Analysis of different algorithms for maximum flow problem

November 7, 2022

Sarvottam Swaroop (2021CSB1129) ,
Simran Kaur (2021CSB1134) ,
Vanshika Dhamija (2021CSB1138)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Sravanthi Chede

Summary: Our project entails the analysis and implementation of two different algorithms for maximum flow problem and analysing the better algorithm.Maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum.There are different algorithms to solve the problem. In our project we are implementing two algorithms :- Ford–Fulkerson algorithm and Dinic's algorithm which are based on the idea of residual graph and level graph respectively. We will show both theoretically as well as experimetally that these algorithms have different time complexities and on the basis of the run time we will find out which algorithm is better
Ford-Fulkerson has a time complexity of O(maxflow*E) where E is the number of edges of the graph. Dinic's algorithm has a time complexity of O($V^2 * E$).

---

1. Introduction

A graph is a non-linear data structure made up of nodes and edges. The edges represent some correlation between the nodes. Nodes are sometimes known as vertices, while edges are lines or arcs that connect any two nodes in the network.
A flow network G D (V, E) is a directed graph in which each edge (u,v) belonging to E has a nonnegative capacity c(u,v) greater than or equal to 0. We further require that if E contains an edge (u,v), then there is no edge (u,v) in the reverse direction.
Maximum flow problems involve finding a feasible flow through a flow network that obtains the maximum possible flow rate. The maximum value of a source to sink flow is equal to the minimum capacity of that source to sink path in the network. Each edge is labeled with capacity. We can solve the max flow problem with the help of different algorithms. Ford-fulkerson and Dinic's algorithm being two of them. Before implementing them, we first have to get a basic idea of Breadth First search and Depth First Search algorithms.

BFS, Breadth-First Search, is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue.The time complexity of BFS is O(V+E) when we use adjacency list and O(V*V) when we use adjacency matrix, where V represents vertices and E represents edges.
DFS, Depth First Search, is an edge-based technique. It uses the Stack data structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped. The time complexity of BFS is O(V+E) in case of adjacency list and O(V*V) in case of adjacency matrix.

Ford-fulkerson algorithm assumes a network as a graph with a starting vertex 'S' and an ending vertex 'E'. The intuition of the algorithm is that as long as there is a path from 'S' to 'E' that can take some flow the entire way, we send it. The path is called an augmenting path. We have to do this until there are no more augmenting paths.Ford Fulkerson algorithm has time complexity O(|E| * f), where |E| is the number of edges and 'f' is the

maximum flow of a network because it depends on the maximum flow of the network.An augmenting path is a simple path from source to sink which do not include any cycles and that pass only through positive weighted edges. A residual network graph indicates how much more flow is allowed in each edge in the network graph. If there are no augmenting paths possible from source to sink , then the flow is maximum.

In Dinic's algorithm we create the level graph by doing BFS from the source node in order to label all levels of the current flow graph.If the sink is never reached while building the level graph, stop and return the max flow. Using only valid edges in the level graph created, perform multiple DFS from the source node to the sink node until a blocking flow is reached, and sum up the bottleneck values of all the augmenting paths that were found to calculate the max flow. A flow is blocking flow if there is no more flow that can be sent through an edge using the level graph.The time complexity of Dinic's algorithm is O(V * V * E). There are fewer than V phases possible, and each phase takes O(V * E) time. Hence its time complexity is O(V * V *E).

2. Implementation

In this section, we'll explain you the working of Ford Fulkerson's and Dinic's algorithm with a sample graphs

2.1. Ford Fulkerson's Algorithm

Here, the working of Ford Fulkerson will be portrayed on the sample graph(Graph 1)

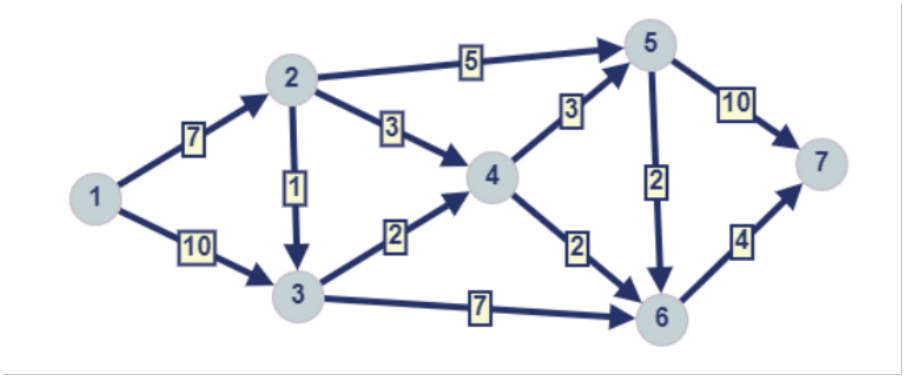

Figure 1: Graph 1

Ford Fulkerson works on the concept of DFS. Select any path from source(1) to sink(7). Let us first consider the path 1-2-5-7. Minimum capacity is 5. So, residual graph 2 will be obtained by taking 5 out from all paths and adding 5 in Maximum flow. The resulting residual graph is Graph 2. And maximum flow =5

