it may did! If you showed you at least understood Dijkstra, you got some credit too.

4) 20 pts

Phantasy Airlines operates a cargo plane that can hold up to 30000 pounds of cargo occupying up to 20000 cubic feet. We have contracted to transport the following items.

Item type	Weight	Volume	Number	Cost if not carried
1	4000	1000	3	$800
2	800	1200	10	$150
3	2000	2200	4	$300
4	1500	500	5	$500

For example, we have contracted 10 items of type 2, each of which weighs 800 pounds and takes up to 1200 cubic feet of space. The last column refers to the cost of subcontracting shipment to another carrier.

For each pound we carry, the cost of flying the plane increases by 5 cents. Which items should we put in the plane, and which should we ship via other carrier, in order to have the lowest shipping cost? Formulate this problem as an integer programming problem.

Suggested Solution:

Assume the data in the table are per item.

Considering x_1, x_2, x_3, x_4 are the items that we will ship for types 1,2,3 and 4 respectively.

It is clear that :

$$0 \le x_1 \le 3 \quad \& \quad 0 \le x_2 \le 10 \quad \& \quad 0 \le x_3 \le 4 \quad \& \quad 0 \le x_4 \le 5$$

Weight constraint :

$$x_1 * 4000 + x_2 * 800 + x_3 * 2000 + x_4 * 1500 \le 30{,}000$$

Volume constraint:

$$x_1 * 1000 + x_2 * 1200 + x_3 * 2200 + x_4 * 500 \le 20{,}000$$

Cost of shipping :

$$C_{Ship} = C_A + (x_1 * 4000 + x_2 * 800 + x_3 * 2000 + x_4 * 1500) * \$0.05$$

Where C_A is the cost of operating empty cargo plane

Cost of sub-contracting

$$C_{Sub} = (3 - x_1) * 800 + (10 - x_2) * 150 + (4 - x_3) * 300 + (5 - x_{4)} * 500)$$

Out objective is to minimize $C_{Ship} + C_{Sub}$

Grading Criteria: Some credit will be deducted if you missed some optimization part. Making this simple question complicated may result deduction.

5) 20 pts
Suppose you are given a set of n integers each in the range 0…K. Give an efficient algorithm to partition these integers into two subsets such that the difference |S1-S2| is minimized, where S1 and S2 denote the sums of the elements in each of the two subsets.

Proposed Solution :

```
a[1]...a[m] = the set of integers
opt[i,k] = true if and only if it is possible to sum to k using elements 1...i

Initialize opt(i,k) = false for all i,k
for(i=1...n) {
  for(k=1..K) {
    if(opt(i,k) == true) {
    for(j=1..m) {
      opt(i,k + a[j]) = true;
    }
  }
}
for(k = floor(K/2)..1) {
  if(opt(n, k)) {
    Return |K - 2k|;   // Returns the difference. Partition can be found by back tracking
  }
}
```

6) 10 pts
Adrian has many, many love interests. Many, many, many love interests. The problem is, however, a lot of these individuals know each other, and the last thing our polyamorous TA wants is fighting between his hookups. So our resourceful TA draws a graph of all of his potential partners and draws an edge between them if they know each other. He wants at least *k* romantic involvements and none of them must know each other (have an edge between them). Can he create his LOVE-SET in polynomial time? Prove your answer.

Proposed Solution :
This is just Independent Set. Proof of NP-completeness was shown in class lecture. We show the correctness as follows: 1) Any Independent Set of size k on the graph will satisfy the requirement that no two love interests know each other (share an edge). 2) If there is a set of k love interests none of which know each other, then there must be a corresponding set of k vertices, such that no two share an edge (know each other).

7) 10 pts
An edge in a flow network is called a bottleneck if increasing its capacity increases the max flow in the network. Give an efficient algorithm for finding all the bottlenecks in the network.

Proposed Solution
Find max flow and the corresponding residual graph, then for each edge increase its capacity by one unit and see if you find a new path from s to t in the residual graph. (Of course you undo this increase before trying the next edge.) If the flow increases for any such edge then that edge must be a bottleneck.

