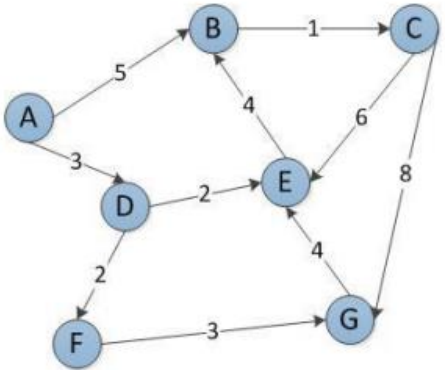
NIKHIL BADYAL
001810501069(A2)

ASSIGNMENT 2

ASSIGNMENT – 2

1. Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. You need to implement greedy version of this search algorithm and test them on following graph (A is the start node and G is the Goal node) and print the Search sequence.



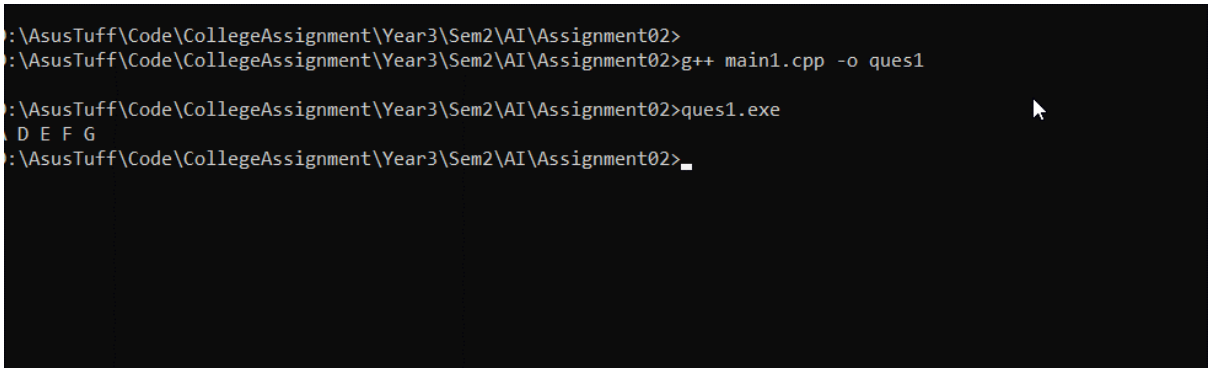
CODE

```
#include <iostream>
#include<vector>
#include<unordered_map>
#include<queue>

using namespace std;

void BestFirstSearch(unordered_map<char, vector<pair<char, int>>>& graph, char
 src, char target){
    priority_queue<pair<int, char>, vector<pair<int, char>>, greater<pair<int,
 char>>> pq;
    unordered_map<char, bool>visited;
    pq.push({0, src});

    while(!pq.empty()){
        char node = pq.top().second;pq.pop();
        cout<<node<<" ";
        if(node == target){
            break;
        }
```

```
        vector<pair<char, int>> neighbor;
        neighbor = graph[node];

        for(int i=0;i<neighbor.size();i++){
            if(!visited[neighbor[i].first]){
                visited[neighbor[i].first] = true;
                pq.push({neighbor[i].second, neighbor[i].first});
            }
        }
    }
}
int main() {
    unordered_map<char, vector<pair<char, int>>> graph;

    graph['A'].push_back({'B', 5});
    graph['A'].push_back({'D', 3});

    graph['B'].push_back({'C', 1});

    graph['C'].push_back({'E', 6});
    graph['C'].push_back({'G', 8});

    graph['D'].push_back({'E', 2});
    graph['D'].push_back({'F', 2});

    graph['E'].push_back({'B', 4});

    graph['F'].push_back({'G', 3});

    graph['G'].push_back({'E', 4});


    BestFirstSearch(graph, 'A', 'G');
}
```

