CS570 Summer 2025: Analysis of Algorithms Exam II

	Points		Points
Problem 1	20	Problem 4	18
Problem 2	9	Problem 5	20
Problem 3	18	Problem 6	15
			Total 100

First name	
Last Name	
Student ID	

Instructions:

1. This is a 2-hr exam. Closed book and notes. No electronic devices or internet access.
2. A single double sided 8.5in x 11in cheat sheet is allowed.
3. 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.
4. No space other than the pages in the exam booklet will be scanned for grading.
5. 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.
6. Do not detach any sheets from the booklet.
7. 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.
8. Do not write your answers in cursive scripts.
9. This exam is printed double sided. Check and use the back of each page.

Student ID:

1) 20 pts (2 pts each)

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]

The Ford-Fulkerson Algorithm finds a maximum flow of a flow network with n vertices, m edges, with all edge capacities ≤ 5 in polynomial time with respect to m and n.

[TRUE/FALSE]

In a flow network, increasing the capacity of an edge that is part of a minimum cut will always increase the value of the maximum flow in the network by at least 1 unit.

[TRUE/FALSE]

If all edge capacities in a flow network are integer multiples of 3, then for any maximum flow f, the value of flow v(f) must also be an integer multiple of 3.

[TRUE/FALSE]

Suppose we run Bellman-Ford on a weighted graph G = (V, E) with destination node t ∈ V and we go past n-1 iterations (where n=|V|). If there is a vertex v ∈ V such that its distance to t further decreases at iteration n, then v is on a negative weight cycle of G.

[TRUE/FALSE]

The memory space required for any iteration-based dynamic programming algorithm with n^2 unique subproblems is $\Omega(n^2)$

[TRUE/FALSE]

In a 0-1 knapsack problem, a solution that uses up all of the capacity of the knapsack will always be optimal.

[TRUE/FALSE]

If removing an internal node u (and its incident edges) from a flow network G disconnects its source from its sink, then the number of iterations required for Ford-Fulkerson to find max-flow in G will be bounded by the total capacity of the edges going into u.

[TRUE/FALSE]

The Sequence Alignment problem discussed at length in lecture (between two strings of size m and n) can be solved given only O(k) memory space where $k = min(m, n)$

[TRUE/FALSE]

The Ford-Fulkerson algorithm can be used to find the maximum flow through a flow network that contains cycles.

Student ID:

2) 9 pts **Select all correct answers!** 3 pts each. No partial credit

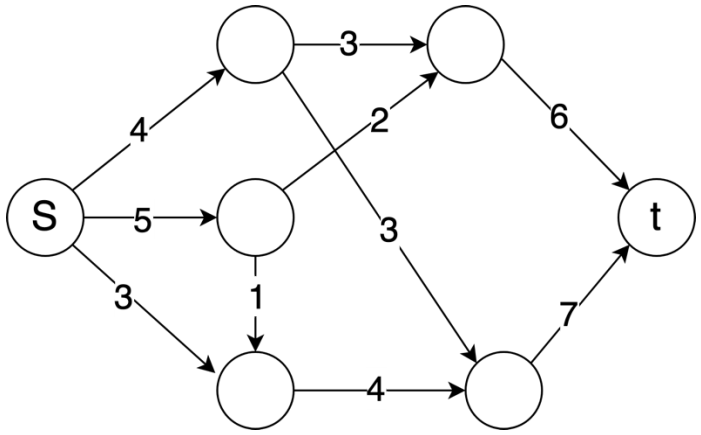
I. Which of the following statements is true about an algorithm that runs in pseudo-polynomial runtime?

(a) Its runtime includes both polynomial and logarithmic terms.
(b) Its runtime is polynomial with at least one of the polynomial terms being a numeric value of the input
(c) The runtime is polynomial in terms of the number of bits in the input.
(d) The runtime is exponential in terms of the number of bits in the input.

II. Which of the following statements is/are True about the Ford-Fulkerson algorithm?

(a) The algorithm repeatedly finds augmenting paths in the residual graph until no more exist.
(b) The algorithm may not terminate if the capacities are irrational numbers.
(c) It may converge to an incorrect value of max flow for non-integer edge capacities.
(d) For integer edge capacities, it terminates in at most v(f) iterations, where v(f) is the maximum flow value.
(e) The capacity values will not influence the running time of the Ford-Fulkerson algorithm, as long as they are positive integers.

