# CS570 Fall 2021: 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:

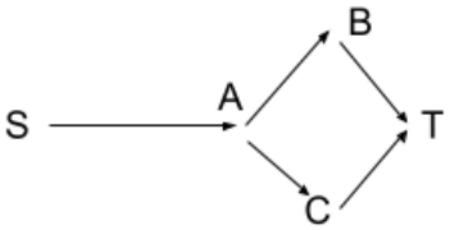
1. This is a 2-hr exam. Open book and notes and internet access. But no internet communications through social media, chat, or any other form is allowed.
2. If a description to an algorithm or a proof is required, please limit your description or proof to within 150 words, preferably not exceeding the 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.
8. This exam is printed double sided. Check and use the back of each page.

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

**[ TRUE/FALSE ] (2 pts)**
If all edge capacities in a Flow Network are integer multiples of 5 then considering any max flow F in this network, flow over each edge due to F must be an integer multiple of 5.

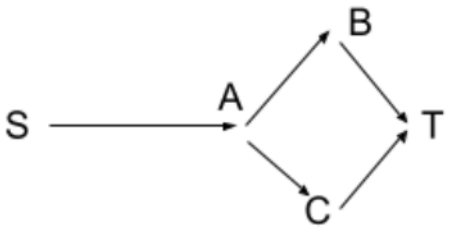
False. There will always exist a flow with flow over each edge being a multiple of 5, but this may not be true of ANY flow.

Counter - example as shown on the side. Let all capacities be 5. Let f(SA) = 5, f(AB) = f(AC) = 2, f(BT) = f(CT) = 3. Then, f is a max flow.




**[ TRUE/FALSE ] (2 pts)**
In a Flow Network G, if the capacity of any edge on a min cut is increased by 1 unit, then the value of max flow in that network will also increase by 1 unit.
Note: an edge on the min cut (A, B) refers to an edge that carries flow out of A (where the source S is) and into B (where the sink T is).

False. As counter - example, consider the network as shown on the side. Let all capacities be 5. The max flow here is of value 5. Consider the cut having AB, AC as the cut-edges. Increasing one of these to a capacity of 6 still keeps max flow at 5.

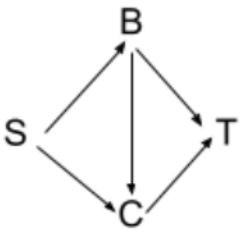



**[ TRUE/FALSE ] (2 pts)**
In a Flow Network with maximum flow value v(F)>0, and with more than one min cut, decreasing the capacity of an edge on any of the min cuts will reduce the value of max flow.
Note: an edge on the min cut (A, B) refers to an edge that carries flow out of A (where the source S is) and into B (where the sink T is).

True. Suppose the capacity of an edge in a min-cut (A,B) is decreased from c to c'. Initially, v(F) must have been c. Now, the max flow must be bounded by the new size of cut (A,B), i.e., c', thus, it does decrease.

**[ TRUE/FALSE ] (2 pts)**
Consider the Ford Fulkerson algorithm and the residual graph used at each iteration. If the Flow Network has no cycles (i.e. the network is a directed acyclic graph), we will not need to include backward edges in the residual graph to achieve max flow.

False. Consider the counter-example shown on the side. This is a DAG (no cycles). Let all capacities be 5. Suppose FF chooses path SBCT in the first step. If back-edges are not added, the residual graph will have T disconnected from S at this stage and terminate - at flow 5. Allowing the back edges however, we get path SCBT as step 2 making the actual max flow of 10.




**[ TRUE/FALSE ] (2 pts)**
Suppose an edge e is not saturated due to max flow f in a Flow Network. Then increasing e's capacity will not increase the max flow value of the network.
True. Since e is not saturated, e is not a cut-edge for any min-cut. Thus, increasing the capacity of e, does not affect the size of any min-cut and hence the max-flow value

remains the same.

**[ TRUE/FALSE ] (2 pts)**
The time complexity of any dynamic programming algorithm with $n^2$ unique subproblems is $\Omega(n^2)$.
True.

**[ TRUE/FALSE ] (2 pts)**
The Bellman-ford algorithm may not be able to find the shortest simple path in a graph with negative cost edges.

True.

