CSCI 570
Exam 2 Solutions

Problem 1 (Questions 1-10) (Random Questions)[Any 10 from the below 18 Questions]
Rubric: 2 points each correct Answer. Max 20 points

1) Max-flow in a flow network can be found with a polynomial-time greedy algorithm - True

2) Suppose we have two minimum s-t cuts (A1, B1) and (A2, B2) and some max flow f. Total flow from A1 to B1 is equal to the total capacities of edges going from A2 to B2. - True

3) A Weakly polynomial time algorithm is always slower than a strongly polynomial time algorithm. - False

4) The main difference between divide and conquer and dynamic programming is that divide and conquer solves problems in a top-down manner whereas dynamic programming does this bottom-up. - False

5) We can solve the weighted interval scheduling problem in linear time using dynamic programming. - False

6) A flow network with unique edge capacities could have more than one min cut. - True

7) Ford-Fulkerson Algorithm can solve the max-flow problem in a flow network in polynomial time if the augmenting path with the fewest number of edges is chosen in each iteration. - True

8) The Ford-Fulkerson Algorithm can find a maximum matching in a bipartite graph in $O(n^3)$ time where n is the number of nodes in the graph. - True

9) In a flow network with integer edge capacities, if the capacity of an edge which is not on a min cut is decreased by k, the value of max flow will go down by at most k-1. - True

10) An algorithm runs in weakly polynomial time if the number of operations is bounded by the numerical value of input terms, but not in the number of integers in the input. - False

11) For every flow network with a unique min cut and where value of max flow is greater than zero, there is always an edge such that increasing the capacity on that edge will increase the value of the maximum flow in the network. - True

12) In a flow network, a flow that contains a negative flow cycle could be a valid flow. - True

13) In a flow network, a valid max flow may contain a positive flow cycle. - True

14) In a flow network, decreasing the capacity of an edge that does not belong to a min cut could result in lowering the value of max flow. - True

15) In the scaled version of the Ford Fulkerson algorithm, if at a given augmentation step the bottleneck value is exactly equal to 64 (the threshold value at that scaling phase) then the bottleneck value for any subsequent augmentation step cannot be greater than 64. - False

16) The Ford-Fulkerson algorithm can be used to find the maximum flow through a graph with cycles. - True

17) The Basic Ford-Fulkerson algorithm (without scaling) can be used to solve the feasible circulation problem in strongly polynomial time. - False

18) Suppose, increasing the capacity of edge e by 1, increases value of max flow f by 1. Then it must be that $f(e)$ = capacity (e). - True

Problem 2(Question 11-12):

Q11: Consider the Edmonds-Karp algorithm applied to the above graph to find a maximal flow. State the augmenting paths found by the algorithm. State for each augmenting path the nodes on the path and the value (of flow) augmented with along the path. (8 pts) Hint: Remember that in the Edmonds-Karp algorithm we always choose the shortest augmentation path from s to t. Otherwise, the algorithm is the same as Ford Fulkerson.

Solution:

Step1: path s -> t flow 2

Step2: path s -> D -> E -> F -> t flow 3

Step3: path s -> D -> E -> G -> H -> t flow 1

Step4: path s -> D -> A -> B -> C -> F -> t flow 2

Step5: path s -> D -> A -> B -> C -> F -> E -> G -> H ->t flow 3

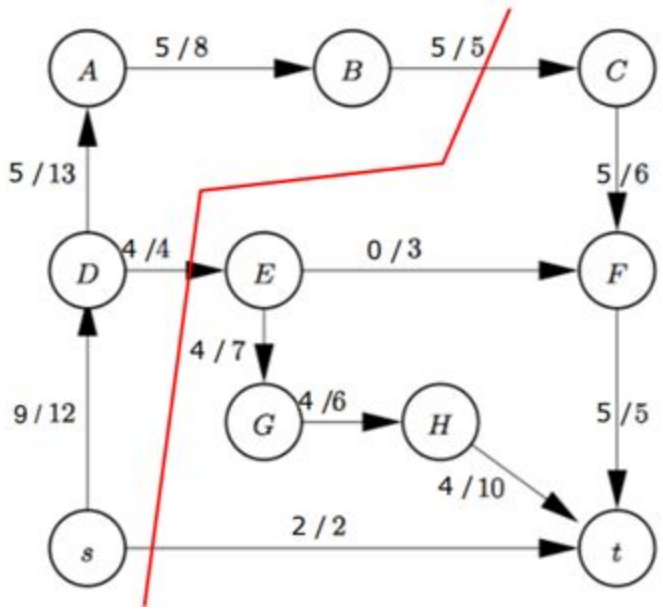
Rubrics:

1.5 pts per step, 1 pt for the path and 0.5 pt for the flow of the path. (total 5 * 1.5 pts = 7.5 pts)

If all the path and values are correct, extra 0.5 pts will be given.