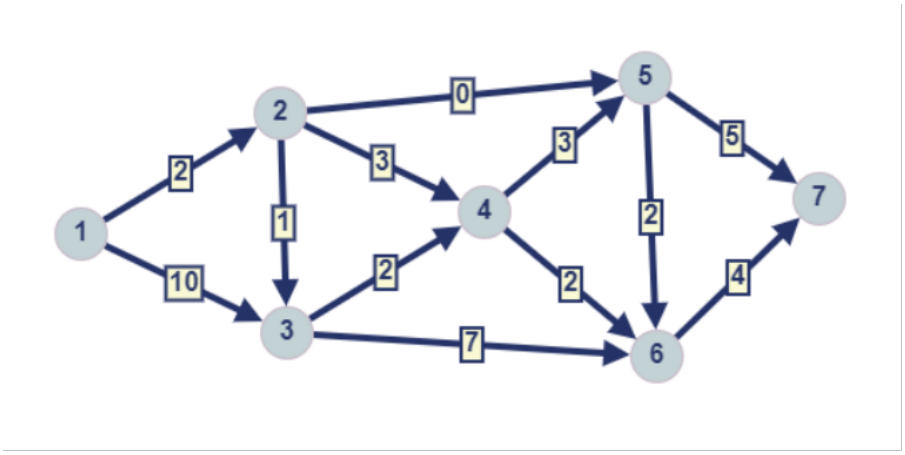

Figure 2: Graph 2

Select another path 1-3-6-7. Minimum capacity is 4. Residual graph is given in Graph 3 and maximum flow is 5+4=9.

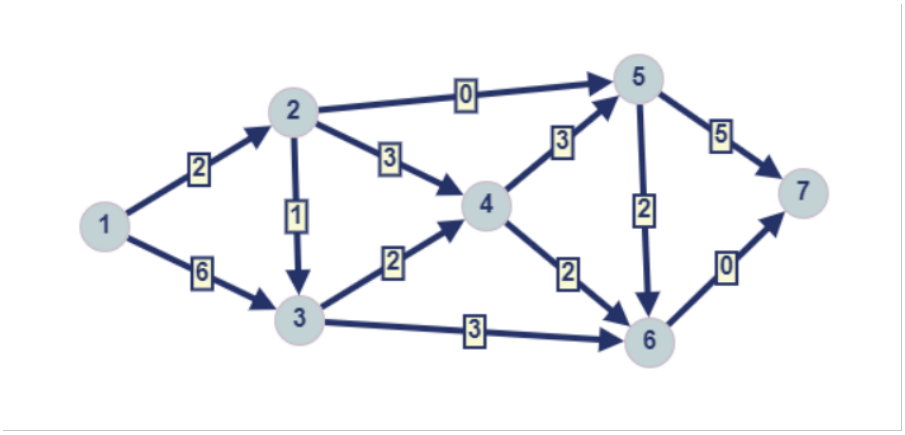

Figure 3: Graph 3

Next path we take is 1-2-4-5-7. Minimum capacity now is 2. Residual graph is given in Graph 4 and maximum flow is 5+4+2=11.

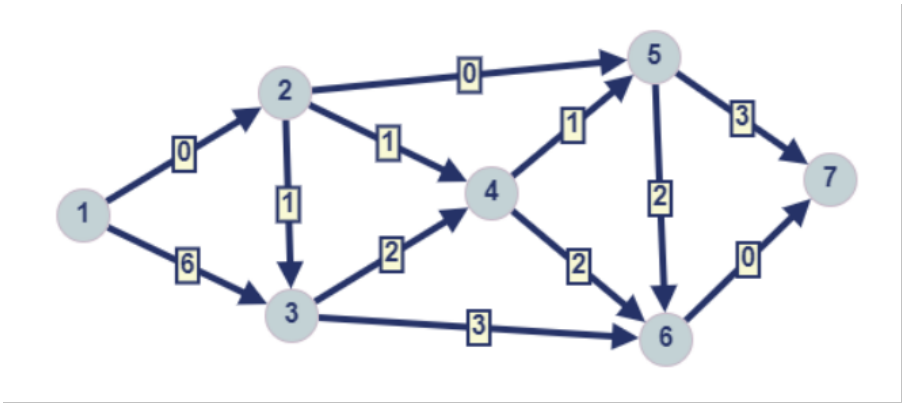

Figure 4: Graph 4

Last possible path is 1-3-4-5-7 with minimum capacity 1. Residual graph is given in Graph 5 and maximum flow is 5+4+2+1=12.

