Advanced Algorithm

FLOW

Hassan Jaber

Contents

Functional Documentation

1. Problem Description:

Our main project to design and implement an algorithm for calculating the edge connectivity of a simple undirected graph. First of all, what is an edge connectivity ?

An edge connectivity of a graph is the minimum number of edges whose removal will disconnect the graph.

2. Solution Description:

In this project we are going to construct a flow network from a undirected graph, so why are we going to try to do that ?

2.1 Definition and Usage of flow network:

One of the usage of flow network is to find the maximum flow using the famous "Ford-Fulkerson Algorithm", so let's explain what's a flow network in graph theory, and why should we calculate the maximum flow of a graph in order to find the edge connectivity of a graph.

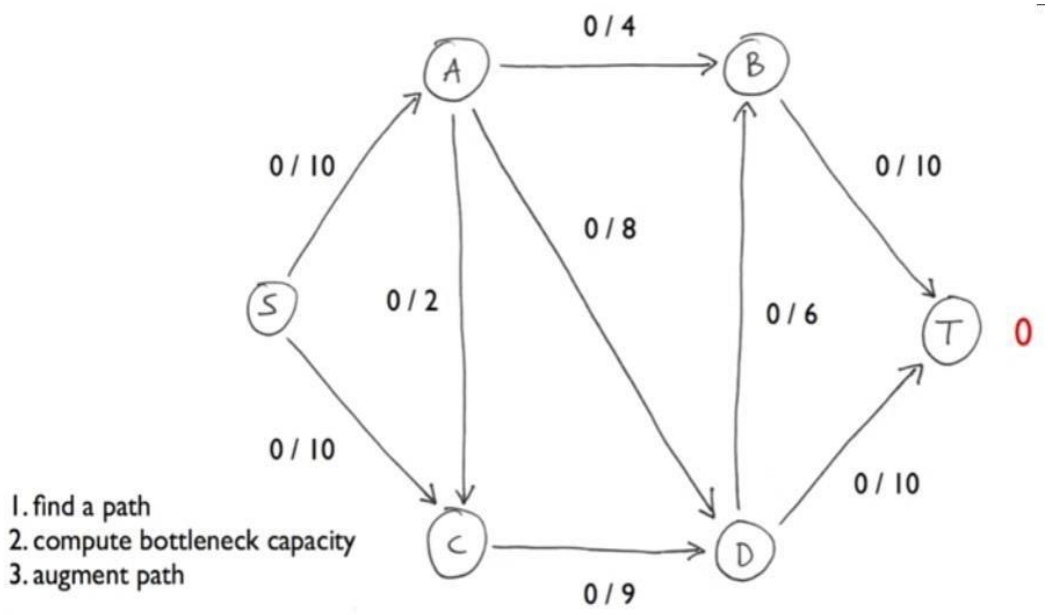
1) A flow of graph is when we have a directed graph where each edge has a capacity and each edge receives a flow, the amount of the edge flow cannot be more than the edge capacity. And the flow network graph always starts with a source node and ends with a sink node.

2) A maximum flow is the maximum amount of flow that the graph would allow from source node to sink node. Hence we are going to calculate the maximum flow using
"Ford-Fulkerson Algorithm".

3) Max-flow Min-cut Theorem: The maximum flow in any graph should always be equal to the number of minimum cuts *(edge connectivity)*. That's why we are going to calculate the max-flow using "Ford-Fulkerson Algorithm".

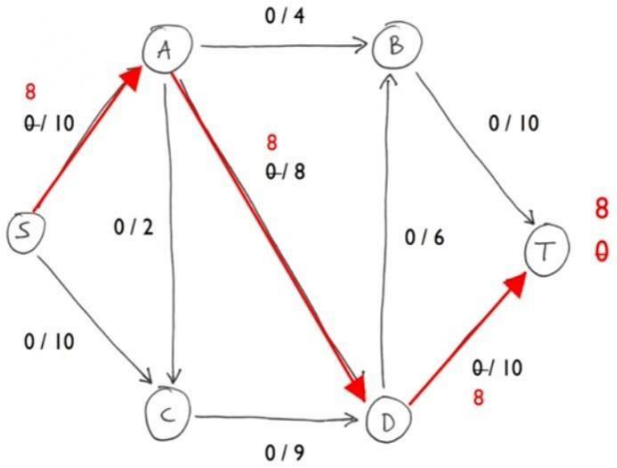
4) Ford-Fulkerson Algorithm: It's an algorithm to find the maximum flow of a flow network by finding an "Augmenting path".
An augmenting path is basically a path whose edges are*(Better explanation is found in the next **section 2.2**)*: a. non-full forward
b. non-empty backward