**[ TRUE/FALSE ] (2 pts)**
If the running time of an algorithm can be represented as a polynomial in terms of the number of bits in the input, then the algorithm is considered to be efficient.

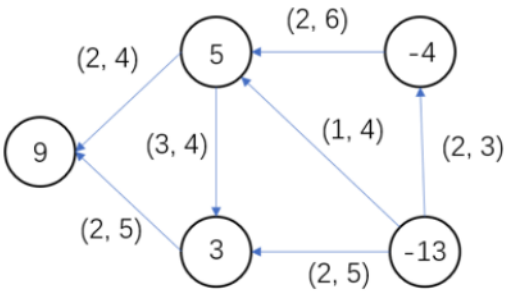
True.

**[ TRUE/FALSE ] (4 pts)**
Recall the coin change problem from lecture where we are trying to pay amount m using the minimum number of coins. We presented two attempts to solve this problem in class, one based on the greedy approach and one using dynamic programming. If coin denominations are limited to 1, 3, and 4 these two approaches will result in the same number of coins to pay any amount m.

False. DP always gives the optimal solution. Greedy does not for m = 6.

2) 20 pts

In the network G below, the demand values are shown on vertices (supply value if negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. You need to show all your steps.




a. Reduce the Feasible Circulation with Lower Bounds problem to a Feasible Circulation problem without lower bounds. (8 pts)

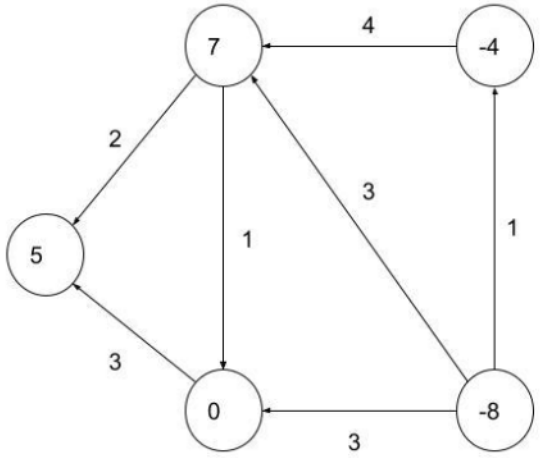
Solution:

Steps to follow :- 1) Assign flows to edges to satisfy lower bounds.

2) At each node v, $L_v$ = (f_in - f_out), followed by D' = D - $L_v$

3) At each edge, cap = cap - LB

Resultant network:




Rubrics:

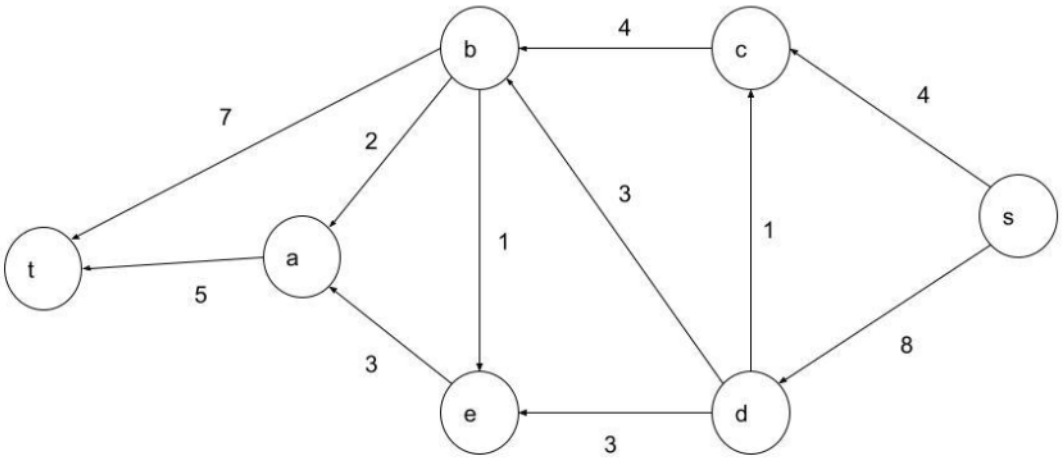
(4 Pts) : If all edge values: (Upper_bound - Lower_bound) is correctly calculated

