SCHOOL OF DATA AND COMPUTER SCIENCE, SUN YAT-SEN UNIVERSITY

E2 15-Puzzle Problem (IDA*)

18340013 Conghao Chen

September 11, 2020

Contents

1 IDA* Algorithm

1.1 Description

Iterative deepening A* (IDA*) was first described by Richard Korf in 1985, which is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.

It is a variant of **iterative deepening depth-first search** that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the **A* search algorithm**.

Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree.

Iterative-deepening-A* works as follows: at each iteration, perform a depth-first search, cutting off a branch when its total cost $f(n) = g(n) + h(n)$ exceeds a given threshold. This threshold starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

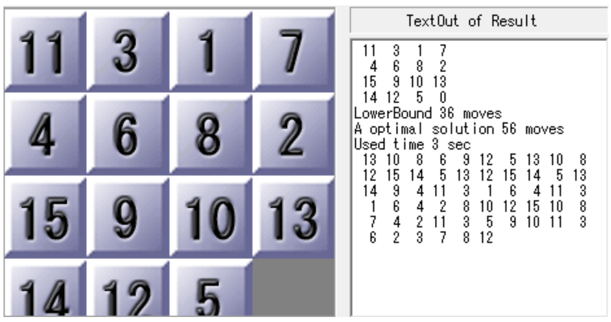
1.2 Pseudocode

```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
cost(node, succ)  step cost function
is_goal(node)     goal test
successors(node)  node expanding function, expand nodes ordered by g + h(node)
ida_star(root)    return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
      path.pop()
    end if
  end for
  return min
end function
```

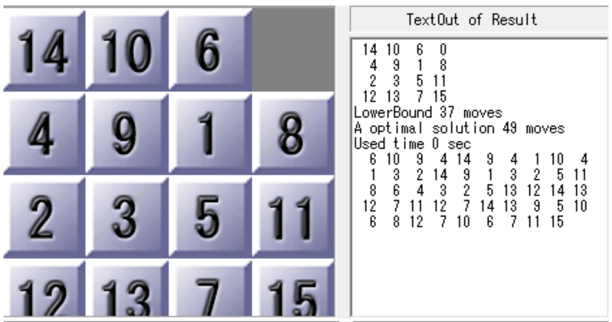
2 Tasks

- Please solve 15-Puzzle problem by using IDA* (Python or C++). You can use one of the two commonly used heuristic functions: h1 = the number of misplaced tiles. h2 = the sum of the distances of the tiles from their goal positions.

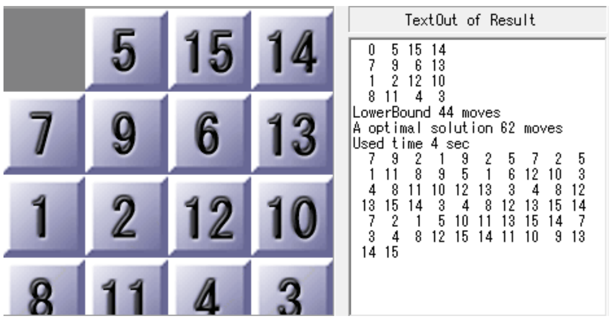
- Here are 4 test cases for you to verify your algorithm correctness. You can also play this game (`15puzzle.exe`) for more information.



