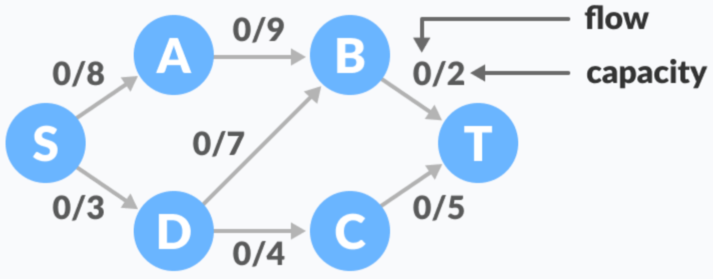
Ford-Fulkerson Algorithm

Ford-Fulkerson algorithm is a greedy approach for calculating the maximum possible flow in a network or a graph.

A term, **flow network**, is used to describe a network of vertices and edges with a source (S) and a sink (T). Each vertex, except **S** and **T**, can receive and send an equal amount of stuff through it. **S** can only send and **T** can only receive stuff.

We can visualize the understanding of the algorithm using a flow of liquid inside a network of pipes of different capacities. Each pipe has a certain capacity of liquid it can transfer at an instance. For this algorithm, we are going to find how much liquid can be flowed from the source to the sink at an instance using the network.



Flow network graph

Terminologies Used

Augmenting Path

It is the path available in a flow network.

Residual Graph

It represents the flow network that has additional possible flow.

Residual Capacity

It is the capacity of the edge after subtracting the flow from the maximum capacity.

How Ford-Fulkerson Algorithm works?

The algorithm follows:

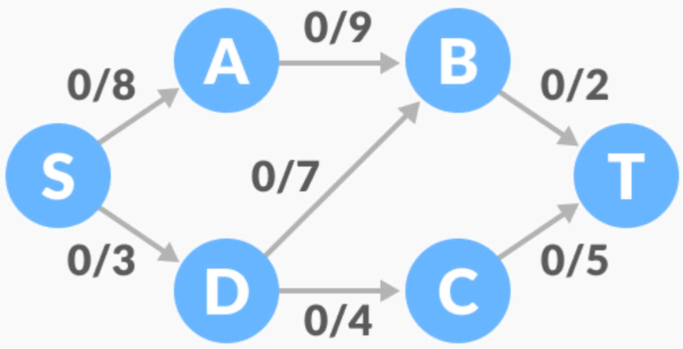
1. Initialize the flow in all the edges to 0.
2. While there is an augmenting path between the source and the sink, add this path to the flow.
3. Update the residual graph.

We can also consider reverse-path if required because if we do not consider them, we may never find a maximum flow.

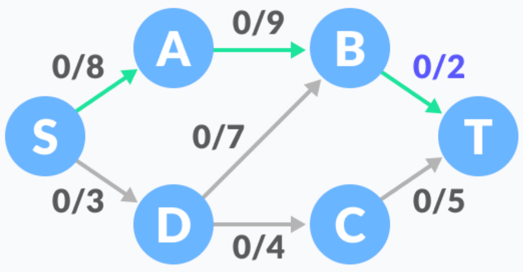
The above concepts can be understood with the example below.

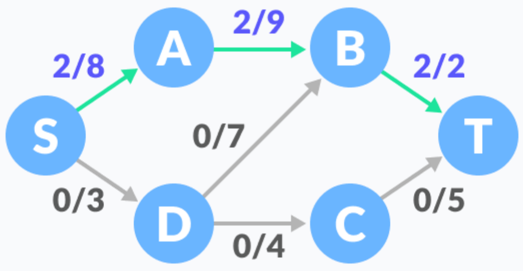
Ford-Fulkerson Example

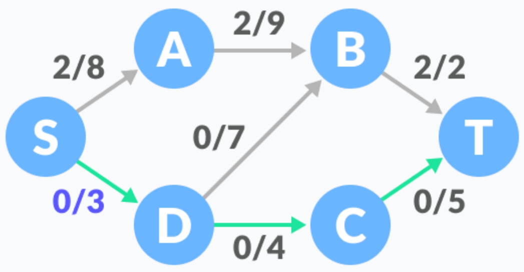
The flow of all the edges is 0 at the beginning.