(4 Pts): If all Node Values (D' = D - (f_in - f_out)) is correctly calculated

(-2 Pts) if no steps explained.

b. Reduce the Feasible Circulation problem obtained in *part a* to a Maximum Flow problem in a Flow Network. (8 pts)

Solution:
The Max-Flow graph is as follows:




Rubrics:

Add S and T (2 pts)
Connect S with correct nodes (2 pts)
Connect T with correct nodes (2 pts)
Give correct capacities to edges (2pts)

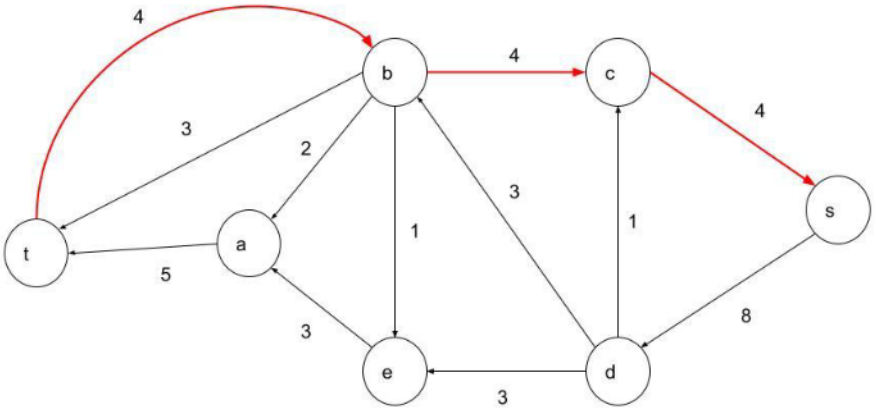
c. Using the solution to the resulting Max Flow problem explain whether there is a Feasible Circulation in G. (4 pts)

Solution:

Candidate Solution 1:

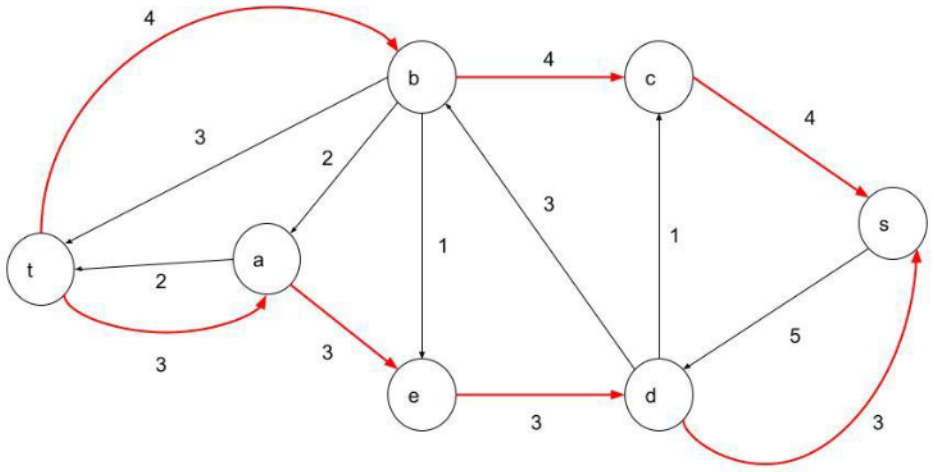
First Augmenting Path: s -> c -> b -> t with flow = 4

Residual Graph 1:



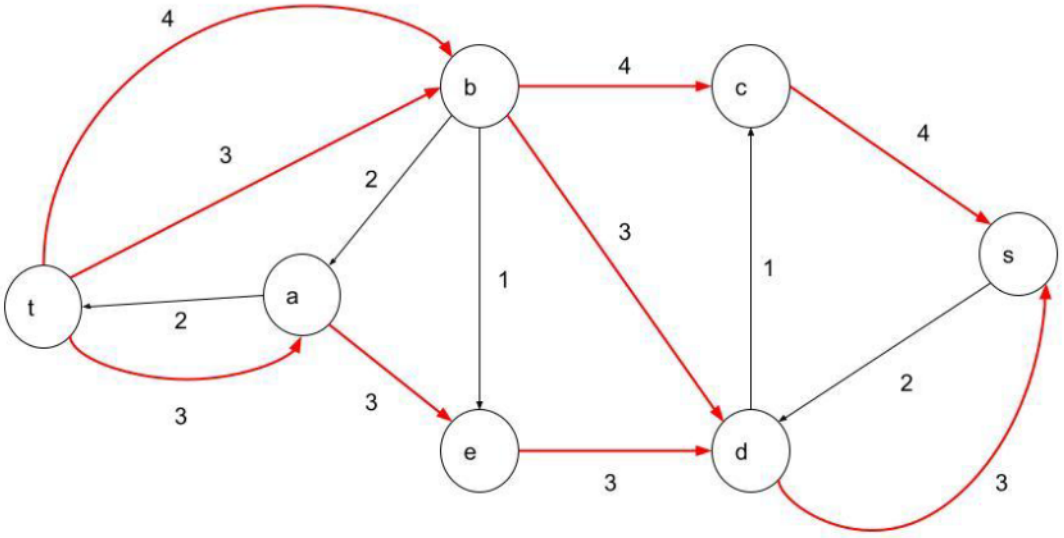

Second Augmenting Path: s -> d -> e -> a -> t with flow = 3

Residual Graph 2:



Third Augmenting path: s -> d -> b -> t with flow 3

Residual Graph 3:




Candidate Solution 2:
First Augmenting Path: s -> c -> b -> t with flow = 4
Second Augmenting Path: s -> d -> b -> a -> t with flow = 2
Third Augmenting Path: s -> d -> b -> t with flow = 1
Fourth Augmenting Path: s -> d -> e -> a -> t with flow = 3

Candidate Solution 3:
First Augmenting Path: s -> d -> e -> a -> t with flow = 3
Second Augmenting Path: s -> d -> b -> a -> t with flow = 2
Third Augmenting Path: s -> d -> c -> b -> t with flow = 1
Fourth Augmenting Path: s -> c -> b -> t with flow = 3
Fifth Augmenting Path: s -> d -> b -> t with flow = 1

Max- Flow = 10

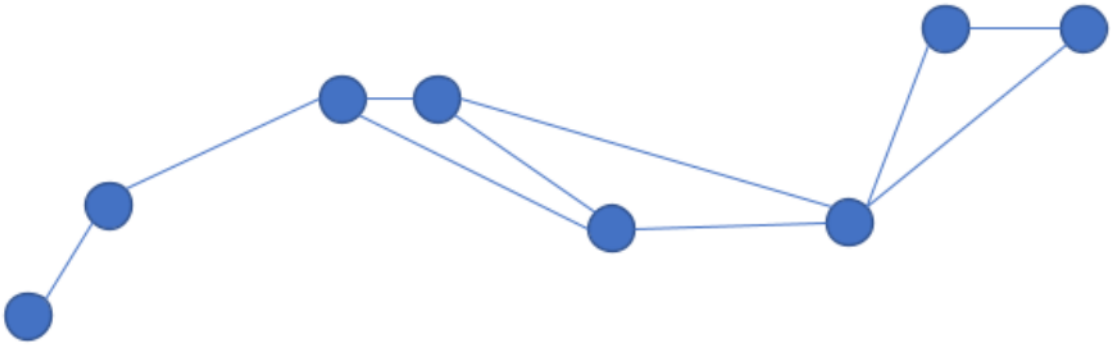
Since, the value of Max-Flow is less than the total demand value D=12, there is **No Feasible solution in the circulation network, and therefore there is no feasible circulation in the circulation with lower bounds network.**

Rubrics:
2 pts: Finding correct Max-Flow and presenting their appropriate residual graphs according to the sequence of augmenting paths that the student has chosen

1 pt: Correctly concluding “No Feasible Circulation”
1 pt: Reasoning behind “No Feasible Circulation”

