CS570
Analysis of Algorithms
Summer 2010
Exam II

Name: ________________________

Student ID: ____________________

____Check if DEN student

	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

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

[[✓ TRUE]/FALSE]
Given a weighted, directed graph with no negative-weight cycles, the shortest path between every pair of vertices can be determined in worst-case time O(V^3).

[TRUE/[✓ FALSE]]
Any problem that can be solved using dynamic programming has a polynomial time worst case time complexity with respect to its input size.

[TRUE/[✓ FALSE]]
Ford-Fulkerson algorithm will always terminate as long as the flow network G has edges with strictly positive capacities.

[TRUE/[✓ FALSE]]
For any graph G with edge capacities and vertices ***s*** and ***t***, there always exists an edge such that increasing the capacity on that edge will increase the maximum flow from s to t. (Assume there is at least one path in the graph from ***s*** to ***t***)

[[✓ TRUE]/FALSE]
For any network and any maximum flow on this network, there always exists an edge such that decreasing the capacity on that edge with decrease the network's max flow.

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

[TRUE/[✓ FALSE]]
For edge any edge e that is part of the minimum cut in G, if we increase the capacity of that edge by any integer k>1, then that edge will no longer be part of the minimum cut.

[TRUE/[✓ FALSE]]
0/1 knapsack problem can be solved in polynomial time using dynamic programming

[[✓ TRUE]/FALSE]
One can find the value of the optimal solution AND the optimal solution to the sequence alignment problem using only O(n) memory where n is the length of the smaller sequence.

[TRUE/[✓ FALSE]]
In a flow network if all edge capacities are irrational (not rational numbers) then the max flow is irrational.

2) 20 pts
You are trying to help a supermarket to distribute n types of coupons to a total of m customers. However, each type of coupon is of interest to only a subset of m customers . No coupon should be sent to a customer who is not interested in it. On the other hand, each customer can receive at most k coupons. No customer can receive duplicated coupons.

Design an efficient network flow based algorithm that given the list of customers of interest for each of the n coupons, and the upper bound k, will determine what coupons should be sent to each customer so as to maximize the total number of coupons sent. Show the running time (using parameters given above). You can assume that m>>n.

Answer:
Construct a flow graph as follows. Let s and t be the source and sink vertices respectively. For each customer, introduce a vertex. Call these b1,b2,...,bm. Likewise, for each coupon introduce a vertex. Call these c1,c2,..cn.

The edges as defined as follows. There is an edge from s to each of the coupons and it has infinite capacity. That is c(s,ci)=infty, for 1<=i<=n.
There is an edge from ci to bj if and only if the customer bj is in interested in the coupon ci. Let its capacity be 1.
There is an edge from every user to the sink t of capacity k. That is c(bj,t)=k, for 1<=j<=m.

Algorithm: Compute the(a) Max Flow of the graph. If f(ci,bj)=1, then send a ci coupon to customer bj.

The running time can be analyzed as follows:
There is a total of O(m+n) nodes and O(mn+m+n) edges in the network. To the running time is O(V^2*E)=O((m+n)^2*mn). It is reasonable to assume m>>n. Thus we can say running time is O(m^3).

Correctness Proof. We show this in two steps. First, we show that any flow translates to a valid solution to the coupon distribution problem and vice versa. Then we show that maximizing flow maximizes the number of coupons distributed.

Assume that we have an integral flow f assigned to the graph. Send f(ci,bj) number of ci coupon to customer bj.

The flow f(ci,bj) can be greater than zero only if the there is an edge between ci and bj. Thus customers are only sent coupons they are interested in. Further f(ci,bj)<=1, thus no one is sent any duplicate coupons. The flow flowing into bj corresponds to the number of coupons received by the customer bj. From the flow conservation constraint at bj, this equals the flow going out of bj which is bounded by k. Thus no customer received more than k coupons. From the conservation constraint at ci, the flow assigned to the edge (s,ci) is the total number of coupons of type ci sent. We have thus met all the constraints of the coupon distribution problem and the flow assignments translates to a valid solution.

Similarly, we can show that any valid solution to the distribution problem can be mapped to a valid flow for the graph. The flow out of the source s is nothing but the total number of coupons distributed. Thus we indeed maximize the total number of coupons.

3) 20 pts

In the first midterm you are asked to solve the following problem:
Given a array A of n positive numbers, find indices i<j such that A[j]/A[i] is maximized.
Note that you cannot reorder the elements in the array.
Now solve the same problem using dynamic programming in O(n) time.

