DFS基本优化

重庆南开信竞入门课程

helang

搜索的进程可以看作是从树根出发，遍历一棵倒置的树——搜索树的过程。而所谓剪枝，就是通过某种判断，避免一些不必要的遍历过程，形象的说，就是剪去了搜索树中的某些“枝条”，故称剪枝。应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条应当舍弃，哪些枝条应当保留的方法。

搜索剪枝的原则:

（1）正确性

必须保证不能丢失正确的解，这是前提。通过解答必须具备的特征，必须满足的条件来考察待判断的枝条能否被剪掉。

（2）准确性

即能够尽可能多的剪去不能得到正确解的枝条。

（3）高效性

设计好剪枝判断方法后，对每一根枝条都执行一次判断。要尽量减少剪枝判断的副作用

NKOJ 3189

有一个方格迷宫，我们可以将它看作一个n*m的矩阵，每个方格表示一个房间，每个方格中都有数字。数字-1表示该房间内有陷阱，不能通过。如果格子里是>=0的数字，表示该房间中有怪兽，数字代表该怪兽的杀伤力，何老板通过该房将会失去对应数值的生命值。

20	100	50	x	x	10
10	x	50	10	10	5
10	x	70	x	10	x
20	30	20	50	20	10

一开始何老板位于左上角的方格（坐标[1,1]位置），他要走到右下角的出口（坐标[n,m]位置），每一步何老板可以往上、下、左、右走。他想知道最少需要失去多少生命值就可以走出迷宫？

n,m<=30

```
void dfs(int x,int y,int tot)//当前走到(x,y),当前已耗费的生命值为tot
{
    if((x==n)&&(y==m))
    {
        if(tot<ans)ans=tot;
        return;
    }
    visit[x][y]=true;    //标记当前点是否已被讨论过
   if(y+1<=m && Map[x][y+1]!=-1 && !visit[x][y+1]) dfs(x,y+1,tot+Map[x][y+1]);
   if(x+1<=n && Map[x+1][y]!=-1 && !visit[x+1][y]) dfs(x+1,y,tot+Map[x+1][y]);
   if(y-1>=1 && Map[x][y-1]!=-1 && !visit[x][y-1]) dfs(x,y-1,tot+Map[x][y-1]);
   if(x-1>=1 && Map[x-1][y]!=-1 && !visit[x-1][y]) dfs(x-1,y,tot+Map[x-1][y]);
    visit[x][y]=false;   //想一想，为何要恢复现场?
}
```

考虑：怎么优化上面的搜索代码?

20	100	50	x	x	10
10	x	50	10	10	5
10	x	70	x	10	x
20	30	20	50	20	10

伪代码：`int Min=min{ map[x][y] }`,即Min记录所有怪兽中，杀伤力最小的

此处`Min=5;`

```
void dfs(int x,int y,int tot)
{
    if(tot>=ans)return;
    if((x==n)&&(y==m))
    {
        if(tot<ans)ans=tot;
        return;
    }

    if(tot+(n-x+m-y)*Min>=ans)return;
```

//优化1：可行性剪枝

如果当前搜索已经无法产生比之前最优解更优的解时，可以提前回溯。
比如：如果当前生命耗费值tot，它比之前已求出的答案ans更劣，则沿着当前方案继续搜下去也不会有更优的解产生，则结束当前搜索

//优化2：最优化剪枝

如果沿着方案搜索，剩下的搜索步骤全部以最理想的方式发展，都无法得到更优的解，则提前回溯。
比如：当前走到(x,y),则距离终点，剩下的步数最少为n-x+m-y步，
假设这些步上全是杀伤力最小的怪兽，
在此最优情况下，也没有之前求出的答案优，则结束当前搜索

```
    visit[x][y]=true;
    if(y+1<=m && Map[x][y+1]!=-1 && !visit[x][y+1]) dfs(x,y+1,tot+Map[x][y+1]);
    if(x+1<=n && Map[x+1][y]!=-1 && !visit[x+1][y])  dfs(x+1,y,tot+Map[x+1][y]);
    if(y-1>=1 && Map[x][y-1]!=-1 && !visit[x][y-1])     dfs(x,y-1,tot+Map[x][y-1]);
    if(x-1>=1 && Map[x-1][y]!=-1 && !visit[x-1][y])     dfs(x-1,y,tot+Map[x-1][y]);
    visit[x][y]=false;
}
```

```
void dfs(int x,int y,int tot)
{
    if(tot<Cost[x][y])Cost[x][y]=tot;   //优化3:记忆化
    else return;
```

