CSCI 570 Homework 4

Due Date: Oct. 29, 2023 at 11:59 P.M.

1. (**Dynamic Programming**) Given n balloons, indexed from 0 to $n-1$. Each balloon is painted with a number on it represented by array $nums$. You are asked to burst all the balloons. If you burst the balloon i you will get $nums[left] \cdot nums[i] \cdot nums[right]$ coins, where left and right are adjacent indices of i. After the bursting the balloon, the left and right then becomes adjacent. Assume, $nums[-1] = nums[n] = 1$ and they are not real therefore you can not burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

   Solution:

   - Let OPT (l, r) be the maximum number of coins you can obtain from balloons l, l+1, . . . , r1, r
   - The key observation is that to obtain the optimal number of coins for a balloon from l to r, we choose which balloon is the last one to burst.
   - Assume that balloon k is the last one you burst, then you must first burst all balloons from l to k 1 and all the balloons from k + 1 to r which are two sub-problems.
   - So we get recurrence relation:

   $$OPT(l, r) =$$

   $$\max_{l \le k \le r} OPT(l, k-1) + OPT(k+1, r) + nums[k] \cdot nums[l-1] \cdot nums[r+1]$$

   - We have the initial condition OPT(l, r) = 0 if $r \le l$.
   - For running time analysis, we in total have $O(n^2)$ and computation of each state takes $O(n)$ time. Therefore, the total time is $O(n^3)$.

   Rubrics (10 points):

   - +2 points for the base case

- +2 points for defining subproblems OPT(l, r)...
- +4 points for recurrence relation
- +2 points for correct time complexity analysis (irrespective of whether the algorithm is correct or not).
- -1 point for every mistake in either of the steps

2. (**Dynamic Programming**) Suppose you are in Casino with your friend, and you are interested in playing a game against your friend by alternating turns. The game contains a row of n coins of values v_i, where n is even. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money you can definitely win if you move first. Analyze the running time of your algorithm.

   Solution:

   - Suppose $OPT(i, j)$ represents the maximum value the user can collect from i-th coin to j-th coin. When we move first, we have two choices to make. Either we choose i-th coin or j-th coin.
   - **Choice 1**: The user chooses the i-th coin with value v_i: The opponent either chooses $(i + 1)$-th coin or j-th coin. The opponent intends to choose the coin which leaves the user with minimum value. The user can collect the value $v_i + min(OPT(i + 2, j), OPT(i + 1, j - 1))$
   - **Choice 2**: The user chooses the j-th coin with value v_j: The opponent either chooses i-th coin or $(j-1)$-th coin. The opponent intends to choose the coin which leaves the user with minimum value. The user can collect the value $v_j + min(OPT(i + 1, j - 1), OPT(i, j - 2))$
   - Continue with a recurrence relation: $OPT(i, j) = max(v_i + min(OPT(i + 2, j), OPT(i + 1, j - 1)), v_j + min(OPT(i + 1, j - 1), OPT(i, j - 2)))$.
   - 2 base cases: $OPT(i, j) = v_i$ and $OPT(i, i + 1) = max(v_i, v_{i+1})$.
   - The running time of this algorithm is $O(n^2)$

   Rubrics (10 points):

   - +2 points for the base case(s)
   - +2 points for defining subproblems OPT(i, j)...
   - +2 points for two choices
   - +2 points for recurrence relation
   - +2 points for correct time complexity analysis (irrespective of whether the algorithm is correct or not).

- -1 point for every mistake in either of the steps

3. (**Dynamic Programming**) 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. Analyze the running time of your algorithm.

   Figure below shows bad tiles in gray and one of the possible ways for Jack to safely cross the bridge alive (See Fig. 1).