3) 20 pts

In the last 570 exam, there were nearly a thousand students that were ready to take the exam in person at 8 different test locations (L1..L8). There were Si students assigned to each room i. The copy center made some big mistakes and instead of delivering Si papers they delivered Ri papers to each room i, where Ri was higher than Si for some rooms and lower for some others. TAs then had to rush and come up with a solution to redistribute the excess papers at each test location to those test locations that had a shortage. And because the time was short, they ruled out sending papers directly between test locations that were far apart from each other on campus. But they did not rule out sending papers indirectly through other test locations. (For example, L1 and L3 in below graph were considered far apart from each other but papers could still be redistributed between them through L2). So with these considerations in mind, they ended up with a connected graph that looked like this:



a) Assuming that the total number of papers delivered was at least equal to the total number of papers required, design a network flow based algorithm to determine how many papers had to be sent (and in which direction) on each edge in the above network in order to supply each room with the required papers. You need to describe exactly how you reduce this problem to a network flow problem. (14 pts)

Solution 3a):

Across all the candidate solutions below, the common steps initially are

1. Use the locations as nodes in the flow network.
2. Convert each undirected edge in the given graph to 2 directed edges in opposite directions in the flow network.
3. No constraints on how many papers MUST be carried along any such edge, so no lower bound. Capacity should be a suitable high value: Infinity (= conservative safest upper bound) OR Summation of Si (= papers needed in total) OR Summation of Ri (= papers available in total) OR $\sum_{\{i \text{ where } Si-Ri>0\}} (S_i - R_i)$ (= sum of shortages of papers) OR $\sum_{\{i \text{ where } Ri-Si>0\}} (R_i - S_i)$ (= sum of surplus of papers). The flow on an edge between two locations can be at most any of these upper-bounds.

Candidate Solution 1: Circulation with lower bounds:

1. Add a source S and edges S -> $L_i$ for each location. Each such edge corresponds to the papers sent from the copy center. Since this is exactly $R_i$, set it's LB/cap to $R_i/R_i$.
2. Add a sink T and edges $L_i$ -> T for each location. Each such edge corresponds to the papers finally retained at location $L_i$. This needs to be at least $S_i$ , thus that's the LB. It

could be much more; a capacity of as low as max {S_i, R_i} works (a safe upper bound is simply a capacity of infinity)

3. Add a demand value of $\sum$ R_i at T and $-\sum$ R_i at S. Alternatively (instead of adding the demands at S,T), adding a T -> S edge suffices. (This is not always equivalent, but works here since the net S-T flow value is fixed due to the LB/cap values of $R_i$/$R_i$ on the edges out of S). The capacity of this back-edge has to be at least $\sum$ R_i.

Solution 2: Relative to solution 1, The lower bounds on the edges from S could be removed by adjusting the capacities/demands around. Consequently,, we have no source, but do have a sink, and each $L_i$ has a demand of '- R_i' and they are all connected to the sink with demand $\sum$ R_i

Solution 3: Relative to solution 2, , we can get rid of the lower bounds on edges $L_i$ -> T. In this case, we have (In addition to the common steps laid out in the beginning) a demand of Si - Ri at location Li and sink T with a demand (sum of Ri - sum of Si). We have edges $L_i$ -> T of capacity $R_i$ - $S_i$ wherever it is positive (but okay to add edges from all the $L_i$ -> T and okay to have bigger capacities).

Solution 4: The circulation network from Solution 3 can be further converted to a simple flow network by getting rid of the demands and adding a super source S* and a super sink T*. S* must connect to all $L_i$ where $R_i$ - $S_i$ is positive, with as much edge capacity. Each $L_i$ with positive $S_i$ - $R_i$ connects to T* with as much capacity and T connects to T* with capacity (sum of Ri - sum of Si). This network must have a max flow of $\sum_{\{i \text{ where } Ri-Si>0\}} (R_i - S_i)$.

Solution 5: A modified version of Solution 4 can get rid of T and all the edges coming in and out of it. This network must have a max flow of value $\sum_{\{i \text{ where } Si-Ri>0\}} (S_i - R_i)$ (note this crucial difference with respect to max flow value of Solution 4).