CS570
Analysis of Algorithms
Summer 2008
Final Exam

Name: _____

Student ID: _____

____4:00 - 5:40 Section ____6:00 – 7:40 Section

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	16	
Problem 4	16	
Problem 5	16	
Problem 6	16	
Total	100	

2 hr exam
Close book and notes

1) 20 pts
For each of the following statements, answer whether it is TRUE or FALSE, and briefly justify your answer.

a) If a connected undirected graph G has the same weights for every edge, then every spanning tree of G is a minimum spanning tree, but such a spanning tree cannot be found in linear time.

T

b) Given a flow network G and a maximum flow of G that has already been computed, one can compute a minimum cut of G in linear time.

F

c) The Ford-Fulkerson Algorithm finds a maximum flow of a unit-capacity flow network with n vertices and m edges in time O(mn) if one uses depth-first search to find an augmenting path in each iteration.

T

d) Unless P = NP, 3-SAT has no polynomial-time algorithm.

F

e) The problem of deciding whether a given flow f of a given flow network G is a maximum flow can be solved in linear time.

F

f) If a decision problem A is polynomial-time reducible to a decision problem B (i.e., $A \leq_p B$), and B is NP-complete, then A must be NP-complete.

F

g) If a decision problem B is polynomial-time reducible to a decision problem A (i.e., $B \leq_p A$), and B is NP-complete, then A must be NP-complete.

T

h) Integer max flow (where flows and capacities are integers) is polynomial time reducible to linear programming .

F

i) It has been proved that NP-complete problems cannot be solved in polynomial time.

F

j) NP is a class of problems for which we do not have polynomial time solutions.

T

2) 16 pts

Suppose that we are given a weighted undirected graph G = (V,E), a source s Є V , a sink t Є V , and a subset W of vertices V, and want to find a shortest path from s to t that must go through at least one vertex in W. Give an efficient algorithm that solves the problem by invoking any given single-source shortest path algorithm as a subroutine a small number (how many?) times. Show that your algorithm is correct.

Min (pathLength(s, w)+ pathLength(w, t)) where w belongs to W

pathLength(i, j) finds the shortest path between I and j. It can be implemented using any existing shortest path algorithm. It will be used 2*size(W) times

16 pts

Suppose that we have n files F_1, F_2, ..., F_n of lengths l_1, l_2, ... , l_n respectively, and want to concatenate them to produce a single file $F = F_1 \circ F_2 \circ ... \circ F_n$ of length $l_1 + l_2 + ... + l_n$. Suppose that the only basic operation available is one that concatenates two files. Moreover, the cost of this basic operation on two files of length l_i and l_j, is determined by a cost function $c(l_i,l_j)$ which depends only on the lengths of the two files. Design an efficient dynamic programming algorithm that given n files F_1, F_2, ..., F_n and a cost function $c(.,.)$, computes a sequence of basic operations that optimizes total cost of concatenating the n input files.

The sub structure of this problem is the time to merge a set of files S. time(S) – the time needed to merge the files and the files in S should be kept for repeated use

```
if size (S) == 2:
    # S1 and S2 are the two files in S
    time (S) = c (length(S1), length(S2))
else:
    # f is a file in S
    # S-f : take f out from S
    time (S) = min (c (length(f), total length of all other files in S)+time (S-f)
```

Return time (S) where S contains all the files

4) 16 pts
Define the language
Double-SAT = { ψ: ψ is a Boolean formula with at least 2 distinct satisfying assignmentg}.

For instance, the formula ψ: (x v y v z) ^ (x′ v y′ ∨ z′) ^ (x′ ∨ y′ ∨ z) is in Double-SAT, since the assignments (x = 1, y = 0, z = 0) and (x = 0, y = 1, z = 1) are two distinct assignments that both satisfy ψ.

Prove that Double-SAT is NP-Complete.

Various methods can be used for reducing SAT to Double-SAT. Since SAT is NP-Complete, Double-SAT is NP complete.

Following is an example for the reduction.

