CS570
Analysis of Algorithms
Spring 2017
Exam II

Name: _____
Student ID: _____
Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	15	
Problem 5	15	
Problem 6	15	
Problem 7	15	
Total		

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

[/FALSE] The value of the max flow is not enough and we need the complete flow
Given the value of max flow, we can find a min-cut in linear time.

[/FALSE]
The Ford-Fulkerson algorithm can compute the maximum flow in polynomial time.

[/FALSE]
A network with unique maximum flow has a unique min-cut.

[TRUE/]
If all of the edge capacities in a graph are an integer multiple of 3, then the value of the maximum flow will be a multiple of 3.

[/FALSE]
The Floyd-Warshall algorithm always fails to find the shortest path between two nodes in a graph with a negative cycle.

[/FALSE] 0/1 knapsack does not have a polynomial time dynamic programming solution—not even a weakly polynomial solution. 0/1 knapsack has a pseudopolynomial time dynamic programming solution which is basically an exponential time solution (with respect to its input size)
0/1 knapsack problem can be solved using dynamic programming in polynomial time, but not **strongly** polynomial time.

[/FALSE]
If a dynamic programming algorithm has n subproblems, then its running time complexity is $O(n)$.

[/FALSE]
The Travelling Salesman problem can be solved using dynamic programming in polynomial time

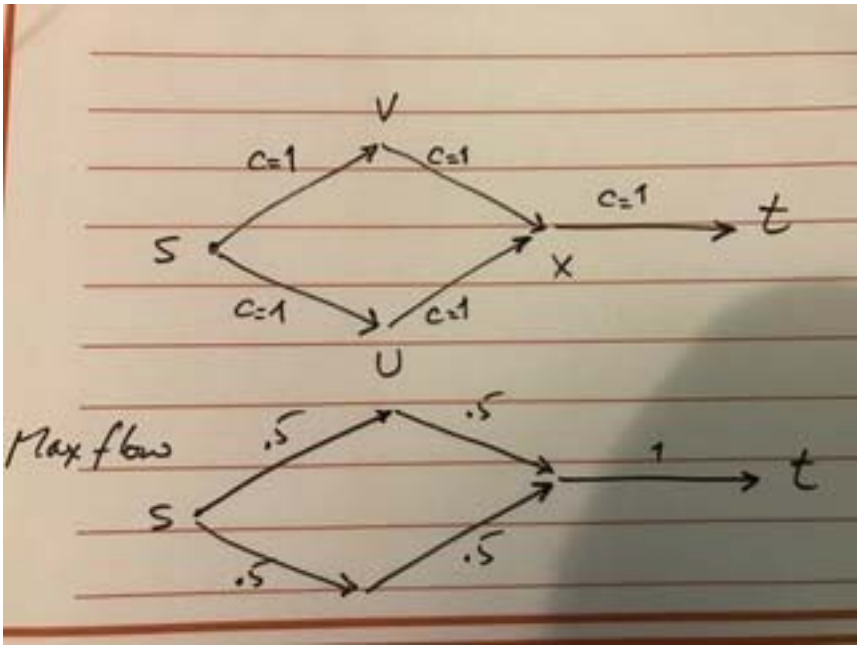
[/FALSE]
If flow in a network has a cycle, this flow is not a valid flow.

[FALSE/] We can't do this for *undirected* graphs
We can use the Bellman-Ford algorithm for undirected graph with negative edge weights.

2) 10 pts

Given $N(G(V,E),s,t,c)$, a flow network with source s, sink t, and positive integer edge capacities *c(e)* for every *e* □ *E*. Prove or disprove the following statement:
A maximum flow in an integer capacity graph must have integral (integer) flow on each edge.

Counterexample:



Grading:
A correct counter example gets full 10 points, otherwise 0 points is given.

Common Mistake:
0 is an integer so it will not count as a non-integral flow.

3) 10 pts

The subset sum problem is defined as follows: Given a set of n positive integers $S=\{a_1, a_2, ..., a_n\}$ and a target value T, does there exist a subset of S such that the sum of the elements in the subset is equal to T? Let's define a boolean matrix M where M(i, j) is true if there exists a subset of $\{a_1, a_2, ..., a_i\}$ whose sum equals j. Which one of the following recurrences is valid? (circle one)

1) $M(i, j) = M(i-1, j)$ Ú $M(i, j- a_i)$

2) $M(i, j) = M(i-1, j)$ Ú $M(i-1, j- a_i)$ This is the correct solution

3) $M(i, j) = M(i-1, j-1)$ Ú $M(i-1, j)$

4) $M(i, j) = M(i, j- a_i)$ Ú $M(i-1, j - a_i)$

Solution:

For computing M(i, j) you either include a_i or you don't. If you include a_i then we have M(i, j) = M(i-1, j-a_i). If you don't include a_i, then M(i, j) = M(i-1, j).

