CSCI 570: Homework set 4

Mounika Mukkamalla

1

Sub Problem:
Let opt[i][j] be maximum number of coins obtained by bursting balloons wisely in the subarray from i to j including i and j. where $0 <= i, j < n$

Recurrence Relation:
For a k in [i, j], we burst k and obtain the maximum coins in the left subarray, [i, k-1] and right subarray, [k+1, j] and sum it up.
opt[i][j] will be maximum of all the k values from i to j
$opt[i][j] = max(opt[i][k-1] + opt[k+1][j] + nums[i-1] * nums[k] * nums[j+1])$ where $i <= k <= j$

Base case:
When i ==j, $opt[i][j] = nums[i-1] * nums[i] * nums[i+1]$ considering $nums[-1] = nums[n] = 1$
When $j < i, opt[i][j] = 0$

c) Algorithm:
1. Define a 2D array opt[n][n] of size nXn.
2. Initialise all the elements of opt array to 0
3. for $len = 1; len <= n; len++$ { // Traverse through all the subarrays from length =1 to n
a. for $i = 0; i <= (n - len); i++$ { // Start from the left
i. $j = i + len - 1$; //Upper bound for the subarray starting from i
1. for $k = i; k <= j; k++$ { // Traverse the subarray
a. local = 0 //a local variable
b. prev = 1, next = 1 // previous and next balloons of k
c. if $i >= 1$ then $prev = nums[i-1]$
d. if $j + 1 < n$ then $next = nums[j+1]$
e. if $(i == j)$ then $opt[i][j] = prev * nums[i] * next$
f. else:
i. if $k > i : local += opt[i][k-1]$
ii. if $k < j : local += opt[k+1][j]$
iii. $opt[i][j] = max(opt[i][j], local + prev * nums[k] * next)$
2. }
ii. }
b. }
4. return opt[0][n-1]

Complexity:
Step 2: $O(n^2)$
Step 3: There are 3 for loops, *len* loop runs $O(n)$ times, i loop runs $O(n)$ times and k loop runs $O(n)$ times.
Hence step 3 overall complexity: $O(n^3)$
Total complexity of the algorithm = $O(n^3)$

2

Sub Problem:
This is similar to min max problem in AI.

We divide the problem involving 2 players. Given n is even and I start first
This algorithm specifically concerns about my winning and opt values represented here are for my winning - doesn't give values for my friend's winning.
When n is even, I will be playing and hence I chose the maximum value I can obtain by either choosing the left or right most value of the array.
But when n is odd, it's my friend's turn, and hence they choose their maximum leading to my minimum possible choice.
Hence let opt[i][j] represent maximum or minimum possible amount of money that I win for the subarray i to j including i and j.
When j-i+1 is even, it's my turn, and I choose maximum and hence opt[i][j] represents maximum amount of money.
When j-i+1 is odd, it's my friend's turn and hence opt[i][j] returns the minimum amount of money that I can choose.

Recurrence Relation:
When j-i+1 is even: $opt[i][j] = max(v_i + opt[i+1][j], v_j + opt[i][j-1])$
When j-i+1 is odd: It's my friend's turn and they choose best for them and I get minimum of the left out which is: $opt[i][j] = min(opt[i+1][j], opt[i][j-1])$

Hence when j-i+1 is even(that is when it's my turn),
$opt[i][j] = max(v_i + min(opt[i+2][j], opt[i+1][j-1]), v_j + min(opt[i+1][j-1], opt[i][j-2]))$ - since opt[i+1][j] and opt[i][j-1] becomes odd and we replace it with our 2nd recurrence relation.