On input $\psi(x_1, \ldots, x_n)$:
1. Introduce a new variable y.
2. Output formula $\psi'(x_1, \ldots, x_n, y) = \psi(x_1, \ldots, x_n)$ ^ (y | not y).
If $\psi(x_1, \ldots, x_n)$ belongs to SAT, then ψ ' has at least 1 satisfying assignment, and therefore $\psi'(x_1, \ldots, x_n, y)$ has at least 2 satisfying assignments as we can satisfy the new clause (y | not y) by assigning either
y = 1 or y = 0 to the new variable y, so $\psi'(x_1, \ldots, x_n, y)$ belongs to Double-SAT. On the other hand, if $\psi(x_1, \ldots, x_n)$ does not belong to SAT, then clearly $\psi'(x_1, \ldots, x_n, y) = \psi(x_1, \ldots, x_n)$ ^ (y | not y) has no satisfying assignment either, so $\psi'(x_1, \ldots, x_n, y)$ does not belong to Double-SAT. Therefore, SAT can be reduced to Double-SAT. Since the above reduction clearly can be done in P time, Double-SAT is NP-Complete.

5) 16 pts

Let X be a set of n intervals on the real line. A subset of intervals $Y \subset X$ is called a tiling path if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The size of a tiling cover is just the number of intervals.

Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1 .. n]$ and $X_R[1 .. n]$, representing the left and right endpoints of the intervals in X.

A set of intervals. The seven shaded intervals form a tiling path

Sort the intervals from left to right by the start position, if the start positions are same, then sort it from the left to right by the end position;

```
FOR (i=1;i<=n; i++) do
        Result[i] = positive unlimited ;
        FOR (j = 1;j < i; j ++) do
                IF (interval i overlap with interval j) then
                        IF (Result[i]> result[j]+1) then
                                Result[i] = result[j]+1;
Return result [n]
```

The complexity is O(n^2)

6) 16 pts
   An edge of a flow network is called *critical* if decreasing the capacity of this edge results in a decrease in the maximum flow. Give an efficient algorithm that finds a critical edge in a network.

   Find a min cut of the network which has the same capacity as the maximum flow. Every outgoing edge from the min cut is a critical edge

CS570
Analysis of Algorithms
Spring 2008
Final Exam (B)

Name: _____

Student ID: _____

Section:__2:00-5:00 or __5:00-8:00

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	20	
Problem 5	10	
Problem 6	20	
Total	100	

Note: The exam is closed book closed notes.

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

[TRUE/FALSE]FALSE
In a flow network whose edges have capacity 1, the maximum flow always corresponds to the maximum degree of a vertex in the network**.**

[TRUE/FALSE]FALSE
If all edge capacities of a flow network are unique, then the min cut is also unique.

[TRUE/FALSE]TRUE
A minimum weight edge in a graph G must be in one minimum spanning tree of G.

[TRUE/FALSE]TRUE
When the size of the input grows, any polynomial algorithm will eventually become more efficient than any exponential one.

[TRUE/FALSE/UNKNOWN]FALSE
NP is the class of problems that are not solvable in polynomial time.

[TRUE/FALSE/UNKNOWN]FALSE
If a problem is not solvable in polynomial time, it is in the NP-Complete class.

[TRUE/FALSE/UNKNOWN]TRUE
Linear programming can be solved in polynomial time.

[TRUE/FALSE] FALSE
$10^{2 \log 4n+3} + 9^{2 \log 3n+21}$ is O(n).

[TRUE/FALSE]FALSE
f(n) = O(g(n)) implies g(n) = O(f(n)).

[TRUE/FALSE]FALSE
If X can be reduced in polynomial time to Y and Z can be reduced in polynomial time to Y , then X can be reduced in polynomial time to Z.

2) 15 pts

Suppose you are given a number x and an array A with n entries, each being a distinct number. Also it is known that the sequence of values A[1],A[2], ...,A[n] is **unimodal**. In other words for some unknown index p between 1 and n, we have A[1] < ...<A[p-1]<A[p]>A[p + 1] > ... > A[n]. (Note that A[p] holds the peak value in the array).

Give a algorithm with running time O(log n) to determine if x belongs to A, if yes the algorithm should return the index j such that A[j] = x. You should justify both the correctness of your algorithm and the running time.