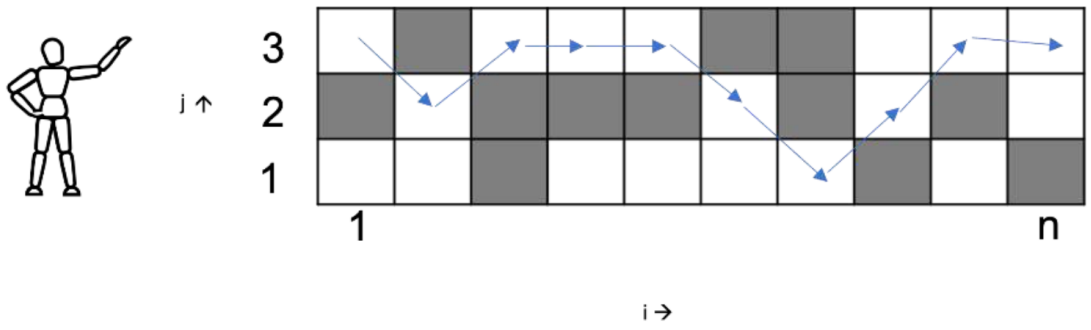

Figure 1

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

OR

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

Rubrics (2 points):

-1 pt: Missing from block or to block.

Variant 1:

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

Variant 2:
In the second variant, $i-1$ is replaced by $i+1$ in each of the terms on the RHS.
Variant 3:

$$OPT1(i) = \begin{cases} 0 & \text{if BadTile}(1,i) = 1 \\ 1 & \text{if } i = 1 \\ OPT2(i-1) + OPT1(i-1) & \text{otherwise} \end{cases} \tag{2}$$

$$OPT2(i) = \begin{cases} 0 & \text{if BadTile}(2,i) = 1 \\ 1 & \text{if } i = 1 \\ OPT2(i-1) + OPT1(i-1) & \\ \quad +OPT3(i-1) & \text{otherwise} \end{cases} \tag{3}$$

$$OPT3(i) = \begin{cases} 0 & \text{if BadTile}(3,i) = 1 \\ 1 & \text{if } i = 1 \\ OPT2(i-1) + OPT3(i-1) & \text{otherwise} \end{cases} \tag{4}$$

Rubrics (3 points):

- 3 pts: Write 3 cases correctly.
- -1 pt: For each case missed.
- -2 pts: If only $j = 2$ case is written and other cases are not mentioned.
- -2 pts: If recurrence relation is partially correct (-1 pt for each

case).

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 $2, 3, ..., n$:
    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 in 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 $OPT1[], OPT2[], OPT3[]$;

for i in 2 to n:

Above recurrence

return $OPT1(n) + OPT2(n) + OPT3(n)$

If variant 3 is written recurrence would be just $j = 2$ case.

Rubrics (4 points):

- 4 pts: Write base case, recurrence relation, 3 cases, and final answer correcltly.
- -1 pt: If where final answer can be found is not mentioned.
- -1 pt: If recurrence relation is not shown or is wrong.
- -1 pt: If 3 cases are not shown or are wring.
- -1 pt: If base condition is not shown or is wrong.

$O(N)$. Yes.

Rubrics (1 point):

- 1 pt: For correct complexity.

4. Given a flow network with the source s and the sink t, and positive integer edge capacities c. Prove or disprove the following statement: if deleting edge e reduces the original maximum flow more than deleting any other edge does, then edge e must be part of a minimum $s-t$ cut in the original graph.

   False. Counter-example:

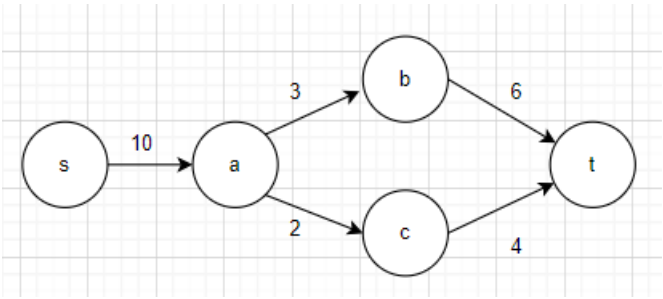

Figure 2

In the above figure, max flow is 5, min-cut is a-b and a-c. However, max-flow is reduced to 0 when we remove edge s-a (which is not a part of the min-cut in the original graph).

5. Given a flow network with the source s and the sink t, and positive integer edge capacities c. Let $s - t$ be a minimum cut. Prove or disprove the following statement: If we increase the capacity of every edge by 1, then $s - t$ still be a minimum cut.

