ARTIFICIAL INTELLIGENCE

NAME : TONMOY BISWAS

CLASS : BCSE -3

SECTION : A3

ROLL NO : 002110501133

SET – 1

1. Graph search algorithms :-
   Create a tree/ graph
   Implement BFS, DFS.
   Report "order of nodes visited" And "solution path" for each of the search techniques .

➢

- BFS :-

CODE

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

class Solution{
    public :
      vector<int> bfs(int n,vector<pair<int,int>> &edges){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        vis[0]=1;
```

```
        queue<int> q;
        q.push(0);
        while(!q.empty()){
            int node=q.front();
            ans.push_back(node);
            q.pop();
            for(auto it:adj[node]){
                if(!vis[it]){
                    vis[it]=1;
                    q.push(it);
                }
            }
        }
        return ans;
    }

};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        // cout<<"Enter the edge no "<<i+1<<":"<<endl;
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    Solution s;
    vector<int> bfs=s.bfs(n,edges);
    cout<<"Nodes visited in bfs traversal"<<endl;
    for(auto it:bfs){
        cout<<it<<" ";
    }
    cout<<endl;
    return 0;
}
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Nodes visited in bfs traversal
0 1 2 3 4 5 6 7 8 9 10
```

- BFS SOLUTION-PATH :-

CODE

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

class Solution{

    public:

    vector<int> bfs(int n,vector<pair<int,int>> &edges){
```

```
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        vis[0]=1;
        queue<int> q;
        q.push(0);
        while(!q.empty()){
            int node=q.front();
            ans.push_back(node);
            q.pop();
            for(auto it:adj[node]){
                if(!vis[it]){
                    vis[it]=1;
                    q.push(it);
                }
            }
        }
        return ans;
    }

    vector<int> bfs_path(int n,vector<pair<int,int>> &edges,int
source,int destination){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        vector<int> parent(n);
        for(int i=0;i<n;i++){
            parent[i]=i;
        }
        vis[source]=1;
        queue<int> q;
        q.push(source);
        while(!q.empty()){
            int node=q.front();
            q.pop();
```

```
            ans.push_back(node);
            if(node==destination){
                break;
            }
            for(auto it:adj[node]){
                if(!vis[it]){
                    q.push(it);
                    vis[it]=1;
                    parent[it]=node;
                }
            }
        }

        return ans;
    }

};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    Solution s;
    cout<<"enter source and destination : "<<endl;
    int source,destination;
    cin>>source;
    cin>>destination;
    vector<int> path=s.bfs_path(n,edges,source,destination);
    cout<<"Search path in bfs"<<endl;
    for(auto it:path){
        cout<<it<<" ";
    }
    cout<<endl;
    return 0;
}
```

OUTPUT

```
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter source and destination
0
9
Search path in bfs
0 1 2 3 4 5 6 7 8 9
```

- DFS :-

CODE

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

class Solution{
    void helper_dfs(vector<vector<int>> &adj,int node,int
parent,vector<int> &vis,vector<int> &ans){
```

```
        vis[node]=1;
        ans.push_back(node);
        for(auto it:adj[node]){
            if(it!=parent){
                if(!vis[it]){
                    helper_dfs(adj,it,node,vis,ans);
                }
            }
        }
    }
    public:
    //Assuming 0 is default root of the graph
    vector<int> dfs(int n,vector<pair<int,int>> &edges){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        //Assuming 0 is default root of the graph
        helper_dfs(adj,0,-1,vis,ans);
        return ans;
    }
};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        // cout<<"Enter the edge no "<<i+1<<":"<<endl;
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    Solution s;
    vector<int> dfs=s.dfs(n,edges);
    cout<<"Nodes visited in dfs traversal"<<endl;
    for(auto it:dfs){
```

```
        cout<<it<<" ";
    }
    cout<<endl;

    return 0;
}
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Nodes visited in dfs traversal
0 1 4 5 8 9 2 6 3 7 10
```

- DFS SOLUTION PATH :-

CODE

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

class Solution{
    void helper_dfs(vector<vector<int>> &adj,int node,int
parent,vector<int> &vis,vector<int> &ans){
        vis[node]=1;
        ans.push_back(node);
        for(auto it:adj[node]){
            if(it!=parent){
```

```
                if(!vis[it]){
                    helper_dfs(adj,it,node,vis,ans);
                }
            }
        }
    }
    public:
    //Assuming 0 is default root of the graph
    vector<int> dfs(int n,vector<pair<int,int>> &edges){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        //Assuming 0 is default root of the graph
        helper_dfs(adj,0,-1,vis,ans);
        return ans;
    }

    void helper_dfs_path(vector<vector<int>> &adj,int node,int
prev,vector<int> &vis,vector<int> &parent){
        vis[node]=1;
        for(auto it:adj[node]){
            if(it!=prev){
                if(!vis[it]){
                    parent[it]=node;
                    helper_dfs_path(adj,it,node,vis,parent);
                }
            }
        }
    }

    vector<int> dfs_path(int n,vector<pair<int,int>>
&edges,int source,int destination){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
```

```
        vector<int> parent(n);
        for(int i=0;i<n;i++){
            parent[i]=i;
        }
        helper_dfs_path(adj,source,-1,vis,parent);
        int cur=destination;
        while(parent[cur]!=cur){
            ans.push_back(cur);
            cur=parent[cur];
        }
        ans.push_back(cur);
        reverse(ans.begin(),ans.end());
        return ans;
    }
};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        // cout<<"Enter the edge no "<<i+1<<":"<<endl;
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    Solution s;
    vector<int> dfs=s.dfs(n,edges);
    cout<<"Nodes visited in dfs traversal"<<endl;
    for(auto it:dfs){
        cout<<it<<" ";
    }
    cout<<endl;
    int source,destination;
    cout<<"Enter the source"<<endl;
    cin>>source;
    cout<<"Enter the destination"<<endl;
    cin>>destination;
    vector<int> path=s.dfs_path(n,edges,source,destination);
    cout<<"Search path in dfs"<<endl;
    for(auto it:path){
```

```
            cout<<it<<" ";
        }
        cout<<endl;

        return 0;
    }
```

OUTPUT

```
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Nodes visited in dfs traversal
0 1 4 5 8 9 2 6 3 7 10
Enter the source
0
Enter the destination
9
Search path in dfs
0 1 5 9
```

SET – 2

1. . Graph search algorithms
   A) Create a tree/ graph
   B) Implement DLS, IDS, IBS.
   Report "order of nodes visited" And "solution path" for each of the search techniques.