Base Case:
When $i == j : opt[i][i] = 0$ // - because in my algorithm we are concerned only about my winning hence when i ==j, it's my friend's turn and I get nothing here.
When $j < i : opt[i][j] = 0$

Algorithm:
1. Define a 2D array opt[n][n] of size nXn array.
2. Initialise the array with 0
3. for $len = 1; len <= n; len++$ { // Traverses the array with lengths from 1 to n
a. for $i = 0; i <= n - len; i++$) { // represents lower bound for the sub array
i. j = i+len-1; // represents upper bound in the sub array
ii. if i==j:
1. $opt[i][j] = 0$
ii. else if j-i+1 is even: $opt[i][j] = max(v_i + opt[i+1][j], v_j + opt[i][j-1])$
iii. else if j-i+1 is odd: $opt[i][j] = min(opt[i+1][j], opt[i][j-1])$
} } 4. return opt[0][n-1]

Complexity:
Step 2: $O(n^2)$
Step3: len loop runs for $O(n)$ times and i loop inside runs $O(n)$ times
Overall complexity for step3: $O(n^2)$
Overall algorithm complexity: $O(n^2)$

3

Sub Problem:
We define opt[j][i] to be the maximum number of steps required to reach from the starting point(before the bridge) to (j, i). where $0 <= j <= 2$ and $0 <= i < n$.
j = 0 implies 1st row, j= 1 implies 2nd row and j=2 implies 3rd row.
Similarly i starts from 0 representing the 1st column.

Recurrence Relation:
If (j, i) is not a bad tile, opt[j][i] = opt[j][i-1] + opt[j-1][i-1] + opt[j+1][i-1]
if (j, i) is a bad tile, opt[j][i] = 0
We reach to position (j, i) either from (j, i-1) or from diagonally left, (j+1, i-1) or from diagonally right, (j-1, i-1) assuming every position is within in the range.
Hence if (j, i) is a good tile,
opt[0][i] = opt[0][i-1] + opt[1][i-1]
opt[1][i] = opt[1][i-1] + opt[0][i-1] + opt[2][i-1]
opt[2][i] = opt[2][i-1] + opt[1][i-1]

Base Case:
For the 1st column, opt[j][0], number of ways to reach this tile is 1.
opt[j][0] = 1
If there is a bad tile at opt[j][i] = 0

Algorithm:
1. Define 2D array opt[3][n] of size 3Xn
2. if (0,0) is bad tile then opt[0][0] = 0 else opt[0][0]=1
3. if(1, 0) is a bad tile then opt[1][0] = 0 else opt[1][0] = 1
4. If (2, 0) is a bad tile then opt[2][0] = 0 else opt[2][0] = 1
5. for $i = 1; i < n; i++$ {
a. for $j = 0; j <= 2; j++$
i. if (j, i) is a bad tile then opt[j][i] = 0 .
b. else:
i. opt[0][i] = opt[0][i-1] + opt[1][i-1]
ii. opt[1][i] = opt[1][i-1] + opt[0][i-1] + opt[2][i-1]
iii. opt[2][i] = opt[2][i-1] + opt[1][i-1]
6. return opt[0][n-1] + opt[1][n-1] + opt[2][n-1]
}
Complexity:
Step 2, 3, 4: O(1)
Step 5.a: O(1) - since j ranges from [0,2] which is some constant
Step 5: for loop runs n-1 times, hence O(n) complexity.
Overall algorithm complexity = O(n)

4

Given statement is false, We disprove it by the following example.
Let's consider a residual graph represented in flow/cap format below.
Dotted lines represent the saturated edges and orange line shows the min cut.

If we remove edge, {b, t} maximum flow is reduced to 4. There is no edge in this graph on whose removal maximum flow is reduced more than removing edge {b, t}
And our min cut contains edges: {{a, b}, {d, b}, {d, t}}
There exists no other possible min cut in graph, G_1, that includes edge {b, t} since incoming total capacity for b is 19 and outgoing total capacity for b is 20. Hence by conservation constraint, edge {b, t} can never be saturated, and hence can never be in any min cut.
Therefore, the given statement is wrong, hence disproved the statement.

