

## FordFulkerson.cpp

```

#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
using namespace std;
// Number of vertices in given graph
#define V 6
/* Returns true if there is a path from source 's' to sink 't' in
   residual graph. Also fills parent[] to store the path */
bool bfs(int rGraph[V][V], int s, int t, int parent[]){
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty()){
        int u = q.front();
        q.pop();
        for (int v=0; v<V; v++){
            if (!visited[v] && rGraph[u][v] > 0){
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    // If we reached sink in BFS starting from source, then return true, else false
    return visited[t];
}
// Returns the maximum flow from s to t in the given graph
int fordFulkerson(int graph[V][V], int s, int t){
    int u, v;
    // Create a residual graph and fill the residual graph with
    // given capacities in the original graph as residual capacities
    // in residual graph
    int rGraph[V][V]; // Residual graph where rGraph[i][j] indicates
                       // residual capacity of edge from i to j (if there
                       // is an edge. If rGraph[i][j] is 0, then there is not)
    for (u = 0; u < V; u++) for (v = 0; v < V; v++) rGraph[u][v] = graph[u][v];
    int parent[V];
    int max_flow = 0; // There is no flow initially
    while (bfs(rGraph, s, t, parent)){
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v]){
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]); }
        // update residual capacities of the edges and reverse edges
        // along the path
        for (v=t; v != s; v=parent[v]){
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow; }
        // Add path flow to overall flow
        max_flow += path_flow;
    }
    return max_flow;
}
int main(){
    int graph[V][V] = { {0, 16, 13, 0, 0, 0}, {0, 0, 10, 12, 0, 0}, {0, 4, 0, 0, 14, 0},
                        {0, 0, 9, 0, 0, 20}, {0, 0, 0, 7, 0, 4}, {0, 0, 0, 0, 0, 0}};
    cout << "The maximum possible flow is " << fordFulkerson(graph, 0, 5); }

```