Advanced Graph Algorithms

Rahul Ranjan

Problem statement - A civil hospital can serve all the districts adjacent to it. Assume a state named Rogipur has no civil hospitals till date. The state officials want to set up new civil hospitals. Given the map of state with coordinate locations of district head quarter, Find the minimum number of civil hospitals needed to serve the whole state.

Solution - We know from the above statement that a hospital can serve its neighbouring districts, so we have a problem of identifying the neighbours connected the districts. We can follow the below steps to identify the connected neighbour districts to find the minimum number of hospitals need to serve the entire state :

- Create an adjacency matrix with 0 and 1, where 1 means the two districts are connected or adjacent to each other whereas 0 means the two districts are not connected as: (Taking undirected graph as input)

```
{{ 0, 1, 1, 0, 0, 0, 0, 0 },
 { 1, 0, 0, 1, 1, 0, 0, 0 },
 { 1, 0, 0, 1, 0, 0, 0, 0 },
 { 0, 1, 1, 0, 0, 0, 0, 1 },
 { 0, 1, 0, 0, 0, 1, 1, 0 },
 { 0, 0, 0, 0, 1, 0, 0, 0 },
 { 0, 0, 0, 0, 1, 0, 0, 1 },
 { 0, 0, 0, 1, 0, 0, 1, 0 }}
```

- We will be using Breadth First Search to obtain our desired output of minimum number of hospitals needed to serve the entire state.
- Start from any districts or vertex
- We are using C++ to implement the program and we have added proper comments in the code for better understanding.

Steps :

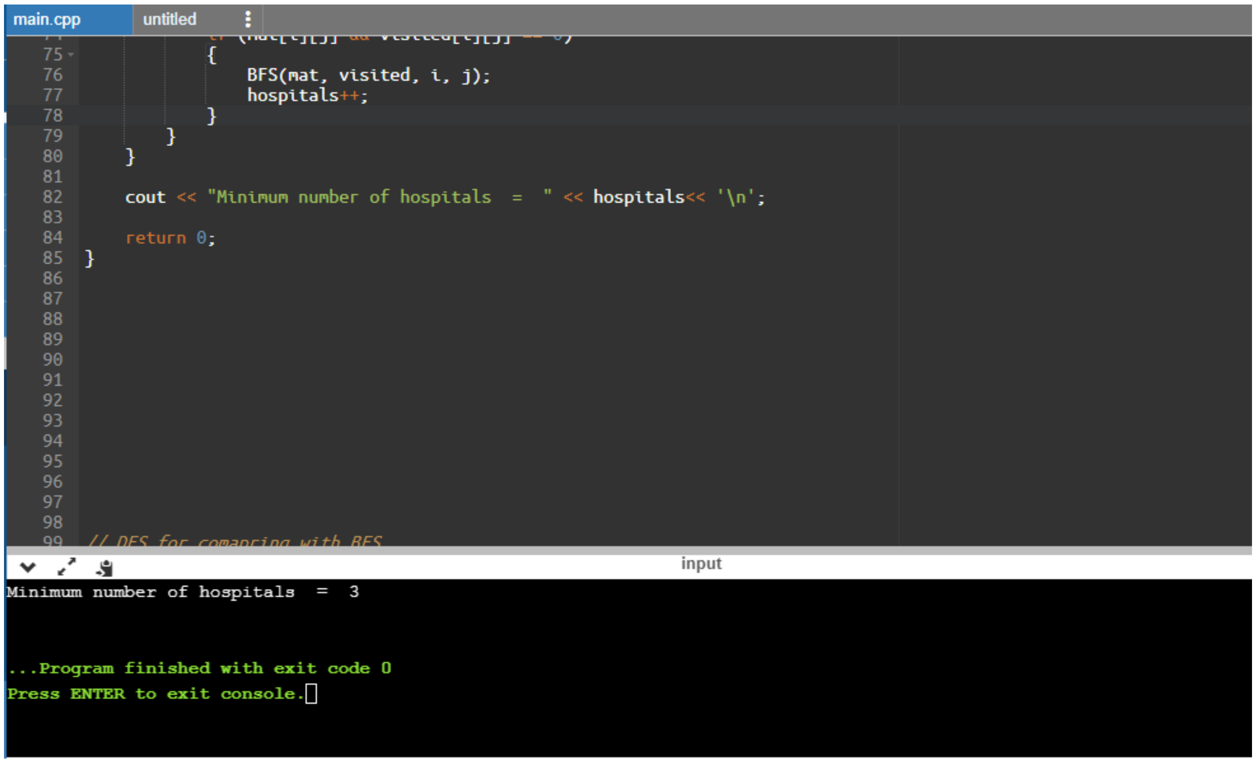
1. We are first initialising the 2D adjacency matrix based on the connectivity of the districts ,i.e, In input adjacency matrix 1 means two districts (nodes) are connected, whereas 0 means not connected.
2. Defined the required variables as appropriate.
3. Thereafter, we are writing a program to calculate the number of times the BFS function is called ,i.e., call BFS() for unvisited node to compute the number of connected districts or adjacent districts. If an unvisited node is found increment hospital number by 1.
4. On calling the BFS() function, we need to initialise the queue (because BFS uses queue data structure) to store the details about visited district nodes and dequeue the source node and We will execute the BFS until the queue is empty.
5. The while loop runs until queue is empty and the program checks for all the 8-Neighbours of a district node to look for adjacent nodes if 1 is found we traverse that node :