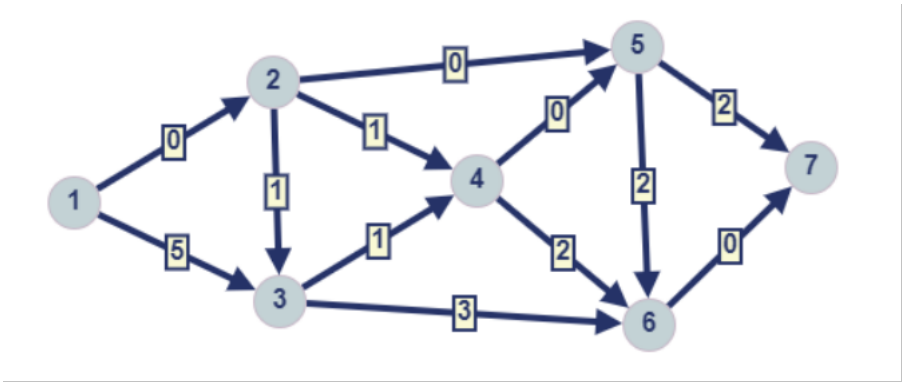

Figure 5: Graph 5

2.2. Dinic's Algorithm

Here, the working of Dinic's algorithm will be portrayed on the sample graph(SampleGraph)

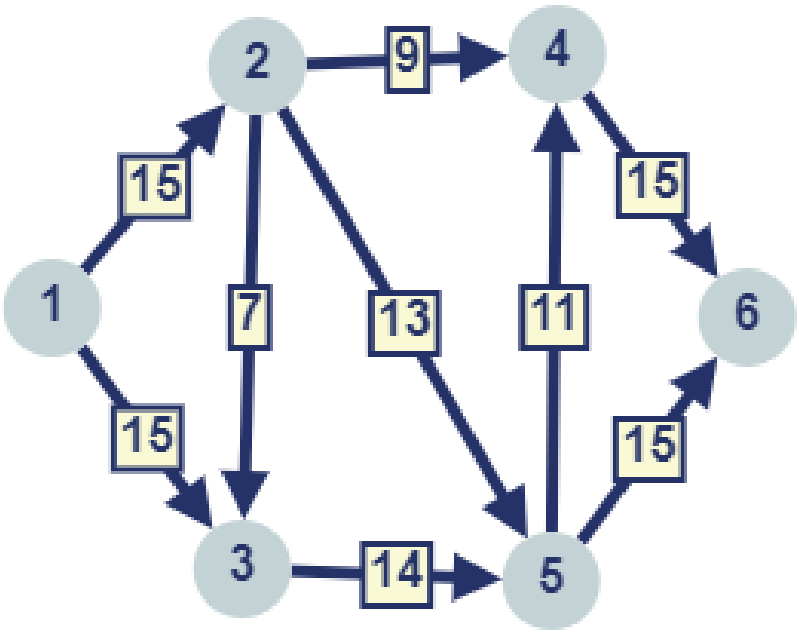

Figure 6: Sample Graph

There are three blocking flows in the given sample graph shown by the given graphs and their level graphs. Let us assume path 1-2-4-6. Here minimum capacity is 9 so we will add 9 to the maximum flow and will decrease capacity of each edge by 9.

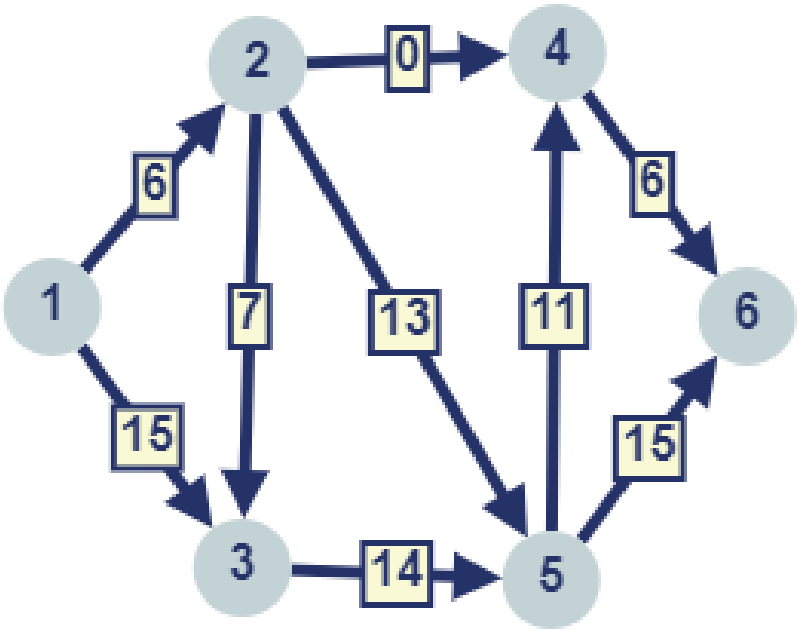

Figure 7: Graph 1

We will get a level graph as shown.