   False. Counter-example:

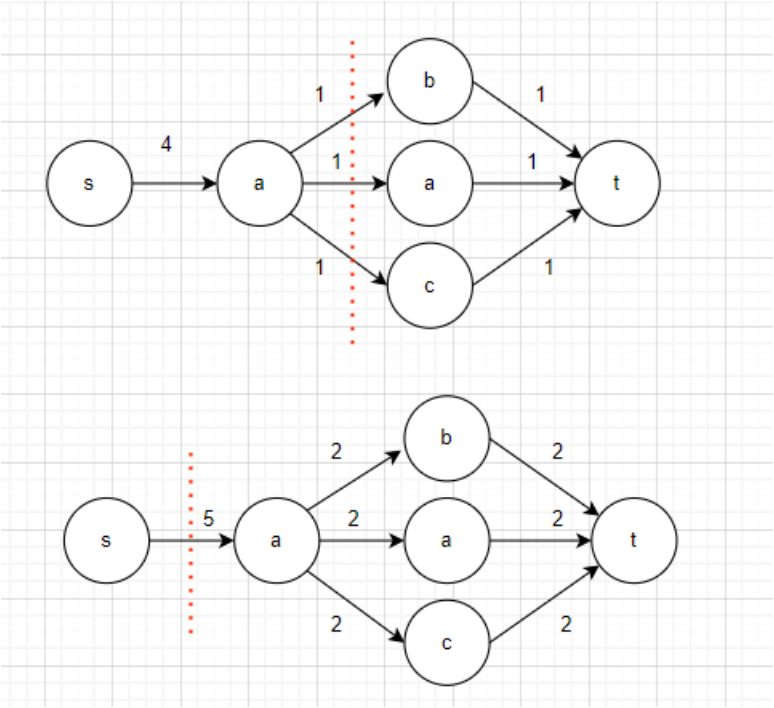

Figure 3

6. (**Network Flow**)

(a) For the given graph G_1 (see Fig. 4), find the value of the max flow. Edge capacities are mentioned on the edges. (You don't have to show each and every step of your algorithm).

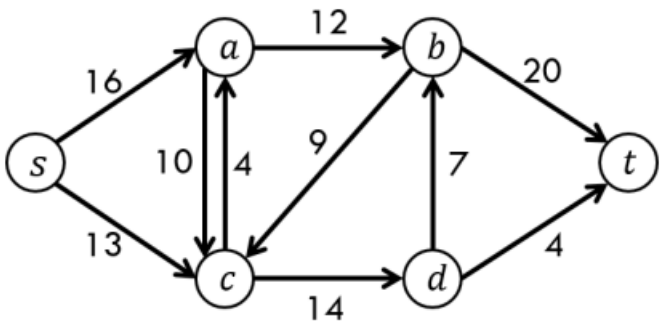

Figure 4: G1

The value of the max flow is 23. See Fig. 5

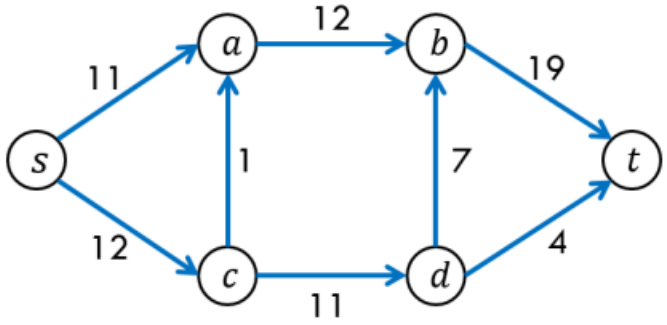

Figure 5

Note: There could be other answers

Solution 2: Find the min-cut (= max-flow) Min cut is edge a-b, d-b and d-t (12 + 7 + 4 = 23)

Rubrics (5 points):

- -0.5 for every edge mistake
- -1 if edge values are correct but the final answer is wrong
- +5 if found using min-cut instead of the residual graph

(b) For the given graph, find the value of the min-cut. (You don't have to show each and every step of your algorithm).