➢

- DLS : -

CODE

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

class Solution{

    public:

    void helper_dfs(vector<vector<int>> &adj,int node,int
parent,vector<int> &vis,vector<int> &ans){
        vis[node]=1;
        ans.push_back(node);
        for(auto it:adj[node]){
            if(it!=parent){
                if(!vis[it]){
                    helper_dfs(adj,it,node,vis,ans);
                }
            }
        }
    }

    bool helper_dls_path(vector<vector<int>> &adj,int node,int
prev,int destination,vector<int> &vis,vector<int> &parent,int
depth){
```

```
        vis[node]=1;
        if(depth<0){
            return false;
        }
        if(node==destination){
            return true;
        }
        for(auto it:adj[node]){
            if(it!=prev){
                if(!vis[it]){
                    parent[it]=node;
                    bool
t=helper_dls_path(adj,it,node,destination,vis,parent,depth-1);
                    if(t){
                        return true;
                    }
                }
            }
        }
        return false;
    }

    vector<int> dls_path(int n,vector<pair<int,int>>
&edges,int source,int destination,int depth){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        vector<int> parent(n);
        for(int i=0;i<n;i++){
            parent[i]=i;
        }
        bool t=helper_dls_path(adj,source,-
1,destination,vis,parent,depth);
        if(t){
            int cur=destination;
            while(parent[cur]!=cur){
                ans.push_back(cur);
                cur=parent[cur];
            }
            ans.push_back(cur);
```

```
            reverse(ans.begin(),ans.end());
        }
        return ans;
    }
};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    Solution s;
    int source,destination,depth;
    cout<<"enter source , destination and depth : "<<endl;
    cin>>source>>destination>>depth;
    vector<int>
path=s.dls_path(n,edges,source,destination,depth);
    if(path.empty()){
        cout<<"Destination not found in the given depth";
    }
    else{
        cout<<"Destination found"<<endl;
        cout<<"DLS path:"<<endl;
        for(auto it:path){
            cout<<it<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter source, destination
0 9
Enter depth
3
Destination found
DLS path:
0 1 5 9
```

```
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter source, destination
0 9
Enter depth
2
Destination not found in the given depth
```

- IDS :-

CODE

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

class Solution{

    public:

    void helper_dfs(vector<vector<int>> &adj,int node,int
parent,vector<int> &vis,vector<int> &ans){
        vis[node]=1;
        ans.push_back(node);
        for(auto it:adj[node]){
            if(it!=parent){
                if(!vis[it]){
                    helper_dfs(adj,it,node,vis,ans);
```

```
                }
            }
        }
    }

    bool helper_dls_path(vector<vector<int>> &adj,int node,int
prev,int destination,vector<int> &vis,vector<int> &parent,int
depth){
        vis[node]=1;
        if(depth<0){
            return false;
        }
        if(node==destination){
            return true;
        }
        for(auto it:adj[node]){
            if(it!=prev){
                if(!vis[it]){
                    parent[it]=node;
                    bool
t=helper_dls_path(adj,it,node,destination,vis,parent,depth-1);
                    if(t){
                        return true;
                    }
                }
            }
        }
        return false;
    }

    vector<int> dls_path(int n,vector<pair<int,int>> &edges,int
source,int destination,int depth){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        vector<int> parent(n);
        for(int i=0;i<n;i++){
            parent[i]=i;
        }
```

```
        bool t=helper_dls_path(adj,source,-
1,destination,vis,parent,depth);
        if(t){
            int cur=destination;
            while(parent[cur]!=cur){
                ans.push_back(cur);
                cur=parent[cur];
            }
            ans.push_back(cur);
            reverse(ans.begin(),ans.end());
        }
        return ans;
    }

    void ids(int n,vector<pair<int,int>> &edges,int source,int
destination,int depth){
        for(int i=0;i<=depth;i++){
            vector<int>
path=dls_path(n,edges,source,destination,i);
            if(path.empty()){
                cout<<"Destination not found for depth:
"<<i<<endl;
            }
            else{
                cout<<"\nDestination found for depth: "<<i<<endl;
                cout<<"path:"<<endl;
                for(auto it:path){
                    cout<<it<<" ";
                }
                cout<<endl;
                break;
            }
        }
    }
};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
```

```
            // cout<<"Enter the edge no "<<i+1<<":"<<endl;
            int x,y;
            cin>>x>>y;
            edges.push_back({x,y});
        }
        Solution s;
        cout<<"enter source , destination and depth : "<<endl;
        int source,destination,depth;
        cin>>source>>destination>>depth;
        s.ids(n,edges,source,destination,depth);

        return 0;
    }
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter source and destination
0 9
Enter depth
3
Destination not found for depth: 0
Destination not found for depth: 1
Destination not found for depth: 2

Destination found for depth: 3
path:
0 1 5 9
```

