CS570
Analysis of Algorithms
Spring 2012
Exam II

Name: _____

Student ID: _____

____12:30 PM Section ____2:00 PM Section

	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 200 words, anything beyond 200 words will not be considered.

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

[FALSE] If f is a maximum s-t flow in a flow network G, then for all edges e out of s, we have $f(e) = c_e$.

[FALSE] Let (A, B) be a minimum s-t cut with respect to the capacities $\{ c_e : e \in E\}$. Now suppose we add 1 to every capacity, then (A,B) is still a minimum s-t cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

[TRUE] If we multiply each edge with the same positive multiple "f", the max flow also gets multiplied by the same factor.

[TRUE] If a problem can be solved by dynamic programming, then it can always be solved by exhaustive search.

[TRUE] Sequence alignment problem between sequence X and sequence Y can be solved using dynamic programming in O(mn) when |X| = m and |Y| = n.

[FALSE] Bellman-Ford algorithm cannot solve the shortest path problem in graphs with negative cost edges in polynomial time.

[FALSE] If a dynamic programming solution is set up correctly, i.e. the recurrence equation is correct and the subproblems are always smaller than the original problem, then the resulting algorithm will always find the optimal solution in polynomial time.

[TRUE] In the Ford–Fulkerson algorithm, choice of augmenting paths can affect the number of iterations.

[FALSE] If in a flow network all edge capacities are distinct, then there exists a unique min-cut.

[TRUE] Max flow in a flow network can be found in polynomial time.

2) 20 pts

We are running a 24-hour restaurant, and need to make sure we have at least n_i people on duty from hour i to $i + 1$. Each of our employees is only willing to work particular hours of the day (for example, some are students and cannot work until evening, others may refuse late night hours). Additionally, each employee j needs to be assigned at least x_j and at most y_j hours per day. Design an algorithm to determine whether it is possible to keep the restaurant running all the time.

Solution:

We set up a bipartite matching as follows. There is a vertex e_i for each employee and h_j for each hour. A unit of flow from an employee to an hour represents that the employee is working that hour. To ensure that each hour has at least n_i people, the edge from h_i to sink T has minimum capacity n_i. The edge from source S to employee e_j has minimum capacity x_j and maximum capacity y_j. Furthermore, there is an edge (e_i, h_j) if and only if e_i can work hour h_j. A maximum flow of value at least Σn_i says that it is possible to keep the restaurant running all the time.

Here is the general checklist for a network flow transformation solutions:

1 – Identify the problem can be solved using network flow? (3 pts)
2 – What are the mapping to the corresponding components in the network flow graph (nodes (4 pts), edges (3 pts), capacities (3 pts))?
3 – How is the answer to the network flow graph related to the answer of the problem? (3 pts)
4 – Proof of correctness (2 pts)
5 – What is the running time (and why?) (2 pts)

3) 20 pts

You are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee has a value v_j, representing how enjoyable their presence would be at the party. Your goal is to determine the best guest list.

a. Suppose we always greedily choose the employee of max value (assuming we haven't already chosen one his immediate superiors or subordinates). Give a counter-example to show this greedy criteria does not work.
b. Give an algorithm that determines the best guest list and analyze its running time.

Solution:

a. v(a)=v(c)=2, v(b)=3. a is b's superior, and b is c's superior.

b. Define *OPT(x)* to be the optimal value for the subtree rooted at *x*.

If *x* is a leaf node, $OPT(x) = v_x$.

OPT(null) = 0.

$OPT(x) = max(v_x + \Sigma_g(OPT(g)$: ***g* is a grandchild of *x*)**, $\Sigma_c(OPT(c)$: ***c* is a child of *x*))**.

To produce the actual guest list given the array you generated, starting from the root node *r*, determine if the value of *r* comes from including *r* (in which case it will equal to $v_x + \Sigma_g(OPT(g)$: ***g* is a grandchild of *x*)). **If it does, then include *r* in the answer, and recursively repeat the process for each of *r*'s grandchildren. If not, then do not include *r*, and recursively repeat for each of *r*'s children.**

To analyze the running time, since each node has a single parent and grandparent, each node will be examined twice. Thus we can fill in the entire array in linear time.

4) 20 pts

An Edge Cover on a graph $G = (V, E)$ is a set of edges $X \subseteq E$ such that every vertex in V is adjacent to an edge in X. Give an algorithm that finds the smallest-size Edge Cover on a **bipartite** graph.

Solution:
We can utilize Network Flow Transformation to solve this problem.

To transform an arbitrary instance of the Bipartite Edge Cover problem into an instance of Network Flow, add a source, with a directed edge to all nodes in L. Add a sink, with a directed edge from all nodes in R. Add direction to all edges from L to R. Give each edge weight 1.

To extract an answer to the Bipartite Edge Cover instance from your solution to Network Flow, each edge from L to R which has flow will be in the Edge Cover solution. For each remaining vertex, choose an arbitrary edge adjacent to it.

Proof of correctness:
Each edge in a valid solution covers either 1 or 2 vertices. By solving max flow, we are maximizing the number of edges which cover 2 vertices. We can then arbitrarily choose the remaining edges to cover each of the remaining vertices.

Complexity analysis:
The flow network has $|V| + |E|$ edges and the possible maximum flow value is $O(|V|)$, therefore, applying Ford-Fulkerson costs $O(|V| (|V|+|E|)$).

5) 20 pts

We're running a lab with C computers and P people working for us. We have a set of possible projects 1, 2, …, n each of which requires some number of people $p_1, p_2, \ldots, p_n$ and some number of computers $c_1, c_2, \ldots, c_n$. Design a dynamic programming algorithm to determine the maximum number of projects we can work on simultaneously. Analyze the running time.

Solution:

Define $OPT(i, p, c)$ to be the maximum number of projects $i, i+1, \ldots, n$ we can work on with p people and c computers.

$$OPT(n+1, p, c) = OPT(i, 0, c) = OPT(i, p, 0) = 0.$$

For any $x < 0$, $$OPT(i, x, c) = OPT(i, p, x) = -\infty.$$

$$OPT(i, p; c) = max(OPT(i+1, p, c), 1 + OPT(i+1, p - p_i, c - c_i))$$

Fill the table in as follows:

for i=n+1 to 1
for p=0 to P
for c=0 to C
Calculate OPT[i,p,c]

The answer is stored in OPT[1,P,C]. Each entry can be calculated in constant time, so the running time is $\Theta(nPC)$.

Here is the general checklist for a dynamic programming solutions:

1 – What exactly are you calculating in your recursive formula? That is, what does OPT(x,y) mean? (2 pts)
2 – Recursive formula (10 pts)
3 – Recursive formula base cases (2 pts)
4 – What order is the array filled in? (3 pts)
5 – What is the running time (and why?) (3 pts)