CS570 Fall 2018: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	20
Problem 2	20	Problem 6	8
Problem 3	16		
Problem 4	16		
	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.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

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

[TRUE/FALSE]
Dynamic programming only works on problems with non-overlapping subproblems.

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

[TRUE/FALSE]
Every flow network with a non-zero max s-t flow value, has an edge e such that increasing the capacity of e increases the maximum s-t flow value.

[TRUE/FALSE]
A dynamic programming solution always explores the entire search space for all possible solutions.

[TRUE/FALSE]
Decreasing the capacity of an edge that belongs to a min cut in a flow network always results in decreasing the maximum flow.

[TRUE/FALSE]
Suppose f is a flow of value 100 from s to t in a flow network G. The capacity of the minimum $s-t$ cut in G is equal to 100.

[TRUE/FALSE]
One can **efficiently** find the maximum number of edge disjoint paths from s to t in a directed graph by reducing the problem to max flow and solving it using the Ford-Fulkerson algorithm.

[TRUE/FALSE]
If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[TRUE/FALSE]
Given a flow network where all the edge capacities are even integers, the algorithm will require at most C/2 iterations, where C is the total capacity leaving the source s.

[TRUE/FALSE]
By combining divide and conquer with dynamic programming we were able to reduce the space requirements for our sequence alignment solution at the cost of increasing the computational complexity of our solution.

2) 20 pts.
Suppose that we have a set of students $S_1, \cdots, S_s$, a set of projects $P_1, \cdots, P_p$, a set of teachers $T_1, \cdots, T_t$. A student is interested in a subset of projects. Each project pi has an upper bound $K(P_i)$ on the number of the students that can work on it. A teacher T_i is willing to be the leader of a subset of the projects. Furthermore, he/she has a upper bound $H(T_i)$ on the number of students he/she is willing to supervise in total. We assume that no two teachers are willing to supervise the same project. The decision problem here is whether there is really a feasible assignment without violating any of the constrains in K and in H.

a) Solve the decision problem using network flow techniques. (15 pts)

The source node is connected to each student with the capacity of 1. Each student is connected to his/her desired projects with a capacity of 1.
Each project is connected to teachers who are willing to supervise it with a capacity of K(P_i). Each teacher is connected to the sink node with the capacity of H(T_i). If there is a max flow that is equal to the number of students, there is a solution to the problem. You can also convert the order of nodes and edges (source->teachers->projects->students->sink) as long as the edges are correct. You can have 2 sets of nodes for projects (inputs and outputs) as long as the output nodes do not constrain the flow.

b) Prove the correctness of your algorithm. (5 pts)

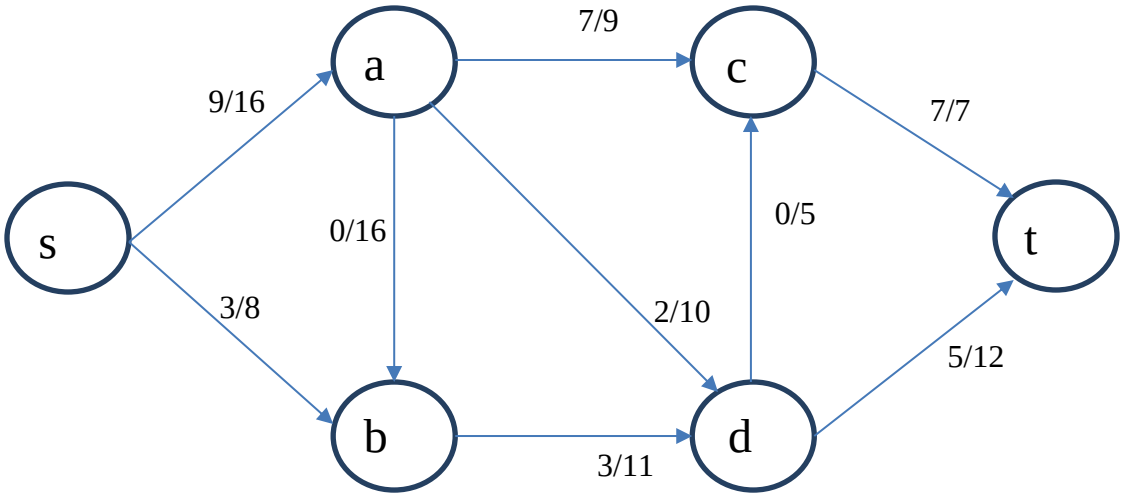
We need to show that
A – If there is a feasible assignment of students and teachers to projects, we can find a max flow of value s (# of students) in G
B – If we find a max flow of value s in G, we can find a feasible assignment of students and teachers to projects.