- IBS :-

CODE

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

class Solution{
    void print_vector(vector<int> &ans){
        for(auto it:ans){
            cout<<it<<" ";
        }
        cout<<endl;
    }
    void helper_dfs(vector<vector<int>> &adj,int node,int
parent,vector<int> &vis,vector<int> &ans,int &cur_b,int
destination,int b){
        if(cur_b>b){
            return ;
        }
        vis[node]=1;
        ans.push_back(node);
        if(node==destination){
            cout<<"Destination found in breadth "<<cur_b<<endl;
            print_vector(ans);
            ans.pop_back();
            cur_b=b+1;
            return ;
        }
        bool leaf=true;
        for(auto it:adj[node]){
            if(it!=parent){
                leaf=false;
                if(!vis[it]){
                    helper_dfs(adj,it,node,vis,ans,cur_b,destinat
ion,b);
                }
            }
        }
        if(leaf){
            cout<<"Destination not found in breadth
"<<cur_b<<endl;
            cur_b++;
            print_vector(ans);
```

```
        }
        ans.pop_back();
    }
    public:
    //Assuming 0 is default root of the graph
    void ibs(int n,vector<pair<int,int>> &edges,int source,int
destination,int b){
        //creating adjacency list
        vector<vector<int>> adj(n);
        for(auto &it:edges){
            adj[it.first].push_back(it.second);
            adj[it.second].push_back(it.first);
        }
        vector<int> vis(n);
        vector<int> ans;
        //Assuming 0 is default root of the graph
        int cur_b=1;
        helper_dfs(adj,source,-1,vis,ans,cur_b,destination,b);
        // return ans;
    }
};

int main(){
    cout<<"Enter the number of nodes"<<endl;
    int n;
    cin>>n;
    vector<pair<int,int>> edges;
    cout<<"Enter the number of edges"<<endl;
    int e;
    cin>>e;
    for(int i=0;i<e;i++){
        // cout<<"Enter the edge no "<<i+1<<":"<<endl;
        int x,y;
        cin>>x>>y;
        edges.push_back({x,y});
    }
    int source,destination,b;
    cout<<"Enter the souce and destination"<<endl;
    cin>>source>>destination;
    cout<<"Enter the depth"<<endl;
    cin>>b;
    Solution s;

    s.ibs(n,edges,source,destination,b);
```

```
    return 0;
}
```

OUTPUT

```
E:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab>cd "e:\bcse 3rd
year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab\Assignment 2\"
&& g++ IBS.cpp -o IBS && "e:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd
sem)\AI Lab\Assignment 2\"IBS
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter the souce and destination
0 9
Enter the depth
2
Destination not found in breadth 1
0 1 4
Destination not found in breadth 2
0 1 5 8
```

```
E:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab>cd "e:\bcse 3rd
year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab\Assignment 2\"
&& g++ IBS.cpp -o IBS && "e:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd
sem)\AI Lab\Assignment 2\"IBS
Enter the number of nodes
11
Enter the number of edges
10
0 1
0 2
0 3
1 4
1 5
2 6
3 7
5 8
5 9
7 10
Enter the souce and destination
0 9
Enter the depth
3
Destination not found in breadth 1
0 1 4
Destination not found in breadth 2
0 1 5 8
Destination found in breadth 3
0 1 5 9
```

2. Implement water jug and 8 puzzle :-

   - WATER JUG :-

CODE

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

bool canMeasureWaterBFS(int x, int y, int t)
```

```
{
    if (x + y < t)
        return false;

    vector<int> vis(x + y + 1, 0);
    queue<int> q;
    q.push(0);
    vis[0] = 1;
    vector<int> dir = {x, -x, y, -y};
    while (!q.empty())
    {
        int top = q.front();
        q.pop();

        for (int i = 0; i < 4; i++)
        {
            int total = top + dir[i];
            if (total == t)
                return true;
            else if (total < 0 || total > (x + y))
            {
                continue;
            }
            else if (!vis[total])
            {
                vis[total] = 1;
                q.push(total);
            }
        }
    }
    return false;
}

int main()
{
    cout<<"enter capacity of jug1 and jug2 and target"<<endl;
    int a,b,c;
    cin>>a>>b>>c;
    bool ans=canMeasureWaterBFS(a,b,c);
    if(ans) cout<<”YES”;
    else cout<<”NO”;
```

OUTPUT

```
E:\bcse 3rd year (2nd sem)\AI LAB practice for viva>cd "e:\bcse 3rd year (2nd sem)\AI LAB practice for viva\set2\" && g++ waterjugBFS.cpp -o waterjugBFS && "e:\bcse 3rd year (2nd sem)\AI LAB practice for viva\set2\"waterjugBFS
enter capacity of jug1 and jug2 and target
5 3 4
YES
```

