CS570 Fall 2019: Analysis of Algorithms Exam II

	Points		Points
Problem 1	20	Problem 4	20
Problem 2	20	Problem 5	20
Problem 3	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.
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]
For any edge *e* that is part of a minimum cut in a flow network G, if we increase the capacity of that edge by any integer k>1, then that edge will no longer be part of a minimum cut.

[TRUE/FALSE]
The sequence alignment algorithm described in class can be used to find the longest common subsequence between two given sequences.

[TRUE/FALSE]
The scaled version of the Ford-Fulkerson algorithm can compute the maximum flow in a flow network in polynomial time.

[TRUE/FALSE]
Given a set of demands D = {dv} on a circulation network G(V,E), if the total demand over V is zero, then G has a feasible circulation with respect to D.

[TRUE/FALSE]
In a flow network, the maximum value of an $s-t$ flow could be less than the capacity of a given $s-t$ cut in that network.

[TRUE/FALSE]
If f is a max $s-t$ flow of a flow network G with source s and sink t, then the capacity of the min $s-t$ cut in the residual graph G_f is 0.

[TRUE/FALSE]
In a graph with negative weight cycles, one such cycle can be found in O(nm) time where n is the number of vertices and m is the number of edges in the graph.

[TRUE/FALSE]

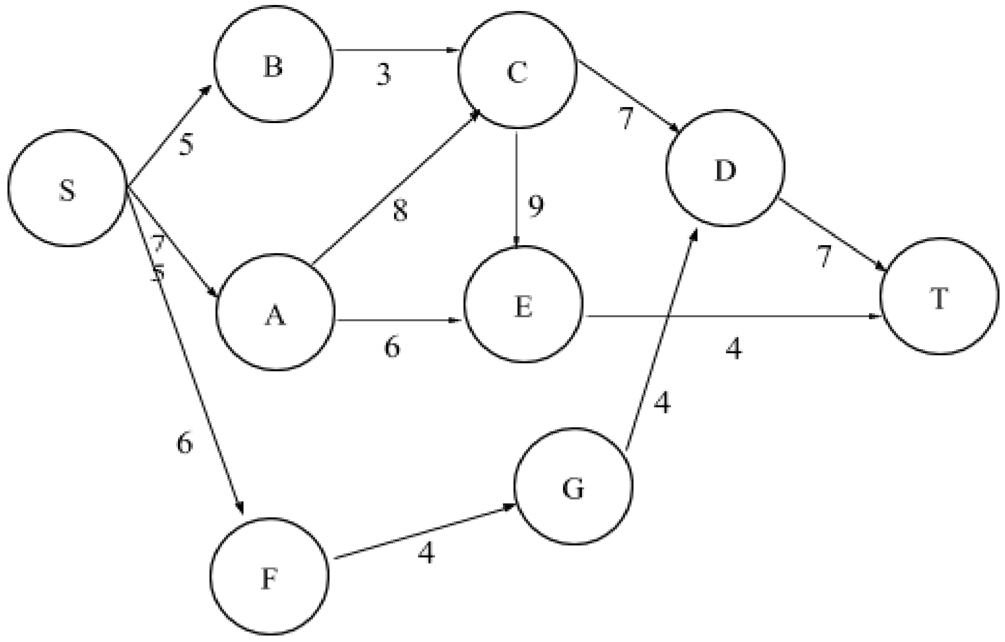
An algorithm runs in weakly polynomial time if the number of operations is bounded by a polynomial in the number of bits in the input, but not in the number of integers in the input.

[TRUE/FALSE]
Let G(V,E) be an arbitrary flow network, with a source s, a sink t. Given a flow f of maximum value in G, we can compute an s-t cut of minimum capacity in O(|E|) time.

[TRUE/FALSE]
The basic Ford-Fulkerson algorithm can be used to compute a maximum matching in a given bipartite graph in strongly polynomial time.