Rubric:
1) incorrect/missing nodes (incorrect nodes * -3)
2) incorrect/missing edges (incorrect edges * -2)
3) extra demand values (extra demands * -1)
4) not mentioning that max flow = number of students (-3)
5) if the proof is only provided for one side (-2)
6) if proof is not sufficient (-1)

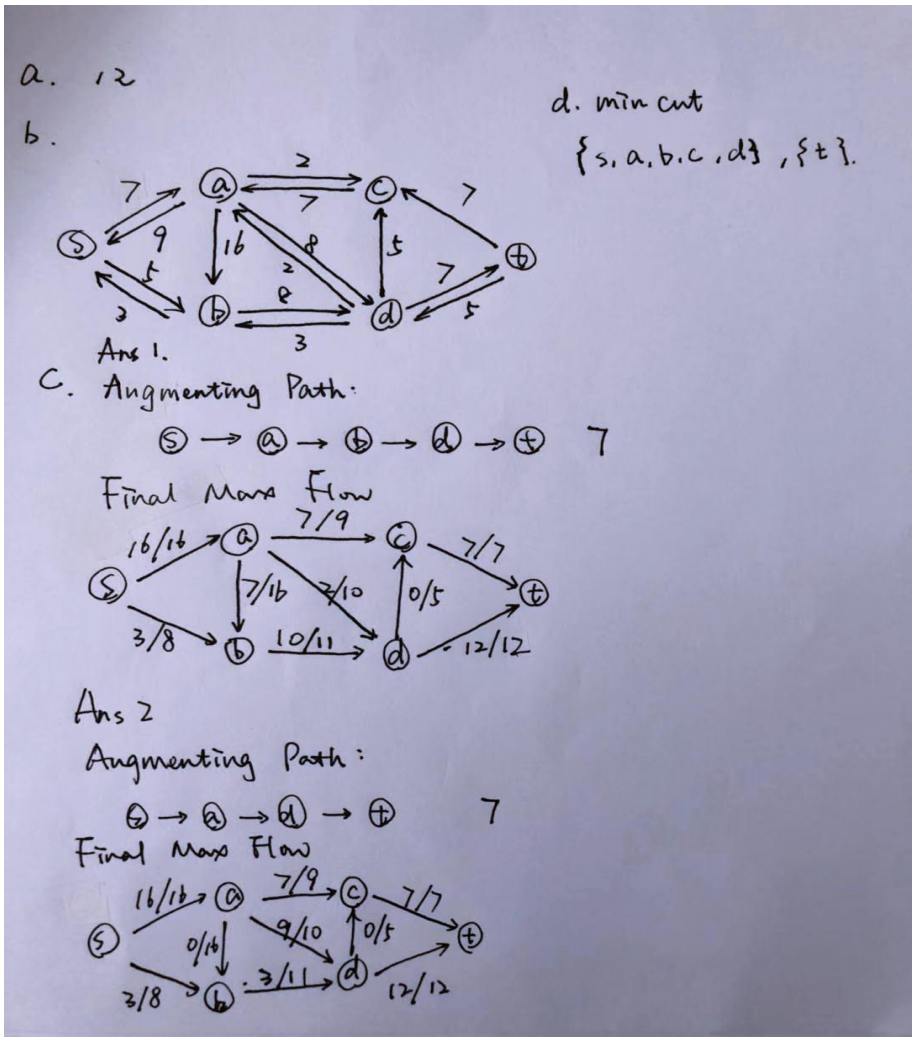
3) 16 pts.
Consider the below flow-network, for which an s-t flow has been computed. The numbers *x/y* on each edge shows that the capacity of the edge is equal to *y*, and the flow sent on the edge is equal to *x*.

a. What is the current value of flow? (2 pts)
b. Draw the residual graph for the corresponding flow-network. (6 pts)
c. Calculate the maximum flow in the graph using the Ford-Fulkerson algorithm. You need to show the augmenting path at each step and final max flow. (6 pts)
d. Show the min cut found by the max flow you found in part c. (2 pts)

Note: extra space provided for this problem on next page



Solution:



Rubric:
a) Incorrect value -> 0 point
b) Every incorrect/missing edge or capacity -> -1point
c) Every possible flow which has a capacity of 19 is correct.
Answers without correct value(other than) 19 will get 0 point.
Answers with correct value/final max flow/final residual graph without augmenting path/detail steps will get 3 points
Answers with correct value/final max flow/final residual graph with augmenting path&detail steps will get 6 points
Answers with a correct augmenting path but incomplete max flow will get 1 point.
d) Incorrect set -> 0 point

4) 16 pts

A symmetric sequence or palindrome is a sequence of characters that is equal to its own reversal; e.g. the phrase "a man a plan a canal panama" forms a palindrome (ignoring the spaces between words).

a) Describe how to use the **sequence alignment algorithm** (described in class) to find a longest symmetric subsequence (longest embedded palindrome) of a given sequence. (14 pts)
Example: CTGACCTAC 's longest embedded palindromes are CTCCTC and CACCAC

Solution: given the sequence S, reverse the string to form S^r . Then find the longest common substring between S and S^r by setting all mismatch costs to ∞ and δ=1 (or anything greater than zero) .

b) What is the run time complexity of your solution? (2 pts)
$O(n^2)$

Rubric 4a)