For example let's consider the following graph:

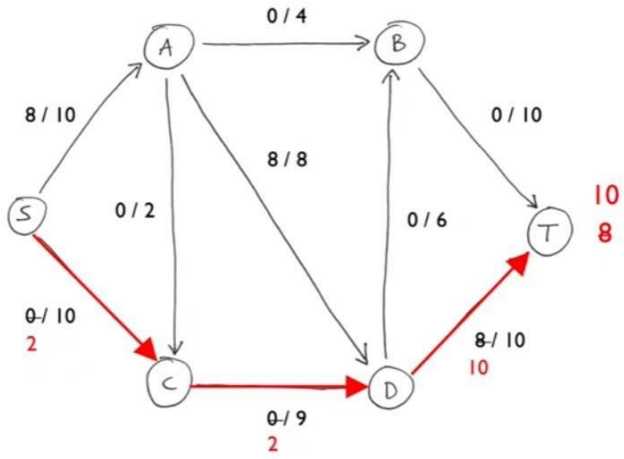


This is a flow graph that begin that starts with a source node (s) and ends with sink node (T).

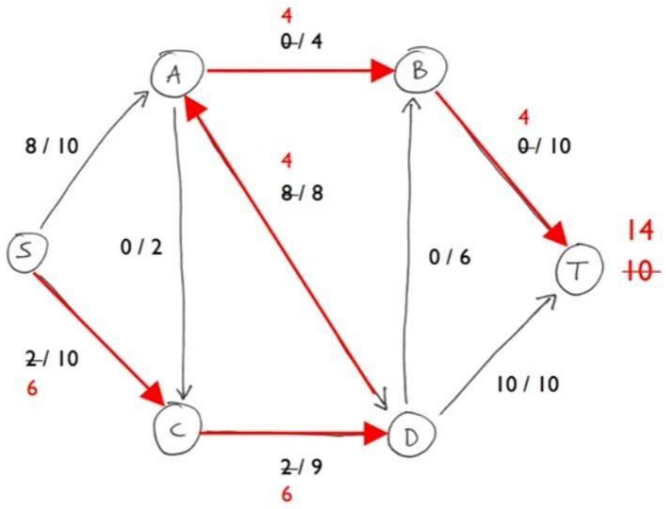
First of all let's choose the following path:



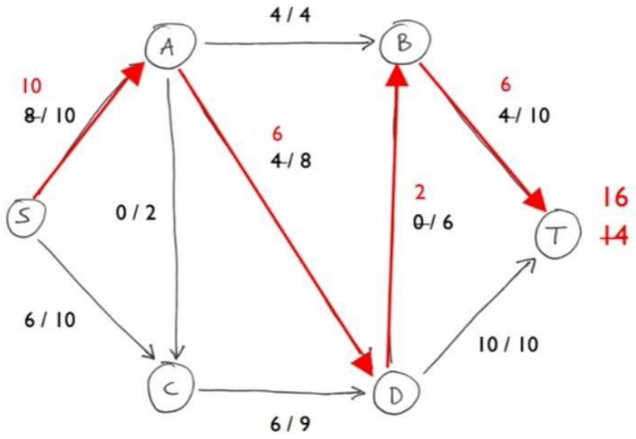
We are done augmenting here using *(non-full forward)* so let's choose another path:



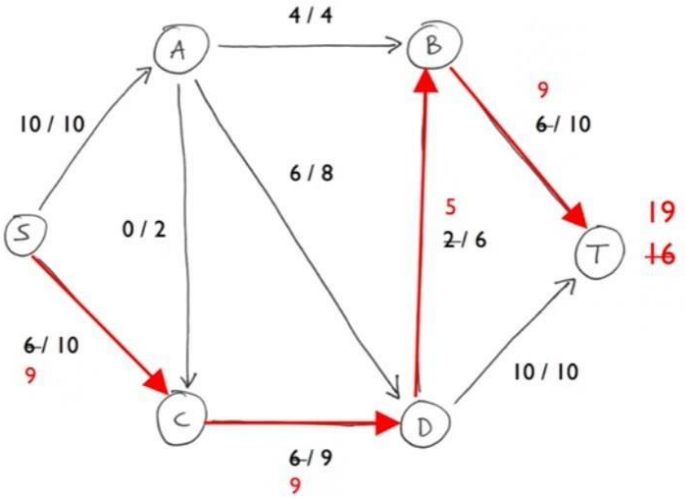
Now let's choose a path specifically with a backward edge because this is how this algorithm works, the backward edge will be from D to A instead of A to D:



Let’s take a new path, in this case we have a bottleneck capacity of 2 from S to A:



Now the last path that we can choose:



Now we can conclude that the max-flow is 19.

Now we can conclude that the max-flow is 19, therefore the minimum cut*(edge connectivity)* of this graph is 19

2.2 Definition and Usage of Augmenting Path:

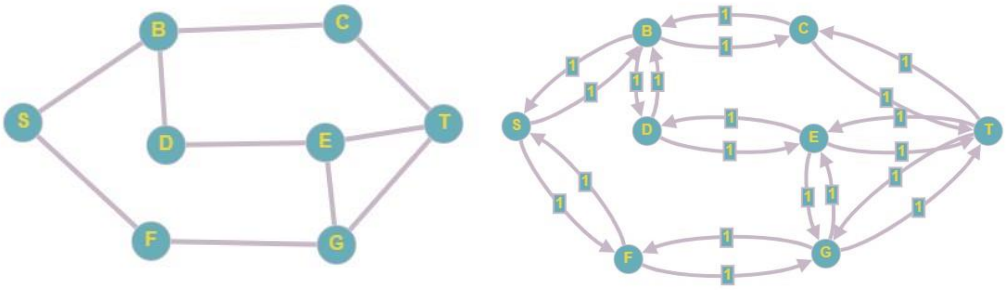
The reason why we are using the above algorithm is to obtain individual augmenting paths that can be used to increment an existing flow. So you may ask what is exactly is an augmenting path ?

Augmenting paths are basically paths from the source to the sink of the network than increase the existing flow of this network. So using edges not used in the flow, or edges partially used, we can obtain paths that increase the flow, but we have to not forget that we can also decrease the used flow for a used edge, so that it can be re-used in another augmenting path *(as we can observe from the example above)*. But in our case this will slightly change but with the same concept and we are going to see why and how in the following sections.

2.3 Constructing a network from Undirected graph:

Since we are dealing with a undirected unweighted graph, we need to construct a network which is directed graph that its edges have capacities. so we can use the Ford-Fulkerson and find the maximum flow.

The simplest way to do that, is to replace every undirected edge between nodes in the graph with two edges that are backwards and forwards between those node, then we are going to assume that each edge in this graph*(backwards or forwards)* have a capacity of 1. So let's see in the following figures as an example of how we build a flow network from a undirected graph:



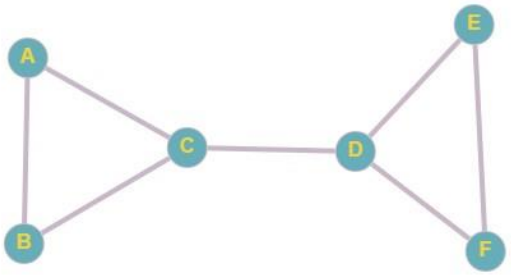
2.3.1 Choosing the source and sink nodes:

After building the network from the undirected graph we should assign the source node of the network with one of the vertices from the undirected graph that have the minimum degree of the graph.
The Degree of a vertex is the number of edges that are incident to the vertex. **The minimum degree** of a graph is minimum of its vertices' degrees.

For example: in the graph above **(*Section 2.3*)** if we assign B as the source node and T as the sink node, if we choose (B, S, F, G, T), (B, D, E, T) and (B, C, T) as augmenting paths we will have a maximum flow equal to 3 therefor edge connectivity equal to 3, and we will get this same value whichever node we choose at a sink other than the source node. The real value of edge connectivity should be 2. So it is better if choose S or F or C or D as a source node so we can have always the right value of edge connectivity.