Solution:

The idea is to first find out p and then break A into two separated sorted arrays, then use binary search on these two arrays to check if x is belong to A.

Let FindPeak() be the function of finding the peak in A . Then FindPeak(A [1 : n]) works as follows:

Look at A [n / 2], there are 3 cases:

(1) If A [n / 2 - 1] < A [n / 2] < A [n / 2] + 1, then the peak must come strictly after n / 2. Run FindPeak(A [n / 2 : n]).

(2) If A [n / 2 - 1] > A [n / 2] > A [n / 2] + 1, then the peak must come strictly before n / 2. Run FindPeak(A [1 : n / 2]).

(3) If A [n / 2 - 1] < A [n / 2] > A [n / 2] + 1, then the peak is A [n / 2], return n/2.

Now we know the peak index(p value). Then we can run binary search on A [1 : p] and A [p +1 : n] to see if x belong to them because they are both sorted.

In the procedure of finding p, we halve the size of the array in each recurrence. The running time T(n) satisfies T (n) = T (n/2) + O(1) . Thus T (n) = O (log n). Also both binary search has running time at most O (log n), so total running time is O (log n).

3) 15 pts
You are given two sequences a[1],...,a[m] and b[1],...,b[n]. You need to find their longest common subsequence; that is, find a subsequence $a[i_1], ..., a[i_k]$ and $b[j_1],...,b[j_k]$, such that $a[i_1] = b[j_1], ..., a[i_k] = b[j_k]$ with k as large as possible. You need to show the running time of your algorithm.

Solutions:

We use dynamic programming to solve this problem.
Notation X(i)= the prefix sequence <x(1), x(2), …, x(i)>
Let Z be a LCS of X and Y, |Z|=k.
If the last character of X and Y are different, Z is a LCS of either X(m-1) and Y(n) or X(m) and Y(n-1).
If the last character of X and Y are the same, Z(k-1) is a LCS of X(m-1) and Y(n-1)
State Transition Equation:

$$OPT(i, j) = \begin{cases} OPT(i-1, j-1)+1 & if\ x(i) = y(j) \\ \max\{OPT(i-1, j), OPT(i, j-1)\} & if\ x(i) \neq y(j) \end{cases}$$

The largest k is OPT(m, n).

Since computing each OPT(i, j) costs O(1), the running time is O(mn).

4) 20 pts
In Linear Programming, variables are allowed to be real numbers. Consider that you are restricting variables to be only integers, keeping everything else the same. This is called Integer Programming. Integer Programming is nothing but a Linear Programming with the added constraint that variables be integers. Prove that integer programming is NP-Hard

Solutions:

Proof by reduction from Satisfiability

Any SAT instance has boolean variables and clauses. Our Integer Programming problem will have twice as many variables, one for each variable and its compliment, as well as the following inequalities:
$0 \le v_i \le 1$ and $0 \le \neg v_i \le 1$
$1 \le v_i + \neg v_i \le 1$
for each clause $C = \{v_1, \neg v_2, \ldots v_i\} : v_1 + \neg v_2 + \ldots + v_i \ge 1$

1. Any SAT problem has a solution in IP.
In any SAT solution, a TRUE literal corresponds to a 1 in IP since, if the expression is SATISFIED, at least one literal per clause is TRUE, so the inequality sum is > 1.

2. Any IP solution gives a SAT solution.
Given a solution to this IP instance, all variables will be 0 or 1. Set the literals corresponding to 1 as TRUE and 0 as FALSE. No boolean variable and its complement will both be true, so it is a legal assignment with also must satisfy the clauses.

Concluding 1 and 2, Satisfiability $<_p$ Integer Programming. Since Satisfiability is NP-complete, Integer Programming is NP-hard

5) 10 pts

A carpenter makes tables and bookshelves, and he wants to determine how many tables and bookshelves he should make each week for a maximum profit. The carpenter knows that a net profit for each table is $25 and a net profit for each bookshelf is $30. The wooden material available each week is 690 units, its working hours are 120 hours per week. The estimated wood and working hours for making a table are 20 units and 5 hours respectively, while for making a bookshelf are 30 units and 4 hours. Formulate the problem using any technique we have covered in class. You do not need to solve it numerically.