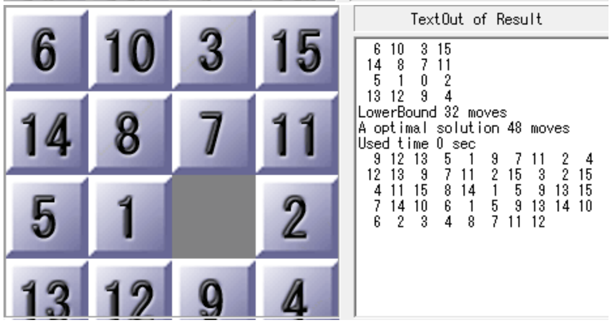
```
TextOut of Result

 11  3  1  7
  4  6  8  2
 15  9 10 13
 14 12  5  0
LowerBound 36 moves
A optimal solution 56 moves
Used time 3 sec
 13 10  8  6  9 12  5 13 10  8
 12 15 14  5 13 12 15 14  5 13
 14  9  4 11  3  1  6  4 11  3
  1  6  4  2  8 10 12 15 10  8
  7  4  2 11  3  5  9 10 11  3
  6  2  3  7  8 12
```



```
TextOut of Result

 14 10  6  0
  4  9  1  8
  2  3  5 11
 12 13  7 15
LowerBound 37 moves
A optimal solution 49 moves
Used time 0 sec
  6 10  9  4 14  9  4  1 10  4
  1  3  2 14  9  1  3  2  5 11
  8  6  4  3  2  5 13 12 14 13
 12  7 11 12  7 14 13  9  5 10
  6  8 12  7 10  6  7 11 15
```



```
TextOut of Result

  0  5 15 14
  7  9  6 13
  1  2 12 10
  8 11  4  3
LowerBound 44 moves
A optimal solution 62 moves
Used time 4 sec
  7  9  2  1  9  2  5  7  2  5
  1 11  8  9  5  1  6 12 10  3
  4  8 11 10 12 13  3  4  8 12
 13 15 14  3  4  8 12 13 15 14
  7  2  1  5 10 11 13 15 14  7
  3  4  8 12 15 14 11 10  9 13
 14 15
```



```
TextOut of Result

  6 10  3 15
 14  8  7 11
  5  1  0  2
 13 12  9  4
LowerBound 32 moves
A optimal solution 48 moves
Used time 0 sec
  9 12 13  5  1  9  7 11  2  4
 12 13  9  7 11  2 15  3  2 15
  4 11 15  8 14  1  5  9 13 15
  7 14 10  6  1  5  9 13 14 10
  6  2  3  4  8  7 11 12
```

- Please send `E02_YourNumber.pdf` to `ai_2020@foxmail.com`, you can certainly use `E02_15puzzle.tex` as the LaTeXtemplate.

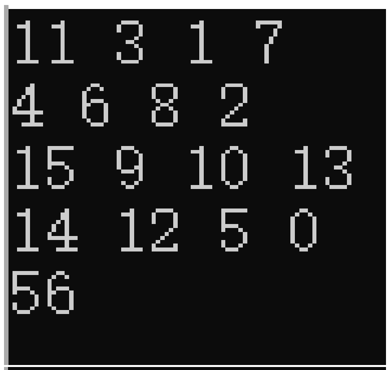
3 Codes

```
#include<iostream>
#include<stdio.h>
#include<cmath>
#include<string>
#include<cassert>
using namespace std;

#define N1 4
#define N2 16
#define LIMIT 100
// express direction
static const int dx[4] = {0, -1, 0, 1};
static const int dy[4] = {1, 0, -1, 0};
static const char dir[4] = {'R', 'U', 'L', 'D'};
//Manhattan distance
int MHD[N2][N2];
```

