Assignment 08 | Advance Algorithms CE-092

Assignment submission for Advance Algorithms subject week 8.

nevilparmar24@gmail.com

Task 1:

FordFulkerson algorithm to find the max flow in the given graph.

Code:

I have implemented this algorithm using bfs and dfs both.
fordFulkerson : Using DFS
fordFulkerson2 : Using BFS

```
/*
* @Author: nevil
* @Date:   2020-09-23 18:46:30
* @Last Modified by:   nevil
* @Last Modified time: 2020-09-23 18:57:36
*/

#include <bits/stdc++.h>
using namespace std;

#define V 6

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] == false && rGraph[u][v] >
0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

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

bool dfsutil(int rGraph[V][V], int s, int t, int
parent[], bool visited[]) {
    visited[s] = true;
    for (int i = 0; i < V; i++) {
        if (visited[i] == false && rGraph[s][i] > 0 ) {
            parent[i] = s;
            dfsutil(rGraph, i, t, parent, visited);
```

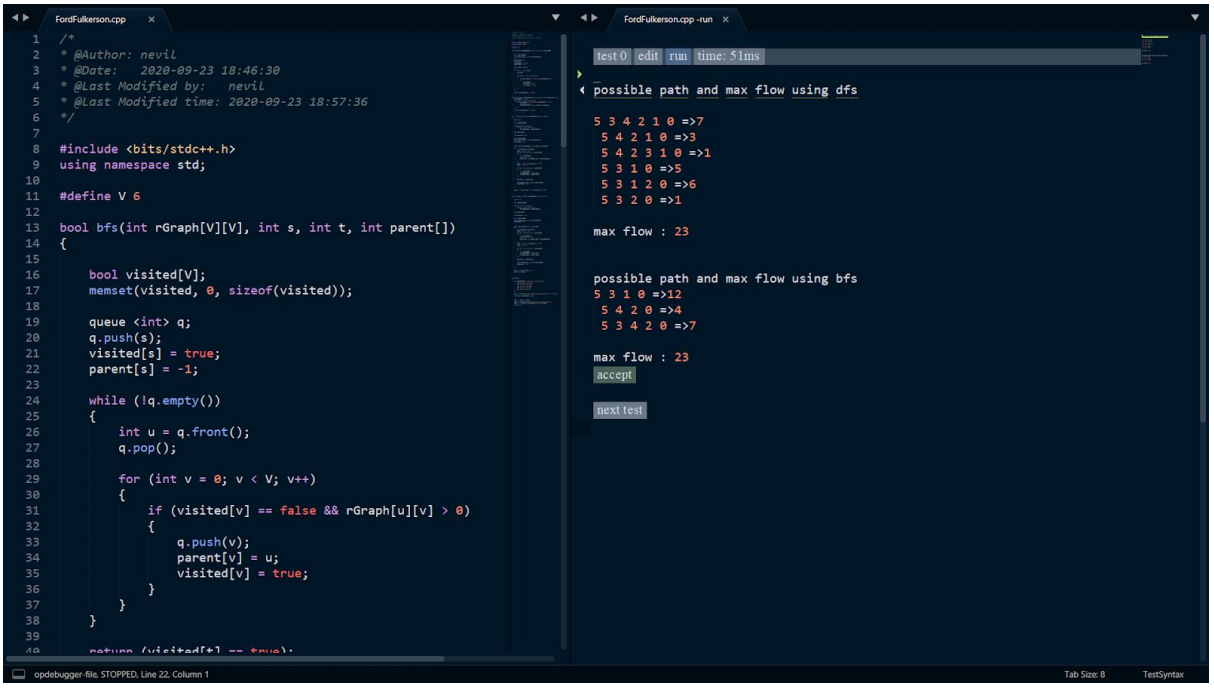
```
        }
    }
    return (visited[t] == true);
}


void  fordFulkerson(int graph[V][V], int s, int t)
{
    int u, v;

    int rGraph[V][V];

    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;

    bool visited[V];
    memset(visited, 0, sizeof(visited));
    parent[s] = -1;


    while (dfsutil(rGraph, s, t, parent, visited))
    {
        int path_flow = INT_MAX;
        cout << t << " ";
        for (v = t; v != s; v = parent[v])
        {
            u = parent[v];
            cout << u << " ";
            path_flow = min(path_flow, rGraph[u][v]);
```