Answer:

```
Let OPT[i] be the max division with numerator A[i].
OPT[1]=1;
OPT[2]=max{OPT[1], A[2]/A[1]}
For i=3:n-1
  OPT[i+1]=max{OPT[i]*A[i+1]/a[i], 1}   (*)
endFor
Find max{OPT[]} to get j.
Find min(A[1..j-1]) to get i.
```

Note that at stage i+1, there are 2 choices:
1. Choose A[i+1] as the numerator
2. Start a new sequence
This constructs the recurrence function (*)
Running time is O(n)+O(n)+O(n)=O(n).

Alternative solution:
Let OPT[i] be the max division up to i. Then we have
OPT[i]=max{OPT[i-1], A[i]/min[i]}
Where min[i] is the minimum element in A[1..i-1].
min[i] can be updated as min[i]=minimum{min[i-1], A[i-1]}
Use a for loop to calculate OPT[1] to OPT[n]. The max division is OPT[n].
Then trace back the array to find i and j.
The running time is O(n) because only one for loop is used and in each iteration the update of OPT[i] and min[i] takes only O(1).

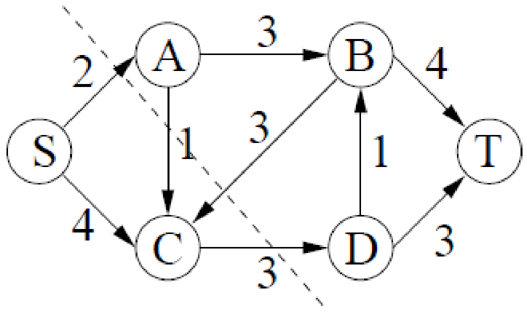
4) 20 pts

In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.

a) Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

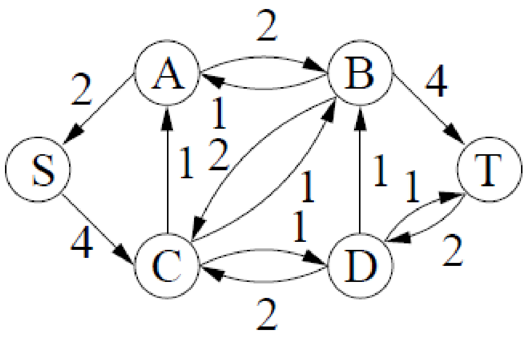
b) List all of the augmenting paths that could be chosen for the third augmentation step.

c) What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent the minimum cut.



a.(8 points) Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

ANSWER.



b. (8 points) List all of the augmenting paths that could be chosen for the third augmentation step.

ANSWER.

S-C-A-B-T, S-C-B-T, S-C-D-B-T, and S-C-D-T.

(each 2 points, if more than 4, minus 2 points)

c. (4 points) What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent the minimum cut.

ANSWER.

5 (2 points)/ See top figure above for the minimum cut.(2 points)

5) 20 pts
In class, we saw how the Bellman-Ford algorithm finds shortest paths in a directed graph with no negative cycles. Use Bellman-Ford as your basis and provide a solution to the following problem:
Given a directed graph, determine if it contains a negative cycle. If it does, output a negative cycle.

Answer:
Let G=(V,E) be the graph and let n be the number of vertices. Let d(i,u,v) denote the cost of the shortest path from u to v using at most i edges. Let P(i,u,v) be the corresponding path.

Given a vertex u, the Bellman-Ford algorithm can be used to compute d(i,u,v) and P(i,u,v) for all vertices v and 1<=i<=n. (In the version described in class, we stopped at i=n-1, but extending it to n is straightforward). The running time is O(|V||E|).

Observe that d(n,u,u)< 0 if an only if the vertex u is a part of a negtive weight cycle(not necessarily simple).

Thus we have the following algorithm to detect and find negative cycles.

For every vertex u, compute d(n,u,u) using Bellman-Ford. If d(n,u,u) >=0 for all u, then the graph has no negative weight cycles.

If there exists a vertex u such that d(n,u,u)<0, then the graph has a negative weight cycle. Output the corresponding path P(n,u,u) as a negative cycle.

The total running time is O(|V|^2. |E|)

note: There are faster ways of doing this. Instead of computing the distances d(i,u,_) from each vertex u, we can compute the shortest paths between all pairs in one go. For instance using Floyd-Warshall algorithm (based on dynamic programming, very similar to Bellmann-ford) in O(|V|^3). There are also tricks we can use so that the Bellmann-Ford does not need to run completely every vertex.

However, these speed ups are not required and any algorithm that runs in polynomial time will get full credit.