1) 20 pts

Perform two iterations (i.e. two augmentation steps) of the scaled version of the Ford-Fulkerson algorithm on the flow network given below. You need to show the value of Δ and the augmentation path for each iteration, and the flow f and $G_f(\Delta)$ after each iteration. (Note: iterations may or may not belong to the same scaling phase)

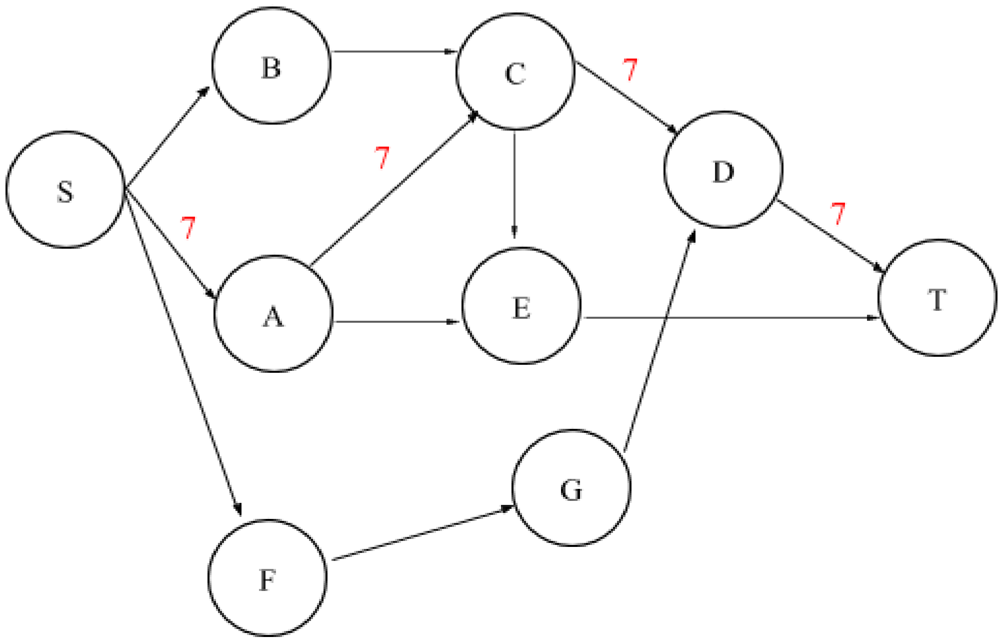


See next page…

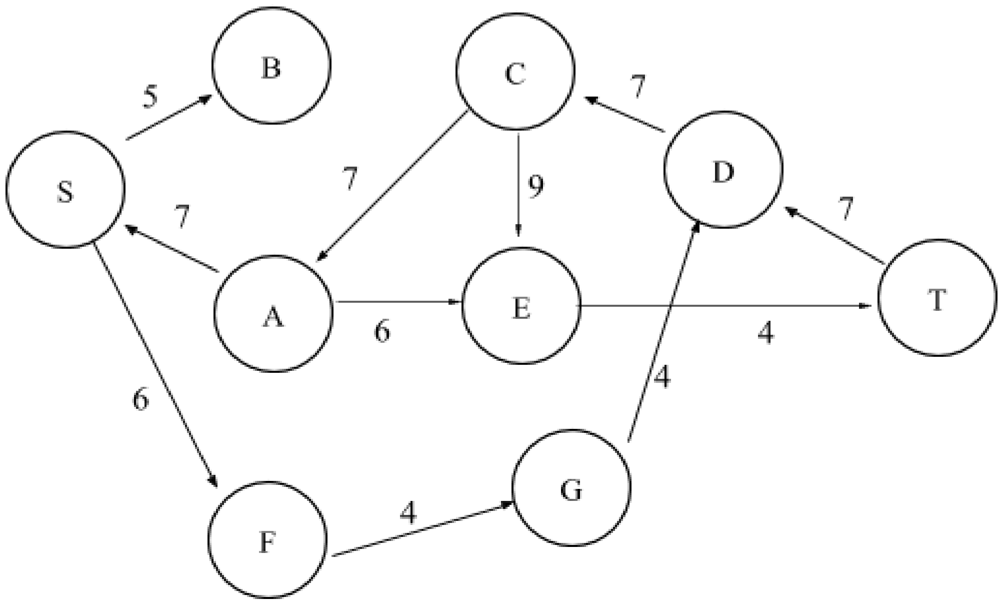
(a) Iteration 1: (8 pts)
Δ is __4________________

