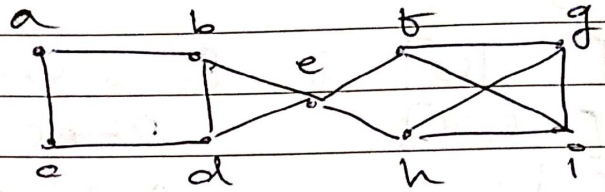
CS 660 - ASSIGNMENT IV

Q1. (s, t) - cut vertex.

A vertex in an undirected connected graph is a cut vertex if and only if removing it would disconnect the graph and divide it into two or more graphs.

e.g.

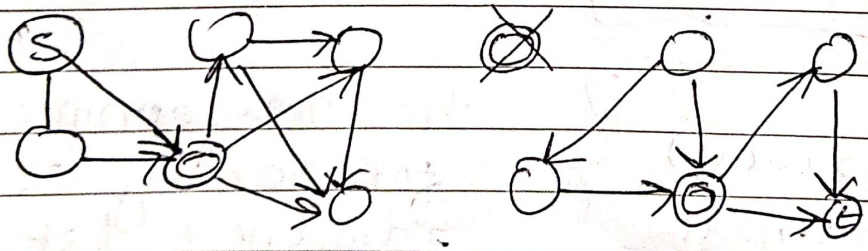


Here, one of the cut vertex is the vertex e since if we remove e, it will split the entire graph into 2 graphs.

Similarly in the given graph,

If we remove the one of the mentioned cut vertex, say, v_2.

It splits into 2 graphs.



Basic Algorithm:

- For every vertex v in Graph G, do the following:

⇒ Remove v from graph G.
⇒ If graph is connected (DFS or BFS)
return "not a cut vertex"
else return "cut vertex (v)".
⇒ add v back to the graph G.
Time Complexity: $O(V * (V+E))$ → DFS.

- In this algorithm, we are checking if there are all vertices connected to each other when we remove each vertex one by one.
- We can check the connectivity if all the vertex are connected. If so, the eliminated vertex is not a cut vertex. Else it is a cut vertex.

Better Algorithm.

- We can use an algorithm in the form of a DFS.
- We can say that u → v, u can be considered as a cut vertex if and only if.

⇒ u is the root of the DFS tree and it has 2 children or more.
⇒ If u has only 1 child v, there must not be an edge in subtree of v such that it connects to any parent of u. or back edge connecting to its ancestor.

This can be performed by creating 2 arrays

parent[] and earliestVisit[]

→ the first condition can easily be implemented by checking for 2 children.

→ The second condition can be used to get earliest visited vertex from tree vertex with u.

min(earliestVisit[u], earliestVisit[x]

Here, x is the point where backedge to u exists.

The algorithm is entirely based on DFS and thus uses the same time as DFS i.e. $O(V+E)$.

Q3. Pg 376 (Prob 18)

Let $G = (V, E)$. $i \to j$ & cap[1...V]

max-no. of students who can take each class.

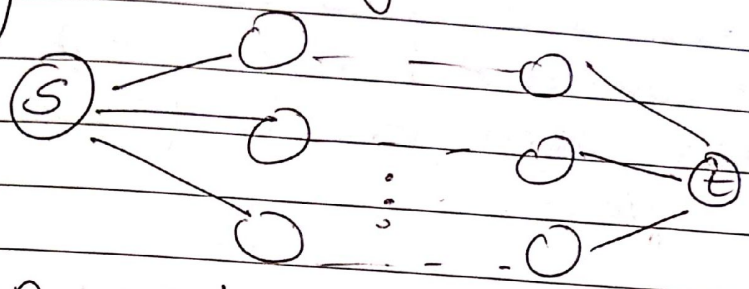
Problem: Find an assignment that maximizes the amount of students in each class with min. students in total.

We can solve this problem by creating a n/w flow diagram.

The network flow diagram can be formed by implementing topological sort

on the existing graph G. This will form the graph to contain a starting vertex (source) s and an ending vertex t (sink)

e.g



We then apply maximum flow algorithm to get the "min no. of students to cover all classes".

since the same students can attend other classes.

To find the max flow, we use ford-fulkerson method which repeatedly finds augmenting paths through the residual graph. & then augments the flow until no more augmenting paths can be found.

A augmenting path is path of edges in residual graph with unused capacity > 0 from s to t.

We use the bottleneck value to augment the flow along a path.

This is done repeatedly, so that during augmentation, the flow values are updated.

When we decrease the flow & along residual edge by bottleneck value

Algo:

→ initialize flow to 0.

