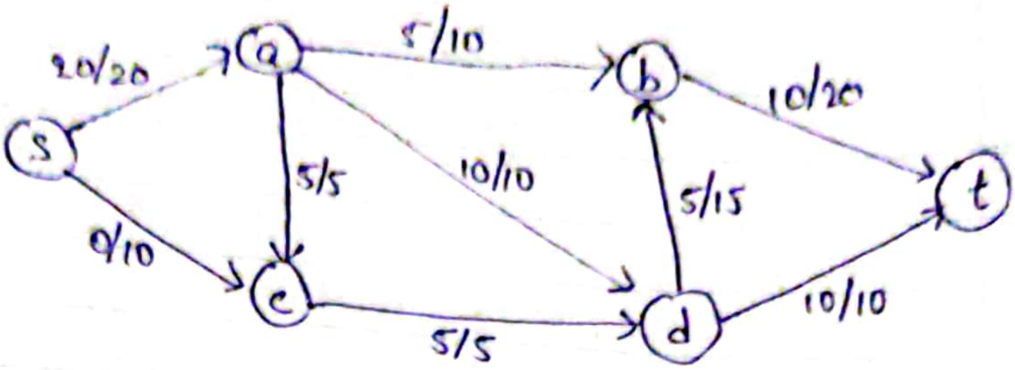
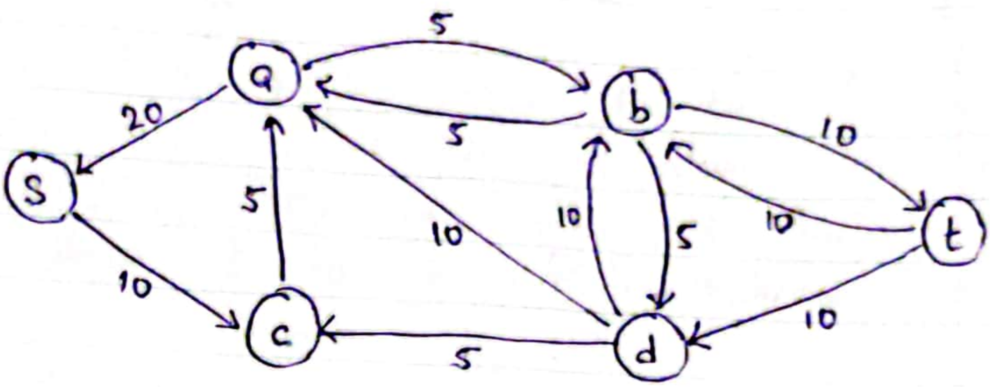
Submitted by: Vijay Anand Varma Indukuri (VXI210000)
Karthik Ragunath Ananda Kumar (KXA200005)

1) Given.

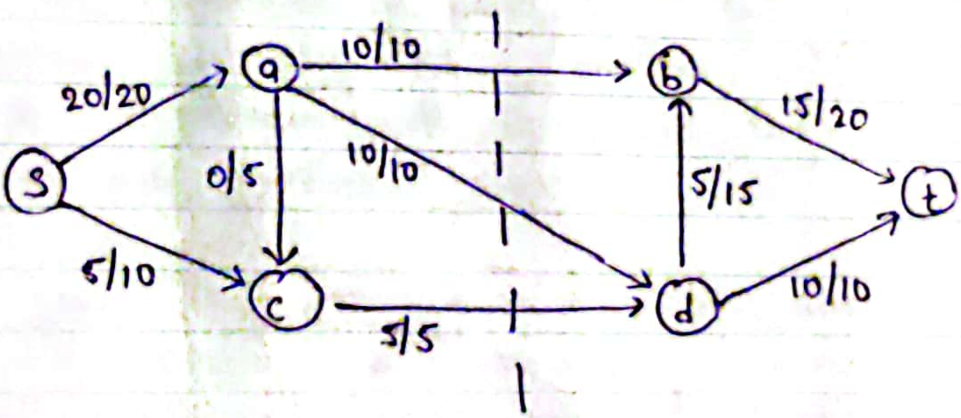


a) Residual Graph:



b) Augmenting path : $s \rightarrow c \rightarrow a \rightarrow b \rightarrow t$

c) Minimum cut from flow is {s, a, c} and {b, d, t}



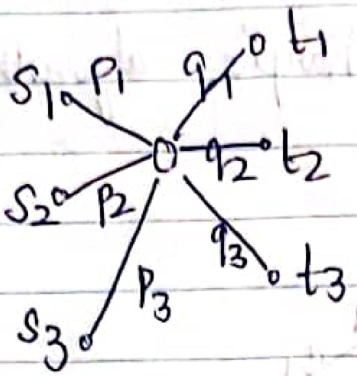
Scanned with CamScanner

2) let original graph be G ~~ast~~ with capacities given. G(V, E)
let's create a new graph G'(V, E) with updated capacities. Updated capaties in new graph will be

$$C_{new} = C_{old}(|E|+1) + 1$$

Relatively, order of capacities will remain same in both the graphs. Because old capacities are all non-negative integers and we are multiplying all these capacities with non-negative value (|E|+1) and adding positive integer 1.
Since we are doing all positive operations on positive capacities, relative order of capacities remain same.
When 2 cuts in G have same min-cut capacities, G' will have different values for these 2 cuts. This is because of multiplying each capacity with relatively large number (|E|+1) and summing up. This is assuming that above 2 cuts have different number of edges involved.
→ Thus, minimum cut in G' will give best minimum cut in G.
→ We can use Edmonds-Karp on G' to get max flow/min cut in $O(nm^2)$ time.

(3)

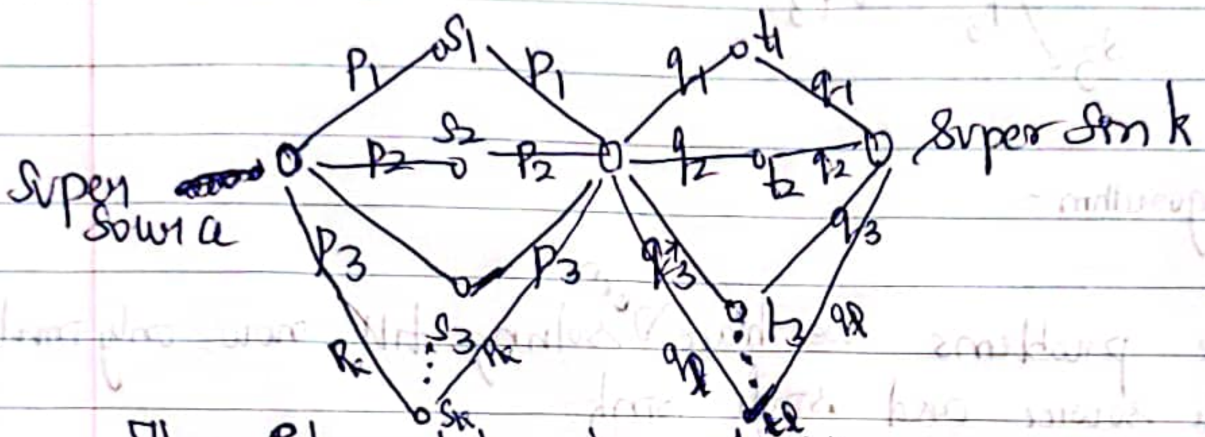


Algorithm:-

The problems we have been solving till now, only involved single source and single sink.
But in this problem there are multiple sources and multiple sinks and it is given that each source s_i produces p_i units of flow and each sink t_i consumes q_i units of flow.
To solve this problem using our typical maxflow/mincut algorithms (let's say Edmund Karp's algorithm with fewest edges).

We can simply add a super source and super sink nodes, with flow from super source to s_1 being p_1, flow from super source to s_2 being p_2 and so on. Similarly flow from sink t_1 to super sink is t_1, from sink t_2 to super sink is t_2 and so on. By this way, flow conservation is not disrupted due to super ~~sink~~ source and super sink nodes added.