III. What is the capacity of the minimum s-t cut in the following flow network?

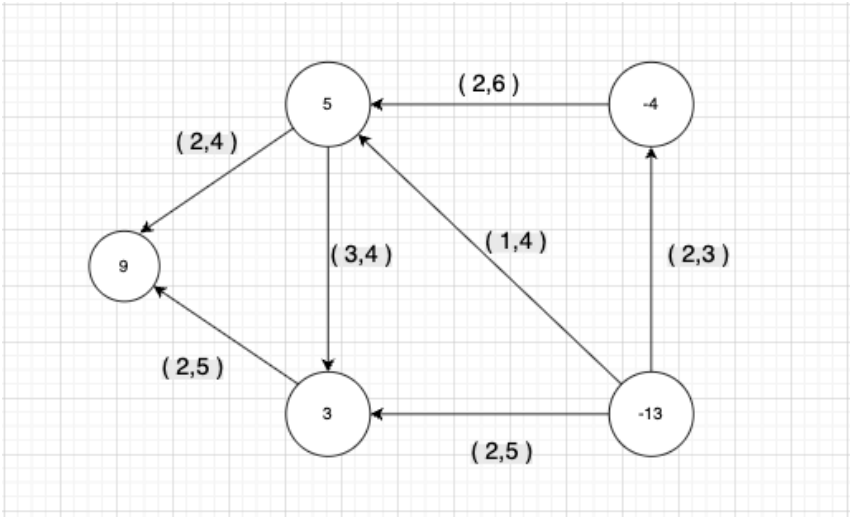


a) 13
b) 12
c) 11
d) 10

3) 18 pts
In the network G below, the demand values are shown on the vertices. Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge.

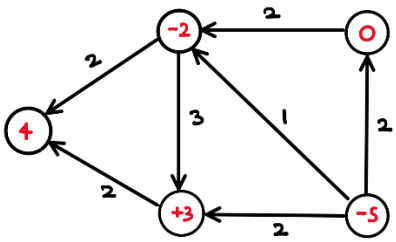
Student ID:

·Answer the questions below to determine if there is any feasible circulation in this graph. You need to show all the steps unless mentioned otherwise.



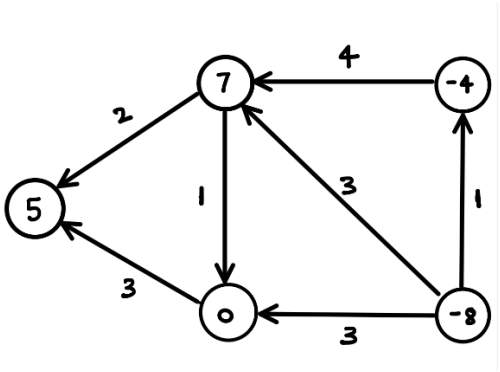
(a) (8 points) Reduce the feasible circulation with lower bound problems to a feasible circulation problem without lower bounds.
(b) (6 points) Reduce the feasible circulation problem obtained in part a to a maximum flow problem in a Flow Network.
(c) (4 points) Find max flow in the flow network found in step b. Use this solution to explain whether or not there exists a feasible circulation in G. (No need to show detailed steps to obtain the max flow value.)

Solution:
(a) $f_0(e) =$ assign each lower bounds to respective edges as follows, where each node represents demand imbalance computed using : $L_v = f^{in}_{(v)} - f^{out}_{(v)}$.

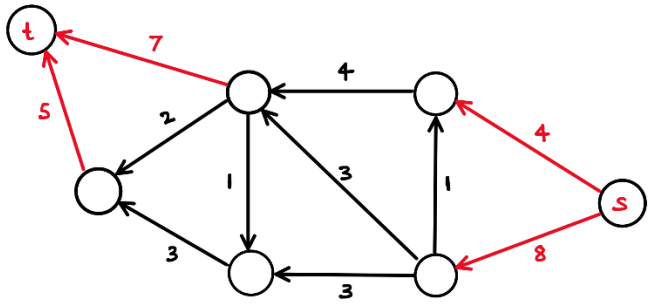


We can create circulation graph without lower bounds using above as follows. The demand for this updated graph is, $D = 12$