→ while ∃ augmenting path p from s to t
augment flow along path
{ subtract bottleneck }
add path flow to flow.

→ return flow.

If $u \to v$ is an edge of G,

$$f'(u \to v) = \begin{cases} f(u \to v) + b(p), & \text{if } u \to v \in p \\ f(u \to v) - b(p), & \text{if } v \to u \in p \\ f(u \to v), & \text{otherwise} \end{cases}$$

max flow = $|f^*|$, after applying BFS of n/w, time complexity = $O((|V|+|E|) \cdot |f^*|)$.
for residual n/w max flow.

Q4. To invite as many friends as possible, the boss needs to make a list of least agreeable friends, the ones with most rivalries

Certificate: a list of invitable friends from { $v_1 \ldots v_n$ } and non-invitable friends.
{ $w_1 \ldots w_n$ }.
To form non-invitable list of friends we include friends with most conflicts.

Independ

Here, Independent Set or set of friends who will be at the party can be the = total friends − non invitable list of friends.

which can be denoted as
I.S. = V − Vertex Cover.

We consider this as a decision problem as we need to identify if there is an independent set of size at least k.

In set cover, we need to find some sets ~~that satisfy so~~ union of which covers all the elements. We want to find two people to invite so that every list of people will be covered.
∴ There is at least 1 person invited in each list.

Instance: Graph G(V, E); +ve integer k

Proposition: Given a graph G, the size of a matching in G is at most size of vertex cover.

Proof: For every edge in the matching at least 1 vertex must be in the cover.

→ We create a graph for the following situation which is an undirected graph where vertices represent list of pairs of people who can be invited & edges represent that people talk to each other.

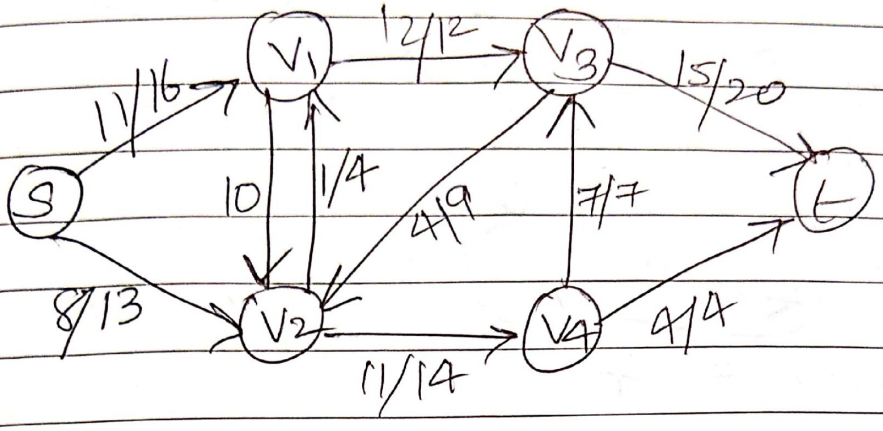
→ We then invert the graph and try to find a list of k people who don't talk to each other & should not be invited.

Independent Set:

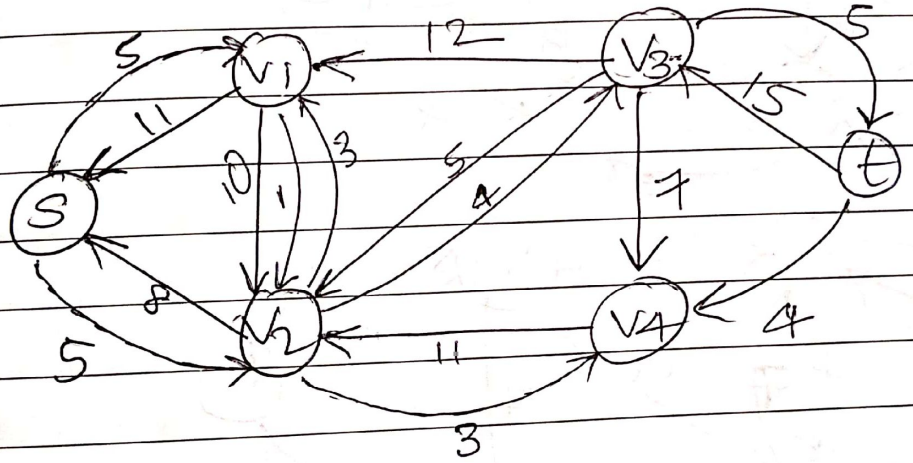
If "G, K" is a YES instance certificate: an I.S. of size k.

We can verify k vertices if they form an I.S. in poly time.

85.



