E2 15-Puzzle Problem (IDA*)

18340052 何泽

September 10, 2020

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.

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

3. Code

- 一开始我是使用python完成的，但是因为py速度过慢，给的第一个例子可能要跑很久很久，于是我就更换使用c开发。
- 首先计算每个数字最少需要几步走到最终位置，并将它作为一个限制，步数最多也不会多于初状态的这个距离
- 按照上、左、下、右的顺序执行，然后按照IDA*的算法执行，每次迭代的判断条件是 `当前的深度+当前的距离<=最初计算的距离`

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<time.h>

int move[4][2]={{-1,0},{0,-1},{0,1},{1,0}};
int map[4][4],map_og[4][4];
int map2[4*4]={0,5,15,14,7,9,6,13,1,2,12,10,8,11,4,3 };
//int map2[4*4]={11,3,1,7,4,6,8,2,15,9,10,13,14,12,5,0 };
//int map2[4*4]={2,10,3,4,1,9,6,7,13,5,11,8,0,14,15,12 };
int limit,path[100];
int flag,length;
int goal[16][2]= {{3,3},{0,0},{0,1}, {0,2},{0,3}, {1,0},{1,1},
{1,2}, {1,3},{2,0}, {2,1}, {2,2},{2,3},{3,0},{3,1},{3,2}};

void change(int*a,int*b){
    int tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

int h(int a[][4]){
    int i,j,cost=0;
    for(i=0;i<4;i++){
        for(j=0;j<4;j++){
```

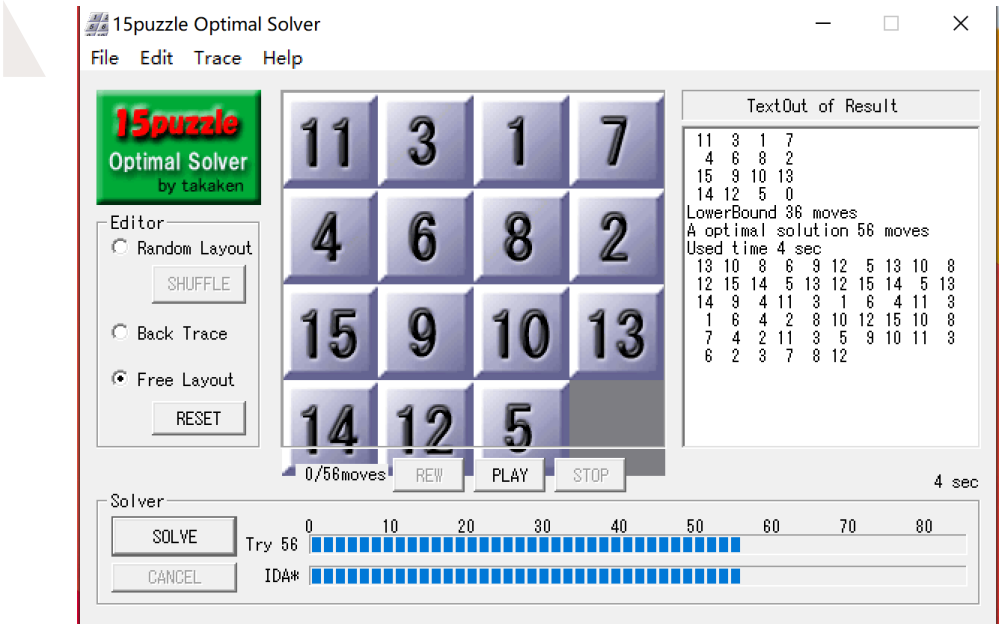
```
            int w=map[i][j];
            cost+=abs(i-goal[w][0])+abs(j-goal[w][1]);
        }
    }
    return cost;
}

void ida_star(int sx,int sy,int len,int pre_move){
    int i,nx,ny;
  if(flag)
    return;
    int dv=h(map);
  if(len==limit){
    if(dv==0){
      flag=1;
      length=len;
      return;
    }
    else
      return;
  }
  else if(len<limit){
    if(dv==0){
      flag=1;
      length=len;
      return;
    }
  }
    for(i=0;i<4;i++){
    if(i+pre_move==3&&len>0)
      continue;
        nx=sx+move[i][0];
        ny=sy+move[i][1];
    if(0≤nx&&nx<4 && 0≤ny&&ny<4){
            change(&map[sx][sy],&map[nx][ny]);
            int p=h(map);
      if(p+len≤limit&&!flag){
                path[len]=i;
                ida_star(nx,ny,len+1,i);
```