Grading:

Recurrence 2 gets full 10 points. Recurrences 1 and 4 get 5 points since they have one of the two cases correct. Recurrence 3 gets 0 points.

4) 15 pts

This problem involves partitioning a given input string into disjoint substrings in the cheapest way. Let $x_1x_2 \ldots x_n$ be a string, where particular value of x_i does not matter. Let C(i,j) (for $i \leq j$) be the given precomputed cost of each substring $x_i \ldots x_j$. A partition is a decomposition of a string into disjoint substrings. The cost of a partition is the sum of costs of substrings. The goal is to find the min-cost partition.
An example. Let "ab" be a given string. This string can be partitioning in two different ways, such as, "a", "b", and "ab" with the following costs C(1,1) + C(2,2) and C(1,2) respectively.

a) Define (in plain English) subproblems to be solved. (5 pts)

Let subproblem OPT(j) be the min-cost partition $x_1x_2 \ldots x_j$

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

$OPT(j) = \min_{1 \leq k < j} (OPT(k-1) + C(k, j))$
$OPT(0) = 0$

c) Compute the runtime of the algorithm. (3 pts)

$\Theta(n^2)$.

Common mistake:

If you write the recurrence based on two parameters of i,j for the corresponding subsequence xi…xj like opt(i,j) you might lose points from 4 to 7 points in part b, depending on your solution. In this case, you probably came up with complexity of $O(n^3)$ instead of $O(n^2)$ which is not optimum compared to the right solution and you might lose other points (2 to 3 points) in part c.

5)

You have two rooms to rent out. There are n customers interested in renting the rooms. The i^{th} customer wishes to rent one room (either room you have) for d[i] days and is willing to pay bid[i] for the entire stay. Customer requests are non-negotiable in that they would not be willing to rent for a shorter or longer duration. Devise a dynamic programming algorithm to determine the maximum profit that you can make from the customers over a period of D days.

Let OPT(d1, d2, i) be the maximum profit obtainable with d1 remaining days for room 1 and d2 remaining days for room 2 using the first i customers.

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

$$OPT(d1, d2, i) = \max(bid[i] + OPT(d1 - d[i], d2, i - 1),\ bid[i] + OPT(d1, d2 - d[i], i - 1),\ OPT(d1, d2, i - 1))$$

Initial conditions
$OPT(d1,d2,0) = 0$
$OPT(d1,d2,i) = -\infty$ if $d1 < d[i]$ or $d2 < d[i]$

b) Compute the runtime of the algorithm. (4 pts)

$O(nD^2)$

Grading Rubric:

Common mistakes:

1. Most common mistake is to model each room separately, so having two recurrence for two rooms. This is not true, and will lead to the same solution for each recurrence.
2. Another common mistake is to model the problem using 2D days (simply add up), this is also not true. i.e. If a customer wants to stay for 4 days and there are 2 days left in both rooms, then this approach of combining two rooms into one incorrectly suggests that the customer can stay.
3. Also, there are some students model the problem using three variables, (days, customers, rooms), this is not true.

4. Using one dimensional array to define subproblem is also wrong.

Credits:

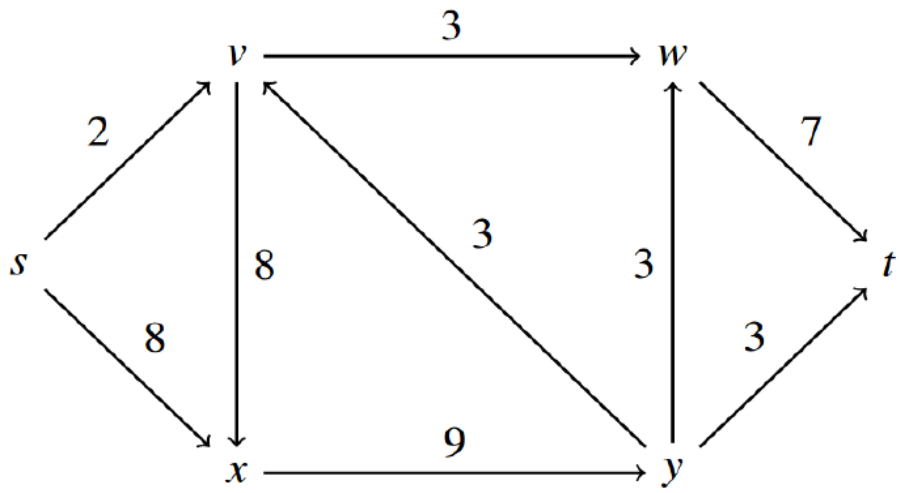
1. Without boundary condition, or incorrect boundary condition : -2 points
2. Every missing recurrence subproblem : -3 points. i.e. the correct recurrence should be max {f_1, f_2, f_3}, but one of them is missing.
3. Incorrect time complexity: -4 points (no partial credit)