记录下之前已搜索过的结果。再次搜到同一点时，若当前方案更优，则更新，否则结束当前搜索
比如Cost[x][y]记录到达(x,y)所需最少耗费值。若再次搜到(x,y),目前的耗费值tot没有之前的优，则结束当前搜索，否则更新Cost[x][y]的值。

```
    if(tot>=ans)return;                 //优化1：可行性剪枝
    if((x==n)&&(y==m))
    {
        if(tot<ans)ans=tot;
        return;
    }
```

如果当前搜索已经无法产生比之前最优解更优的解时，可以提前回溯。
比如：如果当前生命耗费值tot比之前已求出的答案ans更劣，则沿着当前方案继续搜下去也不会有更优的解产生，则结束当前搜索

```
    if(tot+(n-x+m-y)*Min>=ans)return;   //优化2：最优化剪枝
```

如果沿着方案搜索，剩下的搜索步骤全部以最理想的方式发展，都无法得到更优的解，则提前回溯。
比如：当前走到(x,y),则距离终点，剩下的步数最少为n-x+m-y步，
假设这些步上全是杀伤力最小的怪兽，
在此最优情况下，也没有之前求出的答案优，则结束当前搜索

```
    visit[x][y]=true;
    if(y+1<=m && Map[x][y+1]!=-1 && !visit[x][y+1])  dfs(x,y+1,tot+Map[x][y+1]);
    if(x+1<=n && Map[x+1][y]!=-1 && !visit[x+1][y])  dfs(x+1,y,tot+Map[x+1][y]);
    if(y-1>=1 && Map[x][y-1]!=-1 && !visit[x][y-1])  dfs(x,y-1,tot+Map[x][y-1]);
    if(x-1>=1 && Map[x-1][y]!=-1 && !visit[x-1][y])  dfs(x-1,y,tot+Map[x-1][y]);
    visit[x][y]=false;
}
```

NKOJ 3189 加强一下

有一个方格迷宫，我们可以将它看作一个n*m的矩阵，每个方格表示一个房间，每个方格中都有数字。数字-1表示该房间内有陷阱，不能通过。如果格子里是>=0的数字，表示该房间中有怪兽，数字代表该怪兽的杀伤力，何老板通过该房将会失去对应数值的生命值。

20	100	50	x	x	10
10	x	50	10	10	5
10	x	70	x	10	x
20	30	20	50	20	10

一开始何老板位于左上角的方格(坐标[1,1]位置)，他要走到右下角的出口(坐标[n,m]位置)，每一步何老板可以往上、下、左、右走。他想知道最少需要失去多少生命值就可以走出迷宫？

n, m<=1000

```
void dfs(int x,int y,int tot)
{   cnt++; if(cnt>100000)return;          //优化4:卡时   类似蒙特卡罗算法
    if(tot<Cost[x][y])Cost[x][y]=tot;     //优化3:记忆化
    else return;
```

记录下之前已搜索过的结果。再次搜到同一点时，若当前方案更优，则更新，否则结束当前搜索
比如Cost[x][y]记录到达(x,y)所需最少耗费值。若再次搜到(x,y),目前的耗费值tot没有之前的优，则结束当前搜索，否则更新Cost[x][y]的值。

```
    if(tot>=ans)return;                   //优化1：可行性剪枝
    if((x==n)&&(y==m))
    {
        if(tot<ans)ans=tot;
        return;
    }
```

如果当前搜索已经无法产生比之前最优解更优的解时，可以提前回溯。
比如：如果当前生命耗费值tot比之前已求出的答案ans更劣，则沿着当前方案继续搜下去也不会有更优的解产生，则结束当前搜索

```
    if(tot+(n-x+m-y)*Min>=ans)return;     //优化2：最优化剪枝
```

如果沿着方案搜索，剩下的搜索步骤全部以最理想的方式发展，都无法得到更优的解，则提前回溯。
比如：当前走到(x,y),则距离终点，剩下的步数最少为n-x+m-y步，
假设这些步上全是杀伤力最小的怪兽，
在此最优情况下，也没有之前求出的答案优，则结束当前搜索

```
    visit[x][y]=true;
    if(y+1<=m && Map[x][y+1]!=-1 && !visit[x][y+1])  dfs(x,y+1,tot+Map[x][y+1]);
    if(x+1<=n && Map[x+1][y]!=-1 && !visit[x+1][y])  dfs(x+1,y,tot+Map[x+1][y]);
    if(y-1>=1 && Map[x][y-1]!=-1 && !visit[x][y-1])  dfs(x,y-1,tot+Map[x][y-1]);
    if(x-1>=1 && Map[x-1][y]!=-1 && !visit[x-1][y])  dfs(x-1,y,tot+Map[x-1][y]);
    visit[x][y]=false;
}
```

搜索的基本优化

- 1.可行性剪枝
- 2.最优化剪枝
- 3.记忆化搜索
- 4.卡时