// C++ program for implementation of Ford Fulkerson algorithm

#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[])

{

    // Create a visited array and mark all vertices as not visited

    bool visited[V];

    memset(visited, 0, sizeof(visited));

    // Create a queue, enqueue source vertex and mark source vertex

    // as visited

    queue <int> q;

    q.push(s);

    visited[s] = true;

    parent[s] = -1;

    // Standard BFS Loop

    while (!q.empty())

    {

        int u = q.front();

        q.pop();

        for (int v=0; v<V; v++)

        {

            if (visited[v]==false && 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] == true);

}

// 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];  // This array is filled by BFS and to store path

    int max\_flow = 0;  // There is no flow initially

    // Augment the flow while tere is path from source to sink

    while (bfs(rGraph, s, t, parent))

    {

        // Find minimum residual capacity of the edges along the

        // path filled by BFS. Or we can say find the maximum flow

        // through the path found.

        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 the overall flow

    return max\_flow;

}