**Implementation of Network Flow Algorithm using Ford-Fulkerson Method**

Theory:

Residual Graph of a flow network is a graph which indicates additional possible flow. If there is a path from source to sink in residual graph, then it is possible to add flow. Every edge of a residual graph has a value called residual capacity which is equal to original capacity of the edge minus current flow. Residual capacity is basically the current capacity of the edge. . Residual capacity is 0 if there is no edge between two vertices of residual graph. We can initialize the residual graph as original graph as there is no initial flow and initially residual capacity is equal to original capacity. To find an augmenting path, we can either do a BFS or DFS of the residual graph. Using BFS, we can find out if there is a path from source to sink. BFS also builds parent[] array. Using the parent[] array, we traverse through the found path and find possible flow through this path by finding minimum residual capacity along the path. We later add the found path flow to overall flow

Code:

#include <iostream>  
#include <bits/stdc++.h>  
using namespace std;  
  
#define V 6  
  
bool bfs(int graph[V][V], int s, int t, int parent[]){  
    bool visited[V];  
    for(int i = 0; i<V; i++) visited[i] = 0;  
  
    queue<int> q;  
    visited[s] = true;  
    q.push(s);  
    parent[s] = -1;  
    // cout<<s<<" ";  
  
    while(!q.empty()){  
        int u = q.front();  
        q.pop();  
  
        for(int v = 0; v<V; v++){  
            if(visited[v]==false && graph[u][v]>0){  
                visited[v] = true;  
                q.push(v);  
                parent[v] = u;  
                // cout<<v<<" ";  
            }  
        }  
    }  
    return (visited[t]==true);  
}  
  
int ford\_fulkerson(int graph[V][V], int s, int t){  
    int rgraph[V][V];  
    for(int u=0; u<V; u++){  
        for(int v=0; v<V; v++){  
            rgraph[u][v] = graph[u][v];  
        }  
    }  
  
    int max\_flow = 0;  
    int parent[V];  
     
    bool bb = bfs(rgraph, s, t, parent);  
  
    while(bfs(rgraph, s, t, parent)){  
        int path\_flow = INT\_MAX;  
        for(int v=t; v!=s; v=parent[v]){  
            int u = parent[v];  
            path\_flow = min(path\_flow, rgraph[u][v]);  
        }  
  
        for(int v=t; v!=s; v=parent[v]){  
            int u = parent[v];  
            rgraph[u][v]-=path\_flow;  
            rgraph[v][u]+=path\_flow;  
        }  
        // cout<<path\_flow<<endl;  
        max\_flow+=path\_flow;  
    }  
  
    return max\_flow;  
}  
  
  
int main(){  
  
    int graph[6][6] = {{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}};  
     
    int max\_flow = ford\_fulkerson(graph, 0, 5);  
    cout<<"max flow: "<<max\_flow<<endl;  
    return 0;  
}

Output:

max flow: 6

Observations:

The above implementation uses adjacency matrix representation though where BFS takes O(V2) time, the time complexity of the above implementation is O(EV3)

Conclusion:

Flow network is a graph which indicates additional possible flow. If there is a path from source to sink in residual graph

Residual capacity is 0 if there is no edge between two vertices of residual graph

References:

<https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/>