Augmentation Path: __SACDT (there are a few different choices of augmentation paths)_____________________

Flow after the first iteration (you can write flow values over each edge carrying flow):



$G_f(\Delta)$ after the first iteration (you can write flow values over each edge carrying flow):

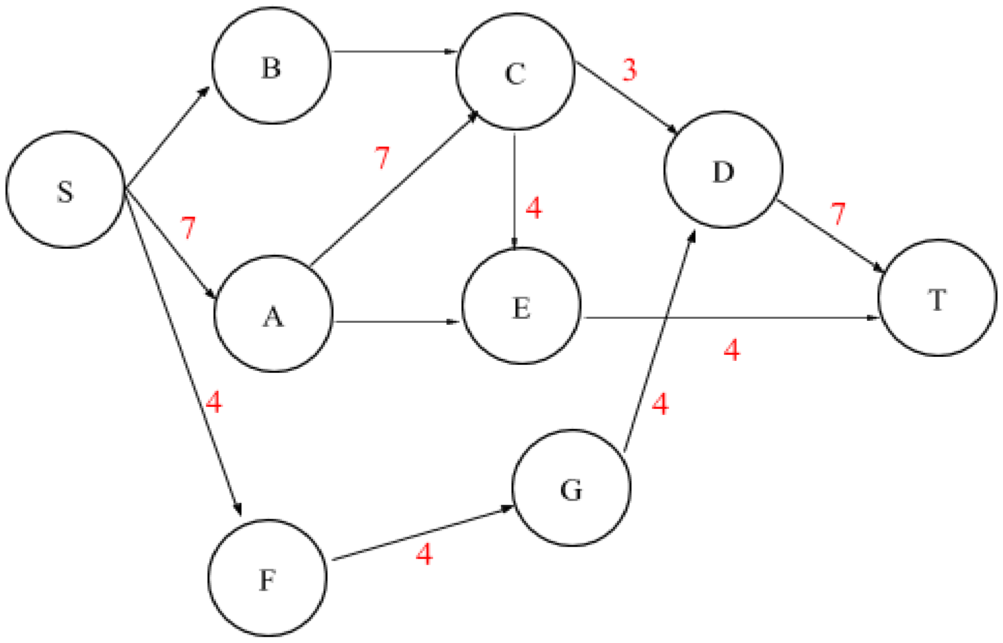


(b) Iteration 2: (8 pts)

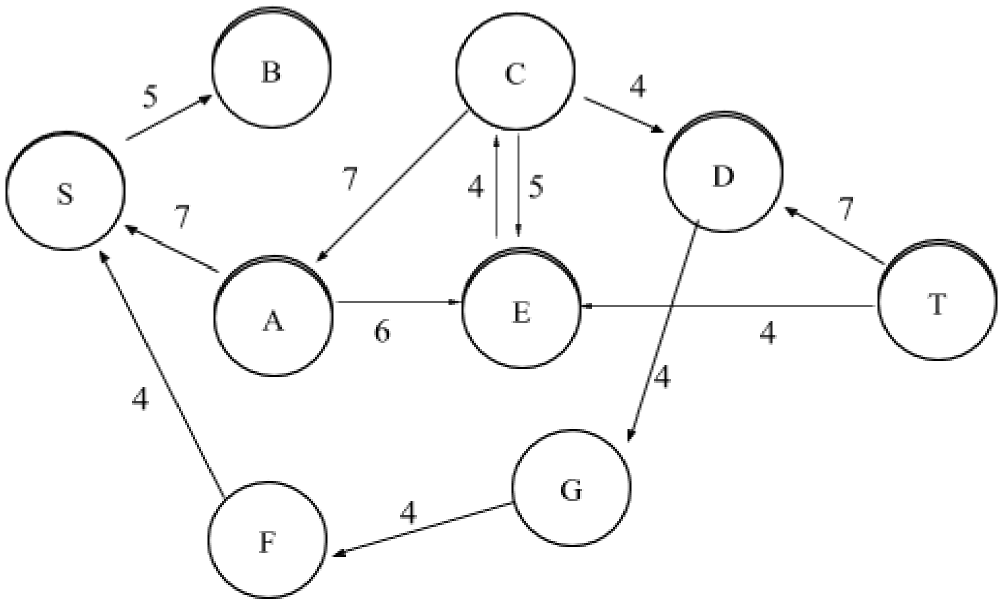
Δ is __4________________

Augmentation Path: __SFGDCET (there are a few different choices of augmentation paths)_____________________

