School of Data and Computer Science, Sun Yat-sen University

Maze Problem

17341078--李焕成

Contents

1 **Task** 2

2 **Codes** 2

3 **Results** 2

1 Task

- Please solve the maze problem (i.e., find the shortest path from the start point to the finish point) by using BFS or DFS (Python or C++)
- The maze layout can be modeled as an array, and you can use the data file MazeData.txt if necessary.
- Please send E01 YourNumber.pdf to ai 201901@foxmail.com, you can certainly use E01 Maze.tex as the LaTeXtemplate.

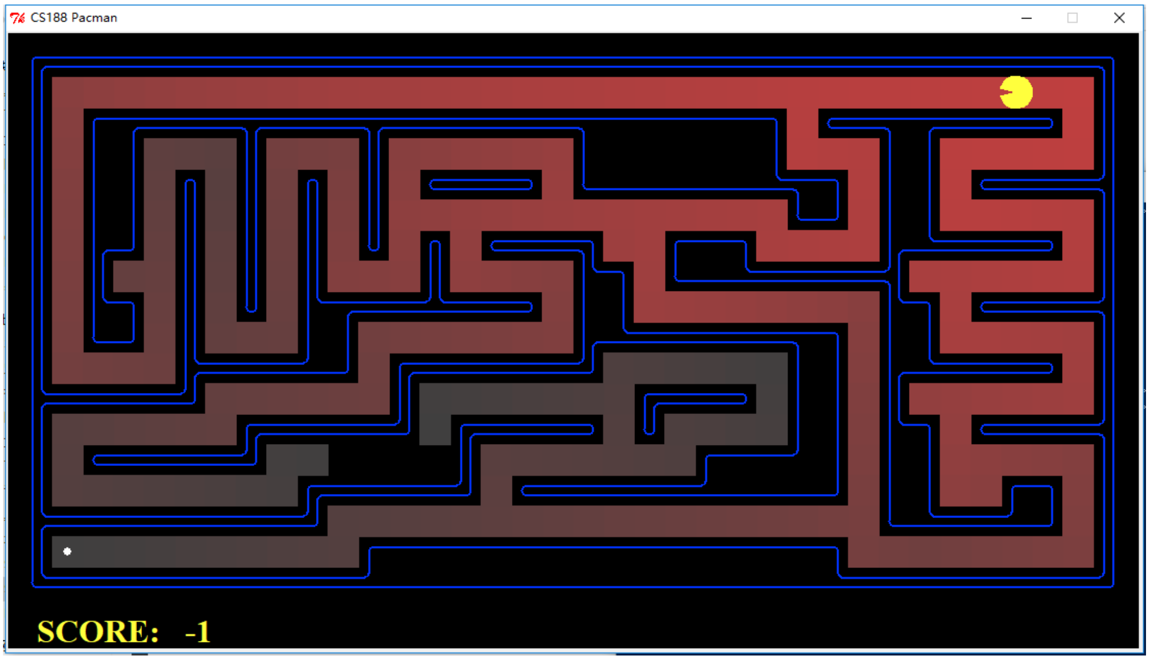


Figure 1: Searching by BFS or DFS

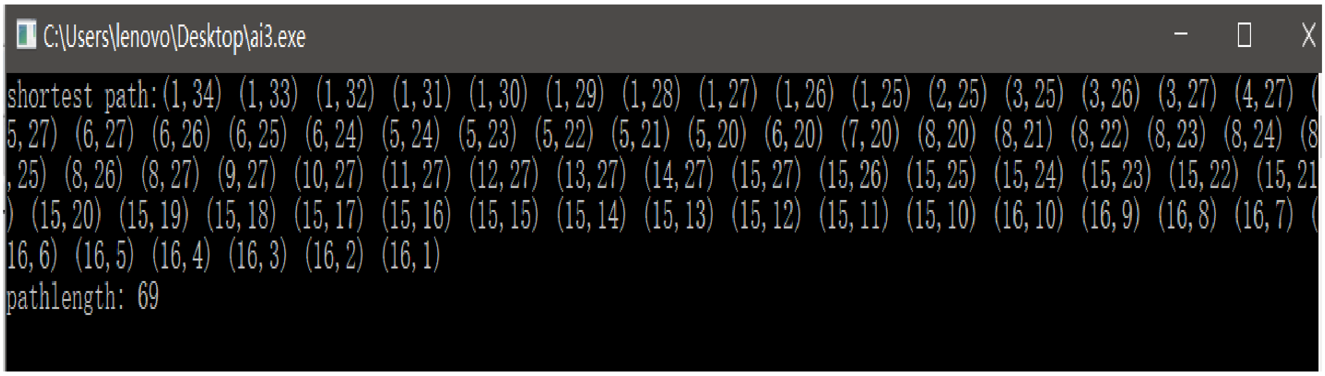
2 Codes

```
#include <iostream>
#include <stack>
#include <vector>
#include<bits/stdc++.h>
using namespace std;

struct Point{
  //行与列
  int row;
  int col;
  Point(int x,int y){
    this->row=x;
    this->col=y;
  }
  bool operator!=(const Point& rhs){
    if(this->row!=rhs.row||this->col!=rhs.col)
      return true;
    return false;
  }

  bool operator==(const Point& rhs) const{
    if(this->row==rhs.row&&this->col==rhs.col)
      return true;
    return false;
  }
};

//func:获取相邻未被访问的节点
//para:mark:结点标记；point：结点；m：行；n：列;endP:终点
//ret:邻接未被访问的结点
Point getAdjacentNotVisitedNode(int** mark,Point point,int m,int n,Point endP){
  Point resP(-1,-1);
  if(point.row-1>=0){
    if(mark[point.row-1][point.col]==0||mark[point.row][point.col]+1<mark[point.row-1][point.col]){//上节点满足条件
        resP.row=point.row-1;
        resP.col=point.col;
        return resP;
    }
  }
  if(point.col+1<n){
    if(mark[point.row][point.col+1]==0||mark[point.row][point.col]+1<mark[point.row][point.col+1]){//右节点满足条件
        resP.row=point.row;
        resP.col=point.col+1;
        return resP;
    }
  }
  if(point.row+1<m){
    if(mark[point.row+1][point.col]==0||mark[point.row][point.col]+1<mark[point.row+1][point.col]){//下节点满足条件
        resP.row=point.row+1;
      resP.col=point.col;
      return resP;
    }
  }
  if(point.col-1>=0){
    if(mark[point.row][point.col-1]==0||mark[point.row][point.col]+1<mark[point.row][point.col-1]){//左节点满足条件
      resP.row=point.row;
      resP.col=point.col-1;
      return resP;
    }
  }
  return resP;
}

//func：给定二维迷宫，求可行路径
//para:maze：迷宫；m：行；n：列；startP：开始结点 endP：结束结点；pointStack：栈，存放路径结点;vecPath:存放最短路径
//ret:无
void mazePath(void* maze,int m,int n, Point& startP, Point endP,stack<Point>& pointStack,vector<Point>& vecPath){