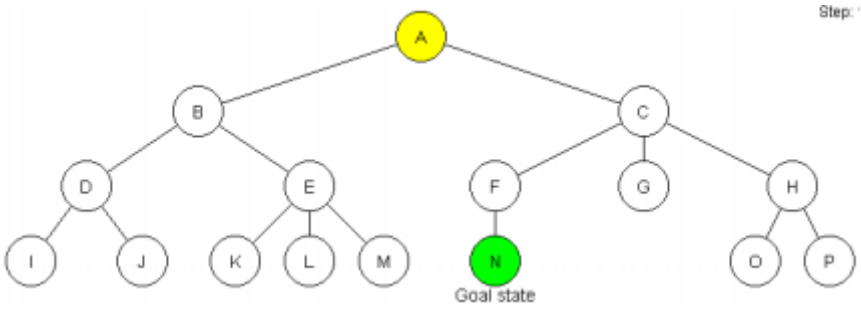
OUTPUT

+



2. Iterative deepening depth-first search (IDDFS) is a state space search strategy in which a depthlimited search is run repeatedly, increasing the depth limit with each iteration until it reaches d, the depth of the shallowest goal state.

Implement this search algorithm and test them on following graph and print the Search sequence (A is the start state and N is the goal state).

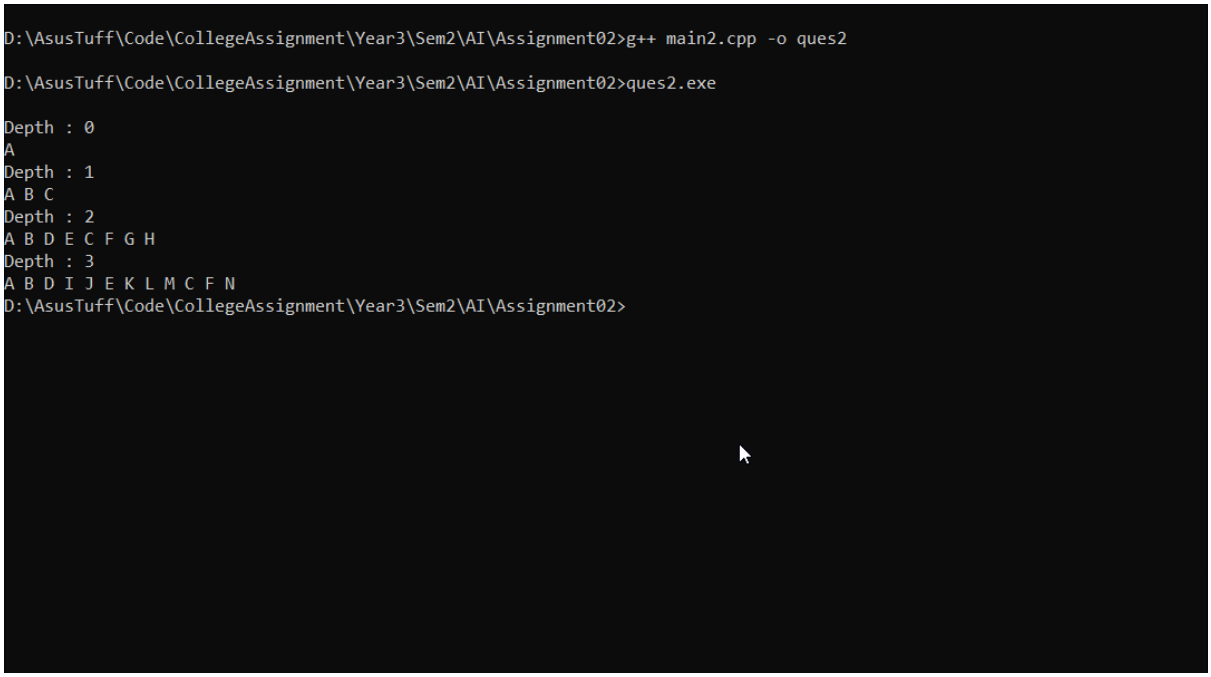


CODE

```
#include <iostream>
#include<vector>
#include<unordered_map>
#include<queue>

using namespace std;

bool dfs(unordered_map<char, vector<char>>& graph, unordered_map<char, bool>&
visited, char u, char target, int depth){
    cout<<u<<" ";
    visited[u] = true;

    if(u == target){
        return true;
    }
    if(depth == 0){
        return false;
    }


    vector<char>neighbor;
    neighbor = graph[u];
    for(int i=0;i<graph[u].size();i++){
        char v = graph[u][i];
        if(!visited[v]){
            if(dfs(graph, visited, v, target, depth-1)){
                return true;
            }
        }
    }
    return false;
}
```

```
void IDDFS(unordered_map<char, vector<char>>& graph, char src, char target, in
t maxDepth){

