CS570
Analysis of Algorithms
Summer 2017
Exam II

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

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

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

[TRUE/FALSE]
In the Ford-Fulkerson algorithm, choice of augmenting paths can affect the min cut.

[TRUE/FALSE]
A dynamic programming algorithm with n^2 unique subproblems, could have a time complexity that is better than $O(n^2)$.

[TRUE/FALSE]
A dynamic programming algorithm with n^2 unique subproblems, could have a memory requirement that is better than $O(n^2)$.

[TRUE/FALSE]
A dynamic programming solution when implemented using iteration will always run faster compared with the same dynamic programming solution implemented using recursion and memoization.

[TRUE/FALSE]
It is possible for a dynamic programming algorithm to have exponential running time.

[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]
Bellman-Ford algorithm is guaranteed to work on directed graphs that don't contain any negative weight cycles.

[TRUE/FALSE]
Bellman-Ford algorithm is not guaranteed to work on all undirected graphs.

[TRUE/FALSE]
If we replace each directed edge in a flow network (except for edges incident on *s* or *t*) with two directed edges in opposite directions with the same capacity and connecting the same vertices, then the value of the maximum flow remains unchanged.

[TRUE/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.

2) 20 pts

A message containing only letters is encoded in the following ways:

'A' → 1
'B' → 2
… …
'Z' → 26

Design an algorithm to calculate the number of ways to decode a given message M. For example, for the message M='12', we can decode it as 'AB' or 'L', so there are 2 ways to decode it.

Maintain an array dp of size n+1, where dp[i] represents the number of ways decoding the first i numbers in the given message.

The solution to our problem will be dp[n] where n=number of input numbers

We define a substring is valid as following:

If the string starts with a "0", it is not valid digit.
If the string is larger than 0 and smaller than 27, it is a valid digit.
If the sting is larger than 26, it is not a valid digit.

We use M[i-1,i] to denote the substring only containing the ith letter and M[i-2,i] to denote the substring containing (i-1)th letter and ith letter.

The base case is dp[0] = 1 as there is only one way to decode empty message. Set dp[1] = 2 if M[-1,1] is valid, dp[1] = 0 if neither M[0,1] nor M[-1,1] is valid, dp[1] = 1 if only M[0,1] is valid

We can calculate dp[i] by considering following cases:

Case 1: both substring M[i-1,i] and M[i-2,i] are valid
Case 2: only M[i-1,i] is valid
Case 3: only M[i-2,i] is valid

In case 1, dp[i] = dp[i-1] + dp[i-2]
In case 2, dp[i] = dp[i-1]
In case 3, dp[i] = dp[i-2]

Below is the pseudocode:

Input: message M
Output: number of ways to decode M
n = length of M
Initialize dp as an array of length (n+1) and all elements are 0
dp[0] = 1
if M[-1,1] is valid::

```
        dp[1] = 2
elseif M[0,1] is not valid or M[-1,1] is not valid:
        dp[1] = 0
else:
        dp[1] = 1
endif
for( i = 2, I <=n, i++)
        if M[i-1,i] is a valid substring:
                dp[i] += dp[i-1]
        endif
        if M[i-2,i] is a valid substring:
                dp[i] += dp[i-2]
        endif
endfor
return dp[n]
```

Grading:

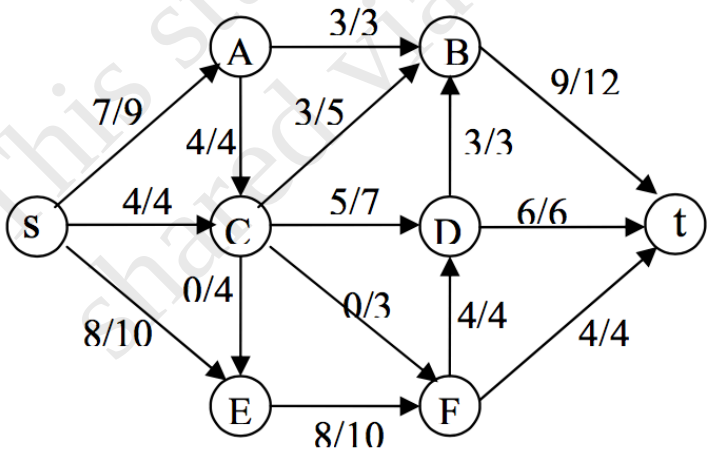
5 points for the meaning of dp[i] and pointing out the solution is dp[n]

6 points for the two initial states, 3 points for each

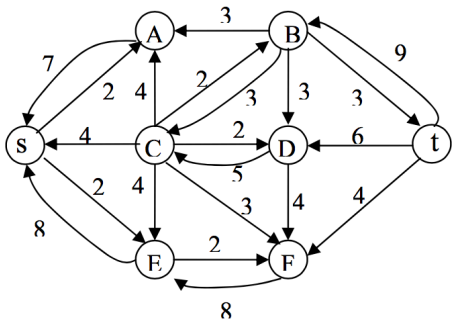
9 points for the analysis of three different cases when calculating dp[i], 3 points for each