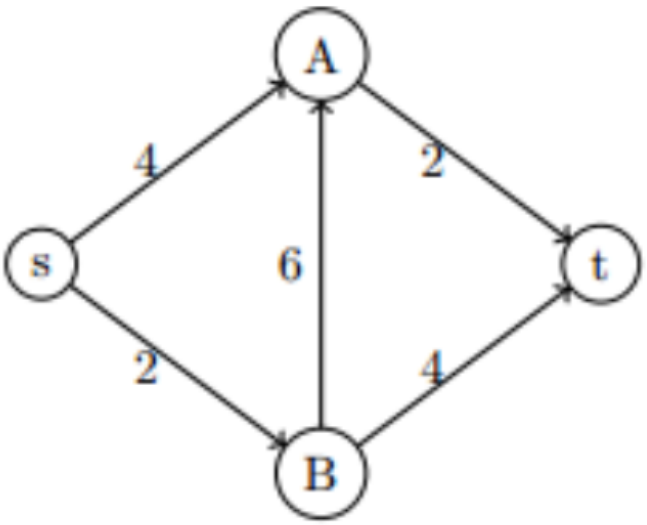

Figure 6

The value of the min cut is 4. See Fig. 7

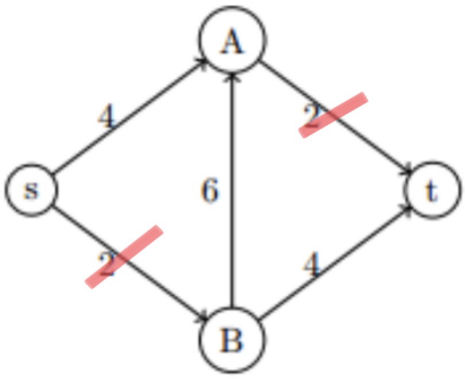

Figure 7

Note: Some possible min-cuts are shown with different colors.

Rubrics (5 points):

- -0.5 for every edge mistake

7. (**Network Flow**) Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. We'll suppose there are n clients, $c_1, c_2, \ldots, c_n$, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations, $b_1, b_2, \ldots, b_k$; the position of each of these is specified by (x, y) coordinates as well. For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a range parameter R which means that a client can only be connected to a base station that is within distance R. There is also a load parameter$L$ which means that no more than$L$ clients can be connected to any single base station. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station. Prove the correctness of the algorithm.

   Solution:

   Connect each client b_i to a base station c_i if that station is within the range R. Assign capacity 1 to those edges. Let s be a source vertex and t be a sink vertex. For every client vertex, add an edge (s, c_i) of capacity 1. For every base station vertex, add an edge (b_i, t) of capacity L.

   Claim: The problem has a solution if and only if the network has a max-flow of value n.

   Proof. Assume that there is a solution. It means that every client is connected to a correspondent base station. So we can push a flow of 1 from the source to each client. On the edges between clients and stations, we assign a flow of 1 or 0, depending whether a client is connected by a particular station or not. On the edges between stations and the sink, we assign a flow value equal to the number of clients covered by that station. This is possible, since we have a valid solution. It follows the max-flow is n.

   Conversely, assume there is a max-flow of value n. This means that each client will get a flow of one unit. We also observe that no base station is overloaded due to the capacity condition L.

Rubrics (10 points):

- 5 pts: Correct construction of the network flow graph.
- 2 pts: Writing correct forward proof.
- 2 pts: Writing correct backward proof.
- 1 pt: Writing the correct claim and mentioning ford-Fulkerson
- -1 pt: Every mistake in edge capacity assignment.

8. (**Network Flow**) You are given a flow network with unit-capacity edges: it consists of a directed graph $G = (V, E)$ with source s and sink t, and $u_e = 1$ for every edge e. You are also given a positive integer parameter k. The goal is delete k edges so as to reduce the maximum $s - t$ flow in G by as much as possible. Give a polynomial-time algorithm to solve this problem. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s - t$ flow in the graph $G' = (V, E - F)$ is as small as possible. Give a polynomial-time algorithm to solve this problem.

Solution:

(a) Assume the value of max-flow of given flow network $G(V, E)$ is g. By removing k edges, the resulting max-flow can never be less than $g - k$ when $|E| \geq k$, since each edge has a capacity 1.

(b) According to max-flow min-cut theorem, there is an $s - t$ cut with g edges. If $g \leq k$, then remove all edges in that $s - t$ cut, decreasing max-flow to 0 and disconnecting 's' and 't'. Else if, $g > k$, then remove 'k' edges from g, and create a new cut with $g - k$ edges.

(c) In both the cases the maxflow cannot be decreased any further thus giving us the maximum reduction in flow.