Solution 6: Create a source S, and connect S -> $L_i$ nodes with capacity $R_i$ and create a sink T, and connect each $L_i$ -> T with capacity $S_i$. This network must have a max flow of value sum (Si)

Solution 7: This one has a different structure with two partitions (L and L') EACH having nodes for ALL 8 locations. Create S, and edges from S -> $L_i$ nodes with capacity $R_i$. Create a sink T, and connect $L'_i$ -> T with capacity $S_i$. Each $L_i$ and its copy $L'_i$ should have edges both ways. But for i \ne j, $L_i$ -> $L'_j$ can be an edge in just this one direction. ALL these edges between L and L' must have the same high capacity as outlined in the beginning of this solution. This network must have a max flow of value sum (Si)

**Final answer (i.e. paper redistribution scheme) required in each solution: Compute the feasible circulation (in solutions 1/2/3) or max flow (in solutions 4/5/6/7) F. From location $L_i$ to an adjacent $L_j$ send F($L_i$ -> $L_j$) - F($L_j$ - $L_i$) papers (or in the reverse direction, whichever gives a positive number).**

Rubrics 3a:

- -2 points for each incorrect/missing demand/LB/capacity/node/edge

- -2 pts: if the final algorithm description of how to compute redistribution solution from the constructed network is missing.
- AT MOST half the points if the fundamental structure of the network is wrong (partial credit subjective to how close the proposed solution is)

a) Prove that you solution is correct (6 pts)

Solution 3b):

Here's the proof for solution 1 (Circulation without lower bounds)

Reduction Claim: **If there is a feasible circulation in the resulting network then it gives a feasible redistribution of papers across the exam rooms. (The solution will actually always exist because we have enough papers and no limits on capacities that stop us from sending papers from one location to another.)**

Proof:

Claim 1) If we have a feasible redistribution of papers to exam rooms we will have a feasible circulation in the circulation network constructed:

- Send flow on each edge (in the appropriate direction) equal to the number of papers that are sent out from room to room.
- Send flow from S to each room of $R_i$
- Send flow from each room to T equal to the final number of papers in that room.
- This will give us a feasible circulation in the network since
  - All lower bound constraints on flow are met (each room got enough papers)
  - All capacity constraints are met (edge capacities were set high enough to allow for movement of any extra papers between locations)
  - All demand conditions are met (especially at T) since redistribution of papers will not cause us to lose or add any flow at any node.

Claim 2) If we have a feasible circulation we can find a feasible circulation of papers to exam rooms

- Send papers equal to flow sent out from room to room
- This will give us a feasible redistribution of papers since
  - All requirements for papers at exam rooms will be met (since lower bounds on flow from room to T are met)
  - There are no physical limits on how many papers can be transported from room to room

Other proofs follow the same format.
- For candidate solutions 4-7, the claim should say "The max flow value is ___ if and only if …" and not "max flow exists if and only if..."
-A common mistake is an attempt to have a circulation formulation as in Solution 3 but missing the sink therein, and then incorrectly reducing it to max flow which looks as the one in Solution 5 (circulation -> flow conversion is not necessary, i.e., has no credit). (The penalty

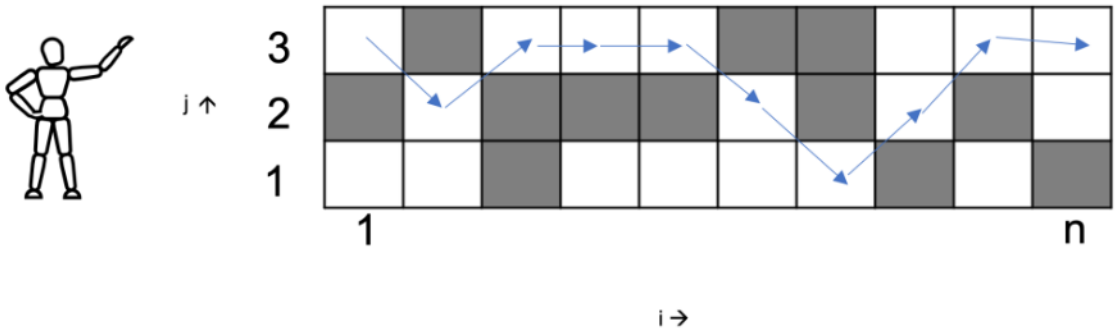
for this error will be reduced from 3.5 to 2.5 IF the claim/final answer is correctly given in terms of the max flow value of the resultant network).

Rubrics 3b:

- **If** the claim is correct,
  - 3 pts: Proof of the Forward claim.
  - 3 pts: Proof of the backward claim.
- Partial credit of at most 2 in total if the claim itself is incorrect (for the proposed solution), but can qualify as a fair attempt..

