Akari 问题解题思路¯

1．首先考虑一定可以或一定不可以放灯泡的点，包括 4，0。

2．然后填放题目中可以直接判断出来的黑格子，比如 2 黑格子周围只有两个可填放的空白格子，则在这两个格子放入灯泡。

3．在此之后再处理剩余的黑格子，3,2,1，其中 3 和 1 共有 4 种情况，2 有六种情况，因此考虑首先处理 3 和 1，最后考虑 2，通过回溯解决，若当前情况不成立，则进行下一个情况，以此类推。

4．在有数字黑格子的灯都满足之后，开始寻找图中还未被点亮的格子，也是运用回溯法（详细在实现难点处）。

数据结构：

```
struct nodes{
    int type;
    int whiteAr;
    int lightAr;
    void operator=(const nodes &D){
        type=D.type;
    }
    bool operator ==(const nodes &d){
        if(type==d.type)return true;
        else return false;
    }
```

```
    bool operator !=(const nodes &d){
        if(type==d.type)return false;
        else return true;
    }
};
```

Type 存储格子类型：

-3：此格子不能放置灯泡。

-2：空格子。

-1：黑色格子。

0,1,2,3,4：1,2,3,4 黑格子。

5：灯泡。

6：被点亮。

WhiteAr：黑格子周围白格子数目。

LightAr：黑格子周围灯的数目。

```
struct black{
    int x,y;
    int type;
};
```

x,y:存储格子位置信息。

Type：存储黑色格子数字。

VecList：用于存储数字黑色格子。

函数用处：

buildList(nodes **r)通过输入的题目来将 1,2,3 黑格子的数字以及位置存在向量表里，并先给 0 格子和 4 格子放置 barrier 和灯泡。

buildR(char *t,nodes **r)构造题目。

complete(nodes **r)判断是否解决题目。

copy(nodes **a,nodes **b)复制题目。

countWhiteLightAround(nodes **r)计算 r 中黑格子周围可放置灯的白色格子和已经放置灯的数目。

canPutLight(nodes **tmp,int x,int y)判断该点能否放置灯泡。

easysolve(nodes **r)首先解决 r 中无需进行猜测的填放。

findwhite(int l,nodes **r)从给点开始遍历寻找没被点亮的点。

hardsolve(int deep,nodes **r)回溯解决需要猜测的填放。

lightup(nodes **tmp,int x,int y)在该点放置灯泡，并通过 countLightAround 和 countWhiteAround 更新黑格子信息。

isSame(nodes **a,nodes **b)判断 a，b 是否相同，辅助 easysolve 结束循环。

printR(nodes **a)输出函数。

setbarrier(nodes **tmp,int x,int y)使该点无法放置灯泡，并通过 countLightAround 和 countWhiteAround 更新黑格子信息。

hardsolve(int deep,nodes **r)通过 blackList 中剩余的黑色格子进行猜测填放。

white(int k,nodes **r)在完成了数字黑格后再寻找还未点亮的点并放置灯泡。

实现难点：

1. 如何判断回溯可以结束，最后通过 complete 函数和全局变量 finish，并在每个函数开始时判断 finish 的值来判断是否可以结束回溯。同时在每个函数内设置了 f 来判断当前设置的灯泡是否合理，不合理则返回 false 来结束这一个回溯。
2. 如何在放置灯泡之前保存上一情况 ,最后考虑直接通过 copy 函数以及建立一个 tmp 变量储存。
3. 写完测试后发现对于处理黑色数字格子 1,2 过多的题目所消耗的时间比较久，原因在与 solve 函数中进行了过多的尝试。
4. 运行的时候发现电脑多次死机，通过任务管理器发现运行时内存占用一直增加，才发现自己忘记写 delete 函数删除变量 tmp 导致函数在回溯过程中一直创建新的空间导致死机。
5. White 函数的判断回溯条件和结束回溯条件 :首先通过 find 函数找到可填放格子位置，然后判断题目是否完成，若不完成，判断是否还有格子可以填放，若没有，返回 false；反之，保存原图表，然后在 find 函数找到的格子处放置灯，然后再次运行 white 函数判断该位置填放是否正确，若正确，则找出解；反之，说明该位置不能放置灯泡，返回 false，然后恢复原图表，在 find 函数找到的格子后面继续寻找空白格子填放，直到题目解决。