3. 设计一个 Las Vegas 算法，求解电路板布线问题

本例给出 Las Vegas 算法求解电路板布线问题的代码。关键步骤为选择当前点周围可行走路线时，使用随机选取的点。
至于分支限界，仅需限制随机选取的步数，达到该步数之后再采用分支限界算法计算剩余步长即可，此处没有进一步做下去。

```
#include<queue>
#include<iostream>
#include <time.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
using namespace std;
//表示方格上位置的结构体
struct Position
{
    int row;
    int col;
};

bool FindPath(Position start,Position finish,int& PathLen,Position
*&path,int **grid,int m,int n)
{
    //找到布线路径，则返回真，否则返回假
    //起点和终点相同，不用布线直接返回
    if((start.row==finish.row) && start.col==finish.col)
    {
        PathLen=0;
        return true;
    }
    //设置方向移动坐标值：右、下、左、上
    Position offset[4];
    offset[0].row=0;
    offset[0].col=1;    //右
    offset[1].row=1;
    offset[1].col=0;    //下
    offset[2].row=0;
    offset[2].col=-1;   //左
    offset[3].row=-1;
    offset[3].col=0;    //上

    //相邻的方格数
    int NumNeighBlo=4;
    Position here,nbr;
```

```
    //设置当前方格，即搜索单位
    here.row=start.row;
    here.col=start.col;

    //由于0和1用于表示方格的开放和封锁，故距离：2-0 3-1
    grid[start.row][start.col]=0; //-2 表示墙壁  -1 表示可行  -3 表示不能当作路线
    //队列式搜索，标记可达相邻方格
    queue<Position> q_FindPath;
    do
    {
        int num=0;//方格未标记个数
        Position  selectPostion[5]; //选择位置保存

        for(int i=0; i<NumNeighBlo; i++)
        {
            //达到四个方向
            nbr.row=here.row+offset[i].row;
            nbr.col=here.col+offset[i].col;
            if(grid[nbr.row][nbr.col]==-1)
            {
                //该方格未标记
                grid[nbr.row][nbr.col]=grid[here.row][here.col]+1;
                if((nbr.row==finish.row)&&(nbr.col==finish.col))
                    break;

                selectPostion[num].row=nbr.row;
                selectPostion[num].col=nbr.col;
                num++;
            }
        }

        if(num >0) //如果标记，则在这么多个未标记个数中随机选择一个位置
            //随机选一个入队
              q_FindPath.push(selectPostion[rand()%(num)]);
        //是否到达目标位置finish
        if((nbr.row==finish.row)&&(nbr.col==finish.col))
            break;
        //活结点队列是否为空
        if(q_FindPath.empty())return false; //无解
        //访问对首元素出队
        here=q_FindPath.front();
        q_FindPath.pop();
```

```
    } while(true);

    //构造布线路径
    PathLen=grid[finish.row][finish.col];
    path=new Position[PathLen]; //路径

    //从目标位置finish开始向起始位置倒推
    here=finish;
    for(int j=PathLen-1; j>=0; j--)
    {
        path[j]=here;
        //找前驱位置
        for (int i = 0; i <=NumNeighBlo; i++)
        {
            nbr.row=here.row+offset[i].row;
            nbr.col=here.col+offset[i].col;
            if(grid[nbr.row][nbr.col]==j)   //距离加2正好是前驱位置
                break;
        }
        here=nbr;
    }

    return true;
}

int main()
{
    int path_len;
    int path_len1;
    int m,n;
    Position *path;
    Position *path1;
    Position start,finish;
    Position start1,finish1;
    cout<<"在一个m*n的棋盘上，请分别输入m和n,代表行数和列数，然后输入回车
"<<endl;
    cin>>m>>n;

    //创建棋盘格
    int **grid = new int*[m+2];
    int **grid1 = new int*[m+2];
    for(int i=0; i < m+2; i++)
    {
        grid[i] = new int[n+2];
```

```
        grid1[i] = new int[n+2];
    }
    //初始化棋盘格
    for(int i=1; i <= m; i++)
    {
        for(int j=1; j <=n; j++)
        {
            grid[i][j]=-1;

        }
    }
    //设置方格阵列的围墙
    for(int i=0; i<=n+1; i++){
        grid[0][i]=grid[m+1][i]=-2;//上下的围墙
    }
    for(int i=0; i<=m+1; i++){
        grid[i][0]=grid[i][n+1]=-2;//左右的围墙

    }
    cout<<"初始化棋盘格和加围墙"<<endl;
    cout<<"--------------- ----------------"<<endl;
    for(int i=0; i < m+2; i++)
    {
        for(int j=0; j <n+2; j++)
        {
            cout<<grid[i][j]<<" ";
        }
        cout<<endl;
    }
     cout<<"--------------------------------"<<endl;


    cout<<"请输入已经占据的位置  行坐标  列坐标,代表此位置不能布线"<<endl;
    cout<<"例如输入  2 2（然后输入回车），表示坐标 2 2 不能布线;当输入的坐标为 0 0（然后输入回车） 表示结束输入"<<endl;


    //添加已经布线的棋盘格
    while(true)
    {
        int ci,cj;
        cin>>ci>>cj;
        if(ci>m||cj>n)
        {
            cout<<"输入非法！！！！！";
```

```
            cout<<"行坐标 < "<<m<<" ,列坐标< "<<n<<" 当输入的坐标为  0,0，结束输入"<<endl;
            continue;
        }else if(ci==0||cj==0){
             break;
