

E2 15-Puzzle Problem (IDA*)

17341111 Xuehai Liu

September 7, 2019

Contents

1	IDA* Algorithm	2
1.1	Description	2
1.2	Pseudocode	2
2	Tasks	2
3	Codes	2
4	Results	9

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

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.zip) for more information.
- Please send E02_YourNumber.pdf to ai_201901@foxmail.com, you can certainly use E02_15puzzle.tex as the L^AT_EX template.

3 Codes

```
\documentclass[a4paper, 11pt]{article}
\usepackage{amsmath}
\usepackage{graphicx}
\usepackage{geometry}
\usepackage{listings}
\usepackage[colorlinks, linkcolor=red]{hyperref}
\geometry{scale=0.8}

\title{
\normalfont \normalsize
```

```

\textsc{School of Data and Computer Science , Sun Yat-sen University} \[25pt] %te
\rule{\textwidth}{0.5pt} \[0.4cm] % Thin top horizontal rule
\huge E2 15-Puzzle Problem (IDA*) \[0.5cm] % The assignment title
\rule{\textwidth}{2pt} \[0.5cm] % Thick bottom horizontal rule
\author{19214808 Yikun Liang}
\date{\normalsize\today}
}

\begin{document}
\maketitle
\tableofcontents
\newpage

\section{IDA* Algorithm}
\subsection{Description}
Iterative deepening A* (IDA*) was first described by Richard Korf in 1985, which i

It is a variant of \textbf{iterative deepening depth-first search} that borrows th

Since it is a depth-first search algorithm, its memory usage is lower than in A*,

\textbf{Iterative-deepening-A* works as follows:} at each iteration, perform a dep
\subsection{Pseudocode}
\begin{figure}[ht]
\centering
\includegraphics[width=17.3cm]{Pic/code}
\end{figure}
\section{Tasks}

\begin{itemize}
\item Please solve 15-Puzzle problem by using IDA* (Python or C++). You ca
\item Here are 4 test cases for you to verify your algorithm correctness.
\begin{figure}[ht]
\centering
\includegraphics[width=8cm]{Pic/case1}
\quad
\includegraphics[width=8cm]{Pic/case2}
\\
\includegraphics[width=8cm]{Pic/case3}
\quad
\includegraphics[width=8cm]{Pic/case4}

\end{figure}
\item Please send \texttt{E02\_YourNumber.pdf} to \texttt{ai\_201901@foxm}
\end{itemize}

```

```

\section{Codes}
\lstset{language=C++}
%%\lstset{language=Python}
\begin{lstlisting}

#include<bits/stdc++.h>
#include <windows.h>
using namespace std;
#define INF 1e9
#define FOUND -2

//node defines a state where the puzzle is
typedef struct node
{
    int arr[4][4]={ {1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}};
    int goal[4][4] = { {1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,0}};
    void setup(int a[4][4])
    {
        for (int i=0;i<4;i++)
            for (int j=0;j<4;j++)
                arr[i][j] = a[i][j];
    }
    // move upward
    bool up()
    {
        for (int i=0;i<4;i++)
            for (int j=0;j<4;j++)
            {
                if(arr[i][j] ==0 && i>0 )
                {
                    int temp = arr[i-1][j];
                    arr[i-1][j]= 0 ;
                    arr[i][j] = temp;
                    return true;
                }
            }
        return false;
    }
    bool down()
    {
        for (int i=0;i<4;i++)
            for (int j=0;j<4;j++)
            {
                if(arr[i][j] ==0 && i<3 )
                {
                    int temp = arr[i+1][j];
                    arr[i+1][j]= 0 ;
                    arr[i][j] = temp;
                    return true;
                }
            }
    }
}

```

```

    }
}
return false;
}
bool left()
{
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
        {
            if(arr[i][j] ==0 && j>0 )
            {
                int temp = arr[i][j-1];
                arr[i][j-1]= 0 ;
                arr[i][j] = temp;
                return true;
            }
        }
    return false;
}
bool right()
{
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
        {
            if(arr[i][j] ==0 && j<3 )
            {
                int temp = arr[i][j+1];
                arr[i][j+1]= 0 ;
                arr[i][j] = temp;
                return true;
            }
        }
    return false;
}
void print()
{
    for (int i=0;i<4;i++)
    {
        for (int j=0;j<4;j++)
            cout<<arr[i][j]<<" ";
        cout<<endl;
    }
}
//this h function uses the 1st estimate function
/*
int h()
{
    int cost = 0;
    for (int i=0;i<4;i++)

```

```

        for (int j=0;j<4;j++)
        {
            if (i==3&&j==3)
            {
                if (arr[3][3]==0)
                    cost++;
                continue;
            }
            if (arr[i][j]!=i*4+j+1)
                cost++;
        }
        return cost;
    }*/

//this uses the manhattan function
int h()
{
    int cost = 0;
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
        {
            int t = arr[i][j];
            if (t==0)
            {
                continue;
            }
            int r = (t-1)/4;
            int c = (t-1)%4;
            cost += abs(r - i) + abs(c - j);
        }
    return cost;
}

bool is_goal()
{
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
        {
            if (arr[i][j]!=goal[i][j])
                return false;
        }
    return true;
}

bool operator == (node &other )
{
    for (int i=0;i<4;i++)
        for (int j=0;j<4;j++)
        {
            if ( arr[i][j] != other.arr[i][j])