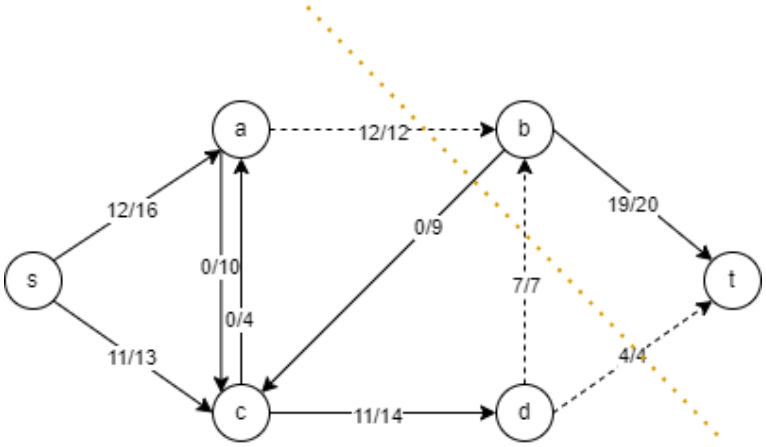

Figure 1: G_1

5

Given statement is false. We disprove it by the following example.
Let's consider a residual graph represented in flow/cap format below.
Dotted lines represent the saturated edges and orange arc shows the min cut.

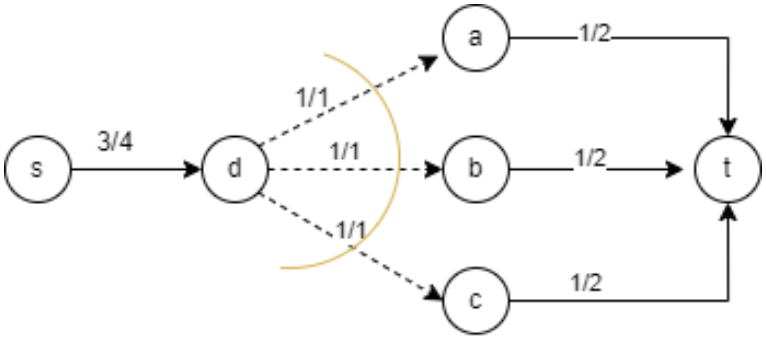

Figure 2: G_f

Here the s-t minimum cut is {{d, a}, {d, b}, {d, c}}
Now let's increase the capacities of every edge by one and the resulting residual graph is below.

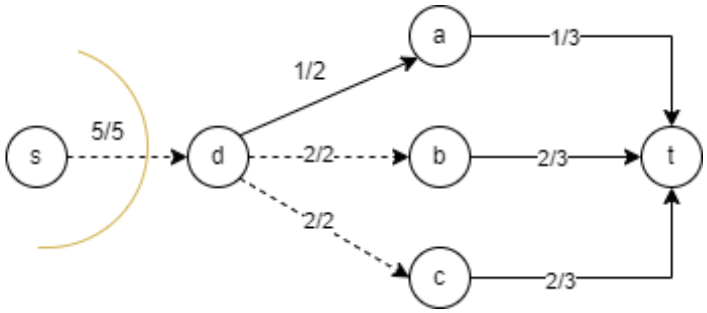

Figure 3: Updated residual graph $G_f^{'}$

Here the s-t minimum cut is {{a, d}}
Hence in both G_f and $G_f^{'}$, s-t min cuts are different and hence given statement is false.

6

a) After applying FF algorithm on G_1, we get the following residual graph,
Dotted lines represent the saturated edges and orange line shows the min cut.

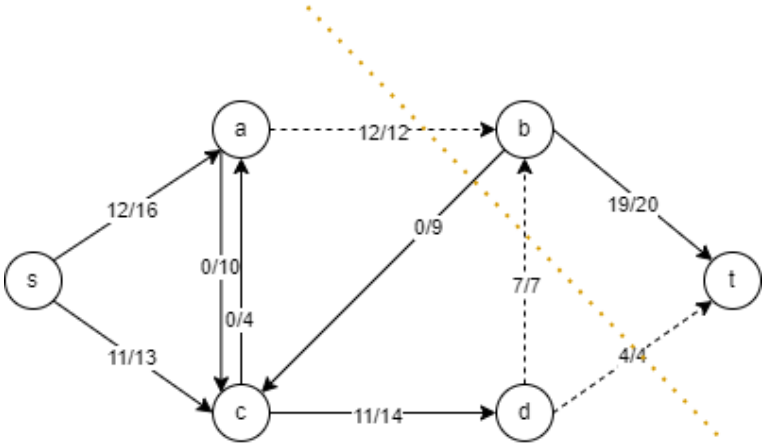

Figure 4: Updated residual graph $G_1^{'}$

Minimum s-t cut here is {{a, b}, {d, b}, {d, t}}
Capacity of edge, {a, b}, $c_{ab} = 12$, and flow, $f_{ab} = 12$
Capacity of edge, {d, b}, $c_{db} = 7$, and flow, $f_{db} = 7$
Capacity of edge, {d, t}, $c_{dt} = 4$, and flow, $f_{dt} = 4$
Maximum flow of G_1 = sum of outgoing edges capacities of Minimum cut of G_1
$\implies$ Max flow = $c_{ab} + c_{db} + c_{dt} = f_{ab} + f_{db} + f_{dt} = 12 + 7 + 4 =$ **23**