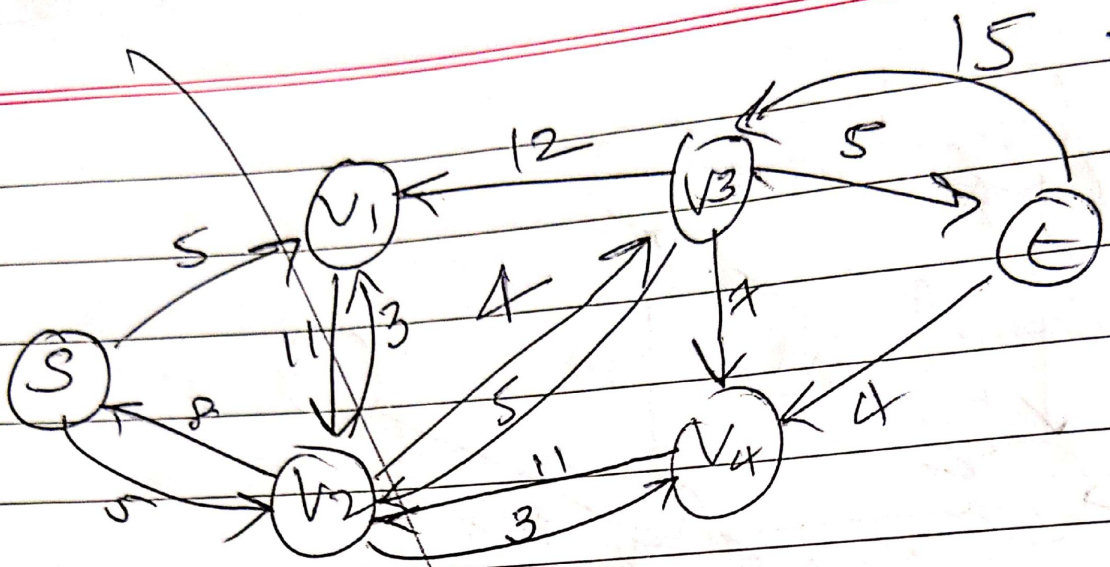
Residual Network:



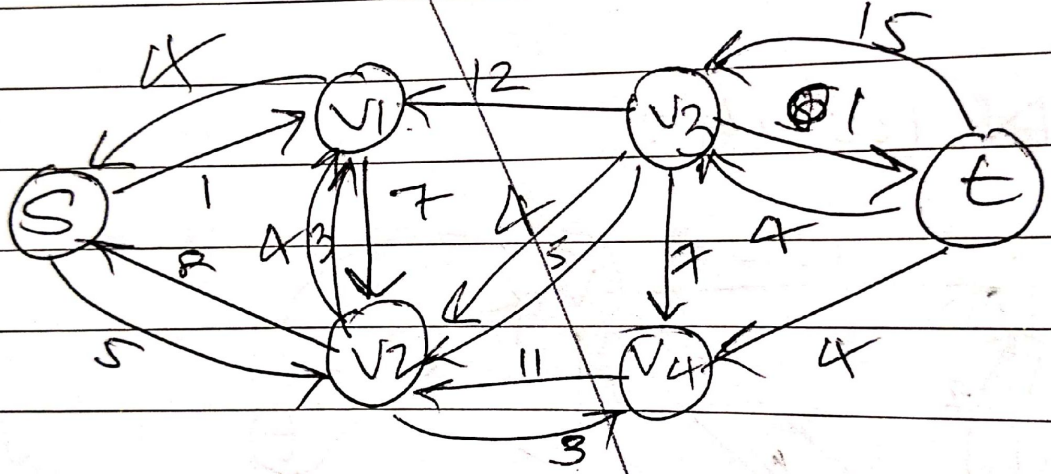
Residual capacity is just capacity left over the existing flow.
In a given flow n/w. $G=(V,E)$ & flow f. an augmenting path p is simple path from source s to sink t in residual n/w.