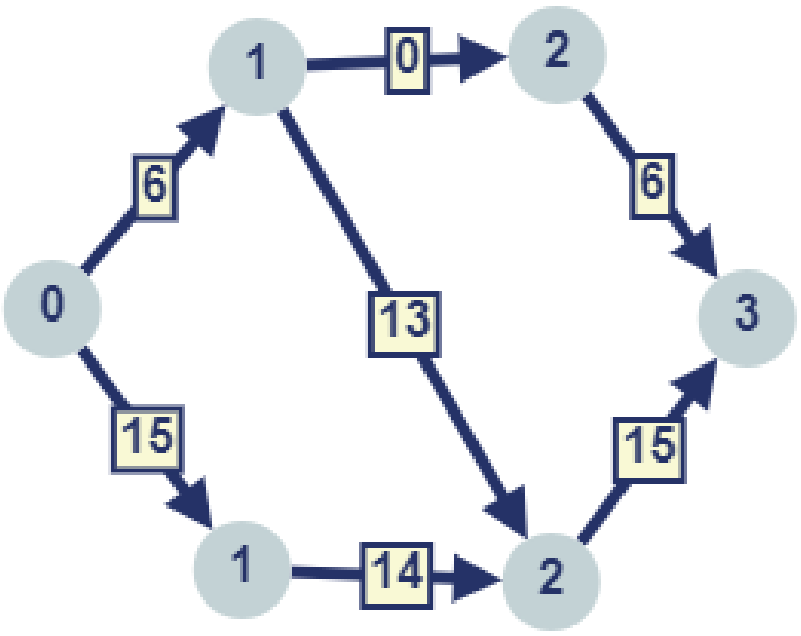

Figure 8: Level Graph of Graph 1

In the path 1-2-5-6 the minimum capacity will be 6 so now the maximum flow will be updated to 9+6 = 15 and the capacities will be decreased from the edges of the path by 6.

