Types of BFS in graph

1. Single Source BFS
2. Multi Source BFS
3. 0 -1 BFS
4. Bi-directional BFS

Single Source BFS

➔In this, there will be only one source point from where the BFS start.

Problem link

https://leetcode.com/problems/keys-and-rooms/

Solution

```
class Solution {

public:


    void bfs(int startRoom, vector<vector<int>>& rooms, vector<bool>
&visited){

        queue<int> q;

        visited[startRoom] = true;

        q.push(startRoom);


        while(!q.empty()){

            int currRoom = q.front();

            q.pop();


            for(int connectedRoom : rooms[currRoom]){

                if(!visited[connectedRoom]){

                    visited[connectedRoom] = true;

                    q.push(connectedRoom);

                }
```

```
            }
        }

    }

    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        int n = rooms.size();
        vector<bool> visited(n,0);

        bfs(0, rooms, visited); // DO bfs starting from room 0

        //Check if all rooms are visited or not
        for(bool i : visited){
            if(!i)
                return false;
        }
        return true;
    }
};
```

➔here 0 is the only one point that will act as source point for the BFS .

- ⇨ And **bfs** function is standard BFS function

Multi Source BFS

➔There will be multiple source point acts as starting point for this type of BFS

Problem:

https://leetcode.com/problems/01-matrix/

code:

```
class Solution {
public:



    bool isvalid(int i,int j,int m,int n)
    {
        if(i==m||j==n||j<0||i<0)
            return false;
        return true;
    }


    vector<vector<int>> dir={{1,0},{0,1},{0,-1},{-1,0}};
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix)
    {
        queue<pair<int,int>> q;
        int m=matrix.size();
        int n=matrix[0].size();
        vector<vector<int>> dis(m,vector<int>(n,-1));
        for(int i=0;i<m;i++)
```

```
            for(int j=0;j<n;j++)
            {
                if(matrix[i][j]==0)
                { //pushing source point
                    q.push({i,j});
                    dis[i][j]=0;
                }
            }
        while(!q.empty())
        {
            pair<int,int> curr=q.front();
            q.pop();
            for(auto& x:dir)
            {
                int a=curr.first+x[0];
                int b=curr.second+x[1];
                if(isvalid(a,b,m,n)&&dis[a][b]==-1)
                {
                    q.push({a,b});
                    dis[a][b]=dis[curr.first][curr.second]+1;
                }
            }
        }
        return dis;
    }
};
```

0 -1 BFS

➔For any unweighted or graph having equal weight shortest path between single source to all other vertices can be found using BFS in O(N)

Sample problems:

chef and reversing(codechef)
Problem statement : for any directed graph with n edges and m vertices what is minimum number of edges to be reversed to have a path from vertex 1 to vertex n

solution:
We will convert the graph into weighted graph for every edge from u to v in original graph we will define its weight as 0 and add a reversed edge from v to u with weight 1
Then apply 0-1 bfs using vertex 1 as source

The shortest path algorithm will always try to use as less reverse paths possible because they have higher weight than original edges.

Code :

```
//pseudocode


unordered_map<int,list<pair<int,int>> adj;


for(int k=0;k<n;k++)
{
        int i,j;
        cin>>i>>j;
        adj[i].push_back(make_pair(j,0));
        adj[j].push_back(make_pair(i,1));
}
```

```
deque<int> q;
int d[n+1]; // initialized to 1e9

d[1]=0;
    q.push_front(1);
    while(!q.empty())
    {
        int node=q.front();
        q.pop_front();
        for(auto neighbour:adj[node])
        {
            int x=neighbour.first;
            int y=neighbour.second;
            if(d[node]+y<d[x])
            {
                d[x]=d[node]+y;
                if(y==0)
                {
                    q.push_front(x);
                }
                else
                {
                    q.push_back(x);
                }
            }
        }
```

```
    }


    if(d[n]==1e9)
    {
        cout<<-1<<endl;
    }
    else
    {
        cout<<d[n]<<endl;
       }
```

Bi-directional BFS

➔traverse the path simultaneously from start node and end node, and merge in the middle

Sample Problem

https://leetcode.com/problems/word-ladder/

code

```
class Solution {

public:

        int ladderLength(string beginWord, string endWord, vector<string>&
wordList) {

                unordered_set<string> s1;

                unordered_set<string> s2;

                unordered_set<string> dict(wordList.begin(),wordList.end());

                if(!dict.count(endWord)) return 0;

                int len=beginWord.size();

                int ans=0;

                s1.insert(beginWord);

                s2.insert(endWord);

                while(!s1.empty() && !s2.empty()){

                        ans++;

                        if(s1.size()>s2.size()){

                                swap(s1,s2);

                        }

                        unordered_set<string> cur;

                        for(string w:s1){
```

```
                        for(int i=0;i<len;i++){
                                char temp=w[i];
                                for(char x='a';x<='z';x++){
                                        w[i]=x;
                                        if(s2.count(w)){
                                                return ans+1;
                                        }
                                        if(!dict.count(w))continue;
                                        dict.erase(w);
                                        cur.insert(w);
                                }
                                w[i]=temp;

                        }
                }

                s1=cur;

        }

        return 0;

    }
};
```