Q12:Give a maximum flow from s to t in the network (state for each edge the flow along the edge), state the value of the maximal flow, and identify a minimum cut. (8 pts) Note: No drawings are necessary but if you wish to provide your solutions on a drawing you can do so.

Solution:



The value of maximum flow is 11

The minimum s-t cut is {s, D, A, B} / {C, E, G, H, F, t} or {s, D, A, B, C, F} / {E, G, H, t}

Rubrics:
4 pts for stating the flow for each edge. every incorrect edge flow value or missing edge flow value will lose 0.5 pts
2 pts for the value of the maximum flow
2 pts for the min s-t cut. Incorrect min s-t cut is 0 point

Problem 3(Question 13):

Say you have an undirected graph G and two subsets of its nodes S={s1, …sk} and T={t1, ….tk}. You are asked to determine if it is possible to find k node-disjoint paths from S to T. In other words, the paths should start at some node in the set S and end at some node in the set T but should not share any nodes with each other. Present a solution to this problem and describe how you determine whether a solution (k node-disjoint paths) exists. Note: You are not asked to find the paths.

Solution:
1- Turn the undirected graph into a directed graph (each undirected edge -> two directed edges in opposite directions). Give each edge a capacity of 1
2- Split each node v, not in the set S or T, into two nodes v' and v'' with an edge going from v' to v'' with capacity of 1. All edges into v will be incident on v' and all edges out of v will be incident on v''
3- Create a source node S* and connect edges with capacity 1 from S* to all si' nodes.
4- Create a sink node T* and connect edges with capacity 1 from tj'' nodes to T*
5- Find max flow F in the resulting flow network. If v(F) is equal to k we can find k node-disjoint paths from S to T

Alternatively (instead of step 3 above), you can place supply values of -1 at all si' nodes and demand values of 1 at tj'' nodes. And check to see if there is a feasible circulation in the resulting circulation network

Rubrics:
5pts for Step 1 and 5pts Step 2, correctly converting the undirected graph into a directed graph.
4pts for Step 3 and Step 4, or the alternative circulation model, correctly building the network flow model.
2pts for Step 5, giving a correct conclusion.

Grading codes in the feedback
A: You didn't correctly convert the undirected graph into a directed graph by splitting each undirected edge into two opposite directed edges.
B: You didn't correctly split each inner node into two nodes or you didn't correctly write the method to handle edges after splitting such as incoming edges, outgoing edges and the edge from v' to v''. Or you forget to set the capacity as 1 for the edge from v' to v''.
C: You didn't correctly build the maxflow graph or the circulation. You may forget to add a source node or a sink node. Or you didn't mention the capacity should be 1.
D: You didn't give a correct conclusion. It's not enough to write "run the max flow" only but you should mention the correct relationship between the max flow and the determination to the original question.

Problem 4 (Question 14):

New semester is coming, and Trojans need to register for courses. There are n trojans (Ti denotes the ith trojan). Each Trojan can register for at most 3 courses. There are m courses and k professors available to teach these courses. ci denotes the ith course and Pi denotes the ith

professor. Each professor Pi teaches certain courses. For example, professor P2 can teach c2 and c5. However, for all the courses that professor Pi teaches, he can teach at most Ni students in total. A given course could be taught by multiple professors. For each registered course, a Trojan can get one unit. Present a solution to find the maximum number of units that can be obtained by all Trojans in the new semester.

Solution:

We construct a graph G which involves:

Nodes: source node S, trojan nodes T_i, course nodes C_i, professor nodes P_i, and sink node X.

Edges: (1) S to all T_i. Set the capacities to 3.

(2) Each T_i to all course nodes. Set the capacities to 1.

(3) Edges of infinite capacity from C_i to P_j if professor j can teach c_i.

(4) Each P_i to X with capacity N_i.

Run max flow algorithm on this graph. The value of max flow is the max units that all trojans can earn.