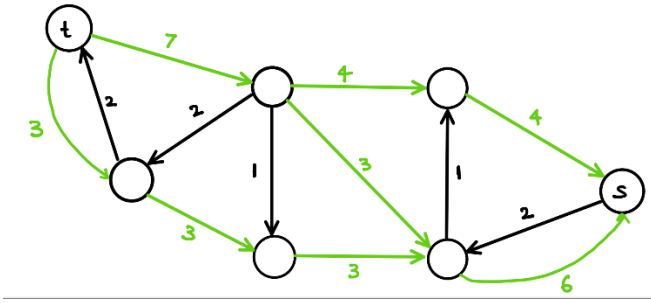
Student ID:



(b)



(c) The max-flow above graph is 10. Since $D \neq 10$, feasible circulation in graph G is not possible



Student ID:

4) 18 pts

A company has n distinct roles to fill and m applicants available. Each applicant i is qualified for a subset of the roles S_i. Each role can be assigned to at most one applicant, and each applicant can take at most one role. Additionally, a subset P of applicants have very little work experience, so to meet their guidelines, the company must make sure that no more than k ($k<m$) of these applicants are hired. Similarly a subset Q of applicants have a lot of work experience and potential to take leadership in future - the company's guideline is to ensure at least l ($l<m$) of these applicants are hired. (Naturally, P and Q are disjoint subsets).

a) Design a network flow algorithm to find an assignment of applicants to roles that maximize the number of roles filled and meets the company guidelines. Clearly describe the construction of the flow network, including the nodes, edges, and capacities. (12 pts)
b) Prove that the assignment obtained from your algorithm corresponds to a valid matching between applicants and roles that meets all the constraints given above. (6 pts)

1- Create a dummy node d and connect s to d with capacity k and connect d to all applicants with little experience with an edge of capacity 1.
2- Create a dummy node d' and connect s to d' with lower bound l and capacity m, and connect d' to all applicants with a lot of experience with an edge of capacity 1.
3- Connect s to all other applicants (that neither have little no lot of experience) with an edge of capacity 1.
4- Connect each applicant node i to the subset of roles Si they qualify for with capacity 1.
5- Connect each role node to t with an edge of capacity 1
6- Connect t to s with an edge of capacity min(m,n)
7- Solve this circulation problem. If there is no feasible circulation then there no feasible assignment of applicants to roles
8- If there is a feasible circulation, set x to the value of flow going through edge t-s in the feasible circulation found.
9- Now we repeat solving the feasible circulation problem (binary search) with an added lower bound constraint y on the edge t-s where the binary search looks for the highest value of y where x < y < min(m,n) for which we have a feasible circulation in this network.

Note: we cannot just do a binary search between 0 and min(m,n) because if we don't find a feasible circulation at min(m,n)/2 it does not necessarily mean that we should go to min(m,n)/4. The reason for not having a feasible circulation at min(m,n)/2 may be that the flow is not sufficient to meet the lower bound l.

Student ID:

2) We should also accept a linear search starting at min(m,n) and iteratively going down by one until feasible circulation is found.

3) With the linear approach above, one can also simply add demands -x and +x to S,T instead of the T-S edge. See if we get feasible circulation, starting at x= min(m,n), x -= 1 each iteration.

Student ID:

5) 20 pts

Suppose you are visiting a new city and planning to book hotels to stay at for D days. There are 3 hotels on a street (lined up next to each other as hotel 1,2,3) where the city government has regulated very low rates to encourage tourism. To prevent visitors from exploiting these rates for long stays, there is a mandated rule that you can **not** book a hotel for two days in a row. So you have to go stay in a different hotel each day. To reduce moving hassles, you want to move only to an adjacent hotel each day (e.g. you would not go directly to hotel 1 from hotel 3 etc.) The rates of the three hotels on day *d* are given by R1(d), R2(d), R3(d) respectively (*d* = 1, 2, ..., D). Design an efficient algorithm to find your minimum hotel expenses for D days, given the constraints above.

a) Define in plain English the subproblems that your dynamic programming algorithm solves. (4 pts)

Subproblems: Opt[i,d] = Min expenses for staying days 1, …, d when the last day (d^{th} day) is stayed at hotel i

b) Define a recurrence relation that expresses the optimal solution (or its value) of each subproblem in terms of the optimal solutions to smaller subproblems. (6 pts)

Recurrence:
Opt[1, d] = R1(d) + Opt[2, d-1]
Opt[2, d] = R2(d) + min(Opt[1, d-1], Opt[3, d-1])
Opt[3, d] = R3(d) + Opt[2, d-1]

c) Using the recurrence formula in part b, write pseudocode using iteration to find the maximum value. Make sure you specify base cases and their values. (6 pts)

Iteration…
Final answer = Min (Opt[1, D] , Opt[2, D] , Opt[3, D])

d) What is the time complexity of your solution? (2 pts)
O(D)

e) Is your algorithm efficient? Provide justification (2 pts)
D represents the size of the input arrays R1, R2, and R3. So the complexity is polynomial time WRT the size of input and therefore the algorithm is efficient.

