导入

OI界流传着这样一句话：

“贪心过样例，暴力出奇迹，骗分最神奇，打表进省一”

信息学

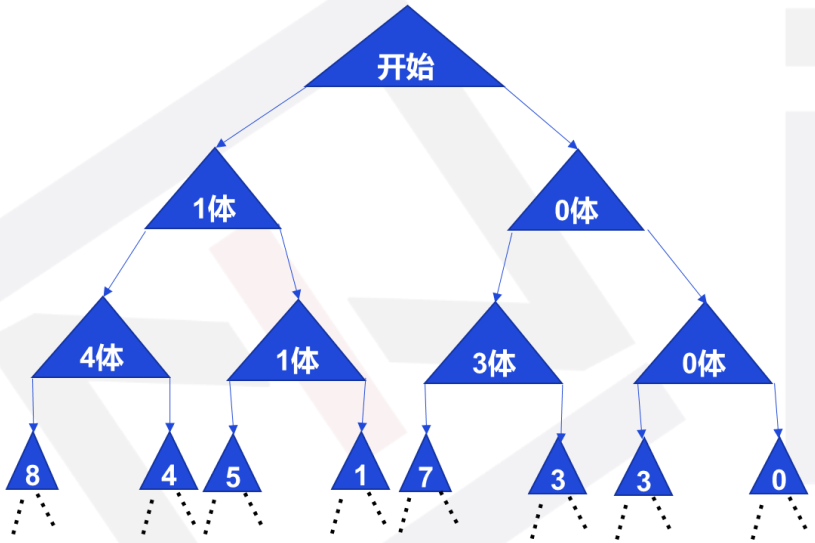
DFS剪枝

西南大学附属中学校

信息奥赛教练组

常见的剪枝方向

可行性剪枝 判断继续搜索是否能得出答案，如果不能，就直接回溯。



一共只要求放n个，超过n的都被return

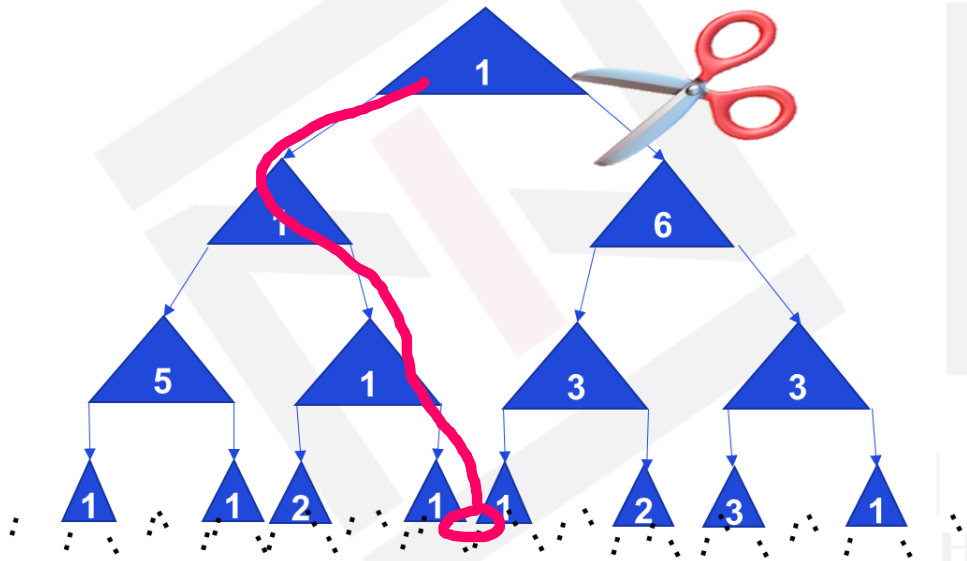
代码例子

```
void dfs(int k){
    if(k>n)  return;
    ....
    return;
}
```

剪枝的几个思路

最优性剪枝

又称为上下界剪枝。
如果当前结点已经无法产生比当前最优解更优的解时，可以提前回溯。



代码例子

```
void dfs(int k){
    if(当前的答案没有已经搜到的答案更优)  return;
    ....
    return;
}
```

进阶一点：
我们还可以利用题目所给条件，
估计出此时条件下答案的‘上下界’，
将它与已经推出的答案相比，
如果不比当前答案小，也可以剪掉。

常见的剪枝方向

记忆化搜索　对于部分重复计算的结果，搜完后，直接存储到数组中方便下次再搜索

F[]		
F[1]=1	F[2]=3	F[3]=6
F[4]=10	未计算	未计算

假如已经算出了f(4)的值

在计算f(5)=f(4)+5时，直接将f(4)的值返回

代码例子

```
int dfs(int k){
    if(f[k]已经计算)  return f[k];
    ....
    return;
}
```