    for(int i=0;i<=maxDepth;i++){
        unordered_map<char, bool>visited;
        cout<<"\nDepth : "<<i<<endl;
        if(dfs(graph, visited, src, target, i)){
            break;
        }
    }
}
int main() {
    unordered_map<char, vector<char>> graph;

    graph['A'].push_back('B');
    graph['A'].push_back('C');

    graph['B'].push_back('D');
    graph['B'].push_back('E');

    graph['C'].push_back('F');
    graph['C'].push_back('G');
    graph['C'].push_back('H');

    graph['D'].push_back('I');
    graph['D'].push_back('J');

    graph['E'].push_back('K');
    graph['E'].push_back('L');
    graph['E'].push_back('M');

    graph['F'].push_back('N');

    graph['H'].push_back('O');
    graph['H'].push_back('P');

    int maxDepth = 3;

    IDDFS(graph, 'A', 'N', maxDepth);

}
```

OUTPUT


D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>g++ main2.cpp -o ques2
D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>ques2.exe
Depth : 0
A
Depth : 1
A B C
Depth : 2
A B D E C F G H
Depth : 3
A B D I J E K L M C F N
D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>

3. Uninformed Depth Limited Search considering 8-Puzzle Problem. Report Order of nodes visited and Solution Path for the search technique.