Flow after the second iteration (you can write flow values over each edge carrying flow):



$G_f(\Delta)$ after the first iteration (you can write flow values over each edge carrying flow):



(c) Can the choice of augmentation paths in the scaled version of Ford-Fulkerson affect the number of iterations? Explain why. (4 pts)

Yes, for example, if we had chosen paths with bottleneck values of 4 (SAET and SFGDT) in the first two iterations, then we should have needed more than two iterations to get to max flow. But with the choice of augmentation paths given above, we were able to get to max flow in 2 iterations.

An intuitive argument: Yes, there are still different choices for augmentation paths within a given scaling phase. The different augmentation paths could have different bottleneck values. This can affect how quickly the value of flow goes up and therefore affect how many iteration we will have to perform. In particular, using aug paths with higher bottleneck capacity can lead to fewer iterations
Common mistakes:

1) Using an example that proves this for the unscaled version but doesn't work for the scaled one.
2) Referencing Edmond Karp and the idea of using 'shortest paths'

Rubric:

Parts a) + b)
Computing the Delta values: 3+3 = 6
Full points for (4,4).
For any Delta in (7/8/9), (Delta, round(Delta/2)) gets -2 penalty.
Most other cases get 0. 1 or 2 partial credit in VERY RARE cases.

Aug path + flow: 2+2 = 4
Aug path flow not equal to the bottleneck edge: -2

Gf(Delta): 3+3 = 6
-1 for 1 edge missing/wrong. -2 for more.
In particular, not filtering edges higher than Delta gets -2. A total of -3 if the both a) and b) have this mistake.

Part c)
0 If the answer is No.
If Yes, 1 for the answer, 3 for the explanation. Parial credit 1/2 if the explanation is incomplete/unclear.

3) 20 pts

The Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other. Each of these operations has unit cost.

For example, the Levenshtein distance between “kitten” and “sitting” is 3. A minimal edit script that transforms the former into the latter is:

kitten -> sitten
sitten -> sittin
sittin -> sitting

We want to design a dynamic programming algorithm that calculates the Levenshtein distance between a string X of length m and another string Y of length n. An edit can be adding, removing, or changing a character in X.

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

OPT(i, j) is the Levenshtein distance between $X_1..X_i$ and $Y_1..Y_j$

Rubrics:

- (3 pts) subproblem is the distance between two substrings of X and Y
- (1 pt) specify two substrings are X1...Xi and Y1...Yj

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

```
OPT(i, j) = OPT(i-1, j-1) if $X_i = Y_j$ ,
            Otherwise, Min( OPT(i-1, j-1) +1     [substitute last letter in X],
                            OPT(i-1, j) +1       [remove last letter in X],
                            OPT(i, j-1) +1   )   [insert a letter in X to match $Y_j$]
```

Rubrics:

- (2 pt) if specify there are two cases (a)X_i = Y_j and (b) X_i != Y_j
- (1 pts) if write correct answer for the case X_i = Y_j
- (3 pts) if write correct answer for the case X_i != Y_j

c) Using the recurrence formula in part b, write pseudocode (using iteration) to compute the Levenshtein distance between strings X and Y. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

```
Initialize OPT to be a (m + 1) * (n + 1) 2d array.
OPT(0,j) = j for all j=1..n
OPT(i,0) = i for all i=1..m

    For i=1 to m
    For j=1 to n
```

```
                If X[i] == Y[j]
                        OPT(i, j) = OPT(i – 1, j – 1)
                ELSE
                        OPT(i, j) = min{OPT(i-1, j-1), OPT(i, j-1), OPT(i-1,j)} + 1
        endfor
    endfor
        RETURN OPT(m, n)
```

Rubrics:
2 pts for Initialization

- (1 pt) specify the first column and the first row need to be initialized specifically
- (1 pt) correct initialization of OPT(i, 0), OPT(j, 0) for i=1...m