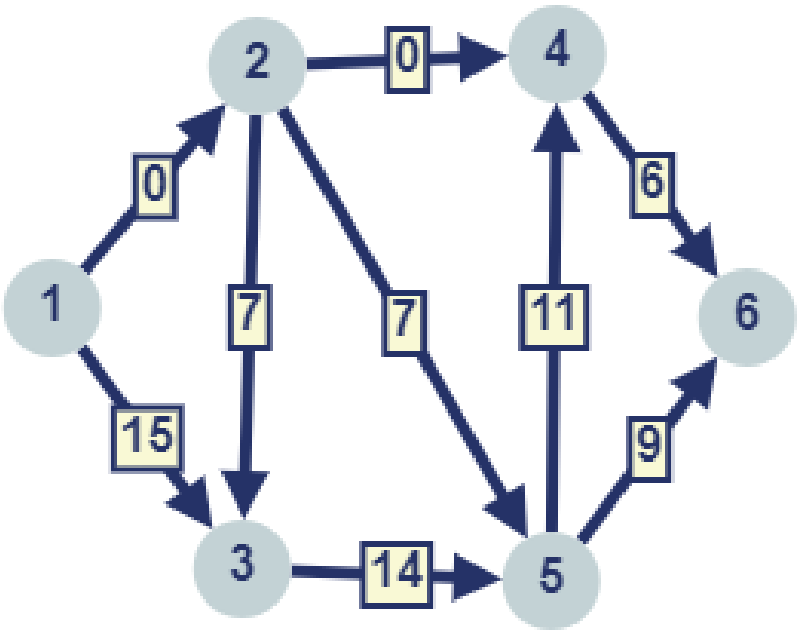

Figure 9: Graph 2

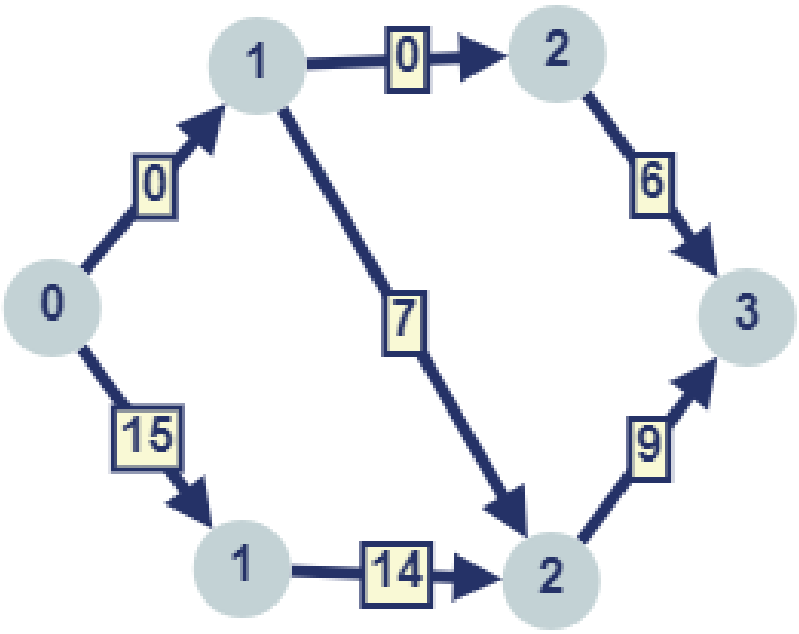

Figure 10: Level Graph of Graph 2

In next path 1-3-5-6 minimum capacity is 9 and the capacities of those particular edges will be decreased, Maximum flow will be updated to 9+6+9=24.

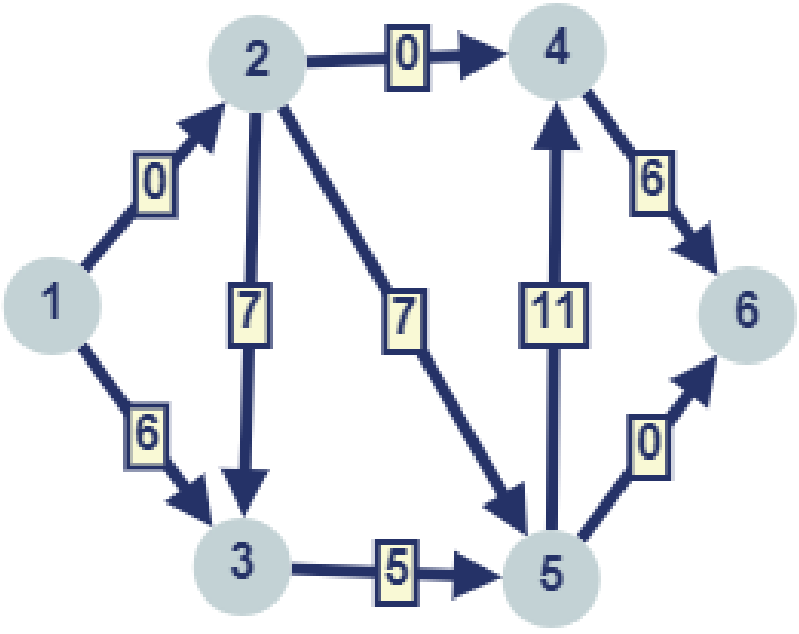

Figure 11: Graph 3

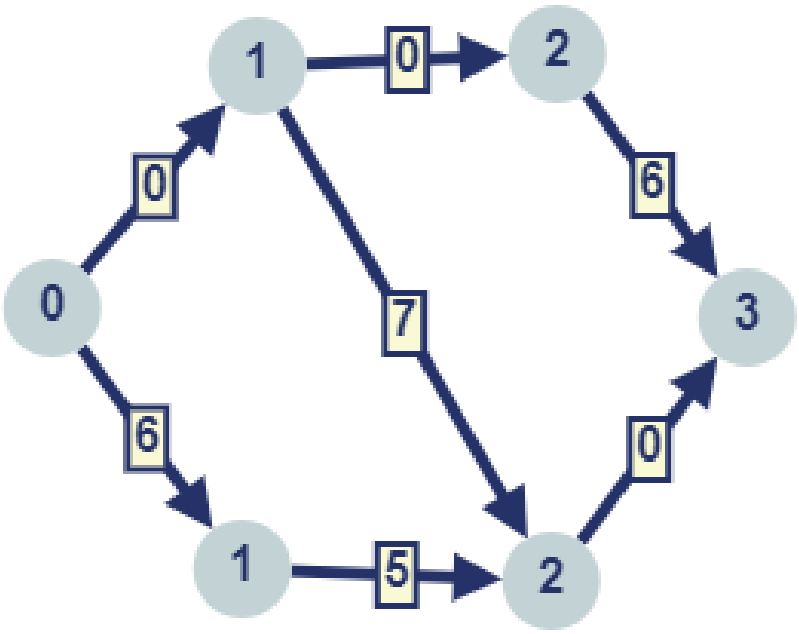

Figure 12: Level Graph of Graph 3

The path 1-3-5-4-6 minimum capacity is 5 so the new graph formed is shown. Maximum flow of 9+6+9+5 = 29 is updated.