(d) The algorithm has polynomial run-time. It takes polynomial time to compute the minimal-cut and linear time in 'k' to remove 'k' edges.

Rubrics (10 points):

- 7 pts: Correct Algorithm proposal.
- 3 pts: Mentioning polynomial time.
- -1 pt: If time to remove k edges is not mentioned.

9. (**Network Flow**) Counter Espionage Academy instructors have designed the following problem to see how well trainees can detect SPY's in an $n \times n$ grid of letters S, P, and Y. Trainees are instructed to detect as many disjoint copies of the word SPY as possible in the given grid. To form the word SPY in the grid they can start at any S, move to a neighboring P, then move to a neighboring Y. (They can move north, east, south or west to get to a neighbor.) The following figure shows one such problem on the left, along with two possible optimal solutions with three SPY's each on the right (See Fig. 3). Give an efficient network flow-based algorithm to find the largest number of SPY's.
Note: We are only looking for the largest **number** of SPY's, not the actual location of the words. No proof is necessary.

We construct a Flow Network as follows, with all edges having capacity 1:

(a) Create one layer of nodes for all the S's in the grid. Connect this layer to a source vertex s, directing edges from s to S.

(b) Create two layers of nodes for all the P's in the grid. Connect the first layer to the S's based on whether or not they are adjacent in the grid, directing edges from S to P. Also, connect the first layer to the second layer by connecting the nodes that correspond to the same location. In other words, we are representing P's as an edge with capacity 1.

(c) Create a layer of nodes for all the Y's in the grid. Connect these to a sink vertex t, directing edges from Y to t. Also, connect them to the second layer of Ps based if the P and Y are adjacent in the grid, directing edges from P to Y.

Claim/Answer: Value of Max Flow in this Flow Network will give us the maximum number of disjoint SPY's.

Note that we don't need to represent nodes S or P with edges of capacity 1 since there are edges of capacity 1 going into S's and edges of capacity 1 leaving Y's which will force these letters to be picked only once.

Rubrics (10 points):
Full points even if the solution splits every node into two nodes and connects respective pair with an edge of unit capacity (to ensure each node is picked only once)

- 2 Points: If solution satisfies the constraint of picking S's only once.
- 2 Points: If solution satisfies the constraint of picking P's only once. (Make sure solution has split P's into two vertices, otherwise the same P can be selected more than once if that P is adjacent to 2 S's and 2 Y's.)
- 2 Points: If solution satisfies the constraint of picking Y's only once.
- 2 Points: If edges are added only between the nodes which are adjacent, *i.e.*, if the solution satisfies the constraint of picking only adjacent S, P, and Y.
- 2 Points: For the correct claim/answer, i.e., if solution associates largest number of disjoint SPY's to the max flow in the network.

10. (**Network Flow**) USC students return to in person classes after a year long interval. There are k in-person classes happening this semester, $c_1, c_2, ..., c_k$. Also there are n students, $s_1, s_2, ..., s_n$ attending these k classes. A student can be enrolled in more than one in-person class and each in-person class consists of several students.

    Each student s_j wants to sign up for a subset p_j of the k classes. Also, a student needs to sign up for at least m classes to be considered as a full time student. (Given: $p_j \geq m$) Each class c_i has capacity for at most q_i students. We as school administration want to find out if this is possible. Design an algorithm to determine whether or not all students can be enrolled as full time students. Prove the correctness of the algorithm.

    We can solve the problem by constructing a network flow graph and then running Ford–Fulkerson algorithm to get the max flow. If the max flow is equal to nm, then all students can be enrolled as full time students. The setup of the graph is described below.

    (a) Place the n students and k classes as vertices in a bipartite style graph. Connect each student s_j with all the classes in p_j using edges with capacity of 1.

    (b) Place a sink vertex which is connected to all the classes (or alternatively students). Also, place a source vertex which is connected to all the students (or alternatively classes).

    (c) Connect all s_j student vertices to the source (or sink) vertex with capacity of m.

    (d) Connect all c_i class vertices to the sink (or source) vertex with capacity of q_i.

