GRAPH - BFS Usage

* DFS visit every node exactly once in some order

But, BFS does the same but can also solve shortest Path Problem in an unweighted graph.

Application :-

1) Reachability
2) Component Numbering
3) Bi-partite

] Use of DFS but can also be done by BFS

4) Cycles
5) Topological sort
6) Shortest Path

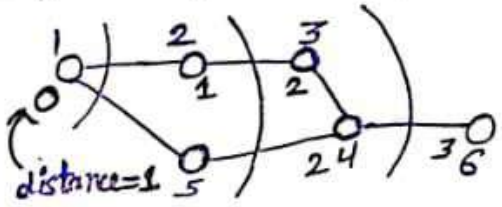
] Use of BFS

Pseudocode :-

```
BFS (x) :
    Q.push (x)
    While (! Q.empty()) {
    curr = q.front();
    q.pop();
    if (visited [curr]) continue;
    visited [curr] = 1;
    for v in neighbours [cur] {
      if (!visited [v])
          q.push [v];
    }
}
```

Uses a Queue (not recursion like DFS)

- Layered Exploration of BFS



Distance = min no. of edges b/w two nodes.

Eg: Node 3 has distance 2 from node 1.

Queue order : 1 → (2,5) → (3,4) → (6)

Execution start at 1.

Distances :

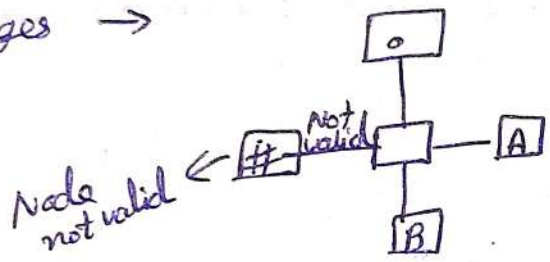
- Node 1 → 0
- Node 2,5 → 1
- Node 3,4 → 2
- Node 6 → 3

* BFS is better for distance finding due to layered explosation.

Q1. Labyrinth [CSES Task 1193]

Only valid nodes are 'A', 'B' & '.'.

Edges →



~~You~~ We have to find set of valid neighbours

* Creating a graph is very difficult here. So we will not use graph here. We can easily map this.

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

const int INF = 100;
using state = pair<int,int>;
#define F first
#define S second

int n,m;
vector<string> arr;

vector<vector<int>> vis,dist;

int dx[] = {0,1,0,-1};
int dy[] = {1,0,-1,0};

bool inside(int x,int y){
    if(0<=x&&x<n&&0<=y&&y<m)return 1;
    else return 0;
}

vector<state> valid_neighbours(state cur){
    vector<state> res;
    for(int dir = 0;dir<4;dir++){
```

```
vector<state> valid_neighbours(state cur){
    vector<state> res;
    for(int dir = 0;dir<4;dir++){
        int nx = cur.F + dx[dir];
        int ny = cur.S + dy[dir];
        if(inside(nx,ny) && arr[nx][ny]!='#'){
            res.push_back(make_pair(nx,ny));
        }
    }
    return res;
}

void printer(){
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cout<<dist[i][j]<<"\t";
        }
        cout<<endl;
    }
    cout<<endl;
```

```cpp
void bfs(state st){
    vis = vector<vector<int>>(n, vector<int>(m,0));
    dist = vector<vector<int>>(n, vector<int>(m,INF));

    queue<state> q;
    // Add source
    q.push(st);
    dist[st.F][st.S]=0;
    // Start BFS
    while(!q.empty()){
        state cur = q.front(); q.pop();
        if(vis[cur.F][cur.S])continue;

        // Explore
        vis[cur.F][cur.S]=1;
        for(state v:valid_neighbours(cur)){                    }
            if(!vis[v.F][v.S] && dist[v.F][v.S] > dist[cur.F][cur.S]+1){
                q.push(v);
                dist[v.F][v.S] = dist[cur.F][cur.S]+1;
            }
        }

        // cout<<cur.F<<" "<<cur.S<<endl;
        // printer();
```

```
void solve(){
    cin>>n>>m;
    arr.resize(n);
    for(int i=0;i<n;i++){
        cin >> arr[i];
    }
    state st,en;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(arr[i][j]=='A')st={i,j};
            else if(arr[i][j]=='B')en={i,j};
        }
    }
    bfs(st);
    printer();
    if(dist[en.F][en.S]==INF){
        cout<<"NO\n";
    }else{
        cout<<"YES\n";
        cout<<dist[en.F][en.S]<<endl;
    }
}
```

Path Printing

How do we tweak above algorithm to include path also ?

→ Changes in bfs func^n

```
void bfs(state st){
    vis = vector<vector<int>>(n, vector<int>(m,0));
    dist = vector<vector<int>>(n, vector<int>(m,INF));
    par = vector<vector<state>>(n, vector<state>(m,{-1,-1}));
    queue<state> q;
    // Add source
    q.push(st);
    dist[st.F][st.S]=0;
    // Start BFS
    while(!q.empty()){
        state cur = q.front(); q.pop();
        if(vis[cur.F][cur.S])continue;

        // Explore
        vis[cur.F][cur.S]=1;
        for(state v:valid_neighbours(cur)){
            if(!vis[v.F][v.S] && dist[v.F][v.S] > dist[cur.F][cur.
            S]+1){
                q.push(v);
                dist[v.F][v.S] = dist[cur.F][cur.S]+1;
                par[v.F][v.S] = cur;
            }
```

```
}
bfs(st);
if(dist[en.F][en.S]==INF){
    cout<<"NO\n";
}else{
    cout<<"YES\n";
    cout<<dist[en.F][en.S]<<endl;

    state cur = en;
    vector<state> path;
    while(cur!=make_pair(-1,-1)){
        pair.push_back(cur);
        cur = par[cur.F][cur.S];
    }

    for(auto v:path){
        cout<<v.F<<" "<<v.S<<endl;
    }
}
```

```
void solve(){
    cin>>n>>m;
    arr.resize(n);
    for(int i=0;i<n;i++){
        cin >> arr[i];
    }
    state st,en;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(arr[i][j]=='A')st={i,j};
            else if(arr[i][j]=='B')en={i,j};
        }
    }
    bfs(st);
    if(dist[en.F][en.S]==INF){
        cout<<"NO\n";
    }else{
        cout<<"YES\n";
        cout<<dist[en.F][en.S]<<endl;

        state cur = en;
        vector<state> path;
        while(cur!=make_pair(-1,-1)){
```

```
vector<state> path;
while(cur!=make_pair(-1,-1)){
    path.push_back(cur);
    cur = par[cur.F][cur.S];
}
reverse(path.begin(),path.end());
for(int i=1;i<path.size();i++){
    int cx = path[i].F-path[i-1].F;
    int cy = path[i].S-path[i-1].S;
    if(cx==1)cout<<"D";
    else if(cx==-1)cout<<"U";
    else if(cy==1)cout<<"R";
    else cout<<"L";
}
```