Flow network graph example

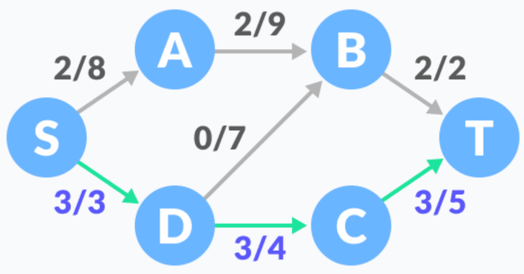
1. Select any arbitrary path from S to T. In this step, we have selected path `S-A-B-T`.


Find a path
The minimum capacity among the three edges is 2 (`B-T`). Based on this, update the `flow/capacity` for each path.


Update the capacities
2. Select another path `S-D-C-T`. The minimum capacity among these edges is 3 (`S-D`).


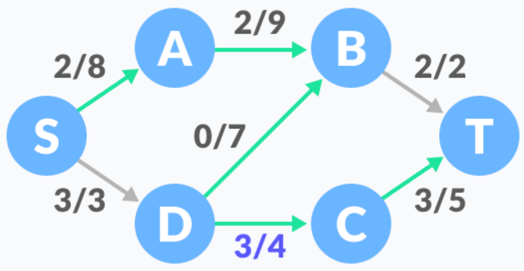
Find next path

Update the capacities according to this.

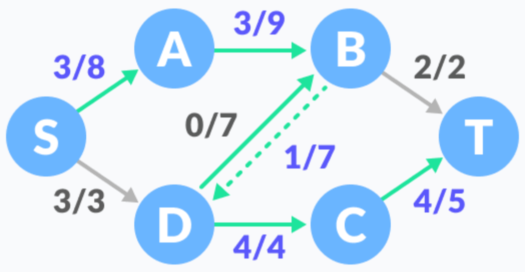


Update the capacities

3. Now, let us consider the reverse-path `B-D` as well. Selecting path `S-A-B-D-C-T`. The minimum residual capacity among the edges is 1 (`D-C`).



Find next path



Updating the capacities.

Update the capacities

The capacity for forward and reverse paths are considered separately.

4. Adding all the flows = 2 + 3 + 1 = 6, which is the maximum possible flow on the flow network.

Note that if the capacity for any edge is full, then that path cannot be used.

Java Examples

```java
// Ford-Fulkerson algorithm in Java

import java.util.LinkedList;

class FordFulkerson {
  static final int V = 6;

  // Using BFS as a searching algorithm
  boolean bfs(int Graph[][], int s, int t, int p[]) {
    boolean visited[] = new boolean[V];
    for (int i = 0; i < V; ++i)
      visited[i] = false;

    LinkedList<Integer> queue = new LinkedList<Integer>();
    queue.add(s);
    visited[s] = true;
    p[s] = -1;

    while (queue.size() != 0) {
      int u = queue.poll();

      for (int v = 0; v < V; v++) {
        if (visited[v] == false && Graph[u][v] > 0) {
          queue.add(v);
          p[v] = u;
          visited[v] = true;
        }
      }
    }

    return (visited[t] == true);
  }

  // Applying fordfulkerson algorithm
  int fordFulkerson(int graph[][], int s, int t) {
    int u, v;
    int Graph[][] = new int[V][V];

    for (u = 0; u < V; u++)
      for (v = 0; v < V; v++)
        Graph[u][v] = graph[u][v];

    int p[] = new int[V];
    int max_flow = 0;

    # Updating the residual calues of edges
    while (bfs(Graph, s, t, p)) {
      int path_flow = Integer.MAX_VALUE;
      for (v = t; v != s; v = p[v]) {
        u = p[v];
        path_flow = Math.min(path_flow, Graph[u][v]);
      }

      for (v = t; v != s; v = p[v]) {
        u = p[v];
        Graph[u][v] -= path_flow;
        Graph[v][u] += path_flow;
      }

      // Adding the path flows
      max_flow += path_flow;
    }

    return max_flow;
  }

  public static void main(String[] args) throws java.lang.Exception {
    int graph[][] = new int[][] { { 0, 8, 0, 0, 3, 0 }, { 0, 0, 9, 0, 0, 0 }, { 0, 0,
0, 0, 7, 2 },
        { 0, 0, 0, 0, 0, 5 }, { 0, 0, 7, 4, 0, 0 }, { 0, 0, 0, 0, 0, 0 } };
    FordFulkerson m = new FordFulkerson();

    System.out.println("Max Flow: " + m.fordFulkerson(graph, 0, 5));

  }
}
```

Ford-Fulkerson Applications

- Water distribution pipeline
- Circulation with demands