IN THE NAME OF GOD

AmirKabir University

Advanced Programming

DR. JAHANSHAHI
OMID RAZZAGHI
9623053

Report midterm project

In the name of God:

In this project we have two parts :1) Implementation of BFS and DFS

2) Beautiful UI

First of all, we go to declare DFS method. DFS method stands for Depth first search method.

DFS

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

At 8-puzzle game we can make the possible movement of empty tile as graph and then do it again until we find the goal state. In this case and DFS method if 8-puzzle is scrambled a lot or there is no solve for it, DFS or BFS method need limit number to know how depth do they search for goal state. We can see pseudocode of DFS method in below image.

$$DFS(root) = DFS(UP) + DFS(DOWN) + DFS(RIGHT) + DFS(LEFT)$$

```
#include ....
vector solution;
vector DFSTraverse(node root){
  if(node == goal){
    isFinished = true
  }

  else{
  solution.push_back(node UP);
  DFSTraverse(node UP);


  solution.push_back(node DOWN);
  DFSTraverse(node DOWN);

  solution.push_back(node RIGHT);
  DFSTraverse(node RIGHT);

  solution.push_back(node LEFT);
  DFSTraverse(node LEFT);
  }

}
```

BFS

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

In this project we use Node class to help search by BFS method. Each node is connected to its children and parent so we can make possible nodes in each level and connect them to children and parent. If we find goal node we can find solution by its connections to parent and grandparent and … .

```cpp
std::vector<Board> BFSTraverse(Board start , Board goal,int numberOfLevel){
    bool isFinished=false;

    Node root {Node(start,Direction::NOTHING)};
    root.fatherPointer = nullptr;

    if(root.table.getTable() == goal.getTable()){
        isFinished = true;
    }

    std::vector<Node> row;
    row.push_back(root);
    Node finalNode;
    for(int levelCounter {0} ; levelCounter < numberOfLevel && !isFinished; levelCounter++){
        std::vector<Node> tempRow {getRow(row.at(0))};
            auto v = getRow(row.at(0));

        if(checkRow(v,goal)!= -1){
            isFinished = true;
            break;
        }
        for(size_t i{1} ; i < row.size() ; i++){
            auto v = getRow(row.at(i));

            if(checkRow(v,goal)!= -1){
                isFinished = true;
                finalNode = v.at(checkRow(v,goal));

                break;
            }
            tempRow.insert(tempRow.cend(),v.begin() , v.end());

        }
        row = tempRow;

    }
    if(isFinished){
        std::vector<Board> solution{traceSolution(finalNode)};
        start.disp();

        return solution;
    }
    else{
        std::vector<Board> nosolution;

        return nosolution;
    }
}
```

UI

For user interface we can use FLTK Framework which is very simple and low size and we can develop it in docker.

For displaying the windows and our app in docker we get help from XLaunch application to see UI window in docker.

Run docker with below code:

```
1. docker run -v %CD%:<WORKDIR>
   -e
   DISPLAY=<YOUR_IP_ADDRESS>:0.0
   -it --rm <CONTAINER> bash -l
```

-e switch connect docker display to Xlaunch and we can see windows in our laptop.