## 4) 20 pts

Jack has gotten himself involved in a very dangerous game called the octopus game where he needs to pass a bridge which has some unreliable sections. The bridge consists of 3n tiles as shown below. Some tiles are strong and can withstand Jack's weight, but some tiles are weak and will break if Jack lands on them. Jack has no clue which tiles are strong or weak but we have been given that information in an array called BadTile(3,n) where **BadTile (j, i) = 1 if the tile is weak and 0 if the tile is strong.** At any step Jack can move either to the tile right in front of him (i.e. from tile (j, i) to (j, i+1)), or diagonally to the left or right (if they exist). (No sideways or backward moves are allowed and one cannot go from tile (1,i) to (3, i+1) or from (3, i) to (1, i+1)). Using dynamic programming find out how many ways (if any) there are for Jack to pass this deadly bridge. Figure below shows bad tiles in gray and one of the possible ways for Jack to safely cross the bridge alive.




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

OPT(j, i) = The number of ways Jack can reach the block (j, i) safely from the bridge start

OR

OPT(j, i) = The subproblems are the number of ways Jack can reach the end safely starting from block (j, i)

Rubrics:
Missing from block or to block : -1

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

Variant 1:

$$
\begin{aligned}
OPT(j, i) &= 0 && \text{if BadTile}(j, i) = 1; \\
&= OPT(j,i-1) + OPT(j+1,i-1) && \text{if } j = 1; \\
&= OPT(j,i-1) + OPT(j+1,i-1) + OPT(j-1,i-1) && \text{if } j = 2; \\
&= OPT(j,i-1) + OPT(j-1,i-1) && \text{if } j = 3;
\end{aligned}
$$

Variant 2 :
In the second variant, i-1 is replaced by i+1 in each of the terms on the RHS

Variant 3 :

OPT1[i] = 0 if BadTile = 1
= 1 if(i ==1)
= opt2[i-1] + opt1[i-1] otherwise

OPT2[i] = 0 if BadTile = 1
= 1 if(i ==1)
= opt2[i-1] + opt1[i-1] +opt3[i-1] otherwise

OPT3[i] = 0 if BadTile = 1
= 1 if(i ==1)
= opt2[i-1] +opt3[i-1] otherwise

Rubrics:
If only j=2 case is written and other cases are not mentioned : -2 (-1 for each case missed)
If recurrence relation is partially correct : -4
Partially wrong recurrence relation -> adding +1 at each step : -2

c) Using the recurrence formula in part b, write pseudocode using iteration to compute the total number of ways to safely cross the bridge. (5 pts)
Make sure you specify
- base cases and their values (2 pts)
- where the final answer can be found (e.g. opt(n), or opt(0,n), etc.) (1 pt)
For Variant 1:

Given BadTile (3,n)
Initialize OPT(3,n) with 0 for each element
OPT(1,1) = 1 if BadTile (1,1) equals to 0, else 0
OPT(2,1) = 1 if BadTile (2,1) equals to 0, else 0
OPT(3,1) = 1 if BadTile (3,1) equals to 0, else 0

```
for i in  2, 3,4, …, n:
        for j in 1, 2, 3:
                if BadTile(j, i) equals to 1:
                        OPT(j, i) = 0
                if j equals to 1:
                        OPT(j, i) = OPT(j, i-1) + OPT(j+1, i-1)
                if j equals to 2:
                        OPT(j, i) = OPT(j,i-1) + OPT(j+1,i-1) + OPT (j-1,i-1)
                if j equals to 3:
                        OPT(j, i) = OPT(j,i-1) + OPT (j-1,i-1)

return OPT(1, n)+OPT(2, n)+OPT(3, n)
```

For Variant 2:

Initialize OPT(5,n)

OPT(0,i) = 0 //outside the grid
OPT(4,i) = 0 // outside the grid

For i in 1 to n:
    For j =1 to 3:
        OPT(j, i) = OPT(j,i-1) + OPT(j+1,i-1) + OPT (j-1,i-1)

return OPT(1, n)+OPT(2, n)+OPT(3, n)

If variant 2 is written recurrence would be just j = 2 case

For Variant 3:

Initialize 3 arrays opt1[], opt2[] opt[3];

For i in 2 to n:
    Above recurrence
return OPT(1, n)+OPT(2, n)+OPT(3, n)

If variant 2 is written recurrence would be just j = 2 case

Rubric:
If where final answer can be found is not mentioned : -1
If recurrence relation is not shown or is wrong : -2
If 3 cases are not shown : -2
If base condition are given according to example : -1

d) What is the complexity of your solution? (1 pt)
Is this an efficient solution? (1 pt)

O(N). Yes.