【记忆化例子：**function \ 滑雪**】

常见的剪枝方向

搜索顺序

在网格类的搜索题目中。
对搜索方向有一定的偏好

A			#		B
		#			
#				#	
		#			

如果从A到B

先搜
右上和右侧
的时间

<

先搜
左上和左侧
的时间

搜索玄学：“从左边搜会T，从右边搜就A了”

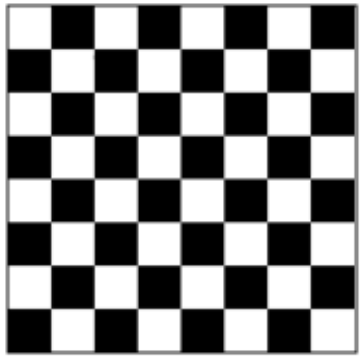
一般来说，搜索时尽量从已知信息较多的地方开始搜，效率会更高

常见的剪枝方向

搜索顺序 、奇偶性剪枝

在网格类的搜索题目中。
对搜索步数的判断

有一个n×m大小的迷宫。其中字符S表示起点，字符D表示出口，字符X表示墙壁，字符.表示平地。你需要从S出发走到D，每次只能向上下左右相邻的位置移动，并且不能走出地图，也不能走进墙壁。每次移动消耗1时间，走过路都会塌陷，因此不能走回头路或者原地不动。现在已知出口的大门会在T时间打开，判断在0时间从起点出发能否逃离迷宫。数据范围n,m≤10,T≤50。



我们记每个格子的行数和列数之和x，如果x为偶数，那么格子就是白色，反之奇数时为黑色。

会发现：

每一走一次肯定会移动到不同颜色的格子上
结论是：走奇数步会改变颜色，走偶数步颜色不变
那么如果起点和终点的颜色一样，而T是奇数的话，就不可能逃离迷宫。
同理，如果起点和终点的颜色不一样，而T是偶数的话，也不能逃离迷宫。
遇到这两种情况时，就不用进行DFS了，直接输出"NO"。

一般题目会问能否在T步到达终点

当T和最短距离D奇偶性不一致的时候，都可以直接判断不可达！

代码例子

```
int main(){
    if(D与T奇偶性不一致)  不可达;
    else dfs(…)
    return 0;
}
```

剪枝就是利用判断来去除一些无效的条件

合理的剪枝可以极大的优化搜索效率

剪枝的原则

正确性

就算当前的剪枝能剪掉大部分无效状态，也不能将带有最优答案分支剪掉

准确性

根据具体问题具体分析,采用合适的判断手段,使不包含最优解的枝条尽可能多的被剪去,以达到程序“最优化”的目的.

高效性

倘若一个剪枝的判断效果非常好,但是它却需要耗费大量的时间来判断、比较,结果整个程序运行起来也跟没有优化过的没什么区别，这样的剪枝没有什么必要。

例1：工作分配问题

设有n件工作分配给n个人。将工作i分配给第j个人所需的费用为c[i][j]。试设计一个算法，为每一个人都分配一件不同的工作，并使总费用达到最小。

设计一个算法，对于给定的工作费用，计算最佳工作分配方案，使总费用达到最小。

样例输入
3
4 2 5
2 3 6
3 4 5
样例输出
9

输入
第一行有1个正整数n (1≤n≤20)。接下来的n行，每行n个数，第i行表示第i个人各项工作费用。

输出
将计算出的最小总费用输出

全排列问题

dfs(k):表示分配到了第k件工作

状态数：n个人，n

边界条件：　k>n　更新最小值

核心代码


```
void dfs(int k){
    if(k>n){
        if(sum<minx) minx=sum;
        return ;
    }
    for(int i=1;i<=n;i++){
        if(!f[i]){
            f[i]=1;
            sum+=a[k][i];
            dfs(k+1);
            sum-=a[k][i];
            f[i]=0;
        }
    }
}
```

时间超限 18%

剪枝：如果当前的sum已经大于等于当前搜索到的最小值，就没有必要搜索下去了

最优性剪枝

核心代码


```
void dfs(int k){
    if(sum>=minx) return ;  //最优性剪枝
    if(k==n+1){
        if(sum<minx) minx=sum;
        return ;
    }
    for(int i=1;i<=n;i++){
        if(!f[i]){
            f[i]=1;
            sum+=a[k][i];
            dfs(k+1);
            sum-=a[k][i];
            f[i]=0;
        }
    }
}
```

剪枝：如果当前的sum已经大于等于当前搜索到的最小值，就没有必要搜索下去了

最优性剪枝

例2：数的划分

将整数n分成k份，且每份不能为空，任意两种划分方案不能相同(不考虑顺序)。
例如：n=7，k=3，下面三种划分方案被认为是相同的。
7=1+1+5
7=1+5+1
7=5+1+1
问有多少种不同的分法。