  //将给定的任意列数的二维数组还原为指针数组，以支持下标操作
  int** maze2d=new int*[m];
  for(int i=0;i<m;++i){
    maze2d[i]=(int*)maze+i*n;
  }

  if(maze2d[startP.row][startP.col]==-1||maze2d[endP.row][endP.col]==-1)
    return ;                //输入错误

  //建立各个节点访问标记，表示结点到到起点的权值，也记录了起点到当前结点路径的长度
  int** mark=new int*[m];
  for(int i=0;i<m;++i){
    mark[i]=new int[n];
  }
  for(int i=0;i<m;++i){
    for(int j=0;j<n;++j){
      mark[i][j]=*((int*)maze+i*n+j);
    }
  }
  if(startP==endP){//起点等于终点
    vecPath.push_back(startP);
    return;
  }

  //增加一个终点的已被访问的前驱结点集
  vector<Point> visitedEndPointPreNodeVec;

  //将起点入栈
  pointStack.push(startP);
  mark[startP.row][startP.col]=true;

  //栈不空并且栈顶元素不为结束节点
  while(pointStack.empty()==false){
    Point
adjacentNotVisitedNode=getAdjacentNotVisitedNode(mark,pointSta
ck.top(),m,n,endP);
    if(adjacentNotVisitedNode.row==-1){ //没有符合条件的相邻节点
      pointStack.pop(); //回溯到上一个节点
      continue;
    }
    if(adjacentNotVisitedNode==endP){//以较短的路劲，找到了终点,

mark[adjacentNotVisitedNode.row][adjacentNotVisitedNode.col]=m
ark[pointStack.top().row][pointStack.top().col]+1;
      pointStack.push(endP);
      stack<Point> pointStackTemp=pointStack;
      vecPath.clear();
      while (pointStackTemp.empty()==false){
        vecPath.push_back(pointStackTemp.top());//这里vecPath存放
的是逆序路径
        pointStackTemp.pop();
      }
      pointStack.pop(); //将终点出栈

      continue;
    }
    //入栈并设置访问标志为true

mark[adjacentNotVisitedNode.row][adjacentNotVisitedNode.col]=m
ark[pointStack.top().row][pointStack.top().col]+1;
```

```
      pointStack.push(adjacentNotVisitedNode);
    }
}

int main(){
        int i,j,start_x,start_y,end_x,end_y,pathlength=0;
   int maze[18][36]={
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,5,1},
        {1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1},
        {1,0,1,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0
,1},
        {1,0,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,0,1,1,1,1,1},

        {1,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0,0,0,0,
0,1},
       {1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,0,0,1,1,1,0,0,0,0,1,1,1,1,1,1,0,1},
        {1,0,1,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,1,1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,
1},