Partial credits:

For common mistakes mentioned above, give maximum of 6 points (depends on the correctness of recurrence equation and run time complexity, if the recurrence equation is not correct then no credit was given even though the complexity is correct)

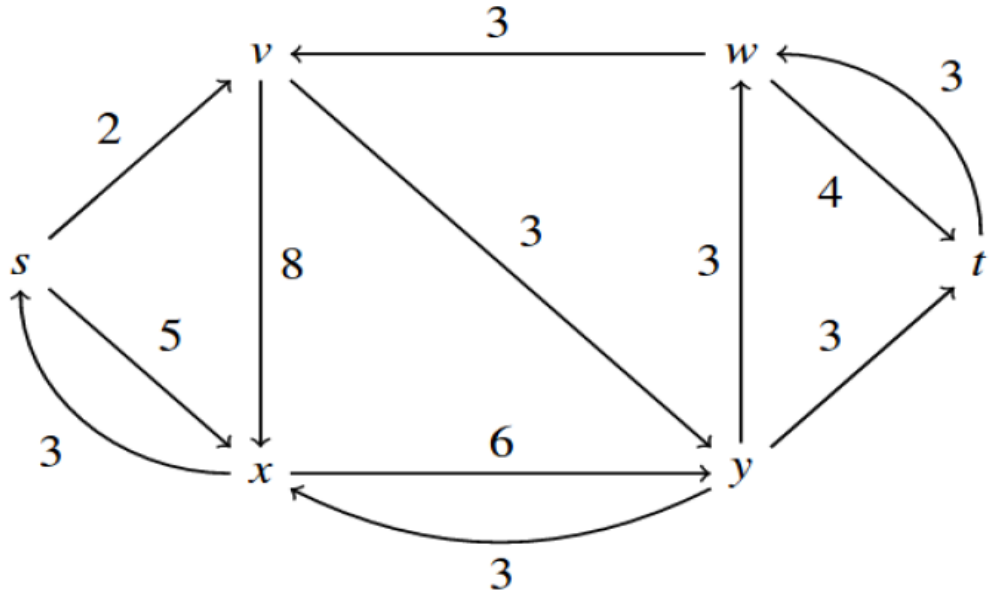
6) 15 pts

You are given the following graph. Each edge is labeled with the capacity of that edge.



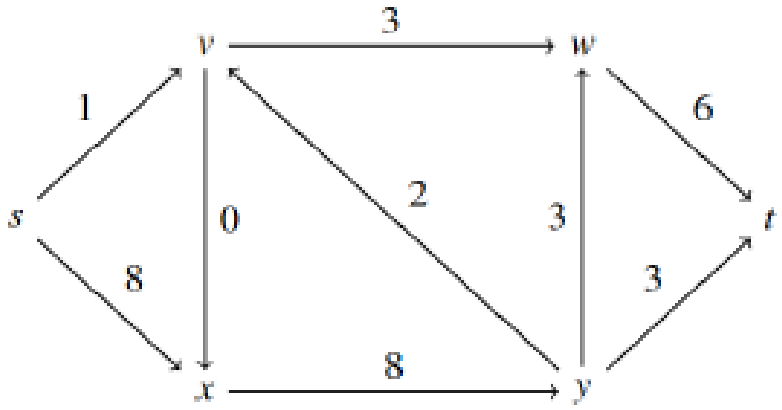
a) Draw the corresponding residual graph after sending as much flow as possible along the path s→x→y→v→w→t. (5 pts)