5) 20 pts
Assume a truck with capacity W is loading. There are n packages with different weights, i.e. [w1, w2, … wn], and all the weights are integers. The company's rule requires that the truck needs to take packages with exactly weight W to maximize profit, but the workers like to save their energies for after work activities and want to load as few packages as possible. Assuming that there are combinations of packages that add up to weight W, design an algorithm to find out the minimum number of packages the workers need to load.

**NOTE: For those who assumed n packages to mean "n types of packages", we've still decided to award points for it out of a maximum possible 13 points. (i.e. for any errors in the attempted solution under this mis-interpretation will have reductions out of the 13 max possible score). This variant of the problem is in fact the one asked in the onlinc version of the exam, please refer to it for solutions and rubrics.**

Solution 1:

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

OPT(w,k) = min packages needed to make up a capacity of exactly w, by considering only the first k packages
OR
Same except, considering packages k onwards up to n (add details below)

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

If $w >= w_k$
$OPT(w,k) = \min\{ 1+OPT(w- w_k , k-1) , OPT(w, k-1) \}$
Else
$OPT(w,k) = OPT(w, k-1)$

(No penalty for missing the latter case IF the base cases in 'c)' include $w < 0$.)

c) Using the recurrence formula in part b, write pseudocode using iteration to compute the minimum number of packages to meet the objective. (5 pts)

1. Initialize for base cases (mentioned below for clarity)

2. For w = 0 to W
For k = 0 to n //can switch the inner-outer loops
If not base case: call recurrence
3. Return ans (mentioned below for clarity)

Make sure you specify
- base cases and their values (2 pts)
OPT(w,0) = inf for all w < 0 (up to -W)
OPT(0,0) = 0
OPT(w,k) = infinity for w < 0 and all k. (Not required if the recurrence has the second case to ensure w never takes negative values)
- where the final answer can be found (e.g. opt(n), or opt(0,n), etc.) (1 pt)
OPT[W, n] in the first definition
OPT[W, 1] in the alternate one

d) What is the complexity of your solution? (1 pt)
Is this an efficient solution? (1 pt)

O(nW) pseudo-polynomial run time
This is not an efficient solution

Solution 2:

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

OPT(w,k) = min packages needed to make up a capacity of exactly w, by considering only the first k packages, AND selecting package k for sure.

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

The recurrence captures all cases of j where j was the highest index used before the $k^{th}$ one:

If $w >= w_k$
OPT(w,k) = 1 + min_{j=0 to k-1} OPT(w- $w_k$ , j)
Else
OPT(w,k) = infinity
(No penalty for missing the latter case IF the base cases in 'c)' include w < 0.)

c) Using the recurrence formula in part b, write pseudocode using iteration to compute the minimum number of packages to meet the objective. (5 pts)
4. Initialize for base cases (mentioned below for clarity). Initialize rest to infinity

5. For w = 0 to W
   For k = 0 to n  //can switch the inner-outer loops
   If (not base case): //call recurrence
   For (j = 0 to k-1)
   OPT(w,k) = min{OPT(w,k), 1 + OPT(w- $w_k$ , j)}
6. Return ans (mentioned below for clarity)

Make sure you specify
- base cases and their values (2 pts)
  OPT(w,0) = inf for all w >0
  OPT(0,0) = 0
  OPT(w,k) = infinity for w < 0 and all k. (Not required if the recurrence has the second case to ensure w never takes negative values)
- where the final answer can be found (e.g. opt(n), or opt(0,n), etc.) (1 pt)

Max_k OPT(W, k)

d) What is the complexity of your solution? (1 pt)
Is this an efficient solution? (1 pt)

$O(n^2W)$  pseudo-polynomial run time
This is not an efficient solution