- if use extra matrix (other than OPT), get 1pt off if no pseudocode of initialization

6 pts for pseudo code

- implement the correct recurrence formula using nested for loop
    - if the recurrence formula get 3 points off, then this part will also get 3 points off
    - get 2pts off if mess up (i-1,j-1), (i-1,j), (i,j-1)
- get 2pts off if the nested for loop is not implemented correctly
    - get 1pt off if the for loop's order is in-compatible with the recurrence formula

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

Runtime: O(mn). Polynomial time

Rubrics:

- (1 pt) O(mn)
- (1 pt) polynomial/strong polynomial

4) 20 pts

Suppose a concert has just ended and C cars are parked at the event. We would like to determine how long it takes for all of them to leave the area. For this problem, we are given a graph representing the road network where all cars start at a particular vertex s (the parking lot) and several vertices (t_1 , t_2 , . . . , t_k) are designated as exits. We are also given capacities (in cars per minute) for each road (directed edges). Give a polynomial-time algorithm to determine the amount of time necessary to get all cars out of the area.

Solution:

We already have a source s (the parking lot) and capacities; however, we have multiple sinks (the exits). We can define a new network based on the road network by adding a sink T and connect each exit to it with a directed edge; the capacity can be any number that is at least as much as the total incoming capacity to that exit. We then run any polynomial-time max flow algorithm on the resulting graph; because it is polynomial in size (has only one additional vertex and at most n additional edges), the resulting runtime to compute v(f*) is polynomial. However, we don't want to stop at computing v(f*), because that's just the rate at which cars leave the parking lot. It takes c/v(f*) minutes for the parking lot to empty.

Rubric:

1) The defined work is fully correct. (6 points)
   a) If there is no sink T, deduct 3 points
   b) If the given road network is not used, deduct 2 points

2) The capacity on the new edges from each exit to the sink is correct. (4 points)

Note there can be multiple edges connected to the exit. And the edges are newly defined, whose capacities are not given. The capacity should be at least as much as the total incoming capacity to that exit or infinity.

3) Run any polynomial-time max flow algorithm on the resulting graph to compute the v(f*). (6 points)

The implementation of max flow is not right, deduct 2 points

4) The value of max flow v(f*) is the rate at which cars leave the parking lot. It takes c/v(f*) minutes for the parking lot to empty. (4 points)
   a) If the min time c/v(f*) is not clearly shown, deduct 2 points

For any attempt. (4 points)

5) 20 pts
There are n companies participating in a trade show. Company *i* gives away goodies worth g_i at their booth to each person visiting their booth. But you might have to wait in a line to get to the booth.

a) Knowing that the wait time at booth *i* is $w_i \geq 0$, formulate a solution that will earn you a minimum of G dollars' worth of goodies without spending more than a total of H hours waiting in lines. Your solution should also indicate, given G and H, if this objective is not possible to achieve. (18 pts)

This is a 0-1 knapsack problem.

The solution can either show a reduction to 0/1 Knapsack or show a direct solution (recurrence formula, pseudocode, etc.)

For a reduction, it is enough to say that:

- H is the capacity of the knapsack
- w_i and g_i are the weight and value of the items respectively
- If the maximum value that we can pack into the knapsack is greater than G, we have a solution, otherwise there is no solution that satisfies the bounds H and G

For the direct solution, we need to

- Define the value of the optimal solution for a unique subproblem
- Present a recurrence formula
- Write pseudocode to show how we can find the value of the optimal solution
- Compare the value of the optimal solution to G and determine if there exists a solution that satisfies the bounds H and G or not.

Rubrics:

- No subproblem definition; - 3 points
- No pseudocode; - 5 points
- Whether efficient; - 1 point
- How to decide whether is possible; -2 points
- Wrong recursive function; - 5 points
- Wrong algorithm; - 18 points
- Incorrect solution based on the assumption; - 5 points
- Miss the boundary case; - 3 points

b) Analyze the complexity of your solution and determine if it is an efficient solution. (2 pts)

Solution runs in O(Hn). This is pseudopolynomial and therefore not efficient.

Additional Space