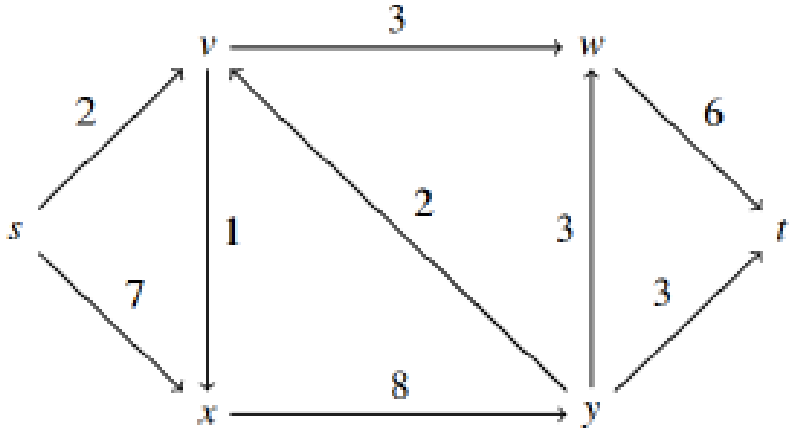
3

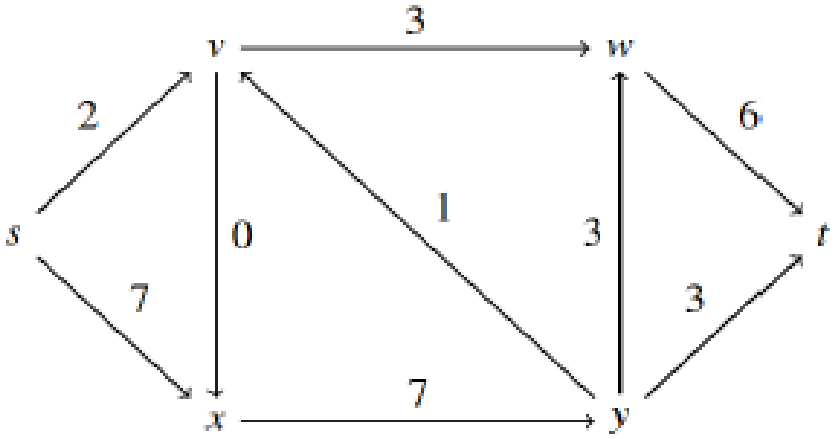


b) Find the value of a max-flow. (5 pts)

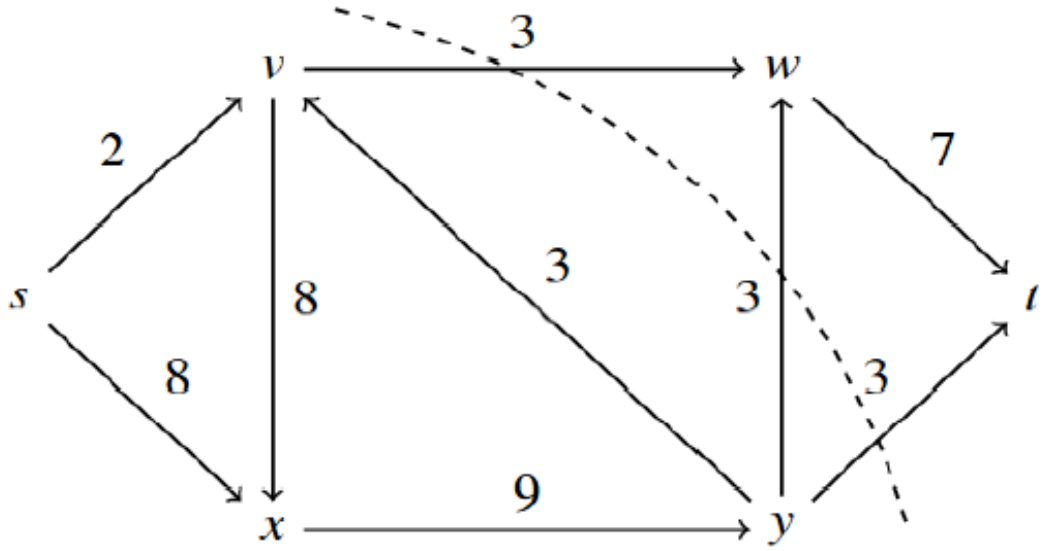
9, multiple answers







c) Find a min-cut? (5 pts)



7) 15 pts

Consider a drip irrigation system, which is an irrigation method that saves water and fertilizer by allowing water to drip slowly to the roots of plants. Suppose that the location of all drippers are given to us in terms of their coordinates (x^d, y^d). Also, we are given locations of plants specified by their coordinates (x^p, y^p).

A dripper can only provide water to plants within distance *l*. A single dripper can provide water to no more than *n* plants. However, we recently got some funding to upgrade our system with which we bought *k* monster drippers, which can provide water supply to three times the number of plants compared to standard drippers. So, we now have *i* standard drippers and *k* monster drippers.

Given the locations of the plants and drippers, as well as the parameters *l* and *n*, decide whether every plant can be watered simultaneously by a dripper, subject to the above mentioned constraints. Justify carefully that your algorithm is correct and can be obtained in polynomial time.

Solution:

Create a graph G=(V, E)

V = {source s, nodes p_1, …, p_j for sets of plants, nodes d_1, …, d_i for standard droppers, nodes for d_{i+1}, …, d_{i+k}, and sink t} (1 pt)

E = {

$c(s, p_x) = 1$, for x = 1 to j (1 pt)

$c(p_x, d_y) = 1$, for x = 1 to j, y = 1 to i+k, (1 pt) and p_x, d_y are close enough (1 pt)

$c(d_y, t) = n$, for y = 1 to I (2 pt)

$c(d_y, t) = 3n$, for y = i to i+k (2 pt)

}

Use Ford-Fulkerson algorithm to find maximum integer flow f in G and justification. (7 pt)#