       {1,0,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,1,1,1,1,1},
        {1,0,1,1,0,1,0,0,0,1,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,0,1,1,0,0,0,0,0,
1},
        {1,0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,0,1,1,1,1,1,1,0,1},
        {1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,0,1,1,0,1,0,0,0,0,0,0
,1},
       {1,0,0,0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,0,0,0,1,1,0,1,1,0,1,1,1,1,1},
        {1,0,1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,1,1,1,0,1,1,0,0,0,0,0
,1},
        {1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,0,0,1,1,0,1},
        {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,
1},
        {1,6,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,
1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}};
   for(i=0;i<18;i++)
   {
     for(j=0;j<36;j++)
     {
       if(maze[i][j]==5)
                              {
                                    start_x=i,start_y=j;
                                    maze[i][j]=0;
                              }
```

```
               if(maze[i][j]==6)
                         {
                              end_x=i,end_y=j;
                              maze[i][j]=0;
                         }
    }
  }
  Point startP(start_x,start_y);
  Point endP(end_x,end_y);
  stack<Point>  pointStack;
  vector<Point> vecPath;
  mazePath(maze,18,36,startP,endP,pointStack,vecPath);

  if(vecPath.empty()==true)
    cout<<"no right path"<<endl;
  else{
    cout<<"shortest path:";
    for(vector<Point>::reverse_iterator
r_iter=vecPath.rbegin();r_iter!=vecPath.rend();++r_iter){

      printf("(%d,%d) ",r_iter->row,r_iter->col);
      pathlength++;
  }
     }
     cout<<endl<<"pathlength: "<<pathlength<<endl;
  //getchar();
}
```

3 Results



```
shortest path:(1,34) (1,33) (1,32) (1,31) (1,30) (1,29) (1,28) (1,27) (1,26) (1,25) (2,25) (3,25) (3,26) (3,27) (4,27) (
5,27) (6,27) (6,26) (6,25) (6,24) (5,24) (5,23) (5,22) (5,21) (5,20) (6,20) (7,20) (8,20) (8,21) (8,22) (8,23) (8,24) (8
,25) (8,26) (8,27) (9,27) (10,27) (11,27) (12,27) (13,27) (14,27) (15,27) (15,26) (15,25) (15,24) (15,23) (15,22) (15,21
) (15,20) (15,19) (15,18) (15,17) (15,16) (15,15) (15,14) (15,13) (15,12) (15,11) (15,10) (16,10) (16,9) (16,8) (16,7) (
16,6) (16,5) (16,4) (16,3) (16,2) (16,1)
pathlength: 69
```