- 10 points for reversing the string and using the sequence alignment algorithm
  - If the recurrence is given without explicitly mentioning the sequence alignment algorithm is fine.
  - Recurrence is discussed in class
- 2 points for allocating the correct mismatch value.
- 2 points for allocating the correct gap score.
- Any other answer which does not use sequence alignment algorithm is wrong. **The question explicitly asks to use the sequence alignment algorithm.** (0 to 2 points)
  - 2 points for correctly mentioning any other algorithm to get palindrome within a string
- Wrong answers
  - Dividing the string into two and reversing the second half.
    - Will not work, as the entire palindrome can be located within the first or the second half
  - Aligning the original string with the copy of the same string
    - This will return the original string and not the palindrome
  - Any other algorithm which finds the palindrome in a string

Rubric 4b)

2 points

0 points for any other answer

5) 20 pts

Let's say you have a dice with m sides numbered from 1 to m, and you are throwing it n times. If the side with number i is facing up you will receive i points. Consider the variable Z which is the summation of your points when you throw the dice n times. We want to find the number of ways we can end up with the summation Z after n throws. We want to do this using dynamic programming.

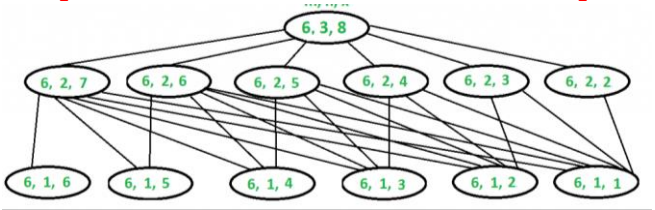
a) Define (in plain English) subproblems to be solved. (4 pts)

OPT(n , Z) = number of ways to get the sum Z after n rolls of dice.

b) Write the recurrence relation for subproblems. (6 pts)

$OPT(n, Z) = \Sigma_{i=1 \text{ to } m} OPT(n-1, Z-i)$

Example: Given, m=6, Calculate: OPT(3,8)
Subproblems are as follows (represented as(m,n,Z):



Rubric:
-6 if not summing over m
-4 if summing over m but OPT is somewhat incorrect.

c) Using the recurrence formula in part b, write pseudocode to compute the number of ways to obtain the value SUM. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

```
OPT(1,i) = 1 for i=1 to MIN(m,SUM)
OPT(1,i) = 0, for i=MIN(m,SUM) to MAX(m,SUM)
For i = 2 to n
     For j = 1 to SUM
          OPT(i, j) = 0
          For k =  1 to MIN (m, i)
               OPT(i, j) = OPT(i, j) + OPT(i-1,j-k)
          Endfor
     Endfor
Endfor
```

```
Return OPT(n,SUM)

Example, Given, m=6, Calculate: OPT(3,8)
OPT Table looks as follows:
```

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0
0	0	1	2	3	4	5	6	5
0	0	0	1	3	6	10	15	**21**

Alternatively, you can initialize OPT(0,0) = 1 and start the iteration of i from 1.

Rubric:
-1 for each initialization missed.
-6 for any incorrect answer. (e.g incorrect recurrence relation, incorrect loops, etc)
i.e. no partial grading for any pseudocode producing incorrect answer.

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (2 pts)

Time Complexity: O(m * n * Z) where m is number of sides, n is the number of times we roll the dice, and Z is given sum. This is pseudo polynomial since the complexity depends on the numerical value of input terms.

Rubric:
Not graded if part c is incorrect.
-2 if stated as polynomial time.
-2 if incorrect complexity equation.

6) 8 pts

a) Given a flow network G=(V, E) with integer edge capacities, a max flow *f* in G, and a specific edge *e* in E, design a linear time algorithm that determines whether or not *e* belongs to a min cut. (6 pts)

Solution:
Reduce capacity of e by 1 unit. - O(1)
Find an s-t path containing e and subtract 1 unit of s-t flow on that path. For this, considering e's adjacent nodes to be (u,v), you can run BFS from source to find an s-u path and BFS from v to find an v-t path which gives you an s-u-v-t path crossing e. - O(|V|+|E|)
Then construct the new residual graph $G_{f'}$ - O(|E|)
If there is an augmenting s-t path in $G_{f'}$ then e does not belong to a min cut, otherwise it does.

b) Describe why your algorithm runs in linear time.

You can find this using BFS from source(only one iteration of the FF algorithm) - O(|V| + |E|)
All the above steps are linear in the size of input, therefore the entire algorithm is linear.

Rubric:
If providing the complete algorithm: full points
- If not decreasing the s-t flow by 1: -1 points
- If running the entire Ford-Fulkerson algorithm to find the flow: -2 points
- If providing the overall idea without description of implementation steps: -3 or -4 points depending on your description

If the provided algorithm is not scalable to graphs having multiple(more than two) min-cuts(for example using BFS to find reachable/unreachable nodes from source/sink): -4 points
- Incomplete or incorrect description of the above algorithm (-1 or -2 additional points depending on your description)

If removing the edge and finding max-flow: -3 or -4 points depending on your description (since this algorithm doesn't run in linear time)
If only considering saturated edges to check for min-cut edges: -7 points
Adding the capacity of e and checking the flow: no points (this approach is not checking for min-cut)
Checking all possible cuts in the graph: no points (this takes exponential time)
Other algorithms: no points