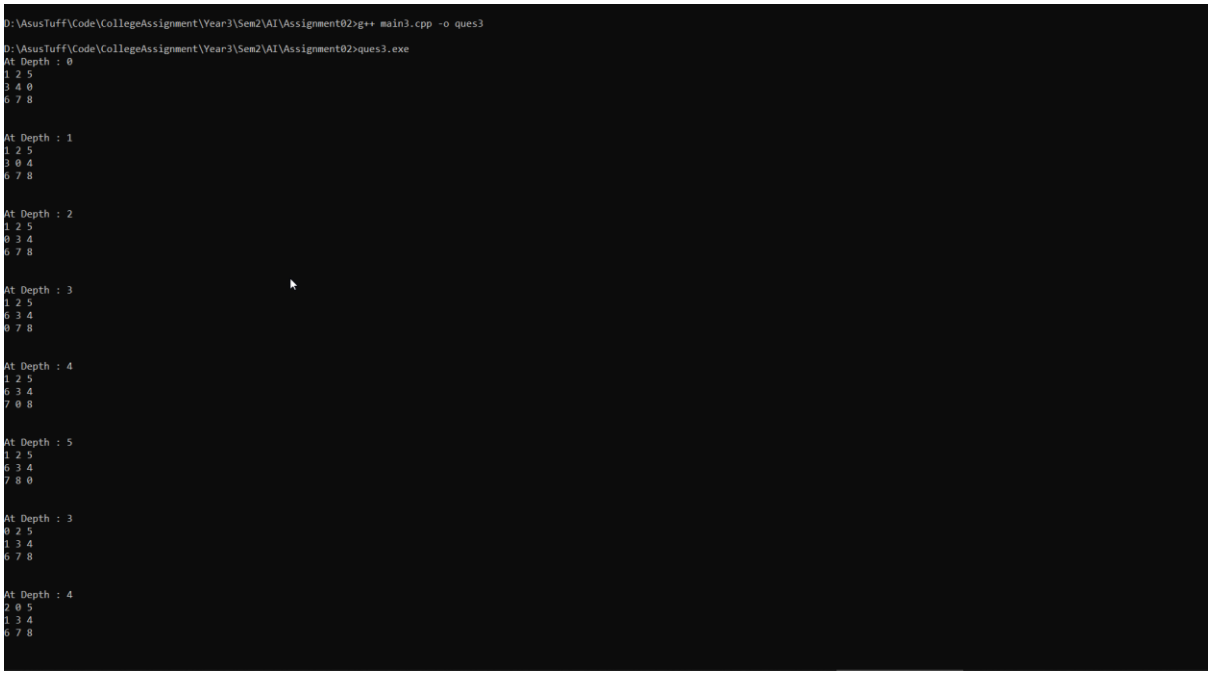
CODE

```
#include <iostream>
#include<vector>
#include<unordered_map>
#include<stack>

using namespace std;
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
int depth=0;
stack<vector<vector<int>>>st;
int cont = 0;
bool dfs(vector<vector<int>>& src, vector<vector<int>>& target, vector<vector<
bool>>& visited, int x, int y, int maxDepth){
    visited[x][y] = true;
    cout<<"At Depth : "<<depth-maxDepth<<endl;
    for(int i=0;i<src.size();i++){
        for(int j=0;j<src[0].size();j++){
            cout<<src[i][j]<<" ";
        }cout<<endl;
    }
    cout<<endl<<endl;

    if(maxDepth == 0){
        cout<<"At max Depth, Now going back : "<<endl;
        return false;
    }

    int flag = 0;
    for(int i=0;i<src.size();i++){
        for(int j=0;j<src[0].size();j++){
            if(src[i][j] != target[i][j]){
                flag = 1;
            }
        }
    }

    if(flag == 0){
```

```
        return true;
    }

    for(int k=0;k<4;k++){
        int di = x + dx[k];
        int dj = y + dy[k];

        if(di>=0 && di<src.size() && dj>=0 && dj < src[0].size() && !visited[d
i][dj]){
            swap(src[x][y], src[di][dj]);
            int flag2 = 0;
            if(dfs(src, target, visited, di, dj, maxDepth-1)){
                st.push(src);
                cont++;
                swap(src[x][y], src[di][dj]);
                flag2=1;
                return true;
            }
            if(flag2 == 0)
                swap(src[x][y], src[di][dj]);

        }
    }
    visited[x][y] = false;
    return false;
}
void DLS(vector<vector<int>>& src, vector<vector<int>>& target, int maxDepth){

     int flag = 0;
    for(int i=0;i<src.size();i++){
        for(int j=0;j<src[0].size();j++){
            if(src[i][j] != target[i][j]){
                flag = 1;
            }
        }
    }
    if(flag == 0){
        cout<<"Source and target both are same"<<endl;
    }
    vector<vector<bool>>visited(src.size(), vector<bool>(src[0].size(), false)
);
    if(dfs(src, target, visited, 1, 2, maxDepth)){
        cout<<"Solution found : "<<endl;
        st.push(src);
        while(!st.empty()){

            vector<vector<int>>v;
            v = st.top();st.pop();
```