Now what about the sink node? How can we choose it?

Let’s take the following graph as an example:



If we take node A as a source node and we choose B as a sink node, and we choose (A,B) and (A,C,D,E,F,D,C,B) as augmenting paths we will have a maximum flow equal to 2 therefor edge connectivity equal to 2, and that’s wrong. The real value of edge connectivity should be 1.
So to prevent for this from happening, the algorithm will take each node other than the source as a sink node and it will calculate the maximum flow in each case, then it will return the minimum of the maximum flow between all those cases as a real maximum flow therefor the edge connectivity of the graph.

2.3.2. Concluding the edge connectivity of the undirected graph:

First of all, since the network that we constructed have a max capacity of 1 for all edges *(We assume in a network that every 2 opposite directed edges between the same 2 nodes as one edge as a capacity of 1 because they are constructed from one undirected edge in the undirected graph),* then we don’t have to worry about the maximum flow in the augmenting path edges like we explained in FordFulkerson algorithm. The flow will always be 1 in each augmenting path.

Secondly, by Max-flow Min-cut theorem, and since every edge of a network have a max capacity of 1 and edge connectivity(minimum cut) of a network is the sum of the capacities of the edges that their cut will disconnect the network, and disconnect the source node from the sink.
Therefor the edge connectivity (minimum cut) of the network will be equal to the exact number of the edges that their cut will disconnect the network.

In conclusion because the network is constructed from the undirected graph therefore the edge connectivity of the network will be equal to the edge connectivity of the undirected graph, but this only true in the optimal case when we choose the optimal sink source.

2.3.3. Boundaries of the maximum flow in our case:

Let’s take in consideration all cases of the network for each sink node. In the optimal case the edge connectivity of a undirected graph is equal to the maximum flow of the network, and edge connectivity is not greater than minimum degree of a graph, therefor maximum flow in that case would be bounded between 1 and the minimum degree, but for all the other cases, the edge connectivity of a undirected

graph would be less than or equal to the maximum flow, hence the maximum flow of the network would be bounded between 1 and the maximum degree.

2.3.4 The difference between this algorithm and Ford-Fulkerson Algorithm:

In the Ford-Fulkerson Algorithm we use the already used edges in some augmenting paths as a backward edges, but in our case since we already have a backward edges for each edge in the network, we only need to increase the capacity of the backward edge by the flow , each time the flow is traversing a forward edge in the augmenting path and we decrease the capacity of the last one simultaneously. In this way, we may use the backward edges later in the other augmented paths.

2.4 Pseudo-code:

Let's consider we have a simple undirected graph G with nodes u,v ∈ V, we construct using this G a Network graph H that have:

- deg(nodes): Degree of a node (number of edges that are incident to the node) - minDegree: minimum degree of the graph nodes.
- s : Source node.
- t : Sink node.
- Flow of the network f_graph.
- Capacity of each edge in the graph c (how much flow can pass between nodes).
- Considering a pair of nodes u,v ∈ V, which V are the nodes in the graph(undirected and network, u and v can be whatever 2 nodes that are connected with each other in the graph).
- Considering nodes a,b,c,… ∈ V, which V are the nodes in the graph(undirected and network, a or b or c or… can be whatever node in the graph).

```
1.Let Graph G(V,E)
2.Let Graph H(V,E)
3.Let list_of_sink_nodes[ ]
4.Let list_of_max_flows[ ]
5.H(V,E) = G(V,E)  /*Let H be a clone of G*/
6.
7.TransformNetwork(H(V,E)):  /*constructing a flow network from H*/
8.              For each undirected edge {u,v} between every 2 nodes u and v in H:
9.              {u,v} = (u,v), (v,u) /*replace every undirected edge with 2 directed edges */
10.             c(u,v) = 1 /*set the capacities of each 2 directed edges with 1*/
11.             c(v,u) = 1
12.
13.     Let minDegree  /*the minimum degree of the graph*/
14.
15..    For each node b in the graph H:
16.                     if deg(b) == minDegre /*check if vertex have minimum degree*/
```

```
17.                 node s = b /*Assign source node with node that have min degree */
18.                 Break
19.
20.                 For each node c in the graph H:
21.                 If c != s
22.                 list_of_sin k_nodes.append(c) /*Put all the other nodes in a list*/
23.                 Return H(V,E)
24.
25.
26.isAugmentingPath(Path P):  /*Function to check if path is augmenting path*/
27.     Let P = this.P
28.     Let Pstate = False    /*State of the path: if path is an augmenting path or not*/
29.
30.         For each edge (u,v)  between 2 nodes u and v in P:
31.         If c(u,v) !=0      /*if a flow can pass between 2 nodes */
32.         Pstate =True 33.         Else:
34.             Pstate = False
35.             Break 36.
37. If Pstate == True: /*update the edges capacities of the augmenting path*/ 38.
For each edge (u,v)  between 2 nodes u and v in P:
39.             c(u,v) -=1 /* we decrease the capacity of the forward edge*/
40.             c(v,u) +=1 /* we increase the capacity of the backward edge */
41.             Return Pstate          /*True if path exists and False if not*/
42.
43.
44.FindMaxFlow(H(V,E)):
45.                     Let flow f_graph = 0 /* we initialize the flow of the network */
46.                     For each node t in list_of_sink_nodes[ ]
47.                     Let P be a path from s to t
48.                     For each path P in the network H(V,E):
49.                     If isAugmentingPath(P) == True:
50.                     f_graph +=1 /*increase the flow of the network*/
51.
52.         List_of_max_flows.append(f_graph) /*append each maximum flow to
52.                                             the list so we can find the
53.                                             minimum of that list later*/
54.                                             For each edge (u,v)  between 2
                                                nodes u and v in H:
55.                                             c(u,v) = 1 /* reset capacities of edges
                                                in H to use the function
56.                                             isAugmentingPath for each case */
57.                                             c(v,u) = 1
58.
59.     f_graph = min(list_of_max_flows) /*min of this list is the edge connectivity*/
60.     Return f_graph
```

3. Correctness and Time complexity analysis

3.1 Correctness:

An algorithm is said to be correct if it has these criteria right:

- A valid input.

- The algorithm should terminate when at a certain point *(here in this algorithm it should terminates when there is no more augmenting paths for all the cases of sink nodes).*

- After terminating we should have a valid output which in our case is a number that indicate the max flow of the optimal case of the network which is equal to the edge connectivity of the input undirected graph, and the edge connectivity should be bounded between 1 and the minimum degree of the graph.

3.2 Proof:

3.2.1 Introduction of the proof :

Definitions:

- k : the edge connectivity of the input graph.

- f : the output number from our algorithm.

- G : the input undirected unweighted Graph G=(V,E) Where V is the set of the vertices and E is the set of the edges.

- s : the Source node of the network.

- t : be the Sink node of the network

- Residual capacity, is the capacity of an edge after having a flow passing through it, so in our case it would be 0.

- Residual Network, is the network obtained after not having any more augmenting paths.

- A minimum cut of a network is equal to the sum of the capacities of the minimum number of edges that their cut would disconnect the network and disconnect the source from the sink. *(In our case the capacity of every edge is equal to 1, and we treat each 2 opposite edge as a single edge because the network is constructed from G, therefor the minimum cut would be equal exactly to the minimum the number of edges that their cut would disjoint the network and disconnect the source from the sink)*.

- Given a connected network H = (V, E, s, t) with capacities of edges equal to 1, H is constructed from G.

- we have to consider that we have many cases of H, where in every case we choose different sink nodes t, but the source node s is always the same.

3.2.2 Proof that f = K:

First, let's consider a pair of nodes s, t ∈ V . A (s − t) cut is a partition of V into S and T = V − S, (V = S U T) such that source node s ∈ S and sink node t ∈ T. The capacity of the cut, represented as c(S, T), is given by:

$$c(S,T) = \sum_{u \in S,\, v \in T} c(u \rightarrow v)$$

Theorem(Max_flow min_cut theorem): For any flow f and any cut (S, T), value(f*) ≤ c(S,T).*

Now , we consider a cut (S, T) in graph H = (V, E , s, t). Any flow in this graph (in each case) must be smaller than or equal to the capacity of this cut, since every flow unit must cross the cut.

Then since this predication is valid for all cuts and for all flows, we deduce that the value of the maximum flow is never greater than the capacity of the minimum cut.