Now, lets run Edmond Karp's algorithm on this modified graph.



The Edmond Karp's algorithm gives max flow/min cut value of this graph as 'X'.

if we find that $X = \sum_{i=1}^{k} \sum_{v \in V} f(s_i, v) = \sum_{i=1}^{k} p_i$

and $X = \sum_{i=1}^{l} \sum_{v \in V} f(t_i, v) = \sum_{i=1}^{l} q_i$

then, we can determine that there exist a flow satisfying production and consumption constraints.

Else, there does not exist a flow satisfying production and consumption constraints

~~Time Complexity~~

Time Complexity :-

Time Complexity is same as time complexity for Edmund Karp's algorithm

$O(m'n'^2)$.

where $m' = m + k + l$.

where $m \Rightarrow$ No. of edges in original graph

$k \Rightarrow$ No. of sources

$l \Rightarrow$ No. of sinks.

$n' = n + 2$

where $n \Rightarrow$ No of vertices in original graph.

4) Let G' be the graph after reducing the capacity of single edge $u \to v$ by 1.

→ If flow through $u \to v$ is less than capacity of $u \to v$ then maximum flow won't change. Just the capacity of $u \to v$ is reduced by 1.

If flow through $u \to v$ is at capacity of $u \to v$:

→ Find a flow path from s to t which contains (u, v). This can be done by using BFS in $O(V+E)$. This is same as finding augmenting path from t to s which includes $v \to u$ edge in residual graph.

→ Now, we reduce the flow of every edge on that path by 1. This will reduce the total flow by 1.

→ Now, running one loop of Ford-Fulkerson (in $O(V+E)$) will search for augmenting path. If we find an augmenting path, then the flow will get increased by 1 and algorithm will terminate. If no augmenting path is found, then flow won't change from previous step.

Running time: The running time of above algorithm is $O(V+E)$ as all the steps given above, finding BFS and one loop of Ford-Fulkerson, takes $O(V+E)$

5a) **Bounded degree spanning tree**

→ Here we do a polynomial time reduction from Hamiltonian path which we know is a NP-complete problem.

Reduction:

→ The k=2 case in bounded degree spanning tree corresponds exactly to Hamiltonian path problem.

→ Above claim is true because a spanning tree with maximum degree 2 is just a path hitting each vertex exactly once.

→ So, given a Hamiltonian path problem, we reduce it to bounded degree spanning tree with k=2.

→ Thus, bounded degree spanning tree is a NP-hard problem.

→ Given a list of edges, we can verify that these edges connect the graph and there are exactly n-1 edges (no cycles) and also that each vertex has degree atmost "k", in polynomial time.

→ Thus, bounded degree spanning tree is also NP.

→ Since, this is both NP and NP-hard, we can say that bounded degree spanning tree is NP-complete.

5b) SET COVER:

→ To prove set cover is NP-hard, we do a polynomial time reduction from Vertex-Cover which we know is a NP complete problem.

Reduction:

→ let's define a vertex cover problem on graph $G(V,E)$

→ let $S=E$ be the set of edges in G.

→ Also define subsets S_k which contains all edges incident to 'k' vertex.

→ let's consider that k sets $S_1, S_2, S_3 - - - S_k$ cover the ground set S, then every edge in E is adjacent to minimum one vertex in $1, 2, - - - k$.

→ This is nothing but forming a vertex cover of size k.

→ If we have vertices $1, 2, 3, --- k$ forming a vertex cover, then S_k covers all edges incident at vertex k. Hence, collection of sets $S_1, S_2, S_3, --- S_k$ form set cover covering S.

→ Since we reduced set cover from vertex cover, we can say that set cover is NP-hard.

→ Provided a collection of subsets of size k, we can iterate over each element in the subsets of collection and mark the elements in S which are covered. Since we can do this verification in polynomial time, we can say that set cover is NP.