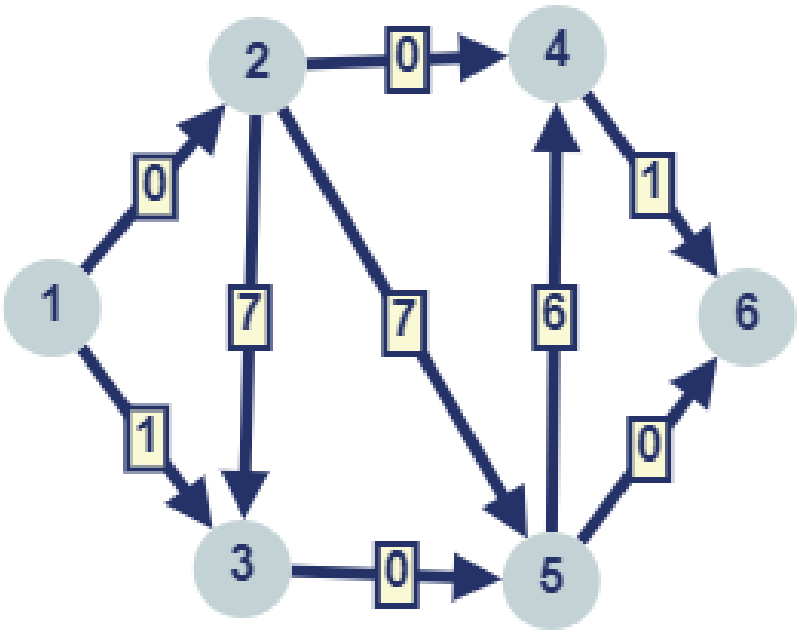

Figure 13: Graph 4

The final graph will be

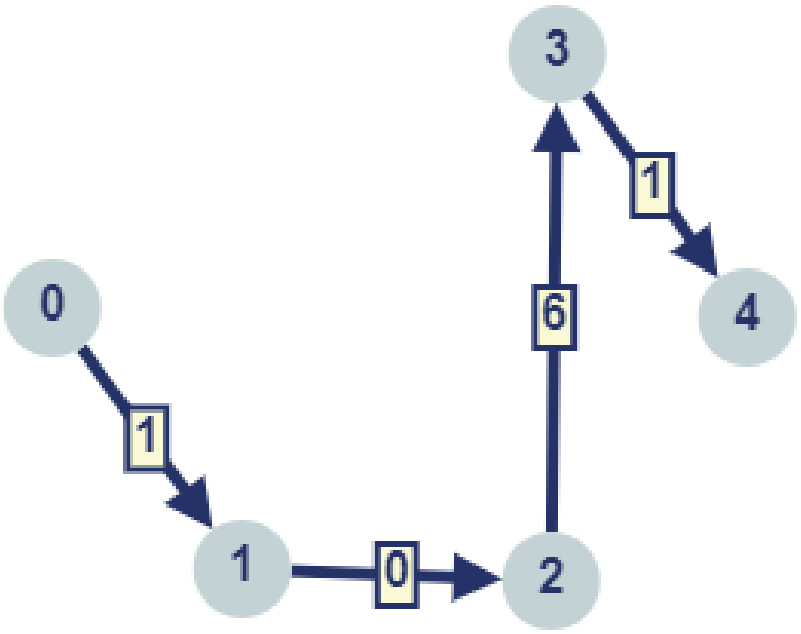

Figure 14: Final Graph

3. Algorithms

Pseudo Algorithms of the 4 main functions used, ie. BFS, DFS, Dinic's algorithm and Ford Fulkerson's algorithm will be shown in this section:

Algorithm 1 Breadth First Search (BFS)

```
Input: Source node s, Graph G, Queue Q
Q.enqueue(s)
mark s as visited
while Q is not empty do
    v=Q.dequeue()
end while
for all neighbours w of v in Graph G do
    if w is not visited  then
        Q.enqueue(w)
        mark w as visited
    end if
end for
```

Algorithm 2 Depth First Search DFS(u , sink , flow)

```
1: if u=sink then
2:    return flow
3: end if
4: for each − vertex − u do
5:    if capacity[u][i]>0 then
6:       countOfU + = 1
7:    end if
8: end for
9: if count[u]= countOfU then
10:   return 0
11: end if
12: for each − vertex − u do
13:   if residualGraph[u][v]>0 then
14:      count[u] + = 1
15:      if level[v]=level[u]+1 then
16:         currentflow = min(flow,residualGraph[u][v])
17:         mincapacity= dfs(v,sink,currentflow)
18:         if (mincapacity>0) then
19:            residualGraph[u][v] - = mincapacity
20:            residualGrapg[v][u] + = mincapacity
21:            return mincapacity
22:         end if
23:      end if
24:   end if
25: end for
```

Algorithm 3 Dinic's Algorithm for max flow

```
1: max-flow=0
2: for each − edge(u, v) − in − G do
3:    flow(u,v)= w(u,v)
4: end for
5: while there is augmenting level path p from s->t in Residual Graph do
6:    Assign level to vertices
7:    while more flow is possible from s->t do
8:       min-cap = minimum residual capacity in path
9:       max-flow = max-flow + min-cap
10:      for each − edge(u, v) − in − path do
11:         flow(u,v)= flow(u,v) - min-cap
12:         flow(u,v)= flow(v,u) + min-cap
13:      end for
14:   end while
15: end while
16: return max-flow
```