```

```

        return false;
    }
    return true;
}

}node;

stack <node> path;

//get the successors of node root
vector<node> successors(node root)
{
    vector<node> v;
    int i=0;
    if(root.left())
    {
        v.push_back(root);
        root.right();
    }
    if(root.right())
    {
        v.push_back(root);
        root.left();
    }
    if(root.up())
    {
        v.push_back(root);
        root.down();
    }
    if(root.down())
    {
        v.push_back(root);
        root.up();
    }
    return v;
}

//judge if node n is in path.
bool in(node n,stack <node> path )
{
    stack<node>temp =path;
    for (int i=0;i<path.size();i++)
    {
        if(temp.empty())
            return false;
        node t = temp.top();
        if(n == t)
            return true;
        temp.pop();
    }
}

```

```

    }
}

void printPath(stack <node>p)
{
    for (int i=0;i<p.size();i++)
    {
        p.top().print();
        p.pop();
    }
}

int search(stack<node> &path, int g, int bound)
{
    node N = path.top();
    int f = g + N.h();
    if(f > bound)
        return f;
    if(N.is_goal())
        return FOUND;
    int min =INF;
    vector<node> v = successors(N);
    for (int i=0;i<v.size();i++)
    {
        if( !in(v[i], path) )
        {
            path.push(v[i]);
            int t = search(path,g+1,bound);
            if(t==FOUND)
                return FOUND;
            if(t<min)
                min = t;
            path.pop();
        }
    }
    return min;
}

int ida_star(node root)
{
    int bound = root.h();
    path.push(root);
    int i = 0;
    while(1)
    {
        cout<<"step_"<<++i<<endl;
        int t = search(path,0,bound);
        cout<<"bound_"<<bound<<endl;
        if(t == FOUND)

```



```

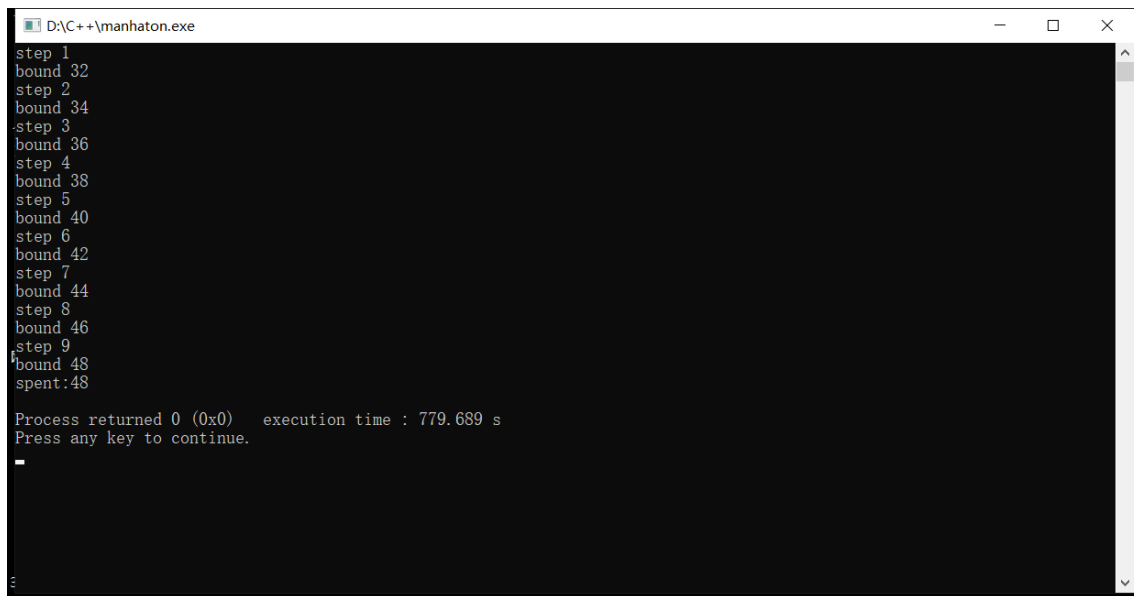
        return bound;
    if (t == INF)
        return -1;
    bound = t;
}
return 0;
}

int main()
{
    node root;
    int a[4][4]={ {11,3,1,7},{4,6,8,2},{15,9,10,13},{14,12,5,0}};
    root.setup(a);
    int b = ida_star(root);
    cout<<b;
    printPath(path);
}

```

4 Results

The first picture is referred to the 4th case in pdf.



```

D:\C++\manhaton.exe
step 1
bound 32
step 2
bound 34
step 3
bound 36
step 4
bound 38
step 5
bound 40
step 6
bound 42
step 7
bound 44
step 8
bound 46
step 9
bound 48
spent:48

Process returned 0 (0x0)   execution time : 779.689 s
Press any key to continue.
_

```

The second picture is referred to the 2nd case in pdf.

```
D:\C++\manhaton.exe
step 1
bound 35
step 2
bound 37
step 3
bound 39
step 4
bound 41
step 5
bound 43
step 6
bound 45
step 7
bound 47
step 8
bound 49
spent:49
Process returned 0 (0x0)   execution time : 74.923 s
Press any key to continue.
```