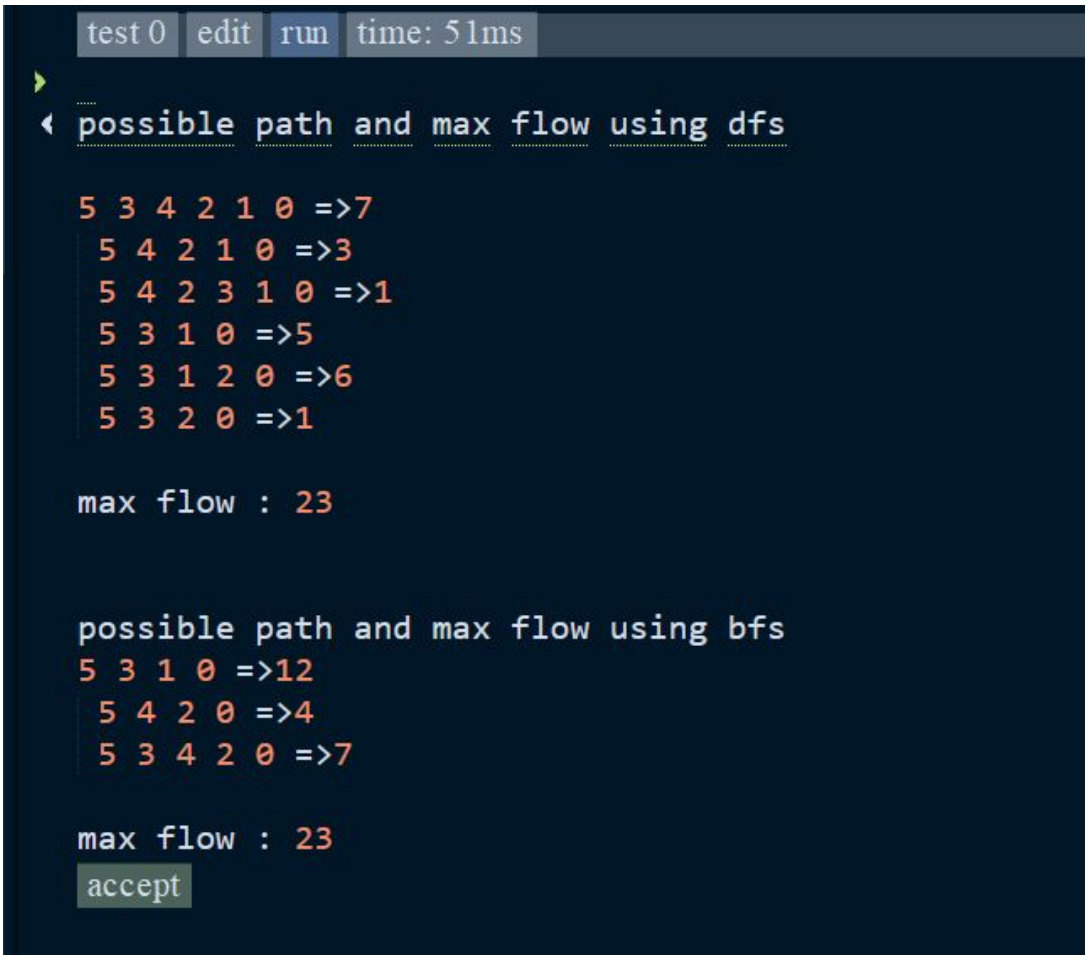
```
        }
        cout << "=>" << path_flow << " ";
        cout << "\n ";

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

        max_flow += path_flow;

        memset(visited, 0, sizeof(visited));
        parent[s] = -1;

    }

    cout << "\nmax flow : " << max_flow << "\n";
}


int fordFulkerson2(int graph[V][V], int s, int t)
{
    int u, v;

    int rGraph[V][V];

    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;

bool visited[V];
memset(visited, 0, sizeof(visited));
parent[s] = -1;


while (bfs(rGraph, s, t, parent))
{
    int path_flow = INT_MAX;
    cout << t << " ";
    for (v = t; v != s; v = parent[v])
    {
        u = parent[v];
        cout << u << " ";
        path_flow = min(path_flow, rGraph[u][v]);

    }
    cout << "=>" << path_flow << " ";
    cout << "\n ";

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

    max_flow += path_flow;

    memset(visited, 0, sizeof(visited));
    parent[s] = -1;
```

```
    }

    cout << "\nmax 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 << "possible path and max flow using dfs \n"
<< "\n" ;
    fordFulkerson(graph, 0, 5);


    cout << endl << endl;
    cout << "possible path and max flow using bfs \n" ;
    cout << fordFulkerson2(graph, 0, 5) << endl;
    return 0;
}
```

Output:


FordFulkerson.cpp
/*
* @Author: nevil
* @Date: 2020-09-23 18:46:30
* @Last Modified by: nevil
* @Last Modified time: 2020-09-23 18:57:36
*/
#include <bits/stdc++.h>
using namespace std;
#define V 6
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] == false && rGraph[u][v] > 0)
{
q.push(v);
parent[v] = u;
visited[v] = true;
}
}
}
FordFulkerson.cpp -run
test 0 edit run time: 51ms
possible path and max flow using dfs
5 3 4 2 1 0 =>7
5 4 2 1 0 =>3
5 4 2 3 1 0 =>1
5 3 1 0 =>5
5 3 1 2 0 =>6
5 3 2 0 =>1
max flow : 23
possible path and max flow using bfs
5 3 1 0 =>12
5 4 2 0 =>4
5 3 4 2 0 =>7
max flow : 23
accept
next test
opdebugger-file, STOPPED, Line 22, Column 1
Tab Size: 8
TestSyntax


test 0 edit run time: 51ms
possible path and max flow using dfs
5 3 4 2 1 0 =>7
5 4 2 1 0 =>3
5 4 2 3 1 0 =>1
5 3 1 0 =>5
5 3 1 2 0 =>6
5 3 2 0 =>1
max flow : 23
possible path and max flow using bfs
5 3 1 0 =>12
5 4 2 0 =>4
5 3 4 2 0 =>7
max flow : 23
accept

Complexity :

The analysis of Ford-Fulkerson depends heavily on how the augmenting paths are found. The typical method is to use breadth-first search to find the path. If this method is used, Ford-Fulkerson runs in polynomial time.

If all flows are integers, then the while loop of Ford-Fulkerson is run at most |F*|. Where F* is the maximum flow. This is because the flow is increased, at worst, by 1 in each iteration.

Finding the augmenting path inside the while loop takes O(V + E), where E is the set of edges in the residual graph. This can be simplified to O(E). So, the runtime of Ford-Fulkerson is O(V + E).

However since we are using an adjacency matrix in the above approach where bfs takes O(V^2) time, the overall time complexity of the above algorithm slightly differs from the adjacency list representation .

Nevil Parmar
CE-092
https://nevilparmar.me