(p-1,q+1)	(p,q+1)	(p+1,q+1)
(p-1,q)	(p,q)	(p+1,q)
(p-1,q-1)	(p,q-1)	(p+1,q-1)

6. We will skip the node if already visited and if it is unvisited then we will mark it as visited and push it into the queue.
7. To check if a node is already visited or not, we will call a function check_neighbour() and this will check if a node is already present in the visited queue then skip it and if unvisited then it is explored.
8. Finally the function will return the minimum number of hospitals needed to serve the entire state of Rogipur.

The time complexity of this approach is O(V+E), where V is the number of districts (vertices) and E is the number of edges (connectivity or path).The given solution is implemented for Undirected graphs.

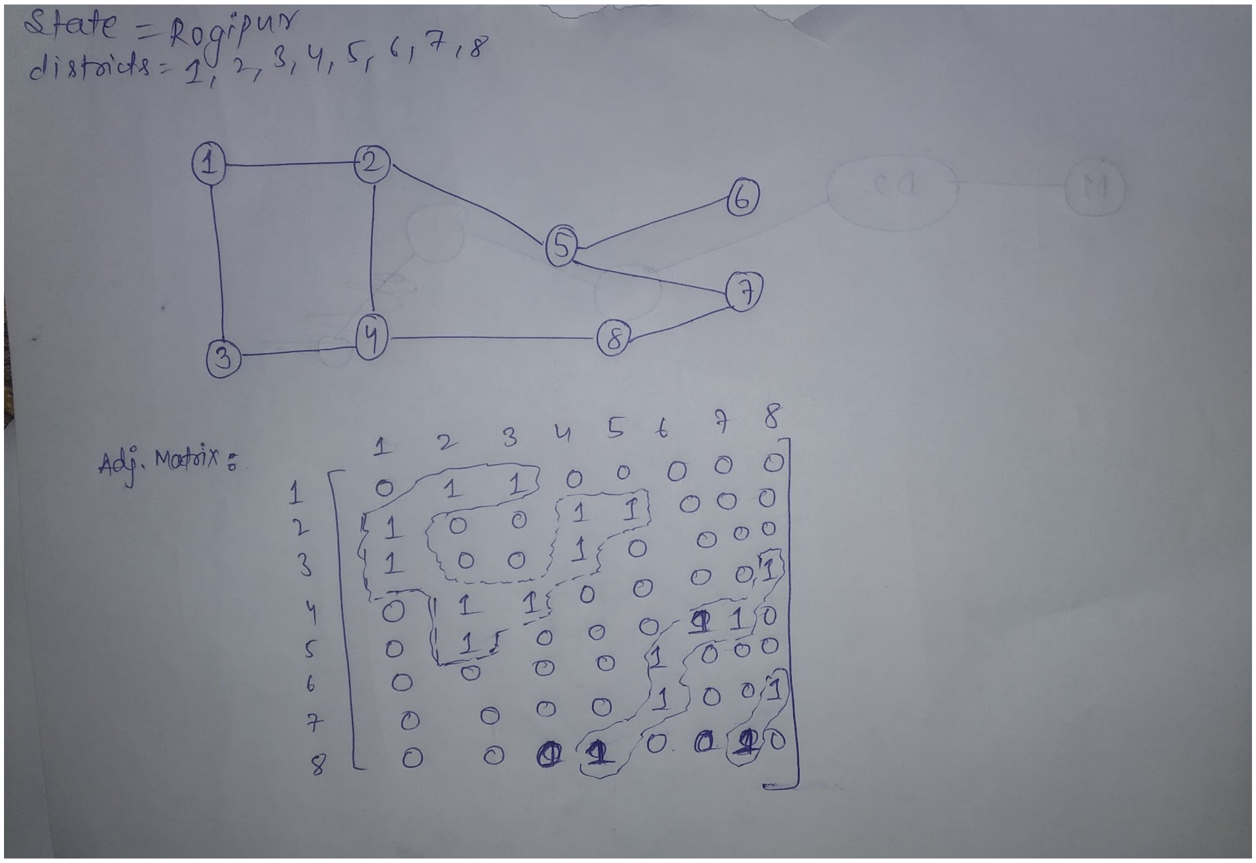
Screenshot of Implementation :



```
            if (mat[i][j] && visited[i][j] == 0)
            {
                BFS(mat, visited, i, j);
                hospitals++;
            }
        }
    }

    cout << "Minimum number of hospitals  =  " << hospitals<< '\n';

    return 0;
}














// DFS for comparing with BFS
```

```
Minimum number of hospitals  =  3


...Program finished with exit code 0
Press ENTER to exit console.
```

Our test approach :



State = Rogipur
districts = 1, 2, 3, 4, 5, 6, 7, 8

Adj. Matrix :

$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 8 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$