```
        if(flag)
                    return;
            }
            change(&map[sx][sy],&map[nx][ny]);
        }
    }
}
int main(){
    int i=0,j,k,l,m,sx,sy,xx,yy;
    char c,g;
  clock_t start,end;
  flag=0,length=0;
  memset(path,-1,sizeof(path));
  for(i=0;i<16;i++){
    if(map2[i]==0){
      map[i/4][i%4]=0;
      map_og[i/4][i%4]=map[i/4][i%4];
      sx=i/4;sy=i%4;
      xx=i/4;yy=i%4;
    }
    else{
      map[i/4][i%4]=map2[i];
      map_og[i/4][i%4]=map[i/4][i%4];
    }
  }
  printf("原矩阵为:\n");
  for (i=0; i<4; i++) {
    for (int y=0; y<4; y++)
      printf("%d ",map[i][y]);
    printf("\n");
  }
  start=clock();
  limit=h(map);
  while(!flag&&length≤50){
    ida_star(sx,sy,0,0);
    if(!flag)
      limit++;
  }
  end=clock();
```

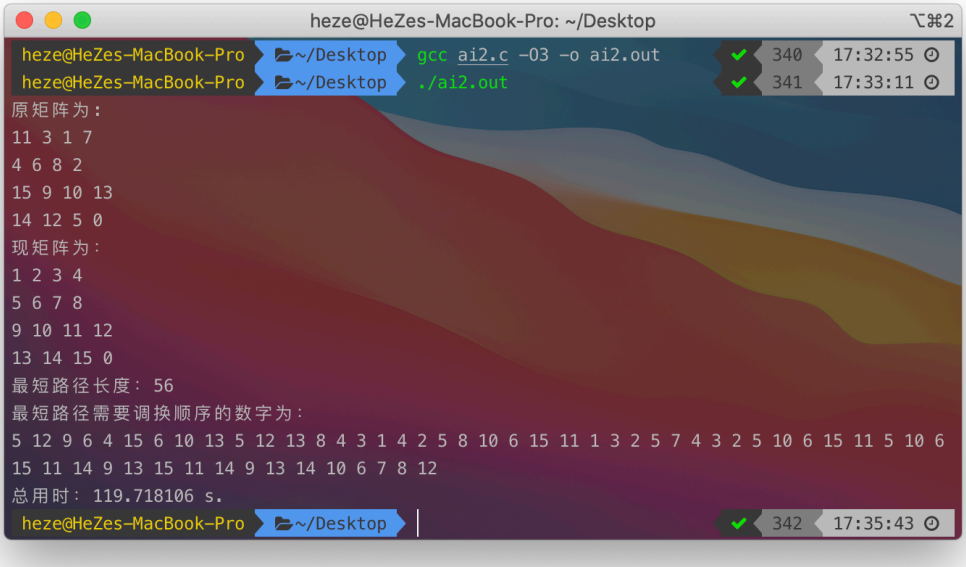
```
105     printf("现矩阵为: \n");
106     for (i=0; i<4; i++) {
107       for (int y=0; y<4; y++)
108         printf("%d ",map[i][y]);
109       printf("\n");
110     }
111     printf("最短路径长度: %d\n",length);
112     printf("最短路径需要调换顺序的数字为: \n");