And Max flow paths are :
s-a-b-t with flow 12
s-c-d-t with flow 4
s-c-d-b-t with flow 7
So total flow is 23

b) After applying FF algorithm on the given graph, we get the following residual graph,
Dotted lines represent the saturated edges and orange line shows the min cut.

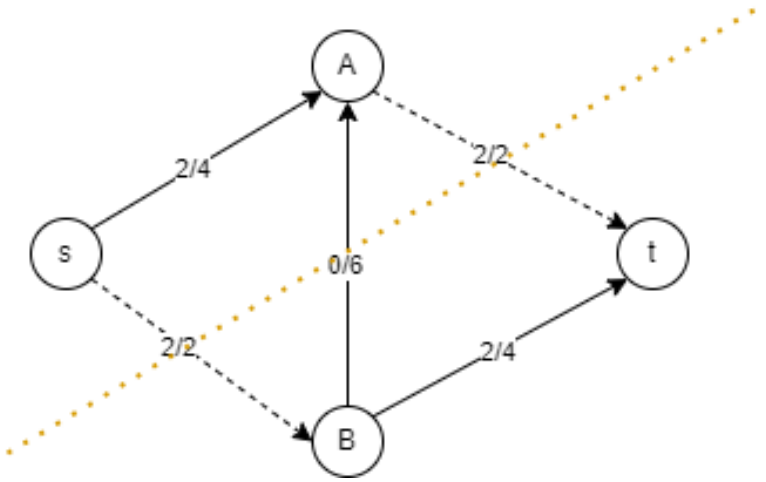

Figure 5: Updated residual graph

Min s-t cut here is {{s, B}, {A, t}}
Capacity of edge, {s, B}, $c_{sB} = 2$, and flow, $f_{sB} = 2$
Capacity of edge, {A, t}, $c_{At} = 2$, and flow, $f_{At} = 2$
Value of min cut = sum of capacities of outgoing edges in min cut
$\implies$ Min cut value = $c_{sB} + c_{At} = 2 + 2 = \mathbf{4}$

And Max flow paths are :
s-A-t with flow 2
s-B-t with flow 2
So total flow is 4 and min cut is 4

7

Let's convert the given problem into Network Flow.
To define NF, we need {V, E, C, s, t}
Vertices, V: set of all clients and base stations as nodes, $\{c_1, c_2, ..., c_n, b_1, b_2, ..., b_k, s, t\}$
where s and t are source and sink vertices respectively.
Edges, E, and capacities for those edges, C:
1. A set of edges is defined from source, s to all the clients, c_i with capacity, 1 - since we wish to connect each client to exactly one base station.
2. A set of edges defined from base stations, b_j to target, t with capacity, L - since no more than L clients can be connected to any single base station.
3. A set of edges is defined from each client, c_i to each base station, b_j if the distance between c_i and b_j is within R, and each edge is given a capacity of 1
So, the graph becomes,

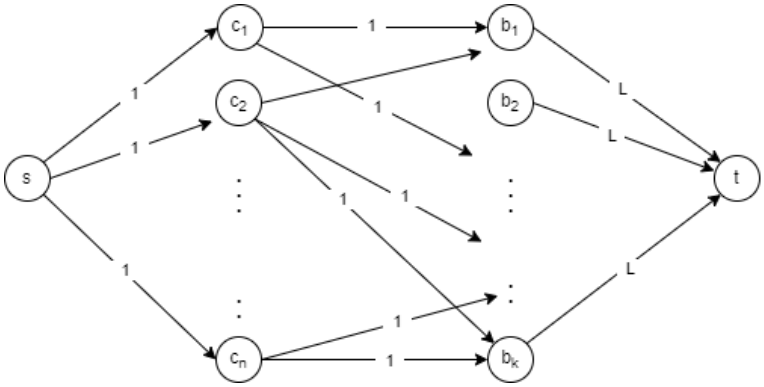

Figure 6: G_1

Claim: Every client can be connected simultaneously to a base station if and only if max flow for the above network is n