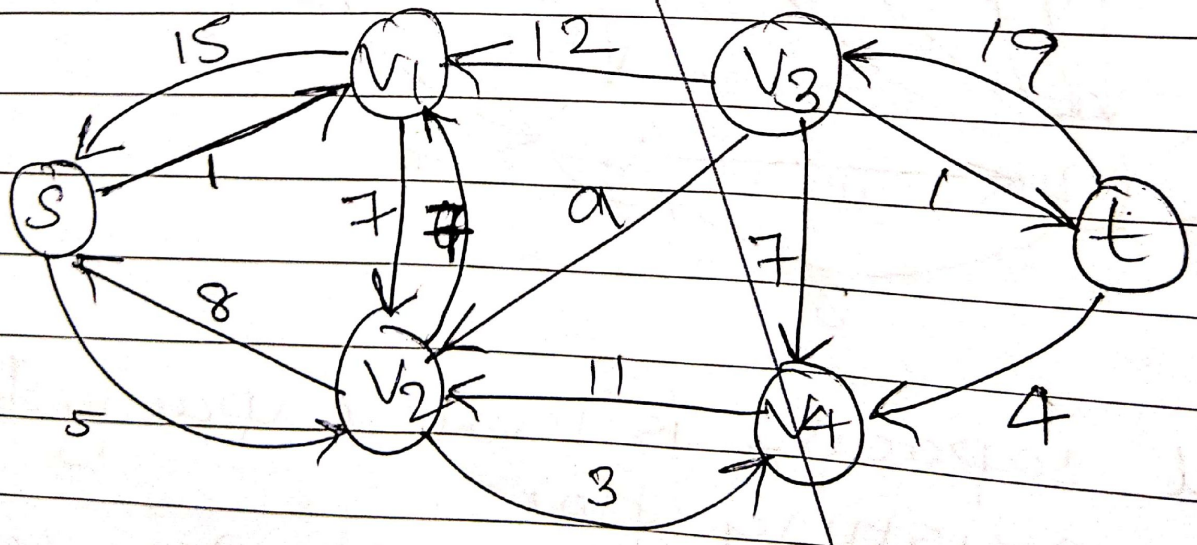
We may increase the flow of an edge (u,v) of an augmenting path by upto a capacity $c(u,v)$

Bottleneck for s, v_1, v_2, v_3, t.



Bottleneck = 4.

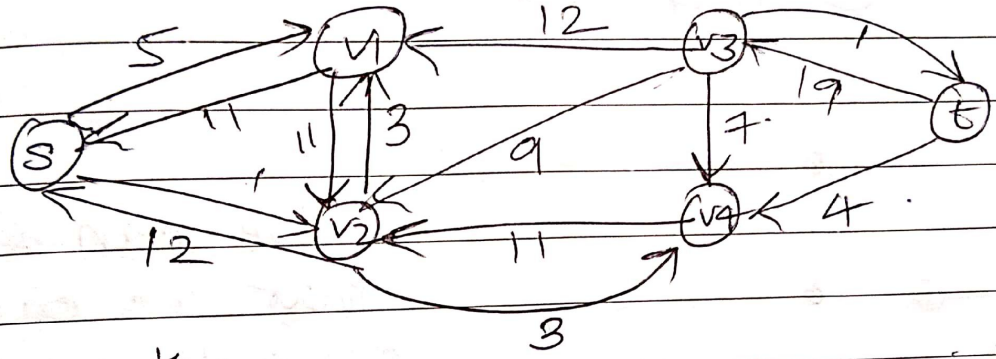




From the residual n/w

∴ V_1 and V_4 don't have any augmented paths connecting to t, we are supposed to ignore V_1 and V_4.

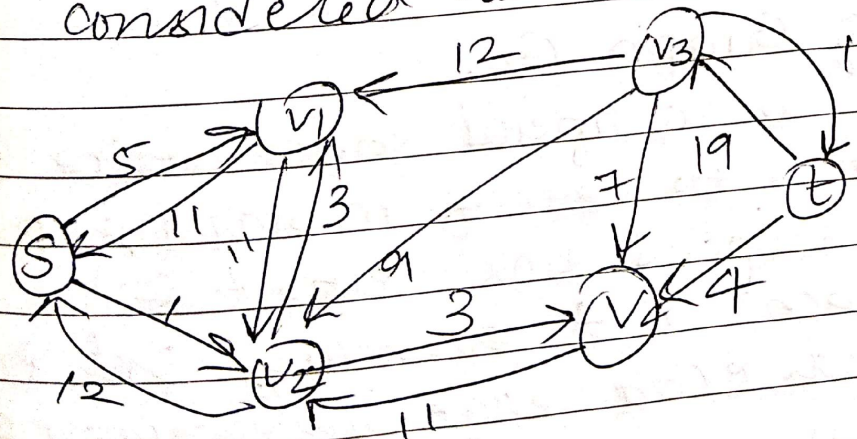
∴ The flow that results from augmenting alng $s \rightarrow v_2 \rightarrow v_3 \rightarrow t$. is.