Solutions:

We formulate the problem using linear programming
Suppose that the carpenter decides to make x tables and y bookshelves.

Based on the description in the problem, we want to maximize $25x + 30y$ subject to
$20x + 30y \leq 690$, and
$5x + 4y \leq 120$ where x, y are integers

6) 20 pts
A variation of the satisfiability problem is the MIN 2-SAT problem. The goal in the MIN 2-SAT problem is to find a truth assignment that minimizes the number of satisfied clauses. Give the best approximation algorithm that you can find for the problem.

Solution:

We assume that no clause contains a variable as well the complement of that variable. (Such clauses are satisfied regardless of the truth assignment used.)

We give a 2-approximation algorithm as follows:

Let I be an instance of MINSAT consisting of the clause set C_I and variable set X_I. Construct an auxiliary graph $G_I(V_I, E_I)$ corresponding to I as follows. The node set V_I is in one-to-one correspondence with the clause set C_I. For any two nodes v_i and v_j in V_I, the edge (v_i, v_j) is in E_I if and only if the corresponding clauses c_i and c_j are such that there is a variable x belongs to X_I that appears in uncomplemented form in c_i and complemented form in c_j, or vice versa.

To construct a truth assignment, we construct an approximate vertex cover V' for G_I such that $|V'|$ is at most twice that of a minimum vertex cover for G_I. Then, construct a truth assignment that causes all clauses in $V_I - V'$ to be false. (For a method to find an approximate vertex cover, please refer to section 11.4 in the textbook.)

CS 570
Analysis of Algorithms
Summer 2007
Final Exam Solutions

Kenny Daniel (kfdaniel@usc.edu)

Question 1

[FALSE] If A is linear time reducible to B ($A \leq B$), and B is NP-complete, then A must be NP-complete.

[FALSE] If B is linear time reducible to A ($B \leq A$), and B is NP-complete, then A must be NP-complete.

[TRUE] If any integer programming optimization problem can be converted in polynomial time to an equivalent linear programming problem, then $P = NP$.

[FALSE] It has been determined that NP Complete problems cannot be solved in polynomial time.

[FALSE] If $P = NP$, then there are still some NP complete problems that cannot be solved in polynomial time.

[TRUE] When we say that a problem X is NP Complete, then it means that every NP complete problem can be reduced to X.

Question 2

Suppose that there is an ordered list of n words. The length of the i-th word is w_i, that is the i-th word takes up w_i spaces. The objective is to break this ordered list of words into lines, this is called a layout. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is $L - K$. The total penalty is the maximum of the line penalties. The problem is to find a layout that minimizes the total penalty. Prove or disprove that the following greedy algorithm correctly solves this problem.

```
For i = 1 to n
    If the i-th word fits on the current line
       Place the i-th word on the current line
    else
       place the i-th word on a new line
```

Solution: This algorithm does not find the optimal layout. For a counterexample, consider the following sentence:

> *"Greedy is not a good algorithm"*

To print this sentence with width 12, the greedy algorithm gives the following layout:

```
123456789012
Greedyisnota
good
algorithm
```

This layout has total penalty 8, because of Line 2. However, this is not optimal, because the following layout has total penalty 4:

```
123456789012
Greedyis
notagood
algorithm
```

Question 3

You are hired by a consulting company which specializes in hiring/firing employees. When the company consults for an organization, it is given a list of all employees of that organization and a list of all pairs of these employees that have personality conflicts. The company then seeks to find the smallest set of employees that must be fired in order to remove all personality conflicts in the organization. Your boss gives you the assignment of designing an efficient algorithm to solve this problem. In particular, he wants your algorithm to be given as input a list of employees and a list of personality conflicts and to output the minimum number of employees that must be fired to remove all personality conflicts.

(a) Show that you can not be expected to design an efficient algorithm to solve this problem.

Solution: This problem can be shown to be NP-hard by reduction from Independent Set (described in Question 4). Suppose we are given an instance of the Independent Set problem, consisting of a graph G such that we want to find the largest set of vertices such that no two vertices share an edge.