Algorithm 4 Ford Fulkerson's Algorithm for max flow

```
max-flow=0
for each − edge(u, v) − in − G do
    flow(u,v)= w(u,v)
end for
while there is a path p from s->t in Residual Graph do
    min-cap = minimum residual capacity in path p
    max-flow = max-flow + min-cap
    for each − edge(u, v) − in − path do
        flow(u,v)= flow(u,v) - min-cap
        flow(u,v)= flow(v,u) + min-cap
    end for
end while
return max-flow
```

4. Time complexity Analysis

In this section we will derive time complexities of the algorithms used

4.1. Breadth First Search(BFS)

Imagine the BFS progressing in levels.
We take a starting vertex S, which is at level - 0. All the adjacent vertices are at level - 1.
Then, we mark all the adjacent vertices of all vertices at level - 1, which don't have a level, to level - 2. So, every vertex will belong to one level only.
And when an element is in a level, we check once for its adjacent vertices, which takes O(|V|) time. As, the frontier covers |V| elements over the course of the algorithm, the total time would become O(|V| * |V|).

4.2. Depth First Search(DFS)

If the graph is represented as adjacency matrix V x V array: To find all of a vertex's outgoing edges, you will have to traverse a whole row of length V in the matrix. Each row in an adjacency matrix corresponds to a node in the graph; each row stores information about the edges that emerge from that vertex. As a result, DFS's temporal complexity in this scenario is O(V * V) = O. (V2).

4.3. Dinic's Algorithm

In Dinic's algorithm there are number of phases with each one having iterations changing flow using shortest path of fixed length let us say l. In the beginning as the Dinitc's algotithm is based on the BFS the BFS builds a time layered network of length l in the time of O(|E|).The layered structure with the absence of any dead end allows for the execution ford fulkerson with time complexity of O(V). The length of l will keep on increasing in each phase and at maximum V-1 phases can be possible. So our maximum time is of the oreder O(E*V*(V-1)) which is equal to O(V*V*E).

4.4. Ford Fulkerson's Algorithm

By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found in the graph. However, there is no certainty that this situation will ever be reached, so the best that can be guaranteed is that the answer will be correct if the algorithm terminates. In the case that the algorithm runs forever, the flow might not even converge towards the maximum flow. However, this situation only occurs with irrational flow values.
When the capacities are integers, the runtime of Ford–Fulkerson is bounded by O(Ef), where E is the number of edges in the graph and f is the maximum flow in the graph. This is because each augmenting path can be found in O(E) time and increases the flow by an integer amount of at least 1, with the upper bound f.
A variation of the Ford–Fulkerson algorithm with guaranteed termination and a runtime independent of the

maximum flow value is the Edmonds–Karp algorithm, which runs in O(V*E*E) time.

5. Tables

In Table 1, you can see the time complexities we discussed about in the previous section.

Time Complexities of different Algorithms in Adjacency Matrix

	Time Complexity
BFS	O(V*V)
DFS	O(V*V)
Dinic's	O(V*V*E)
Ford-Fulkerson	O(maxflow*|E|)

Table 1: Time Complexity of BFS, DFS, Dinic's and Ford-Fulkerson's algorithm

6. Comparison of both Algorithms

From the previous discussions, we can see that Time complexity of Dinic's algorithm is lesser than Ford Fulkerson's algorithm making it more efficient. We'll take few cases and paste the outputs directly from console window to show the same:
For the case with 1000 vertices,

