CS570
Analysis of Algorithms
Fall 2010
Exam II

Name: _____

Student ID: _____

____Monday ____Friday ____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.

[FALSE]
If you have non integer edge capacities, then you cannot have an integer max flow.

[TRUE]
The maximum value of an s-t flow is equal to the minimum capacity of an s-t cut in the network.

[FALSE]
For any graph there always exists an edge such that increasing the capacity on that edge will increase the maximum flow from source s to sink t. (Assume that there is at least one path in the graph from source s to sink t.)

[FALSE]
If a problem can be solved using both the greedy method and dynamic programming, greedy will always give you a lower time complexity.

[FALSE]

There is a feasible circulation with demands $\{d_v\}$ if $\sum_v d_v = 0$.

[FALSE]
The best time complexity to solve the max flow problem is O(Cm) where C is the total capacity of the edges leaving the source and m is the number of edges in the network.

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

[FALSE]
0-1 knapsack problem can be solved using dynamic programming in polynomial time.

[TRUE]
Bellman-Ford algorithm solves 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.

2) 20 pts

You are given an n-by-n grid, where each square (i, j) contains $c(i, j)$ gold coins. Assume that $c(i, j) \geq 0$ for all squares. You must start in the upper-left corner and end in the lower-right corner, and at each step you can only travel one square down or right. When you visit any square, including your starting or ending square, you may collect all of the coins on that square. Give an algorithm to find the maximum number of coins you can collect if you follow the optimal path. Analyze the complexity of your solution.

We will solve the following subproblems: let $dp[i, j]$ be the maximum number of coins that it is possible to collect while ending at (i, j).

We have the following recurrence: $dp[i, j] = c(i, j) + max(dp[i - 1, j], dp[i, j - 1])$

We also have the base case that when either $i = 0$ or $j = 0$, $dp[i, j] = c(i, j)$.

There are n^2 subproblems, and each takes $O(1)$ time to solve (because there are only two subproblems to recurse on). Thus, the running time is $O(n^2)$.

3) 20 pts

King Arthur's court had n knights. He ruled over m counties. Each knight i had a quota q_i of the number of counties he could oversee. Each county j, in turn, produced a set S_j of the knights that it would be willing to be overseen by. The King sets up Merlin the task of computing an assignment of counties to the knights so that no knight would exceed his quota, while every county j is overseen by a knight from its set S_j. Show how Merlin can employ the Max-Flow algorithm to compute the assignments. Provide proof of correctness and describe the running time of your algorithm. (You may express your running time using function $F(v, e)$, where $F(v, e)$ denotes the running time of the Max-Flow algorithm on a network with v vertices and e edges.)

We make a graph with $n+m+2$ vertices, n vertices $k_1,\ldots,k_n$ corresponding to the knights, m vertices $c_1,\ldots,c_m$ corresponding to the counties, and two special vertices s and t.

We put an edge from s to k_i with capacity q_i. We put an edge from k_i to c_j with capacity 1 if county j is willing to be ruled by knight i. We put an edge of capacity 1 from c_j to t.

We now find a maximum flow in this graph. If the flow has value m, then there is a way to assign knight to all counties. Since this flow is integral, it will pick one incoming edge for each county c_j to have flow of 1. If this edge comes from knight k_i, then county j is ruled by knight i.

The running time of this algorithm is $F(n+m+2, \sum_j |S_j|+n+m)$.

4) 20 pts

You are going on a cross country trip starting from Santa Monica, west end of I-10 freeway, ending at Jacksonville, Florida, east end of I-10 freeway. As a challenge, you restrict yourself to only drive on I-10 and only on one direction, in other words towards Jacksonville. For your trip you will be using rental cars, which you can rent and drop off at certain cities along I-10. Let's say you have n such cities on I-10, and assume cities are numbered as increasing integers from 1 to n, Santa Monica being 1 and Jacksonville being n. The cost of rental from a city i to city j (>i) is known, C[i,j]. But, it can happen that cost of renting from city i to j is higher than the total costs of a series of shorter rentals.

(A) Give a dynamic programming algorithm to determine the minimum cost of a trip from city 1 to n.

(B) Analyze running time in terms of n.

Denote OPT[i] to be the rental cost for the optimal solution to go from city i to city n for $1 \le i \le n$. Then the solution we are looking for is OPT[1] since objective is to go from city 1 to n. Therefore OPT[i] can be recursively defined as follows.