We can express this as a company as follows: Each vertex v is represented by an employee e_v. For each edge (u, v), we say that employees e_u and e_v have personality conflicts. It can then be shown that a solution to this employee problem gives a solution to the Independent Set problem, and so the employee hiring/firing problem must be NP-hard.

(b) Your boss now wants you to design an approximation algorithm for the problem. Can you do this? If yes, provide the solution. If no, explain why.

Solution: As in part (a), Independent Set can be reduced to the employee problem, and Independent Set cannot be approximated if $P \neq NP$.

Question 4

We read about the INDEPENDENT SET problem in class. Here is the statement for your reference. Given a graph $G(V, E)$, find an independent set S as large as possible. (An independent set S is a subset of V such that no two nodes in S share an edge). The goal of this problem is to find an integer programming formulation of INDEPENDENT SET.

(a) What are the decision variables here

Each vertex v is represented by a variable x_v which is either 0 or 1.

(b) What is the objective function of the Integer program

The objective is to maximize the sum: $\max \sum_{v \in S} x_v$.

(c) What are the constraints in the Integer Program

For each edge (u, v), we add one constraint such that $x_u + x_v \leq 1$.

Question 5

In the first midterm, you were given a greedy algorithm for the coin change problem. The greedy algorithm gave the optimal solution for one denomination of coins but did not give the optimal solution for all denominations of coins. Here, you are required to give an algorithm that will optimally solve the coin change problem for any denomination of coins. Formally, If you are given k denominations of coins of values $d_1, d_2, \ldots, d_k$ where each d_i is an integer and given an integer amount X, give an algorithm that gives you the smallest number of coins of the denominations such that the sum of the coins is X.

Solution:

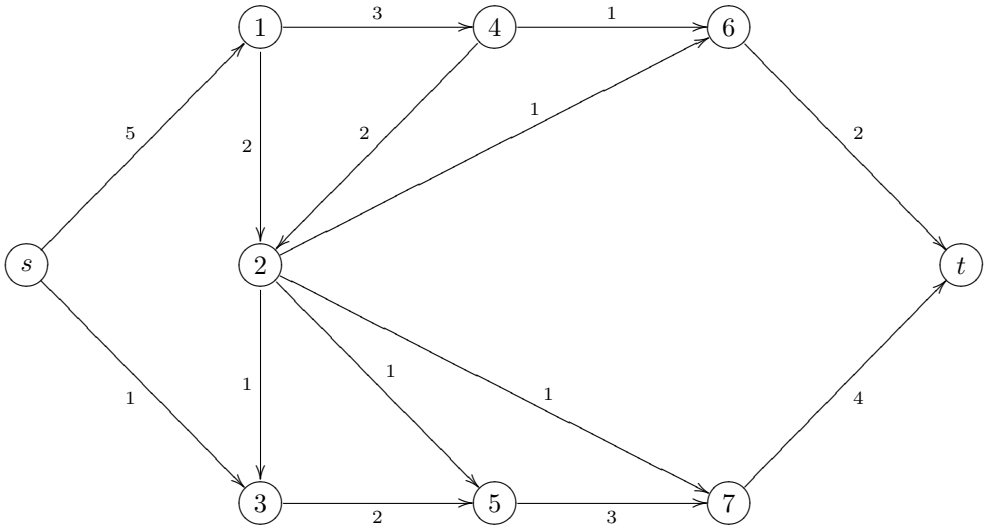
```
int change(X) {
    if (X < 0) return ∞;
    else if (X = 0) return 0;
    else return min_{i=1..k} change(X - di);
}
```

Question 6

(a) Give a maximum s-t flow for the following graph, by writing the flow f_e above each edge e. The printed numbers are the capacities.

(b) Prove that your given flow is indeed a max-flow.

Solution: The figure below shows a flow of 6. This is a max flow, since we the edges going out of s form a cut of weight $5 + 1 = 6$, and so 6 is the min cut / max flow.


1
3
4
1
6
5
2
2
1
2
s
2
t
1
1
1
1
4
3
2
5
3
7