Rubrics:

1. 1pt for creating each type of nodes (S, T, C, P, X, 5 points in total)
2. 2pt for setting the edges and their capacities correctly
   (S->T, T->C, C->P, P->X, 8 points in total)
3. 3pt for running max flow algorithm and returning the answer.

Problem 5 (Question 14-17):

A secret document has been encoded using the following mapping function:

A encoded as 1

B encoded as 2

C encoded as 3

…

Y encoded as 25

Z encoded as 26

We need to find out the total number of ways that a given string within this document can be

decoded.

For example, the string ‘26’ can be decoded two ways: { ‘BF’, ‘Z’ }

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

Solution:

Opt(i): the number of ways that a string of size i can be decoded.
Rubrics:

The above answer gets the full credit.

If the answer is partially correct or only an explanation is given instead of defining the subproblem (2-3 points will be given depending on the answer)

b) Write the recurrence relation for subproblems (5 pts)

Let c(i) denote the i'th number.

If c(i) = 0 and (c(i-1) = 1 or c(i-1) = 2): (2 point)

OPT(i) = OPT(i-2)

else If (c(i-1) =1 and c(i) != 0) or (c(i-1)=2 and 0 < c(i) < 7): (2 points)

OPT(i) = OPT(i-1) + OPT(i-2)

else: (1 points)

OPT(i) = OPT(i-1)

c) Using the recurrence formula in part b, write pseudocode to compute the total number of ways to decode the string. (5 pts)

Make sure you have the base cases and their values defined. (2 pts)

initialize OPT[0] = 1, OPT[1] = 1

for i = 2 to n:

If c(i) = 0 and (c(i-1) = 1 or c(i-1) = 2):

OPT[i] = OPT[i-2]

else if (c(i-1) =1 and c(i) != 0) or (c(i-1)=2 and 0 < c(i) < 7):

OPT[i] = OPT[i-1] + OPT[i-2]

else:

OPT[i] = OPT[i-1]

print OPT[n]

Rubrics:

2 points for correct initial values

2 points for giving the right recurrence loop

1 point for correct final answer

2 point for writing the correct conditions

Problem 6 (Question 18-21):

Your favorite ice cream store Bevrören Spüllen is opening up exactly k stores on Yonge Street and have hired you to help them optimize their store locations. Yonge Street has n blocks 1, 2,…, n from South to North, and for each block i there is an expected revenue ri>0 The company has decided that if they open a store on block i, they will not open another store on block i, or the two blocks to its South (i-1 and i-2) or the two blocks to its North (i+1 and i+2). To find the maximum possible revenue you have decided to give dynamic programming a shot. So here are the steps you need to follow.

Note: You can assume that it is possible to open k stores on Yonge Street that meet given constraints

Question 18

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

Solution:

opt[i,t] be the maximum revenue that is possible to get when t stores are built in blocks 1,...,i.

Rubrics:

1pt for giving a reasonable denotation.

3pts for giving a sound and sufficient explanation. (if the design doesn't cover all possible cases or has a flaw, then it only gets 1 or 2 pts).

Question 19

Write the recurrence relation for subproblems. (4 pts)

Solution:

opt[i,t] = max(opt[i-1,t], opt[i-3,t-1]+r[i])

Rubrics:

4pts for giving a correct formula. (must in accordance with the previous solution)

Question 20

Using the recurrence formula in part b, write pseudocode to compute the maximum revenue possible. (4 pts)

Make sure you have initial values properly assigned.

Solution:

```
For i in [-3, 0]:
        Opt[i,0] = 0
        For t in [1, k]:
                Opt[i,t] = -Inf
For i in [1, n]:
        Opt[i, 0] = Opt[i-1, 0]
        For t in [1, k]:
                Opt[i, t] = max(Opt[i-1, t], Opt[i-3, t-1]+r[i])

Return Opt[n, k] as the answer.
```

Rubrics:

2pts for giving the right recurrence loop.

1pt for giving the correct initial values and boundaries.

1pt for giving the correct final answer.

Question 21

Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not. (4 pts)

Solution:

For the algorithm described above, since it has O(n*k) subproblems in total, and for each subproblem it costs constant time (O(1) time) to solve, the sum-up time cost should be O(n*k).

Furthermore, because k<n (otherwise it wouldn't be possible to open k stores in n blocks), we can say that the algorithm is O(n^2). It is a polynomial complexity.

Rubrics:
1pt for giving the right time complexity.
2pts for a comprehensive explanation of the complexity.
1pt for correctly stating if it's polynomial.