拉斯维加斯算法结合分枝限界算法解决电路板布线问题

一、算法说明：

拉斯维加斯算法的一个显著特征是它所作的随机性决策有可能导致算法找不到所需的解。由于这个算法比较难懂，没有思路编写。于是就先学习--Las Vegas 算法解决 N 皇后问题，Las Vegas 解决 N 皇后问题是采用随机放置位置策略和结合分枝限界相结合。综合解决方案**电路板布线问题**采用了**机放置位置策略**和**结合分枝限界**相结合的方式来解决。关键代码如下：

* 标号①处：这里是**当前点相邻四个点**是否可以放置，如果可以放置用 selectPostion 保存下来，并用 num 记录有多少个位置可以放置。
* 标号②处：这里利用上面保存的可以放置的点，然后**随机取其中一个加入队列**。这就是 Las Vegas 的精髓！
* 标号③处：是初始化时间种子，是随机选择的关键，这个虽然简单，但是由于随机函数不了解，造成了很大伪随机。

```
srand( (unsigned)time( NULL ) );//初始化时间种子 ---------- ③
int num=0;//方格未标记个数
Position  selectPostion[5]; //选择可以到达位置保存
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;   ---------- ①

        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
     //将这个邻居入队

     q_FindPath.push(selectPostion[rand()%(num)]);  ---------- ②
```

二、结果分析

红色字表示　　输入
蓝色字表示　　输出结果
运行时间表示　算法复杂度

1）样例一：3*3 棋盘

---------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
3 3 （回车）
初始化棋盘格和加围墙
---------- ----------
-2 -2 -2 -2 -2
-2 -1 -1 -1 -2
-2 -1 -1 -1 -2
-2 -1 -1 -1 -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　表示坐标 2 2 不能布线;当输入的坐标为 0 0 表示结束输入
2 1（回车）
2 3（回车）
3 3（回车）
0 0（回车）
布线前的棋盘格

-2 -2 -2 -2 -2
-2 -1 -1 -1 -2
-2 -3 -1 -3 -2
-2 -1 -1 -3 -2
-2 -2 -2 -2 -2

请输入起点位置坐标
1 1（回车）
请输入终点位置坐标
3 1（回车）

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

$ 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

(1,1) (1,2) (2,2) (3,2) (3,1)

路径长度：5

布线完毕,查找 4 次
运行时间: 12 ms

2）样例二：5*5 棋盘

---------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
5 5（回车）
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　表示坐标 2 2 不能布线;当输入的坐标为 0 0 表示结束输入
3 1（回车）
3 2（回车）
3 4（回车）
3 5（回车）
4 5（回车）
0 0（回车）
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 -3 -3 -1 -3 -3 -2
-2 -1 -1 -1 -1 -3 -2
-2 -1 -1 -1 -1 -1 -2
-2 -2 -2 -2 -2 -2 -2

请输入起点位置坐标
1 1（回车）
请输入终点位置坐标
5 2（回车）

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

$ 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

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

路径长度：8

布线完毕，查找 7 次

运行时间：16 ms

3）样例三：10*10 棋盘

```
---------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
10  10（回车）
初始化棋盘格和加围墙
--------------- ----------------
-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -2 -2 -2 -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  表示坐标 2 2 不能布线;当输入的坐标为 0 0 表示结束输入
5 1（回车）
5 2（回车）
5 3（回车）
5 4（回车）
5 5（回车）
5 7（回车）
5 8（回车）
5 9（回车）
0 0（回车）
布线前的棋盘格
-------------------------------
-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -3 -3 -3 -3 -3 -1 -3 -3 -3 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
```

```
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2
-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
------------------------------
请输入起点位置坐标
1 1（回车）
请输入终点位置坐标
9 9（回车）

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
```

(1,1) (2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) (1,7) (1,8) (2,8) (2,9) (2,10) (3,10) (4,10) (4,9) (4,8) (4,7) (4,6) (5,6) (6,6) (7,6) (8,6) (9,6) (9,7) (9,8) (9,9)

路径长度：27

布线完毕,查找 8 次
运行时间: 31 ms

代码：

```
#include<queue>
#include<iostream>
#include <time.h>
#include<stdio.h>
#include<stdlib.h>
#include <windows.h>
#include<math.h>

using namespace std;

//表示方格上位置的结构体
struct Position
{
    int row;
    int col;
};

//分支限界算法
bool FindPath(Position start,Position finish,int& PathLen,Position *&path,int **grid,int m,int n)
{
    //找到最短布线路径，则返回真，否则返回假

    //起点和终点想同，不用布线
    if((start.row==finish.row) && start.col==finish.col)
    {
        PathLen=0;
        return true;
    }


    //设置方向移动坐标值：东、南、西、北
    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 表示强  -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++;
            }
```

```
        }

         //  printf("---------%lld\n",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
           // printf("---------%d\n",rand()%(num));
            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);

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

    //从目标位置 finish 开始向起始位置回溯
    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()
{

    cout<<"---------分支限界法之布线问题--------"<<endl;
    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;

DWORD startTime, stopTime;
startTime = GetTickCount();//程序开始时间


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);
}
stopTime = GetTickCount();//程序结束时间



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;
          printf("运行时间: %lld ms\n", stopTime - startTime);
    }else
    {
       cout<<endl;
        cout<<"经过多次尝试，仍然没有找到路线"<<endl;
    }

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