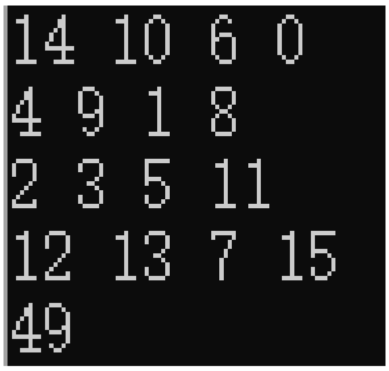
```
struct Puzzle{
        int f[N2],space,h;              //store the information
};

Puzzle state;
int limit;                              //Depth-limit
int path[LIMIT];                        //store the path(direction)

int geth(Puzzle puzzle){
        int sum=0;
        for(int i=0;i<N2;i++) {
                if(puzzle.f[i]==N2) continue;
                sum+=MHD[i][puzzle.f[i]-1];
        }
        return sum;
}

bool solved(){
        for(int i=0;i<N2;i++) if(state.f[i]!=i+1) return false;
        return true;               //the goal
}

bool dfs(int depth,int prev){
        if(state.h==0) return true;
        //If the current depth plus heuristic exceeds the limit, use
           pruning algorithm.
        if(depth+state.h>limit) return false;
        int sx=state.space/N1;
        int sy=state.space%N1;
        Puzzle temp;
        for(int r=0;r<4;r++) {
                int tx=sx+dx[r];
                int ty=sy+dy[r];
                if(tx<0||ty<0||tx>=N1||ty>=N1)continue;
                if(max(prev,r)-min(prev,r)==2)continue;
                temp=state;
                //Calculate the difference of Manhattan distance and
                   exchange the puzzle pieces
                state.h-=MHD[tx*N1+ty][state.f[tx*N1+ty]-1];
                state.h+=MHD[sx*N1+sy][state.f[tx*N1+ty]-1];
                swap(state.f[tx*N1+ty],state.f[sx*N1+sy]);
                state.space=tx*N1+ty;
                if(dfs(depth+1,r)){
                        path[depth]=r;
                        return true;
                }
                state=temp;
        }
        return false;
}

//Iterative deepening
string ID(Puzzle puzzle) {
        puzzle.h=geth(puzzle);      //Manhattan distance of initial
           state
        for(limit=puzzle.h;limit<=LIMIT;limit++) {
                state=puzzle;
                if(dfs(0,-100)){
                        string ans="";
                        for(int i=0;i<limit;i++) ans+=dir[path[i]];
                        return ans;        //store the path
                }
        }

        return "No solution!";
}

int main(){
        for(int i=0;i<N2;i++)
                for(int j=0;j<N2;j++)
                        MHD[i][j]=abs(i/N1-j/N1)+abs(i%N1-j%N1);
                //calculate manhattan distance
        Puzzle fifpuzzle;
        for(int i=0;i<N2;i++){
                cin>>fifpuzzle.f[i];            //input the data
                if(fifpuzzle.f[i]==0){
                        fifpuzzle.f[i]=N2;
                        fifpuzzle.space=i;
                }
        }
        string ans=ID(fifpuzzle);
        cout<<ans.size()<<endl;  //the size of the path is the solution.
        return 0;
}
```

4 Results

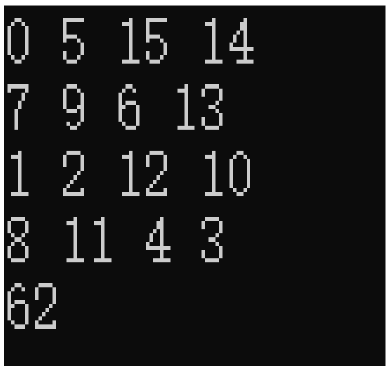
After understanding the knowledge about IDA* in class, it is not difficult to implement the code. However, due to the complexity of this problem and this method, the code needs to take a long time to be executed, which requires patience. In particular, I chose to use Python programming language to complete this program at first, but I didn't get the result after waiting for 30 minutes. I think it's probably because Python always runs more slowly than C++, finally I chose to use C++ programming language. The program written by using C++ takes less time to run than Python. In addition, the heuristic function is h2. I have tested six test cases, and the average running time is about 2-3 minutes. After inputting the 4x4 matrix of 15-Puzzle, the optimal solution will be displayed in 2-3 minutes. The result of the above 4 test cases is as follows:



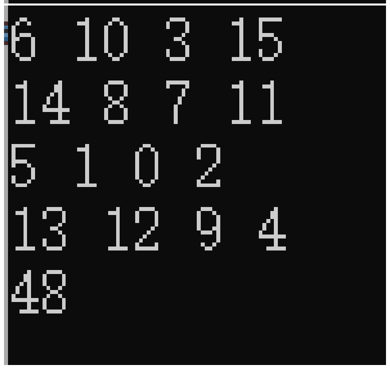
```
11 3 1 7
4 6 8 2
15 9 10 13
14 12 5 0
56
```



```
14 10 6 0
4 9 1 8
2 3 5 11
12 13 7 15
49
```



```
0 5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
62
```



```
6 10 3 15
14 8 7 11
5 1 0 2
13 12 9 4
48
```