Proof:
1. If every client can be connected simultaneously to a base station then max flow = n
If every client is connected to a base station then there exists exactly one outgoing edge from each client, c_i to some base station, b_j. By conservation constraint rule, incoming edge for every client should have a flow of 1.
According to our network flow problem, there is only one incoming edge to client, c_i which is from source with capacity of 1.
Since flow should be 1 for these incoming edges and capacity is 1, all these edges are saturated.
Hence the min cut we get in the network flow graph is from the source to all the clients.
Let's represent the cut. Dotted lines represent saturated edges and orange arc represent cut.

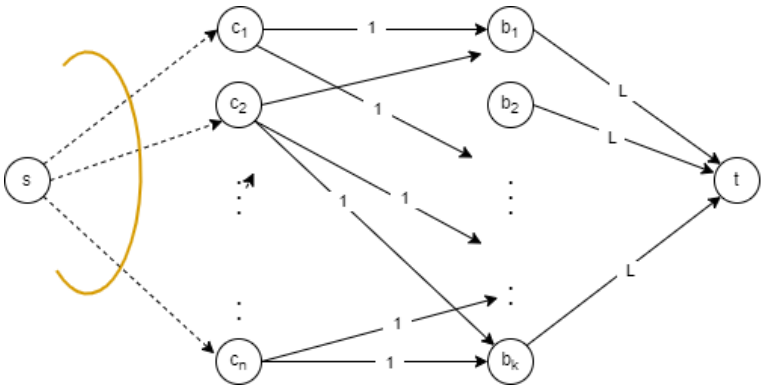

Figure 7: Just for visual view of cut - edge flows are not defined.

Therefore, max flow we could get is $1 + 1 + ... + 1(ntimes) = n$.
2. If max flow = n then there exists an assignment for each client.
Capacity of flow from source = sum of capacities of all edges from source to clients = n
If max flow = n, then all these edges from source are saturated.
By conservation constraint, If an incoming flow is 1 for a client, c_i there exists an outgoing flow from c_i to some b_j(according to our edges defined in the network).
Since every client incoming flow is one, there exists exactly one outgoing saturated edge for every client to some base station.
Hence there exists assignment for each client.
Hence proved both ways.

8

Given edge capacities of all edges = 1. Hence the maximum flow, $|f|$ is O(V) where V is the number of vertices.
Therefore, FF algorithm has complexity: $O(|f|(V+E)) = O(VE+V^2) = O(VE)$ - which is polynomial

Algorithm:
1. Find the min cut using FF algorithm
2. Define set F and initialize to empty set;
3. If number of edges in min cut, $|min\_cut|$ is greater than or equal to k then delete k edges in the min cut and add these to set F
4. If total number of edges in the graph $< k$ then delete all edges and add these to set F.
5. If number of edges in min cut, $|min\_cut|$ is less than k then delete all edges in the min cut and delete remaining, $k - |min\_cut|$ edges in the remaining graph and add these to set F.

Complexity:
Step1: $O(V^2)$
Step3, if true: $O(k)$
Step4, if true: $O(k)$
Step5, if true: $O(k)$
Step 3, 4, 5: removing the edges can take linear time in k if we use hashmap to store all the edges. - Hence finding the edges take constant time updating all removing edges take linear time in k.
Max number of edges that can be removed from the graph = $O(E)$ Hence step 3, 4, 5 is removing edges and their complexity becomes: $O(E)$ Hence overall complexity = $O(E + VE) = O(VE)$