- 8 PUZZLE :-

CODE

```cpp
#include <bits/stdc++.h>
using namespace std;
#define N 3

// state space tree nodes
struct Node
{
    // stores the parent node of the current node
    // helps in tracing path when the answer is found
    Node* parent;

    // stores matrix
    int mat[N][N];

    // stores blank tile coordinates
    int x, y;

    // stores the number of misplaced tiles
    int cost;

    // stores the number of moves so far
    int level;
};

// Function to print N x N matrix
int printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
```

```
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}

// Function to allocate a new node
Node* newNode(int mat[N][N], int x, int y, int newX,
              int newY, int level, Node* parent)
{
    Node* node = new Node;

    // set pointer for path to root
    node->parent = parent;

    // copy data from parent node to current node
    memcpy(node->mat, mat, sizeof node->mat);

    // move tile by 1 position
    swap(node->mat[x][y], node->mat[newX][newY]);

    // set number of misplaced tiles
    node->cost = INT_MAX;

    // set number of moves so far
    node->level = level;

    // update new blank tile coordinates
    node->x = newX;
    node->y = newY;

    return node;
}

// bottom, left, top, right
int row[] = { 1, 0, -1, 0 };
int col[] = { 0, -1, 0, 1 };
int calculateCost(int initial[N][N], int final[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        if (initial[i][j] && initial[i][j] != final[i][j])
        count++;
    return count;
}
```

```
// Function to check if (x, y) is a valid matrix coordinate
int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}

// print path from root node to destination node
void printPath(Node* root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

// Comparison object to be used to order the heap
struct comp
{
    bool operator()(const Node* lhs, const Node* rhs) const
    {
        return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);
    }
};
void solve(int initial[N][N], int x, int y,
        int final[N][N])
{
    // Create a priority queue to store live nodes of
    // search tree;
    priority_queue<Node*, std::vector<Node*>, comp> pq;

    // create a root node and calculate its cost
    Node* root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);

    // Add root to list of live nodes;
    pq.push(root);

    // Finds a live node with least cost,
    // add its childrens to list of live nodes and
    // finally deletes it from the list.
    while (!pq.empty())
    {
        // Find a live node with least estimated cost
```

```
        Node* min = pq.top();

        // The found node is deleted from the list of
        // live nodes
        pq.pop();

        // if min is an answer node
        if (min->cost == 0)
        {
            // print the path from root to destination;
            printPath(min);
            return;
        }

        // do for each child of min
        // max 4 children for a node
        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                // create a child node and calculate
                // its cost
                Node* child = newNode(min->mat, min->x,
                              min->y, min->x + row[i],
                              min->y + col[i],
                              min->level + 1, min);
                child->cost = calculateCost(child->mat, final);

                // Add child to list of live nodes
                pq.push(child);
            }
        }
    }
}

// Driver code
int main()
{
    // Initial configuration
    // Value 0 is used for empty space
    int initial[N][N] =
    {
        {1, 2, 3},
        {5, 6, 0},
        {7, 8, 4}
```

```
    };

    // Solvable Final configuration
    // Value 0 is used for empty space
    int final[N][N] =
    {
        {1, 2, 3},
        {5, 8, 6},
        {0, 7, 4}
    };

    // Blank tile coordinates in initial
    // configuration
    int x = 1, y = 2;

    solve(initial, x, y, final);

    return 0;
}
```

OUTPUT

```
E:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab>cd "e:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab\Assignment 2\"
&& g++ 8puzzle.cpp -o 8puzzle && "e:\bcse 3rd year (2nd sem)\AI LAB (3rd year 2nd sem)\AI Lab\Assignment 2\"8puzzle

1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4
```

SET – 3

1. Implement UCS, best first greedy, A* search.
   Report "order of nodes visited" And "solution path" for each of the search techniques .

➢

- UCS :-

CODE

```
#include<bits/stdc++.h>
using namespace std;
vector<int> dijkstra(int n,vector<vector<pair<int,int>>> &adj)
{
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<
int,int>>> pq;
    vector<int> visited(n,0);
    vector<int> distance(n,INT_MAX);
    distance[0]=0;
    pq.push({0,0});
    while(!pq.empty())
    {
        int node=pq.top().second;
        int dist=pq.top().first;
        visited[node]=1;
        pq.pop();
        for(auto it:adj[node])
        {
            int adjnode=it.first;
            int wt=it.second;
            if(distance[adjnode]>wt+distance[node])
            {
                distance[adjnode]=wt+distance[node];
                pq.push({distance[adjnode],adjnode});
            }
        }
    }
    return distance;
}
```

```
void dijkstra_path(int src,int dest,int
n,vector<vector<pair<int,int>>> &adj,vector<int> &parent)
{
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<
int,int>>> pq;
    vector<int> visited(n,0);
    vector<int> distance(n,INT_MAX);
    distance[src]=0;
    pq.push({0,src});
    while(!pq.empty())
    {
        int node=pq.top().second;
        int dist=pq.top().first;
        visited[node]=1;
        pq.pop();
        for(auto it:adj[node])
        {
            int adjnode=it.first;
            int wt=it.second;
            if(distance[adjnode]>wt+distance[node])
            {
                parent[adjnode]=node;
                distance[adjnode]=wt+distance[node];
                pq.push({distance[adjnode],adjnode});
            }
        }
    }
    cout<<"the parent vector is:";
    for(auto it:parent)
    cout<<it<<" ";
    cout<<endl;
    int node=dest;
    if(distance[node]==INT_MAX){
        cout<<"No path exist"<<endl;
        return;
    }
    cout<<"The soln path is:";
    cout<<node<<"->";
    while(parent[node]!=-1)
    {
        if(node==src)
        cout<<parent[node];
        else
        cout<<parent[node]<<"->";
        node=parent[node];
```

```
    }
    cout<<endl;
}
int main(){
    int n,x,y;
    cout<<"Enter the number of elements in the graph:";
    cin>>n;
    int e;
    cout<<"Enter the number of edges in the graph:";
    cin>>e;
    vector<vector<pair<int,int>>> adj(n);
    int wt;
    for(int i=0;i<e;i++)
    {
        // cout<<"Enter the edge:";
        cin>>x>>y;
        // cout<<"Enter the edge weight=";
        cin>>wt;
        // cout<<"Edge ["<<x<<","<<y<<"]="<<wt<<endl;
        adj[x].push_back({y,wt});
        adj[y].push_back({x,wt});
    }
    vector<int> parent(n,-1);

    int src,des;
    cout<<"enter source and destination : "<<endl;
    cin>>src>>des;
    dijkstra_path(src,des,n,adj,parent);
}
```

OUTPUT

```
Enter the number of elements in the graph:11
Enter the number of edges in the graph:10
0 1 1
0 2 2
0 3 5
1 4 2
1 5 3
5 8 4
5 9 1
2 6 9
3 7 10
7 10 6
Enter the source nd the destination node respectively:0 9
the parent vector is:-1 0 0 0 1 1 2 3 5 5 7
The soln path is:9->5->1->0->
```