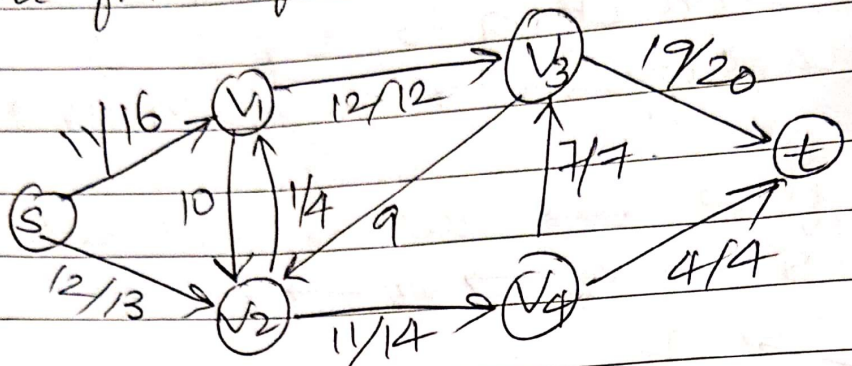
Here, the residual capacity is considered as 4.

∴ ~~The new flow of the graph~~. and the above graph network is induced by the flow.

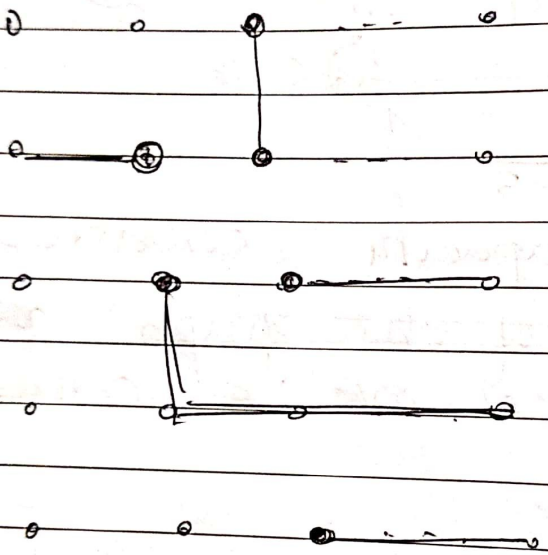
∴ The new flow of the network can be considered as.



∴ The final flow is



Q2.



We can say that we can convert the input grid to flow n/w. $G = (V, E)$.

There are m starting pnts & 4n-4 boundary pnts.

- Construction of Graph G.

→ introduce an artificial source vertex s and connect s to each of m starting points.

→ Introduce artificial sink vertex t & connect each 4n-4 boundary pnts to t.

→ Each undirected edge b/w points $u \in V$ in the original grid is changed into 2 directed edges (u, v) & (v, u) in G.

→ Assign edge capacity = 1 to each edge in G & every vertex capacity = 1 to each vertex in G.

Algorithm:

To determine if there is an escape in the original grid.

→ Find maximum flow through G (Apply Ford Fulkerson's algo)

→ If value of maximum flow in G is m, then there is an escape in the original grid.

→ else: there is no escape

Ford Fulkerson's Algo:

~~path~~ initialize flow to 0.

path = findAugmentingPath( )

while pathExists :

augment flow along path

path = findAugmentingPath( )

~~return flow~~.

Note: Each $u \to v \in E$ has a capacity $c(u \to v) > 0$

$c(u \to v) = 0$ if $u \to v \notin E$.

The running time for ford-fulkerson is $O(VE^2)$.

→ No. of pnts in input grid = n^2 pnts.

→ No. of undirected graph = $2n^2 - 2n$.

→ No. of vertices = $2 + n^2 = O(n^2)$.

→ No. of edges = $m + (4n-4) + 2(2n^2 - 2n)$

$= m + 4n^2 - 4 = \underline{O(n^2)}$.

→ $O(E+V) \cdot |f^*| \Rightarrow$ Ford-Fulkerson algo.

→ $|F^*|$ is max.

$\therefore |F^*| \leq 4n-4$ & $|E|+|V| = O(n^2)$

∴ Running time (higher bound)

$= O((E+V) \cdot |f^*|)$

$= O(n^3)$.

(b) Suppose the input to escape problem consists of single integer n to list of m terminal vertices.

→ Certificate: a list of all the ~~edges~~ paths from source to boundaries $p(u,v)$.

→ Verify: Prove that the m terminal vertices are an Independent Set.

→ Proof: Using the problem similar to the above part (a),

→ we can compute the answer by determining if the soln is NP-complete if we can find soln in poly time.

→ We consider this as a decision problem as we need to identify if there are an independent set containing m terminal vertices of size m.

→ But since we already know that the Independent set is NP-complete, we can say that the problem of grid is NP complete and solvable in polynomial time.