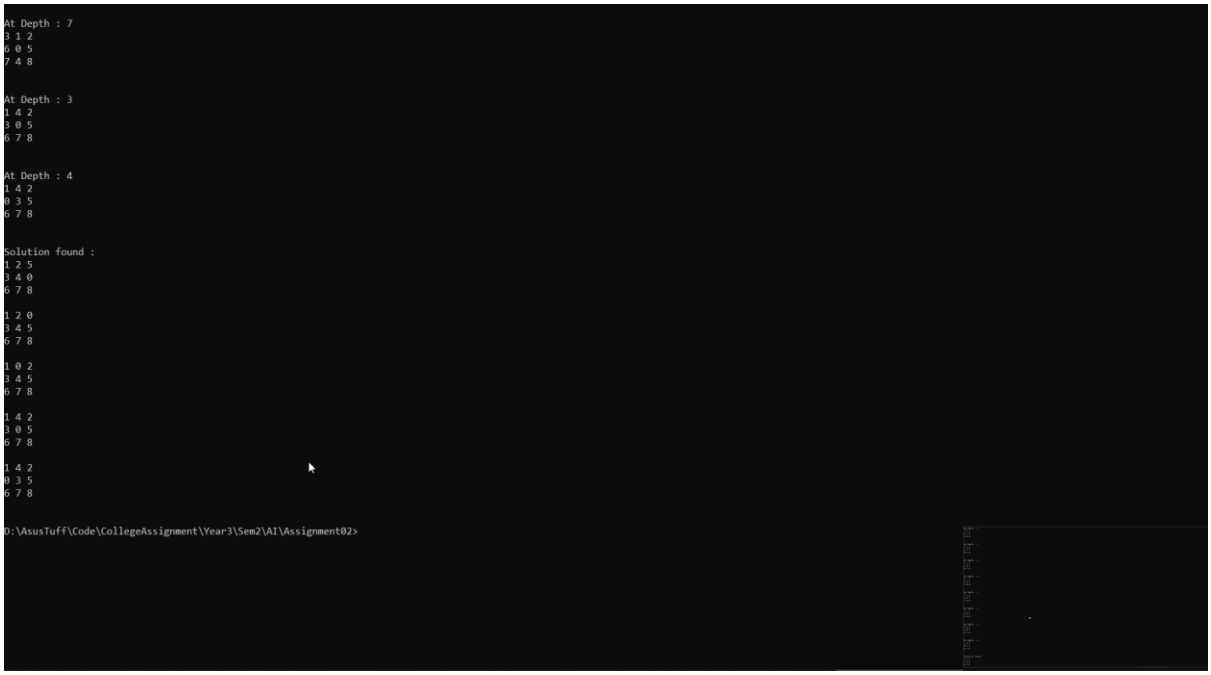
```
            for(int i=0;i<src.size();i++){
                for(int j=0;j<src[0].size();j++){
                    cout<<v[i][j]<<" ";
                }cout<<endl;
            }
            cout<<endl;
        }
    }else{
        cout<<"Solution does not found : ";
    }

}
int main() {
    vector<vector<int>> src = {{1,2,5},
                               {3,4,0},
                               {6,7,8}};


    vector<vector<int>> target = {{1,4,2},
                                  {0,3,5},
                                  {6,7,8}};

   int maxDepth = 8;
   depth=maxDepth;
   DLS(src, target, maxDepth);

}
```

OUTPUT


D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>g++ main3.cpp -o ques3
D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>ques3.exe
At Depth : 0
1 2 5
3 4 0
6 7 8
At Depth : 1
1 2 5
3 0 4
6 7 8
At Depth : 2
1 2 5
0 3 4
6 7 8
At Depth : 3
1 2 5
6 3 4
0 7 8
At Depth : 4
1 2 5
6 3 4
7 0 8
At Depth : 5
1 2 5
6 3 4
7 8 0
At Depth : 3
0 2 5
1 3 4
6 7 8
At Depth : 4
2 0 5
1 3 4
6 7 8


At Depth : 2
1 0 5
3 2 4
6 7 8
At Depth : 3
1 5 0
3 2 4
6 7 8
At Depth : 3
0 1 5
3 2 4
6 7 8
At Depth : 4
3 1 5
0 2 4
6 7 8
At Depth : 5
3 1 5
6 2 4
0 7 8
At Depth : 6
3 1 5
6 2 4
7 0 8
At Depth : 7
3 1 5
6 2 4
7 8 0
At Depth : 1
1 2 5
3 4 8
6 7 0
At Depth : 2
1 2 5
3 4 8


At Depth : 7
2 4 5
1 0 8
3 6 7
At Depth : 3
1 2 5
3 0 8
6 4 7
At Depth : 4
1 2 5
0 3 8
6 4 7
At Depth : 5
1 2 5
6 3 8
0 4 7
At Depth : 5
0 2 5
1 3 8
6 4 7
At Depth : 6
2 0 5
1 3 8
6 4 7
At Depth : 7
2 5 0
1 3 8
6 4 7
At Depth : 4
1 0 5
3 2 8
6 4 7
At Depth : 5
1 5 0


At Depth : 7
3 1 2
6 0 5
7 4 8
At Depth : 3
1 4 2
3 0 5
6 7 8
At Depth : 4
1 4 2
0 3 5
6 7 8
Solution found :
1 2 5
3 4 0
6 7 8
1 2 0
3 4 5
6 7 8
1 0 2
3 4 5
6 7 8
1 4 2
3 0 5
6 7 8
1 4 2
0 3 5
6 7 8
D:\AsusTuff\Code\CollegeAssignment\Year3\Sem2\AI\Assignment02>