- A* :-

CODE

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

class Astar{
    vector<int> order_of_node;
    vector<int> parent;
    // vector<vector<pair<int,int>>> adj;
    unordered_map<int,unordered_map<int,int>> &adj;
    vector<int> dis;
    vector<int> vis;
    int src,des;
    int n;

    int g(int next,int cur){
        int ans= adj[cur][next]+dis[cur];
        if(ans<dis[next]){
            dis[next]=ans;
        }
    }
```

```
    int h(int next){
        if(next==des){
            return 0;
        }
        return 1;
    }

    int fn(int cur,int next){
        return g(next,cur)+h(next);
    }
    public:

    Astar(unordered_map<int,unordered_map<int,int>> &adj,int
n,int src,int des):adj(adj){
        this->n=n;
        this->src=src;
        this->des=des;
    }

    void a_star(){
        parent.resize(n,-1);
        dis.resize(n,10000000);
        vis.resize(n);
        dis[src]=0;
        // vis[src]=0;
        priority_queue<pair<int,int>,vector<pair<int,int>>,greate
r<pair<int,int>>> pq;
        pq.push({h(src),src});
        while(!pq.empty()){
            int node=pq.top().second;
            vis[node]=1;
            order_of_node.push_back(node);
            pq.pop();
            if(h(node)==0){
                break;
            }
            for(auto it:adj[node]){
                int next=it.first;

                if(!vis[next]){
                    int fval=fn(node,next);
                    pq.push({fval,next});
                    parent[next]=node;
                }
            }
```

```
        }
        if(parent[des]!=-1){
            cout<<"path found:"<<endl;
            vector<int> path;
            int cur=des;
            while(cur!=-1){
                path.push_back(cur);
                cur=parent[cur];
            }
            reverse(path.begin(),path.end());
            cout<<"path: "<<endl;
            for(auto it:path){
                cout<<it<<" ";
            }
            cout<<endl;
        }
        else{
            cout<<"No path found"<<endl;
        }
        cout<<"Order of node visited: "<<endl;
        for(auto it:order_of_node){
            cout<<it<<" ";
        }
        cout<<endl;
    }
};

int main(){
    unordered_map<int,unordered_map<int,int>> adj;
    int n,e;
    cout<<"Enter the number of nodes"<<endl;
    cin>>n;
    cout<<"Enter the number of edges"<<endl;
    cin>>e;
    for(int i=0;i<e;i++){
        int x,y,wt;
        cin>>x>>y>>wt;
        adj[x][y]=wt;
        adj[y][x]=wt;
    }
    int src,des;
    cout<<"Enter the src"<<endl;
    cin>>src;
    cout<<"Enter the des"<<endl;
    cin>>des;
```

```
    Astar S(adj,n,src,des);
    S.a_star();
}
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1 1
0 2 2
0 3 5
1 4 2
1 5 3
5 8 4
5 9 1
2 6 9
3 7 10
7 10 6
Enter the src
0
Enter the des
9
path found:
path:
0 1 5 9
Order of node visited:
0 1 2 4 5 9
```

- BEST FIRST GREEDY :-

CODE

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

class GreedyBestFirst{
    vector<int> order_of_node;
    vector<int> parent;
    // vector<vector<pair<int,int>>> adj;
    unordered_map<int,unordered_map<int,int>> &adj;
```

```
    vector<int> dis;
    vector<int> vis;
    int src,des;
    int n;

    int g(int next,int cur){
        int ans= adj[cur][next]+dis[cur];
        if(ans<dis[next]){
            dis[next]=ans;
        }
    }

    int h(int next){
        if(next==des){
            return 0;
        }
        return 1;
    }

    int fn(int cur,int next){
        return g(next,cur)+h(next);
    }
    public:

    GreedyBestFirst(unordered_map<int,unordered_map<int,int>>
&adj,int n,int src,int des):adj(adj){
        this->n=n;
        this->src=src;
        this->des=des;
    }

    void greedy(){
        parent.resize(n,-1);
        dis.resize(n,10000000);
        vis.resize(n);
        dis[src]=0;
        // vis[src]=0;
        priority_queue<pair<int,int>,vector<pair<int,int>>,greater<p
air<int,int>>> pq;
        pq.push({h(src),src});
        while(!pq.empty()){
            int node=pq.top().second;
            vis[node]=1;
            order_of_node.push_back(node);
            pq.pop();
```

```
            if(h(node)==0){
                break;
            }
            for(auto it:adj[node]){
                int next=it.first;
                //assuming all step cost is 1 hence the heuristic is
consistent. hence graph search strategy will work
                if(!vis[next]){
                    int fval=fn(node,next);
                    pq.push({fval,next});
                    parent[next]=node;
                }
            }
        }
        if(parent[des]!=-1){
            cout<<"path found:"<<endl;
            vector<int> path;
            int cur=des;
            while(cur!=-1){
                path.push_back(cur);
                cur=parent[cur];
            }
            reverse(path.begin(),path.end());
            cout<<"path: "<<endl;
            for(auto it:path){
                cout<<it<<" ";
            }
            cout<<endl;
        }
        else{
            cout<<"No path found"<<endl;
        }
        cout<<"Order of node visited: "<<endl;
        for(auto it:order_of_node){
            cout<<it<<" ";
        }
        cout<<endl;
    }
};

int main(){
    unordered_map<int,unordered_map<int,int>> adj;
    int n,e;
    cout<<"Enter the number of nodes"<<endl;
    cin>>n;
```

```
    cout<<"Enter the number of edges"<<endl;
    cin>>e;
    for(int i=0;i<e;i++){
        int x,y,wt;
        cin>>x>>y>>wt;
        adj[x][y]=wt;
        adj[y][x]=wt;
    }
    int src,des;
    cout<<"Enter the src"<<endl;
    cin>>src;
    cout<<"Enter the des"<<endl;
    cin>>des;
    GreedyBestFirst S(adj,n,src,des);
    S.greedy();
}
```