9

Let's convert the given problem into Network Flow.
To define NF, we need {V, E, C, s, t}
For the given n X n grid, let's define nodes for letter S, P, Y.
Let S be set of all nodes s_i denoting the location of letter S in the given in the format, (x, y) where x is row, and y is column.
Similarly, Y are defined.
For P, we define 2 nodes, p_{i1}, p_{i2} containing the same information - denoting the location of the letter P in the format, (x, y) where x is row and y is column.
$S = \{s_1, s_2, ...s_l\}$
$P = \{p_{11}, p_{21}, ...p_{m1}, p_{12}, p_{22}, ...p_{m2}\}$
$Y = \{y_1, y_2, ...y_n\}$
Vertices, V: $\{s_1, s_2, ...s_l, p_{21}, ...p_{m1}, p_{12}, p_{22}, ...p_{m2}, y_1, y_2, ...y_n, s, t\}$
where s and t are source and sink respectively.
Edges E and capacities, C for these edges are defined as:
1. A set of edges is defined from source to all the nodes in S with edge capacity to be 1 - since each s_i is considered only once.
2. A set of edges is defined from all nodes in Y to sink t with edge capacity 1 - since each y_i is considered only once.
3. A set of edges are defined from S to P when p_{j1} is the immediate neighbour(is in the north or south or east or west) of s_i with edge capacity 1 - since each S-P is considered only once.
4. A set of edges is defined from P to Y when y_k is the immediate neighbour of p_{j2} with edge capacity 1 - since P-Y is considered only once.
5. A set of edges from p_{j1} and p_{j2} with edge capacity 1 - since we need to have disjoint S-P-Y words. There might be a case where 2 S's are connected to single P and that single P is connected to 2 Y's. This case is easily handled by giving 2 nodes for P and restricting the outflow of P to just one edge capacity and hence only one S-P-Y word is taken making the selection disjoint.
So, the graph becomes,

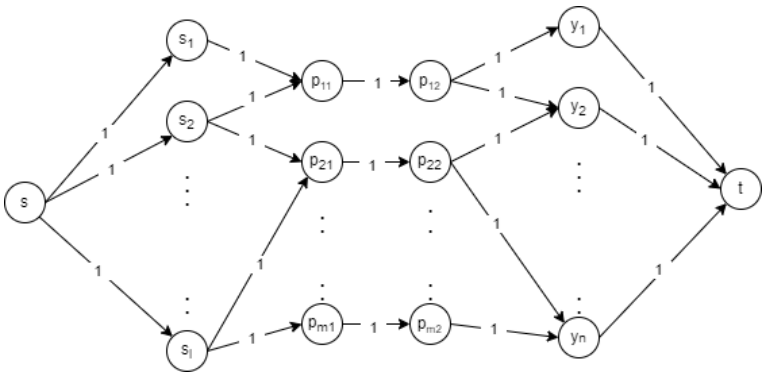

Figure 8: G1

Claim: Largest number of SPY's is k if and only if the max flow for the above flow network is k.

10

Let's convert the given problem into Network Flow.
To define NF, we need {V, E, C, s, t}
Let all students be marked as nodes and all classes be marked as nodes.
Vertices, V: Set of all students nodes, classes nodes, source and sink, $\{s_1, s_2, ..., s_n, c_1, c_2, ..., c_k, s, t\}$ where

s and t are source and sink nodes
Edges E and capacities C for these edges are defined as:
1. A set of edges is defined from source to students with capacity m: since, If source to students all capacities are saturated then all students are full time
2. A set of edges is defined from student, s_i to subset p_i of classes with edge capacity 1 - representing that student s_i can be enrolled for subset p_i classes.
3. A set of edges is defined from class, c_j to sink, t with edge capacity q_j - since each class, c_j has atmost q_j number of students enrolled.
So, the graph becomes:

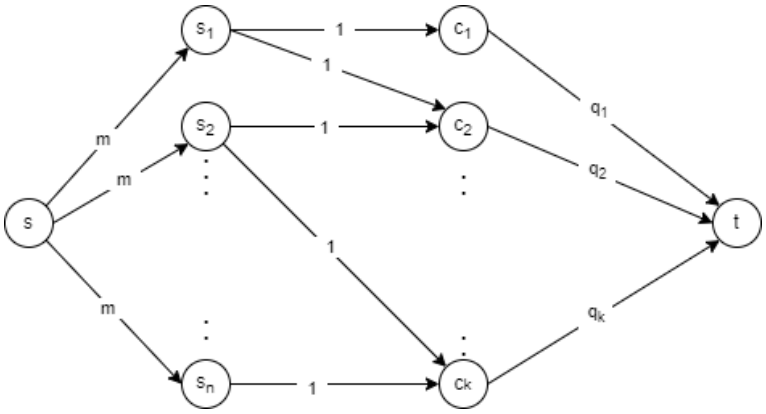

Figure 9: Visual representation of network flow, G1

Claim: There exists an assignment in which students are full time iff max flow of the above network is nm.