For each case of sink node, the algorithm of finding maximum flow terminates when no augmenting path can be found. Then there would be a set of nodes that are reachable from the source node s in the resulting residual network.
Let R be this set of nodes, s ∈R, and let R′ be the set of the unreachable nodes from s, so t ∈R'.

Clearly, R′ = V − R, so pair (R, R′) is a cut. Upon termination of each case, each edge formed by a given node u ∈ R and a given node v ∈ R′ carries a flow equal to its original capacity, in the direction u → v, otherwise the residual capacity of the pair of nodes wouldn't be 0, and v would be reachable in the residual network.

Since all these edges carry the flow in the direction u → v, there is no flow unit being carried from R′ to R, and no flow unit can traverse this cut more than once. Then we achieved a total flow in the network which is maximum flow.

Thus, the maximum flow from s to t, is equals to c(R,R') (minimum cut) so we can replace the inequality with equality in Max_flow min_cut theorem, value(f*) = c(R,R') where here f* is the maximum flow. Then the maximum flow of the network is equal

to the minimum number of edge of the network that their cut disconnect the network and disconnect source node from sink node **(already explained why in the definition above)**

In conclusion, since the network H is constructed from the graph G, and they have the same edges and vertices, and we concluded before that the minimum cut of the network in each case is equal to the minimum number of edges of the network that their cut disconnect the network and disconnect source node from sink node ,and since the edge connectivity of a undirected graph is equal to the minimum number of edges that their cut would disconnect the graph but in graph we don't care about sink and source being disconnected or not, as long as the removal of those edges disconnect the graph, unlike when we are calculating the minimum cut of the networks, so that's why f – *(output of our algorithm)* – which is the minimum between all the maximum flows*(minimum cuts)* of all the cases of the network would be the edge connectivity k of Graph G.

Finally, if we consider to choose whatever minimum cut from our cases of the network as the edge connectivity of G we would get one of the graph cuts, but it's not guaranteed that it's the minimum cut which is the true edge connectivity, that's why f should always be the minimum between those cuts. Hence **f = k**.

3.3 Time complexity analysis

Let G=(V,E) be the input graph.
V is the set of vertices, E is the set of edges
Let Δ be the maximum degree of G

First of all, transforming a undirected graph into a flow network we have to use Depth-first search (DFS) or breadth-first search(BFS) because we have to pass through every node and edge in order to construct the network and one of those searches will have a time complexity of O(|E|+|V|)

Secondly, The Ford-Fulkerson algorithm depends mainly on finding the augmented path, this path can be found using Depth-first search (DFS) or breadth-first search(BFS) because we have to pass through every node and edge to find the augmenting path so, the time complexity of one of those searches will be O(|E|+|V|).

Thirdly, we shouldn't forget that, each time we search and find an augmenting path we should upgrade the Flow with 1 until we reach the maxFlow, so the time complexity of the Ford-Fulkerson that include finding augmenting path then upgrading the flow of the network till we don't have augmenting paths any more, would be O((|E|+|V|)*maxFlow) which is O(|E|*maxFlow).

Finally, we have to find the maxFlow for every node of a graph as a sink except the source node that we chose, therefor the time complexity of Ford-Fulkerson in our project would be O(|E|*maxFlow*(|V|-1)) ~ O(|E|*maxFlow*|V|), and we should consider that in an optimal tree the maxFlow is bounded by 1 and the minimum

degree of the graph but in the worst case scenario it would be bounded between 1 and the maximum degree of the graph
Hence the final time complexity of all the algorithm would be : O(|E|+|V|) + O(|E|*Δ*|V|) = **O(|E|*Δ*|V|).**

4. Input and output description

- Input: The format of the input file would be:
    1.An integer that represent the number of vertices in the graph
    2.Another integer that represent the number of the edges in the graph
    3.A list of vertices [V1,V2,V3,V4…., Vn]
    4.A two-dimensional list [[V1,V2],[V1,V3],[V1,V3]…..[Vi,Vj]] that determine the connection (edges) between each 2 vertices (V1,V2), (V1,V3), (V1,V4)…..(Vi,Vj).

- Output: The output would be an integer which represent the edge connectivity, we discussed in the previous sections how we can get it and how it is bounded.

5. References:

The images used in this report are from:
https://www.youtube.com/watch?v=Tl90tNtKvxs&ab_channel=MichaelSambol