OUTPUT

```
Enter the number of nodes
11
Enter the number of edges
10
0 1 1
0 2 2
0 3 5
1 4 2
1 5 3
5 8 4
5 9 1
2 6 9
3 7 10
7 10 6
Enter the src
0
Enter the des
7
path found:
path:
0 3 7
Order of node visited:
0 1 2 4 5 3 9 8 6 7
```

- WATER JUG (using ucs):-

CODE

```
#include <bits/stdc++.h>

using namespace std;

set<pair<int, int>> s;//closed list
int total_count=0;

struct Node
{
```

```
    pair<int, int> val;
    int level;
    int cost;
    Node *parent;
};

int getCost(pair<int, int> &a, int target)
{
    return min(abs(a.first - target) + a.second, a.first +
abs(a.second - target));
}

Node *createNode(int x, int y, Node *parent, int level)
{
    Node *node = new Node();
    node->val.first = x;
    node->val.second = y;
    node->parent = parent;
    node->level = level;
    node->cost = 10000000;
    return node;
}

void printNode(Node *node)
{
    if (node == NULL)
    {
        return;
    }
    pair<int, int> &p = node->val;
    cout << p.first << " " << p.second << endl;
}

void printPath(Node *node)
{
    if (node == NULL)
    {
        return;
    }
    printPath(node->parent);
    printNode(node);
}

struct com
{
```

```
    bool operator()(Node *a, Node *b)
    {
        return (a->cost) > (b->cost);
    }
};

vector<pair<int, int>> expand(int x, int y, pair<int, int> p)
{
    vector<pair<int, int>> arr;
    arr.push_back({p.first, 0});
    arr.push_back({0, p.second});
    //l to r
    arr.push_back({(min(x, p.first + p.second)), (max(p.first +
p.second - x, 0))});
    //r to l
    arr.push_back({(max(p.second+p.first-
y,0)),(min(p.second+p.first,y))});
    arr.push_back({p.first,y});
    arr.push_back({x,p.second});
    return arr;
}

void solve(int x, int y, int target)
{
    Node *root = createNode(0, 0, NULL, 0);
    root->cost = getCost(root->val, target);
    priority_queue<Node *, vector<Node *>, com> pq;
    // graph search
    s.insert(root->val);
    pq.push(root);
    while (!pq.empty())
    {
        total_count++;
        cout<<total_count<<" ";
        if(total_count>10000000){
            return ;
        }
        Node *min = pq.top();
        pq.pop();
        if(min->cost==0){
            cout<<endl<<endl;
            cout<<"solution path:"<<endl;
            printPath(min);
            return ;
        }
```

```
            vector<pair<int, int>> arr = expand(x, y, min->val);
            for(auto &it:arr){
                if(!s.count(it)){
                    Node *child=createNode(it.first,it.second,min,min-
>level+1);
                    child->cost=getCost(child->val,target);
                    pq.push(child);
                    s.insert(child->val);
                }
            }
        }
    }

    int main()
    {
        int x,y,target;
        x=5,y=3,target=4;
        solve(x,y,target);
        return 0;
    }
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

solution path:
0 0
5 0
2 3
2 0
0 2
5 2
4 3
4 0
```

- WATER JUG (using A*) :-

CODE

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

set<pair<int, int>> s;//closed list
```

```
int total_count=0;

struct Node
{
    pair<int, int> val;
    int level;
    int cost;
    Node *parent;
};

int getCost(pair<int, int> &a, int target)
{
    return min(abs(a.first - target) + a.second, a.first +
abs(a.second - target));
}

Node *createNode(int x, int y, Node *parent, int level)
{
    Node *node = new Node();
    node->val.first = x;
    node->val.second = y;
    node->parent = parent;
    node->level = level;
    node->cost = 10000000;
    return node;
}

void printNode(Node *node)
{
    if (node == NULL)
    {
        return;
    }
    pair<int, int> &p = node->val;
    cout << p.first << " " << p.second << endl;
}

void printPath(Node *node)
{
    if (node == NULL)
    {
        return;
    }
    printPath(node->parent);
    printNode(node);
```

```
}

struct com
{
    bool operator()(Node *a, Node *b)
    {
        return (a->cost + a->level) > (b->cost + b->level);
    }
};

vector<pair<int, int>> expand(int x, int y, pair<int, int> p)
{
    vector<pair<int, int>> arr;
    arr.push_back({p.first, 0});
    arr.push_back({0, p.second});
    //l to r
    arr.push_back({(min(x, p.first + p.second)), (max(p.first +
p.second - x, 0))});
    //r to l
    arr.push_back({(max(p.second+p.first-
y,0)),(min(p.second+p.first,y))});
    arr.push_back({p.first,y});
    arr.push_back({x,p.second});
    return arr;
}

void solve(int x, int y, int target)
{
    Node *root = createNode(0, 0, NULL, 0);
    root->cost = getCost(root->val, target);
    priority_queue<Node *, vector<Node *>, com> pq;
    // graph search
    s.insert(root->val);
    pq.push(root);
    while (!pq.empty())
    {
        total_count++;
        cout<<total_count<<" ";
        if(total_count>10000000){
            return ;
        }
        Node *min = pq.top();
        pq.pop();
        if(min->cost==0){
            cout<<endl<<endl;
```