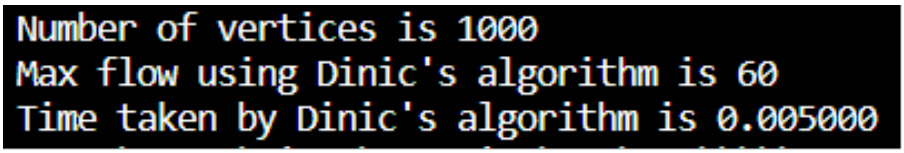

Figure 15: Time taken by Dinic's Algorithm

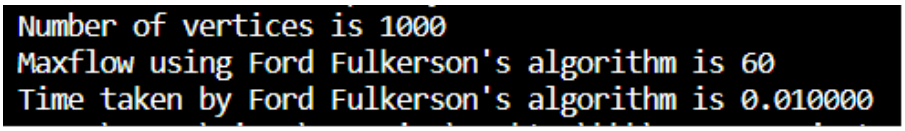

Figure 16: Time taken by Ford Fulkerson's Algorithm

For 5000 vertices,

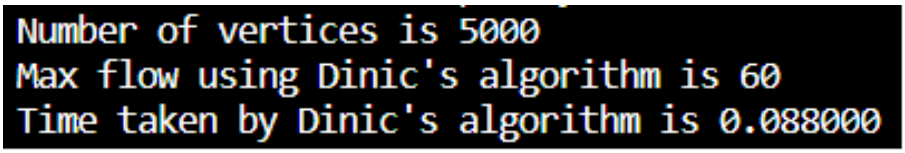

Figure 17: Time taken by Dinic's Algorithm

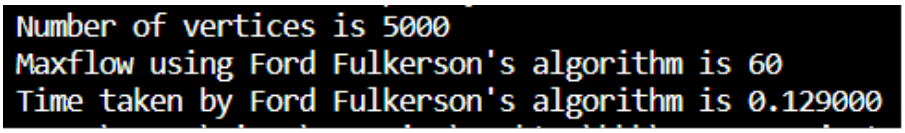

Figure 18: Time taken by Ford Fulkerson's Algorithm

For 6914 vertices,

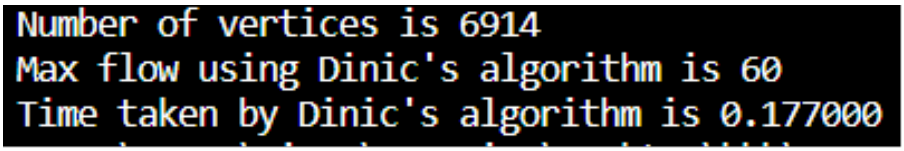

Figure 19: Time taken by Dinic's Algorithm

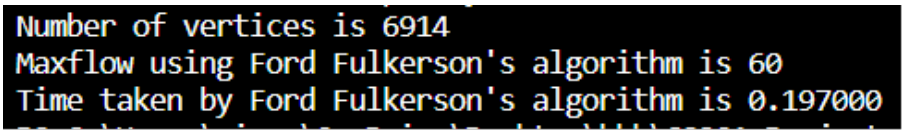

Figure 20: Time taken by Ford Fulkerson's Algorithm

For 10,000 vertices,

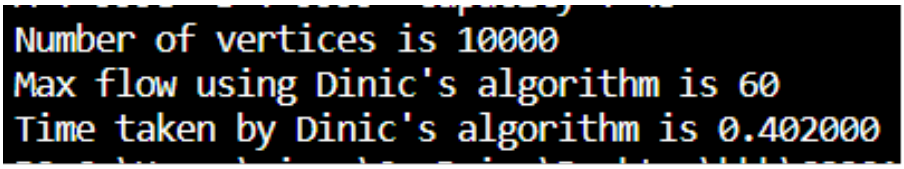

Figure 21: Time taken by Dinic's Algorithm

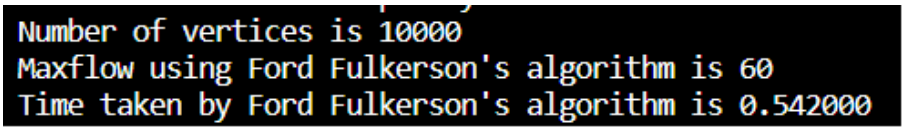

Figure 22: Time taken by Ford Fulkerson's Algorithm

7. Applications

Ford-Fulkerson and Dinic's algorithm can be applied to find the maximum flow between single source and single sink in a graph. In many fields, there exist various crucial applications that can be handled as maximum flow problems. Examples are:
1.Electrical power
2.Airline scheduling
3.Communication networks: Calls in a communication network for particular services
4.Computer sciences: MAximizing packet flow in computer networks
5.Current through electrical networks: : To find the maximum current that could flow the wires without causing any short circuit.
6.Liquids through pipes: In the water/sewage management system, to find maximum capacity of the network.
7. In traffic movements, to find how much traffic can move from a City AA to City BB.

8. Conclusions

1. We studied how we can find out the maximum flow in a graph using two different algorithms namely Ford-Fulkerson and Dinic's algorithm.
2. We did a detailed analysis of their time complexities and showed that dinic's algorithm is more time efficient practically.
3. We also studied about the basic algorithms like BFS, DFS involved in Dinic's and Ford Fulkerson's algorithms
4. We looked at various applications where these algorithms can come in use.

9. Bibliography and citations

Acknowledgements

We wish to thank our instructor, Dr Anil Shukla and our Teaching Assistant Sravanthi Chede for their guidance and valuable inputs provided during the project

References

[1] Nadia Abd-Alsabour. Maximum flow. Technical report, Cairo University, Egypt.

[2] Yefim Dinitz. Dinitz' algorithm: The original version and even's version.

[3] L. Ford and D. Fulkerson. *Flows in Networks.* 1962.

[4] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms.* MIT Press, 1990.

[5] Wikipedia. Maximum flow problem. Technical report, Wikipedia.