→ Since set cover is both NP and NP-hard, set cover is NP-complete.

Scanned with CamScanner

5c) KNAPSACK:

→ To prove KNAPSACK is NP-hard, we do a polynomial time reduction from subset sum which we know is a NP complete problem.

Reduction:

→ We can easily reduce an instance of subset sum problem to an instance of knapsack problem. We just create a knapsack problem such that

$$V_i = S_i = X_i$$
$$k = B = t$$

Thus, new instance will be

$$\sum_i V_i \geq k \Rightarrow \sum_i x_i \geq t$$
$$\Rightarrow \sum_i x_i = t$$
$$\sum_i S_i \leq B \Rightarrow \sum_i x_i \leq \cancel{B}t$$

→ Suppose we have "Yes" answer to new problem, it means we can find such a subset $X \subseteq [1, 2, \dots n]$ that satisfies left part of deduction. Then this subset S is also a ~~problem~~ solution to right part. So, we must have "Yes" answer to original problem.

→ Suppose we have "No" answer, it means there is no subset S that satisfies the left part. So, answer to original problem must also be "No".

→ Thus, knapsack is a N.P-hard problem.

→ Also given a subset of indices we can easily verify if the total size is at most "B" and total value is atleast "k" in linear time.

→ Thus, knapsack is in NP.

→ Since, knapsack is both NP and NP-hard, we can say that knapsack is NP-complete.

5d) We prove FSAT is NP-Hard by doing polynomial time reduction of 3SAT which we know is a NP-complete problem.

Reduction:

→ We know each clause in 3SAT will be in the form $(x \vee y \vee z)$. We reduce this single clause into 2 clauses of 3 CNF form.

$$(x \vee y \vee z) \equiv (x \vee y \vee a) \wedge (z \vee \bar{a} \vee \text{false})$$

→ let's consider each term individually

$(x \vee y \vee a)$ → 'a' is set to false, when either one of x and y is true.

$(a \equiv \overline{x \vee y})$ → 'a' is set to true, when both x and y are false.

→ In this way, we will always have false literal in this term.

$(z \vee \bar{a} \vee \text{false})$ → We can directly see that there is a false literal in this term.

→ Thus by reducing each term in the form $(x \vee y \vee z)$ into $(x \vee y \vee a) \wedge (z \vee \bar{a} \vee \text{false})$, we reduce 3SAT into FSAT.

→ Since we are splitting one term into just 2 terms, this is definitely a polynomial time reduction. Since 3SAT is NP complete, above reduction shows that FSAT is NP-hard.

→ Given specific literals, we can simply verify the given FSAT in polynomial time. Thus FSAT is NP.

→ Since, FSAT is both NP and NP-hard, FSAT is NP-complete.

5c) **Almost Independent Set:**

We prove almost Independent set is NP-hard by doing a polynomial time reduction from Independent Set problem which we know is NP-complete.

Reduction:

→ Let's consider a graph $G(V,E)$ which has a maximum independent set of size k.

→ We will create a new graph $G'(V',E')$ from G by adding one new vertex "x".

→ let's denote independent set of $G(V,E)$ as "S".

→ After adding x to vertex list in G', we connect x to all verteces that are not present in independent set "S".

i.e

$$V' = V \cup \{x\}$$

$$E' = E \cup \forall_{v \in (V-S)} \{x \to v\}$$

Above mentioned are the vertices and edges in G'.

→ Now the maximum independent set in G' will be $S' = S \cup \{x\}$ with size of $k+1$. We can easily verify this by saying that all vertices adjacent to "x" are not present in "S".

→ If we observe, S' is also almost independent set of $G'(V',E')$. This is because we cannot add any other vertex to S'. Because all other vertices are sandwiched between one vertex in S and "x". Thus if we add any other vertex, this new vertex will have 2 adjacent verteces in S'.

→ Thus adding one vertex and edges as mentioned above will convert the max indepentdent set into almost independent set problem.

→ Hence we can say that AIS is NP-hard.