3) 20 pts

Consider the flow network in the following figure with the indicated s-t flow f: every edge is labeled **a**/**b** where **a** is the flow and **b** is the capacity of the edge.



a) Draw the residual network with respect to the flow f. (5 pts)



Grading:
Each wrong edge (-0.5 points)

b) Is f a valid flow? Explain why. (5 pts)

Yes,
1) for any node in this graph, the out degree equals the in degree and (2.5 points)
2) the flow doesn't exceed any edge capacities. (2.5 points)

c) Is f a maximum flow in G? Explain why. (5 pts)

Yes. By following Ford-Fullkerson algorithm, this is the max-flow in G since there is no more s-t paths in the residual graph.

d) Give a minimum s-t cut of the (original) network by specifying below the two sets S and T of the corresponding partition of the nodes, where s is in S, and t is in T. (5 pts)

The residual network does not contain any path from s to t, hence f is a maximum flow. A minimum cut can be derived by letting S be the set of nodes reachable from s in the residual network and T the set of remaining nodes:
S = { s, A, E, F } T = { t, B, C, D }

4) 20 pts

Assume we have N workers. Each worker is assigned to work at one of M factories. For each of the M factories, they will produce a different profit depending on how many workers are assigned to that factory. We will denote the profits of factory i with j workers by $P(i,j)$. Develop a dynamic programming solution to find the assignment of workers to factories that produces the maximum profit, by completing the following steps

a) Describe a unique subproblem (in English) (5 pts)

Assignment of j workers to the first m factories that produces the maximum profit

Grading:
- Didn't mention maximum profit (-2 point)
- Didn't mention how many workers (-1 point)
- Didn't mention how many factories (-1 point)

b) Give a recurrence formula to compute the value of an optimal solution. (5 pts)

Let $B(m, j)$ denotes the maximum profit of the assignment of j workers to the first m factories.
An optimal solution of j workers to the first m factories assigns some number of workers, k, to factory m, and must assign j-k workers with maximal profit to the first m - 1 factories, which, by induction, has cost $B(m-1, j-k)$. Thus, we have

$$B(m, j) = \max_{0 \le k \le j} P(i, k) + B(m-1, j-k).$$

Grading:
- Wrong recurrence formula in dynamic programming recurrence formula form (-2 points)
- Formula not in dynamic programming recurrence formula form (-5 points)

c) Write pseudo-code to compute the value of an optimal solution using iteration. (5 pts)

```
B(0,0) = 0
For m = 1, 2, …, M
        For j = 1, 2, …, N
                B(m, j)  = max_{0≤k≤j} P(i, k) + B(m-1, j-k)
        Endfor
Endfor
```

Grading:
- No initialization (-1 point)
- Used only one for-loop (-2 points)
- Didn't include recurrence formula (-3 points)

d) What is the complexity of the code in section c? (5 pts)

The time to compute each value $B(m, j)$ is $O(N)$ and since there are MN subproblems, the total runtime is $O(N^2M)$.

Grading:
- Wrong time to compute each value $B(m, j)$ (-2 points)
- Wrong number of subproblems (-2 points)
- Missing O( ) notation (- 1 point)

5) 20 pts
Formulate the following optimization problem as a network flow problem:
There are m houses $H_1, \ldots, H_m$ for sale, n buyers $B_1, \ldots, B_n$ looking to buy a house, and k real-estate agents $A_1, \ldots, A_k$. Each agent A_i knows a subset $h_i \subseteq \{H_1, \ldots, H_m\}$ of the properties and a subset $b_i \subseteq \{B_1, \ldots, B_n\}$ of the potential buyers. Due to workload constraints, each agent A_i can close at most a_i real-estate transactions. What is the maximum total number of transactions that can be arranged?

Construct a network with the nodes s (source), $H_1, \ldots, H_m, A_1, \ldots, A_k, A'_1, \ldots, A'_k, B_1, \ldots, B_n$, and t (sink). Note that there are two nodes for each agent. There is an edge of capacity 1 from s to each H_j node. There is an edge of capacity 1 from H_j to A_i if and only if $H_j \in h_i$. There is an edge from A_i to A'_i of capacity a_i. There is an edge of capacity 1 from A'_i to B_j if and only if $B_j \in b_i$. There is an edge of capacity 1 from each B_j node to t.

Grading:
- Wrong capacities (-1 point for each group of edges)
- Not mentioning which houses (/buyers) get connected to which agents (-1 point)
- If your three layers are H—A—B: This does not limit agent A_i being in at most a_i transactions (even if you add a demand of a_i to node A_i) (-10 points)

- If your three layers are A—H—B: This does not limit a house H_j being assigned to only one buyer (-10 points)
- If your three layers are A—B—H: This does not limit a buyer B_j being assigned to only one house (-10 points)