    Claim: The enrollment is feasible if and only if there exists a max flow of nm.
    Proof of Correctness
    Forward Claim: The max flow is nm if the problem has a feasible solution.
    Proof: If there exists a feasible solution for the problem, this means that

all the in-person classes has at max of q_i students and each student was able to sign up for at least m classes. By the law of conservation, the outward flow at the student vertex must be equal to the incoming flow to student vertex. Therefore, all the edges from source to the student vertices will be saturated with value of m, as each of the student was able to sign up for his/her classes. Similarly all the incoming flow at student vertex will be distributed to some of the out going edge with value of 1. This total flow from student vertices to the class vertex will be nm (again, by the law of conservation). And that is will eventually flow to the sink vertices with edges of max capacity q_i. Therefore, if the problem has a feasible solution the the max flow will be nm.

Backward Claim: The problem has a feasible solution if the max flow is nm.

Proof: If the problem has a max flow of nm. This means that along any path from source to sink vertex there is a student who was able to sign up for a particular class. Simply, we just have to follow the flow along a path.

Because the flow is max flow, this means that each of the student vertices receive a flow of m and the outward flow at each of the student vertices will be exactly m. Thus, the student will be able to sign up for at least m classes, which satisfies the constraint of the problem. Also, by the construction of network flow each class can accommodate at most q_i, which satisfies another constraint as the flow along the edge will be at most q_i (that is less than or equal to the capacity). Hence, the problem has a feasible selection if the max flow is nm.

Rubrics (5 points):

- 3 pts: Correct construction of the the network flow graph.
- -1 pt: For each incorrect edge weight.
- 2 pts: Accurate proof of correctness.

11. If there exists a feasible solution to part (a) and all students register in exactly m classes, the student body needs a student representative

from each class. But a given student cannot be a class representative for more than r (where $r < m$) classes which s/he is enrolled in. Design an algorithm to determine whether or not such a selection exists. Prove the correctness of the algorithm. (Hint: Use part (a) solution as starting point).

We can solve the problem by starting from the solution from part (a). We will modify the constructed network flow graph slightly and then will run Ford–Fulkerson algorithm to get the max flow. If the max flow is equal to the number of classes k, then a selection exists otherwise no such selection exists.
The setup of the graph is described below.

(a) Use the same nodes as constructed in Part (a) of the solution.

(b) The course selections (edges between students and courses) for each student will be reduced to what is available in the solution from part (a) (our feasible solution), *i.e.*, we will remove some edges between students and courses that they did not get to sign up for.

(c) The capacity on student to class edges will same remain same (*i.e.*, 1).

(d) Assign 1 as the capacity from classes to sink (or alternatively source) vertex. (As there can only be 1 student representative from each class.)

(e) Assign r capacity for edges between source (or alternatively sink) and student vertices.

Claim: The selection is feasible if and only if there exists a max flow of k.
Proof of Correctness
Forward Claim: The max flow is k if the problem has a feasible selection.
Proof: If there exists a feasible selection for the problem, this means that all the in-person classes had a valid selection and each student was at max selected for r selections. By the law of conservation, the outward

flow at the class vertex must be equal to the incoming flow to class vertex. Therefore, all the edges from sink to the class vertices will be saturated with value of 1, as each of the class had exactly 1 selection. This means that, k was the inward flow at class vertices. (law of conservation). Similarly, the number of selections per student will have the maximum value of r classes. And the total flow from source vertex to the student vertices will be k (again, by the law of conservation). Therefore, if the problem has a feasible solution the the max flow will be k.

Backward Claim: The problem has a feasible selection if the max flow is k.

Proof: If the problem has a max flow of k. This means that along any path from source to sink vertex there is a selection of student from a class. Simply, we just have to follow the flow along a path.

Because the flow is max flow, this means that all the class vertices receive a flow of 1 and the outward flow at the class vertices will be exactly 1 (and hence the flow is k). Thus, only 1 selection of student per class is made, which satisfies the constraint of the problem. Therefore, each class has selection of 1 student only. Also, by the construction of network flow each student can only be selected for at max r classes, which satisfies another constraint as the flow along the edge will at most r (that is less than or equal to the capacity). Hence, the problem has a feasible selection if the max flow is k.

Rubrics (5 points):

- 3 pts: Correct construction of the the network flow graph.
- -1 pt: For each incorrect edge weight.
- 2 pts: Accurate proof of correctness.