4. Implement Uninformed Bi-directional Iterative Broadening Search considering Tree. Report Order of nodes visited and Solution Path for the search technique

CODE

```
#include <iostream>
#include<vector>
#include<unordered_map>
#include<queue>

using namespace std;

bool BFS(vector<vector<int>>& graph, queue<int>& q, vector<bool>& visited, vec
tor<int>& parent, int maxBreadth, char n){

    int u = q.front();
    cout<<u<<" ";
    q.pop();
    int count = 0;
    for (int i=0;i<graph[u].size() && count<maxBreadth;i++)
    {
        int v = graph[u][i];

        if (!visited[v]){
            count++;
            parent[v] = u;
            visited[v] = true;
            q.push(v);
        }

    }
    return false;
}


int checkMeet(vector<bool>& s_visited, vector<bool>& t_visited, int V)
{
    int intersectNode = -1;
    for(int i=0;i<V;i++)
    {
        if(s_visited[i] && t_visited[i])
            return i;
    }
    return -1;
}
```

```
void printPath(vector<int>& s_parent, vector<int>& t_parent, int s, int t, int
 intersectNode)
{
    vector<int> path;
    path.push_back(intersectNode);
    int i = intersectNode;
    while (i != s)
    {
        path.insert(path.begin(), s_parent[i]);
        i = s_parent[i];
    }
    i = intersectNode;
    while(i != t)
    {
        path.push_back(t_parent[i]);
        i = t_parent[i];
    }

    vector<int>::iterator it;
    for(it = path.begin();it != path.end();it++)
        cout<<*it<<" ";
    cout<<"\n";
};

bool BS(vector<vector<int>>& graph, int src, int target, int maxBreadth){

    vector<bool> s_visited(graph.size(), false), t_visited(graph.size(), false
);
    vector<int> sParent(graph.size(), 0), tParent(graph.size(), 0);

    queue<int> s_queue, t_queue;

    int intersectNode = -1;

    s_queue.push(src);
    s_visited[src] = true;
    sParent[src]=-1;

    t_queue.push(target);
    t_visited[target] = true;
    tParent[target] = -1;

    while (!s_queue.empty() && !t_queue.empty())
    {
        BFS(graph, s_queue, s_visited, sParent, maxBreadth, 's');
        BFS(graph, t_queue, t_visited, tParent, maxBreadth, 't');
```

```
        intersectNode = checkMeet(s_visited, t_visited, graph.size());

        if(intersectNode != -1)
        {
            cout << "\nPath exist between " << src << " and "<< target << "\n"
;
            cout << "Meet at: " << intersectNode << "\n";

            printPath(sParent, tParent, src, target, intersectNode);
            return true;
        }
    }
    return false;
}
void BIBS(vector<vector<int>>& graph, char src, char target, int maxDepth){


    for(int i=0;i<=maxDepth;i++){
        cout<<"\nBreadth : "<<i<<endl;
        if(BS(graph, src, target, i)){
            break;
        }else{
            cout<<"path doesn't exist"<<endl;
        }
    }
}
int main() {
    vector<vector<int>> graph(15);

    graph[0].push_back(4);
    graph[1].push_back(4);
    graph[2].push_back(5);
    graph[3].push_back(5);
    graph[4].push_back(6);
    graph[5].push_back(6);
    graph[6].push_back(7);
    graph[7].push_back(8);
    graph[8].push_back(9);
    graph[8].push_back(10);
    graph[9].push_back(11);
    graph[9].push_back(12);
    graph[10].push_back(13);
    graph[10].push_back(14);


    graph[4].push_back(0);
    graph[4].push_back(1);
    graph[5].push_back(2);
    graph[5].push_back(3);
```

```
    graph[6].push_back(4);
    graph[6].push_back(5);
    graph[7].push_back(6);
    graph[8].push_back(7);
    graph[9].push_back(8);
    graph[10].push_back(8);
    graph[11].push_back(9);
    graph[12].push_back(9);
    graph[13].push_back(10);
    graph[14].push_back(10);


    BIBS(graph, 0, 14, 6);

}
```

OUTPUT