Student ID:

6) 15 pts
You are given eight jobs, each with a start time, a finish time, and a profit (weight):

Job	Start	Finish	Profit
1	1	4	5
2	6	10	2
3	5	9	3
4	4	7	4
5	3	8	6
6	0	6	8
7	3	5	1
8	8	11	4

The goal is to choose a subset of mutually compatible jobs (no time overlap) that can be completed with maximum total profit.

a) Define the subproblems and write a recurrence relation that expresses the value of the optimal solution in terms of the optimal solutions to smaller subproblems. (4 pst)

b) Numerically solve the problem using iteration, i.e., tabulation and show the value of the optimal solution. Show all your work. (7 pts)

c) Use the solution in part b to find the optimal solution, i.e. the subset of jobs that achieves the optimal value. Show all your work. (4 pts)

Solution 1:

a) Sort given jobs based on their finish time in ascending order
   Prev(i) = jobs that finish earlier than job i and are compatible with job i
   OPT(i) = best total profit among schedules that ends at job i
   OPT(i) = max_{c in prev(i)} OPT(c) + p_i

b) Sorted jobs (with updated job indices in later context) :

Job	Start	Finish	Profit
1(1)	1	4	5
7(2)	3	5	1
6(3)	0	6	8
4(4)	4	7	4
5(5)	3	8	6
3(6)	5	9	3
2(7)	6	10	2
8(8)	8	11	4

   OPT(1) = 5
   OPT(2) = 1
   OPT(3) = 8

OPT(4) = OPT(1) + 4 = 9
OPT(5) = 6
OPT(6) = OPT(1) + 3 = 8
OPT(7) = OPT(3) + 2 = 10
OPT(8) = OPT(4) + 4 = 13

OPT value = max_i OPT(i) = 13

c) We run the top down pass to get the subset.
The solution is found at Opt(8) so 8 goes into the subset. Opt(8) is the result of OPT(4) + 4 (as opposed to Opt(1/2/3) +4), so 4 goes into the subset. Then, Opt(4) is the result of OPT(1) + 4, so 1 goes into the subset. Opt(1) does not have any elements in Prev(i), so we end, giving us the answer {1,4,8}.

Solution 2:

a) Sort given jobs based on their finish time in ascending order
Prev(i) = the last job that finish earlier than job i and is compatible with job i
OPT(i) = best total profits achievable from first i jobs
OPT(i) = max{OPT(i-1), OPT(prev(i)) + p_i}

b) Sorted jobs (with updated job indices in later context) :

Job	Start	Finish	Profit
1(1)	1	4	5
7(2)	3	5	1
6(3)	0	6	8
4(4)	4	7	4
5(5)	3	8	6
3(6)	5	9	3
2(7)	6	10	2
8(8)	8	11	4

OPT(1) = 5
OPT(2) = OPT(1) = 5
OPT(3) = max{OPT(2), p3} = 8
OPT(4) = max{OPT(3), OPT(1) + 4} = max{8, 9} = 9
OPT(5) = max{OPT(4), p5} = max{9, 6} = 9
OPT(6) = max{OPT(5), OPT(2) + 3} = max{9, 8} = 9
OPT(7) = max{OPT(6), OPT(3) + 2} = max{9, 10} = 10
OPT(8) = max{OPT(7), OPT(5) + 4} = max{10, 13} = 13

c) OPT = OPT(8) = 13
Subset = {1,4,8} via top-down pass. The solution is found at Opt(8) which is the result of OPT(5) + 4 (as opposed to Opt(7)), so 8 goes into the subset and 5 is examined next. Opt(5) equals Opt(4) so 5 is skipped and 4 examined next. Then, Opt(4) is the result of OPT(1) + 4, so 4 goes into the subset and 1 examined next. Opt(1) is obtained by including job 1 so 1 goes into the subset, and no previous job to jump to.

Student ID:

Student ID:

Additional Space

Student ID:

Additional Space