113     for (i=0; i<length; i++) {
114       if(path[i]==0){
115         printf("%d ",map_og[xx-1][yy]);
116         change(&map_og[xx][yy],&map_og[xx-1][yy]);
117         xx-=1;
118       }
119       else if(path[i]==1){
120         printf("%d ",map_og[xx][yy-1]);
121         change(&map_og[xx][yy],&map_og[xx][yy-1]);
122         yy-=1;
123       }
124       else if(path[i]==2){
125         printf("%d ",map_og[xx][yy+1]);
126         change(&map_og[xx][yy],&map_og[xx][yy+1]);
127         yy+=1;
128       }
129       else if(path[i]==3){
130         printf("%d ",map_og[xx+1][yy]);
131         change(&map_og[xx][yy],&map_og[xx+1][yy]);
132         xx+=1;
133       }
134     }
135     double endtime=(double)(end-start)/CLOCKS_PER_SEC;
136     printf("\n总用时: %f s.\n",endtime);
137       return 0;
138 }
139
```

4. Results

- 第一个例子：

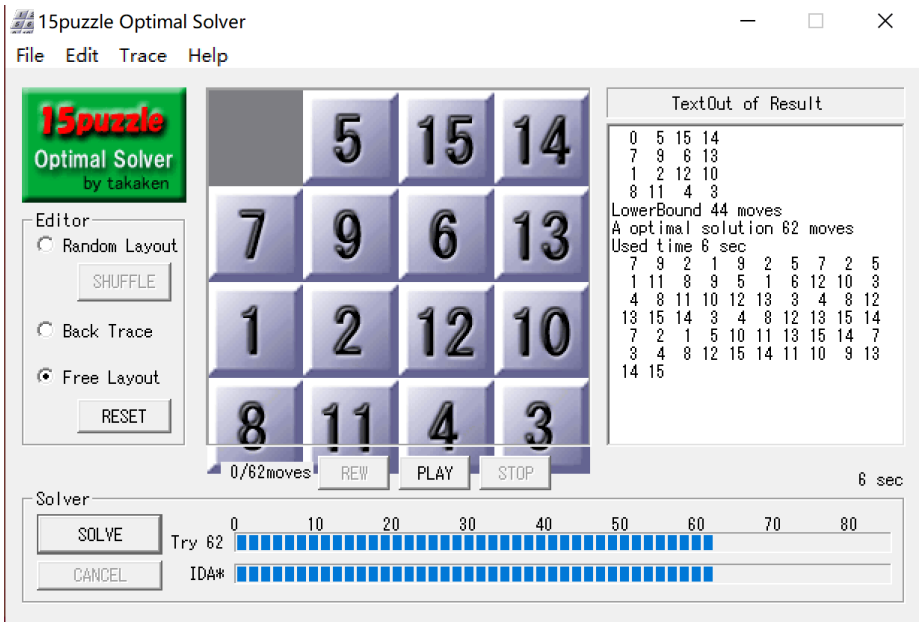


  - 运行（开了$O3$ 优化）：

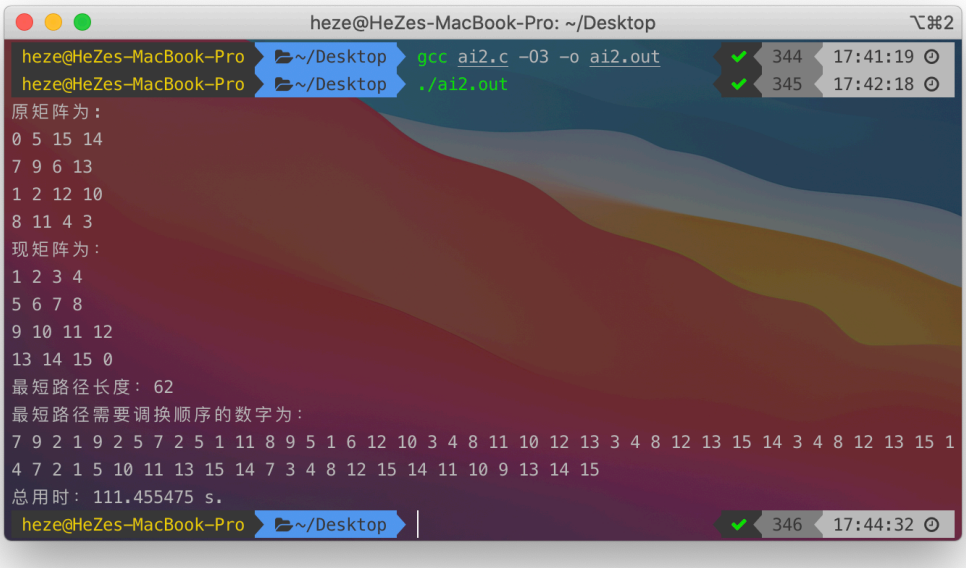


  - 可以看出步数一致，是56步，但是虽然我的路径与那个软件的不一致，但是是因为有多个解，我的运行结果这个路径也是可以的。

- 第二个例子



- 运行



可以看出结果完全一致，算法正确。