1. If there exists and assignment in which all the students are full time then max flow is nm.
If all the students are full time then according to the question, each student was able to enroll in m number of classes. Hence there are minimum m outgoing edges from the student node.
By conservation constraint, incoming edge has capacity m and outgoing has minimum m flow, then all the incoming edges are saturated.
Hence all the edges from source to student are saturated and hence the min cut would include all the edges from source to student and hence max flow is sum of all capacities from source to students = m + m +...+m = n*m

2. If max flow is nm then there exists an assignment in which all the students are full time.
If max flow is nm then all the edges from source to students are saturated and by conservation constraint, each student has incoming flow, m and hence outgoing flow should be m.
Therefore there are m outgoing edges from each student and hence each student is able to enroll for minimum m classes and hence there exists an assignment in which each student is able to enroll for m classes and therefore there exists an assignment in which all students are full time.

Hence proved both ways.

2. Let's convert the given problem into Network Flow.
To define NF, we need {V, E, C, s, t}
Vertices, V: Set of students nodes, classes nodes, source and sink $\{s_1, s_2, ..., s_n, c_1, c_2, ..., c_k, s, t\}$ where s and t are source and sink nodes
Edges E and capacities C for these edges are defined by:
1. A set of edges is defined from source, s to classes, c_i with edge capacities 1 - since each class has single student representative.
2. A set of edges is defined from classes, c_i to student s_j with edge capacities 1. These edges are defined

as the following:
a. From 10a) when max flow is nm, there exists assignment in which each student has m number of classes assigned to them.
b. So, now we have students to classes assignment.
c. using this assignment, define edges from class to all the enrolled students.
3. A set of edges from students, s_j to sink, t is defined with edge capacities, r - since a student can't be class representative for more than r classes.

So, the graph becomes:

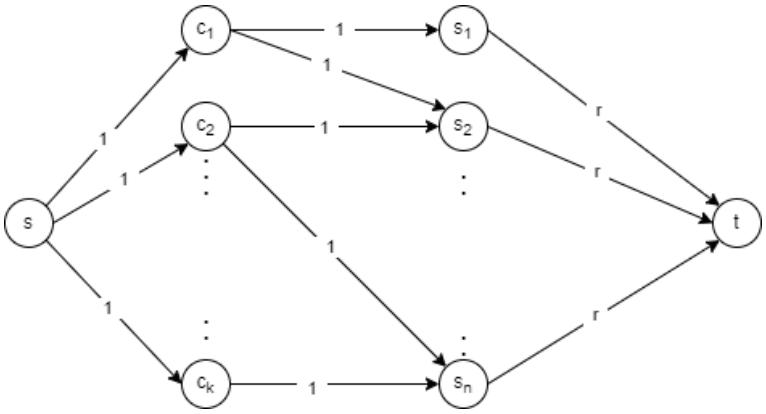

Figure 10: Visual representation of network flow, G2

Claim: There exists a student representative from each class if and only if max flow is k.
Proof:
1. If there exists a student representative from each class then the max flow is k.
If there exists one student representative from each class then there is only one saturated edge outgoing edge from each class, and hence outgoing flow is one.
By conservation constraint rule, since outgoing flow is one, incoming flow has to be 1.
This applies to all the classes and hence incoming edges flow to all the classes is 1 and all these are saturated.
Hence all the outgoing edges from source are saturated and hence min cut includes these edges, therefore max flow is sum of edge capacities of these edges which is 1+1+...+1 k times which is k
Hence max flow is k.

2. If max flow is k then there exists one student representative for each class.
If max flow is k, then all the outgoing edges from source are saturated, and hence incoming flow for each class is 1.
Therefore by conservation constraint rule, outgoing flow has to be one for each class.
Hence there exists exists one student assigned for each class.
Hence proved both ways.