```
            cout<<"solution path:"<<endl;
            printPath(min);
            return ;
        }
        vector<pair<int, int>> arr = expand(x, y, min->val);
        for(auto &it:arr){
            if(!s.count(it)){
                Node *child=createNode(it.first,it.second,min,min-
>level+1);
                child->cost=getCost(child->val,target);
                pq.push(child);
                s.insert(child->val);
            }
        }
    }
}

int main()
{
    int x,y,target;
    x=5,y=3,target=4;
    solve(x,y,target);
    return 0;
}
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14

solution path:
0 0
5 0
2 3
2 0
0 2
5 2
4 3
4 0
```

- WATER JUG (using bfg) :-

CODE

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

set<pair<int, int>> s;//closed list
int total_count=0;

struct Node
{
    pair<int, int> val;
    int level;
    int cost;
    Node *parent;
};

int getCost(pair<int, int> &a, int target)
{
    return min(abs(a.first - target) + a.second, a.first +
abs(a.second - target));
}

Node *createNode(int x, int y, Node *parent, int level)
{
    Node *node = new Node();
    node->val.first = x;
    node->val.second = y;
    node->parent = parent;
    node->level = level;
    node->cost = 10000000;
    return node;
}

void printNode(Node *node)
{
    if (node == NULL)
    {
        return;
    }
    pair<int, int> &p = node->val;
    cout << p.first << " " << p.second << endl;
}

void printPath(Node *node)
```

```
{
    if (node == NULL)
    {
        return;
    }
    printPath(node->parent);
    printNode(node);
}

struct com
{
    bool operator()(Node *a, Node *b)
    {
        return (a->cost) > (b->cost);
    }
};

vector<pair<int, int>> expand(int x, int y, pair<int, int> p)
{
    vector<pair<int, int>> arr;
    arr.push_back({p.first, 0});
    arr.push_back({0, p.second});
    //l to r
    arr.push_back({(min(x, p.first + p.second)), (max(p.first +
p.second - x, 0))});
    //r to l
    arr.push_back({(max(p.second+p.first-
y,0)),(min(p.second+p.first,y))});
    arr.push_back({p.first,y});
    arr.push_back({x,p.second});
    return arr;
}

void solve(int x, int y, int target)
{
    Node *root = createNode(0, 0, NULL, 0);
    root->cost = getCost(root->val, target);
    priority_queue<Node *, vector<Node *>, com> pq;
    // graph search
    s.insert(root->val);
    pq.push(root);
    while (!pq.empty())
    {
        total_count++;
        cout<<total_count<<" ";
```

```
        if(total_count>10000000){
            return ;
        }
        Node *min = pq.top();
        pq.pop();
        if(min->cost==0){
            cout<<endl<<endl;
            cout<<"solution path:"<<endl;
            printPath(min);
            return ;
        }
        vector<pair<int, int>> arr = expand(x, y, min->val);
        for(auto &it:arr){
            if(!s.count(it)){
                Node *child=createNode(it.first,it.second,min,min-
>level+1);
                child->cost=getCost(child->val,target);
                pq.push(child);
                s.insert(child->val);
            }
        }
    }
}

int main()
{
    int x,y,target;
    x=5,y=3,target=4;
    solve(x,y,target);
    return 0;
}
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10

solution path:
0 0
5 0
2 3
2 0
0 2
5 2
4 3
4 0
```

- 8 PUZZLE (using ucs) ;-

CODE

```
#include <bits/stdc++.h>
using namespace std;
#define N 3

int total_count = 0;

struct Node
{
    Node *parent;

    int mat[N][N];

    int x, y;

    int cost;

    int level;
};

// Function to print N x N matrix
void printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d ", mat[i][j]);
```

```
        printf("\n");
    }
}

// Function to allocate a new node
Node *newNode(int mat[N][N], int x, int y, int newX, int newY, int
level, Node *parent)
{
    Node *node = new Node;

    node->parent = parent;

    memcpy(node->mat, mat, sizeof node->mat);

    swap(node->mat[x][y], node->mat[newX][newY]);

    node->cost = INT_MAX;

    node->level = level;

    node->x = newX;
    node->y = newY;

    return node;
}

// bottom, left, top, right
int row[] = {1, 0, -1, 0};
int col[] = {0, -1, 0, 1};

int calculateCost(int initial[N][N], int final[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (initial[i][j] && initial[i][j] != final[i][j])
                count++;
    return count;
}

int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}
```

```
void printPath(Node *root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

struct comp
{
    bool operator()(const Node *lhs, const Node *rhs) const
    {
        return (lhs->level) > (rhs->level); // evaluation funtion
    }
};

int isSame(Node *a, Node *b)
{
    if(a==NULL || b==NULL){
        return 0;
    }
    return calculateCost(a->mat, b->mat) == 0;
}

void solve(int initial[N][N], int x, int y, int final[N][N])
{

    priority_queue<Node *, std::vector<Node *>, comp> pq;

    Node *root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);

    pq.push(root);

    while (!pq.empty())
    {
        total_count++;
        cout << total_count << " ";
        if(total_count>10000){
            break;
        }
        Node *min = pq.top();
```

```
        pq.pop();

        // heuristic=0 -> goal node
        if (min->cost == 0)
        {
            printf("\n\n");
            printPath(min);
            return;
        }

        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                Node *child = newNode(min->mat, min->x,
                                      min->y, min->x + row[i],
                                      min->y + col[i],
                                      min->level + 1, min);
                child->cost = calculateCost(child->mat, final);
                if (!isSame(min->parent, child))
                {
                    pq.push(child);
                }
            }
        }
    }
}

int main()
{

    // int initial[N][N] =
    //     {
    //         {1, 2, 3},
    //         {5, 6, 0},
    //         {7, 8, 4}};

    // int final[N][N] =
    //     {
    //         {1, 2, 3},
    //         {5, 8, 6},
    //         {0, 7, 4}};

    // int x = 1, y = 2;
```

```
    int initial[N][N] =
        {
            {1, 2, 0},
            {4, 6, 3},
            {7, 5, 8}};

    int final[N][N] =
        {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 0}};

    int x = 0, y = 2;

    solve(initial, x, y, final);

    return 0;
}
```

OUTPUT

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

1 2 0
4 6 3
7 5 8

1 2 3
4 6 0
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0
```

- 8 PUZZLE (using bfg) :-

CODE