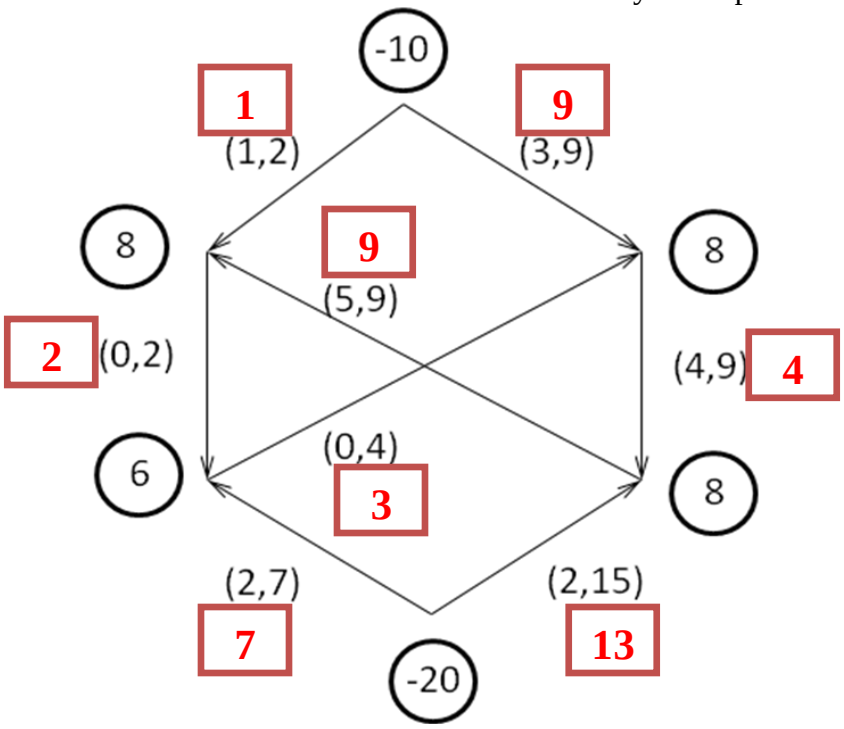
$$OPT[i] = \begin{cases} 0 & if \quad i = n \\ \min_{i \le j \le n} \left(C[i,j] + OPT[j] \right) & otherwise \end{cases}$$

Proof: The car must be rented at the starting city i and then dropped off at another city among i+1,…,n. In the recurrence we try all possibilities with j being the city where the car is next returned. Furthermore, since C[i,j] is independent from how the subproblem of going from city j,…,n is solved, we have the optimal substructure property.

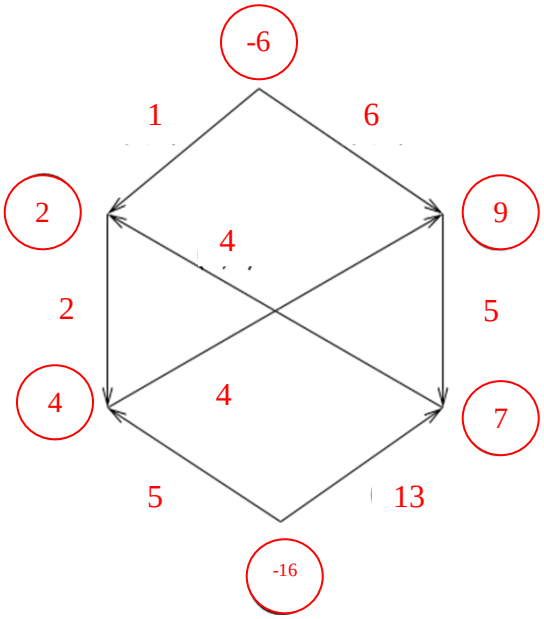
For the time complexity, there are n subproblems to be solved each of which takes linear time, O(n). These subproblems can be computed in the order OPT[n], OPT[n-1],…,OPT[1], in a linear fashion. Hence the overall time complexity is $O(n^2)$.

5) 20 pts

Solve the following feasible circulation problem. Determine if a feasible circulation exists or not. If it does, show the feasible circulation. If it does not, show the cut in the network that is the bottleneck. Show all your steps.



First we eliminate the lower bound from each edge:



Then, we attach a super-source s^* to each node with negative demand, and a super-sink t^* to each node with positive demand. The capacities of the edges attached accordingly correspond to the demand of the nodes.

We then seek a maximum s^*-t^* flow. The value of the maximum flow we can get is exactly the summation of positive demands. That is, we have a feasible circulation with the flow values inside boxes shown as above.