        }else{
            grid[ci][cj]=-3;
        }

    }
    //布线前的棋盘格
    cout<<"布线前的棋盘格"<<endl;
    cout<<"--------------------------------"<<endl;
    for(int i=0; i < m+2; i++)
    {
        for(int j=0; j <n+2; j++)
        {
            cout<<grid[i][j]<<" ";
        }
        cout<<endl;
    }
     cout<<"--------------------------------"<<endl;
    cout<<"请输入起点位置坐标"<<endl;
    cin>>start.row>>start.col;
    cout<<"请输入终点位置坐标"<<endl;
    cin>>finish.row>>finish.col;

    srand( (unsigned)time( NULL ) );
    int time=0; //为假设运行次数
    // 初始值值拷贝
    start1=start;
    finish1=finish;
    path_len1=path_len;
    path1=NULL;
    for(int i=0; i < m+2; i++)
    {
        for(int j=0; j <n+2; j++)
        {
           grid1[i][j]=grid[i][j];
        }
    }

    bool result=FindPath(start1,finish1,path_len1,path1,grid1,m,n);
    while(result==0 && time < 100 ){
```

```
        // 初始值值拷贝
        start1=start;
        finish1=finish;
        path_len1=path_len;
        path1=NULL;
        for(int i=0; i < m+2; i++)
        {
            for(int j=0; j <n+2; j++)
            {
                grid1[i][j]=grid[i][j];
             }
        }

        time++;
        cout<<endl;
        cout<<"没有找到路线,第"<<time<<"次尝试"<<endl;
        result=FindPath(start1,finish1,path_len1,path1,grid1,m,n);
    }

    if(result)
    {
        cout<<"--------------------------------"<<endl;
        cout<<"$ 代表围墙"<<endl;
        cout<<"# 代表已经占据的点"<<endl;
        cout<<"* 代表布线路线"<<endl;
        cout<<"= 代表还没有布线的点"<<endl;
        cout<<"--------------------------------"<<endl;

        for(int i=0; i <= m+1; i++)
        {
            for(int j=0; j <=n+1; j++)
            {
                if(grid1[i][j]==-2)
                {
                    cout << "$ ";
                }
                else if(grid1[i][j]==-3)
                {
                    cout << "# ";
                }
                else
                {
                    int r;
                    for(r = 0; r < path_len1; r++)
```

```
                {
                    if(i==path1[r].row && j==path1[r].col)
                    {
                        cout << "* ";
                        break;
                    }
                    if(i == start1.row && j == start1.col)
                    {
                        cout << "* ";
                        break;
                    }
                }
                if(r == path_len1)
                    cout << "= ";
            }
        }
        cout << endl;
    }
    cout<<"--------------------------------"<<endl;
    cout<<"路径坐标和长度"<<endl;
    cout<<endl;
    cout<<"("<<start1.row<<","<<start1.col<<")"<<" ";
    for(int i=0; i<path_len1; i++)
    {
        cout<<"("<<path1[i].row<<","<<path1[i].col<<")"<<" ";
    }
      cout<<endl;
      cout<<endl;
      cout<<"路径长度: "<<path_len1+1<<endl;

      cout<<endl;
      time++;
      cout<<"布线完毕,查找"<<time<<"次"<<endl;
  }else
  {
      cout<<endl;
      cout<<"经过多次尝试，仍然没有找到路线"<<endl;
  }

  system("pause");
  return 0;
}
```

执行结果样例:

```
ubuntu@VM-8-5-ubuntu:~/lhc/test$ ./a.out
---------分支限界法之布线问题--------
在一个m*n的棋盘上，请分别输入m和n,代表行数和列数，然后输入回车
5 5
初始化棋盘格和加围墙
-------------- ----------------
-2 -2 -2 -2 -2 -2 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -2 -2 -2 -2 -2 -2
--------------------------------
请输入已经占据的位置  行坐标  列坐标,代表此位置不能布线
例如输入  2 2（然后输入回车），表示坐标 2 2 不能布线;当输入的坐标为 0 0（然后输入回车） 表示结束输入
3 3
3 4
3 5
0 0
布线前的棋盘格
--------------------------------
-2 -2 -2 -2 -2 -2 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -3 -3 -3 -2
-2 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -2
-2 -2 -2 -2 -2 -2 -2
--------------------------------
请输入起点位置坐标
2 3
请输入终点位置坐标
4 5

没有找到路线,第1次尝试

没有找到路线,第2次尝试

没有找到路线,第3次尝试

没有找到路线,第4次尝试

没有找到路线,第5次尝试

没有找到路线,第6次尝试

没有找到路线,第7次尝试
--------------------------------
$ 代表围墙
# 代表已经占据的点
* 代表布线路线
= 代表还没有布线的点
--------------------------------
$ $ $ $ $ $ $
$ = = = = = $
$ = * * = = $
$ * * # # # $
$ * = = * * $
$ * * * * = $
$ $ $ $ $ $ $
--------------------------------
路径坐标和长度

(2,3) (2,2) (3,2) (3,1) (4,1) (5,1) (5,2) (5,3) (5,4) (4,4) (4,5)

路径长度: 11

布线完毕,查找8次
```