```
#include <bits/stdc++.h>
using namespace std;
#define N 3

int total_count = 0;

struct Node
{
    Node *parent;

    int mat[N][N];

    int x, y;

    int cost;

    int level;
};

// Function to print N x N matrix
void printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}

// Function to allocate a new node
Node *newNode(int mat[N][N], int x, int y, int newX, int newY,
int level, Node *parent)
{
    Node *node = new Node;

    node->parent = parent;

    memcpy(node->mat, mat, sizeof node->mat);

    swap(node->mat[x][y], node->mat[newX][newY]);
```

```
    node->cost = INT_MAX;

    node->level = level;

    node->x = newX;
    node->y = newY;

    return node;
}

// bottom, left, top, right
int row[] = {1, 0, -1, 0};
int col[] = {0, -1, 0, 1};

int calculateCost(int initial[N][N], int final[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (initial[i][j] && initial[i][j] != final[i][j])
                count++;
    return count;
}

int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}

void printPath(Node *root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

struct comp
{
    bool operator()(const Node *lhs, const Node *rhs) const
    {
        return (lhs->cost) > (rhs->cost); // evaluation funtion
```

```
    }
};

int isSame(Node *a, Node *b)
{
    if(a==NULL || b==NULL){
        return 0;
    }
    return calculateCost(a->mat, b->mat) == 0;
}

void solve(int initial[N][N], int x, int y, int final[N][N])
{

    priority_queue<Node *, std::vector<Node *>, comp> pq;

    Node *root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);

    pq.push(root);

    while (!pq.empty())
    {
        total_count++;
        cout << total_count << " ";
        if(total_count>10000){
            break;
        }
        Node *min = pq.top();

        pq.pop();

        // heuristic=0 -> goal node
        if (min->cost == 0)
        {
            printf("\n\n");
            printPath(min);
            return;
        }

        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                Node *child = newNode(min->mat, min->x,
```

```
                                        min->y, min->x + row[i],
                                        min->y + col[i],
                                        min->level + 1, min);
                child->cost = calculateCost(child->mat, final);
                if (!isSame(min->parent, child))
                {
                    pq.push(child);
                }
            }
        }
    }
}

int main()
{

    // int initial[N][N] =
    //     {
    //         {1, 2, 3},
    //         {5, 6, 0},
    //         {7, 8, 4}};

    // int final[N][N] =
    //     {
    //         {1, 2, 3},
    //         {5, 8, 6},
    //         {0, 7, 4}};

    // int x = 1, y = 2;

    int initial[N][N] =
        {
            {1, 2, 0},
            {4, 6, 3},
            {7, 5, 8}};

    int final[N][N] =
        {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 0}};

    int x = 0, y = 2;

    solve(initial, x, y, final);
```

```
    return 0;
}
```

OUTPUT

```
1 2 3 4 5

1 2 0
4 6 3
7 5 8

1 2 3
4 6 0
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0
```

- 8 PUZZLE (using A*) : -

CODE

```
#include <bits/stdc++.h>
using namespace std;
#define N 3

int total_count = 0;

struct Node
{
    Node *parent;
```

```
    int mat[N][N];
    int x, y;
    int cost;
    int level;
};

// Function to print N x N matrix
void printMatrix(int mat[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}

// Function to allocate a new node
Node *newNode(int mat[N][N], int x, int y, int newX, int newY, int
level, Node *parent)
{
    Node *node = new Node;

    node->parent = parent;

    memcpy(node->mat, mat, sizeof node->mat);

    swap(node->mat[x][y], node->mat[newX][newY]);

    node->cost = INT_MAX;

    node->level = level;

    node->x = newX;
    node->y = newY;

    return node;
}
// bottom, left, top, right
int row[] = {1, 0, -1, 0};
int col[] = {0, -1, 0, 1};

int calculateCost(int initial[N][N], int final[N][N])
{
```

```
    int count = 0;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (initial[i][j] && initial[i][j] != final[i][j])
                count++;
    return count;
}

int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}

void printPath(Node *root)
{
    if (root == NULL)
        return;
    printPath(root->parent);
    printMatrix(root->mat);

    printf("\n");
}

struct comp
{
    bool operator()(const Node *lhs, const Node *rhs) const
    {
        return (lhs->cost) > (rhs->cost); // evaluation funtion
    }
};

int isSame(Node *a, Node *b)
{
    if(a==NULL || b==NULL){
        return 0;
    }
    return calculateCost(a->mat, b->mat) == 0;
}
void solve(int initial[N][N], int x, int y, int final[N][N])
{

    priority_queue<Node *, std::vector<Node *>, comp> pq;

    Node *root = newNode(initial, x, y, x, y, 0, NULL);
    root->cost = calculateCost(initial, final);
```

```
    pq.push(root);
    while (!pq.empty())
    {
        total_count++;
        cout << total_count << " ";
        if(total_count>10000){
            break;
        }
        Node *min = pq.top();

        pq.pop();
        // heuristic=0 -> goal node
        if (min->cost == 0)
        {
            printf("\n\n");
            printPath(min);
            return;
        }

        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y + col[i]))
            {
                Node *child = newNode(min->mat, min->x,
                                      min->y, min->x + row[i],
                                      min->y + col[i],
                                      min->level + 1, min);
                child->cost = calculateCost(child->mat, final);
                if (!isSame(min->parent, child))
                {
                    pq.push(child);
                }
            }
        }
    }
}

int main()
{
    int initial[N][N] =
        {
            {1, 2, 0},
            {4, 6, 3},
            {7, 5, 8}};
```

```
    int final[N][N] =
        {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 0}};

    int x = 0, y = 2;

    solve(initial, x, y, final);

    return 0;
}
```

OUTPUT

```
1 2 3 4 5

1 2 0
4 6 3
7 5 8

1 2 3
4 6 0
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0
```