输入：
n，k(1< =n <= 200，1<= k <= 6)

输出：
一个整数，即不同的分法

样例输入
7 3
样例输出
4

分析

全排列问题

dfs(k):表示分配到了第k个位置

状态数：n

dfs(x,k):表示前一个分配的数为x，分配到了第k个位置

状态数：x~n

题目说明：

例如：n=7，k=3，下面三种划分方案被认为是相同的。

7=1+1+5

7=1+5+1

7=5+1+1

保证后一个数≥前一个数即可 （下界）

那么$a[i]$的下界为$a[i-1]$

假如我们已经分好了$a[1]-a[i-1]$，还剩下要分的$m=n-\sum_{x=1}^{i-1}a_x$,我们还剩下要分的份数为$k-(i-1)=k-i+1$,那么$a[i]$最大为剩下数和的平均数，即$a[i]<=(m/(k-i+1))$

```
dfs(x,k,t) //从1开始，要分的份数，还剩多少数
```

下界为x，上界为t/k

核心代码


```
void dfs(int x,int k,int t) { //从1开始，要分的份数，还剩的数的总和
        if(k==1){
                sum++;
                return;
        }
         for(int i=x;i<=t/k;i++)//剪枝，下界是上一个划分的数,上界是剩下总和的平均值
             dfs(i,k-1,t-i);
}
```

例3：function

对于一个递归函数w(a,b,c)

（1）如果a≤0 or b≤0 or c≤0就返回值1；

（2）如果a>20 or b>20 or c>20就返回w(20,20,20)；

（3）如果a<b并且b<c 就返回w(a,b,c-1)+w(a,b-1,c-1)-w(a,b-1,c)

（4）其它的情况就返回w(a-1,b,c)+w(a-1,b-1,c)+w(a-1,b,c-1)-w(a-1,b-1,c-1)

如果出现w(30,-1,0)既满足条件1又满足条件2，这种时候我们就按最上面的条件来算，所以答案为1。

输入
若干行，每行三个数a，b，c，当a=b=c=-1时结束。
保证输入的数在long long范围内，且为整数。

输出
对于每一组有效的a，b，c，输出w(a,b,c)的值ans。
格式为w(a,_b,_c)_=_ans，　‘_’为空格。

样例输入
1 1 1
2 2 2
-1 -1 -1
样例输出
w(1, 1, 1) = 2
w(2, 2, 2) = 4

根据题意，不难写出这样的代码：

```
long long w(long long a,long long b,long long c){
    if(a<=0||b<=0||c<=0) return 1;
    else if(a>20||b>20||c>20) return w(20,20,20);
    else if(a<b&&b<c) return w(a,b,c-1)+w(a,b-1,c-1)-w(a,b-1,c);
    else return w(a-1,b,c)+w(a-1,b-1,c)+w(a-1,b,c-1)-w(a-1,b-1,c-1);
}
```

递归的状态有大量的重合计算，不妨用一个数组将计算的结果保存

```
long long w(long long a,long long b,long long c){
    if(a<=0||b<=0||c<=0) return 1;
    else if(a>20||b>20||c>20){
        if(f[a][b][c]==-INF) f[a][b][c]=w(20,20,20);
        return f[a][b][c];
    }
    else if(a<b&&b<c){
        if(f[a][b][c-1]==-INF)   f[a][b][c-1]=w(a,b,c-1);
        if(f[a][b-1][c-1]==-INF) f[a][b-1][c-1]=w(a,b-1,c-1);
        if(f[a][b-1][c]==-INF)   f[a][b-1][c]=w(a,b-1,c);
        return f[a][b][c-1]+f[a][b-1][c-1]-f[a][b-1][c];
    }
    else{
        if (f[a-1][b][c]==-INF)     f[a-1][b][c] = w(a-1,b,c);
        if (f[a-1][b-1][c]==-INF)   f[a-1][b-1][c] = w(a-1,b-1,c);
        if (f[a-1][b][c-1]==-INF)   f[a-1][b][c-1] = w(a-1,b,c-1);
        if (f[a-1][b-1][c-1]==-INF) f[a-1][b-1][c-1] = w(a-1,b-1,c-1);
        return f[a-1][b][c]+f[a-1][b-1][c]+f[a-1][b][c-1]-f[a-1][b-1][c-1];
    }
}
```

f数组要初始化，用初始化的值代表未计算

例3：逃离迷宫

有一个n * m大小的迷宫。其中字符‘ S’表示起点，字符‘ D’表示出口，字符‘ X’表示墙壁，字符‘ .'表示平地。你需要从‘ S’走到 ‘D’,每次只能向上下左右相邻的位置移动，并且不能走出地图，也不能走进墙壁
每次移动消耗1时间，走过路都会塌陷，因此不能走回头路或原地不动。现在已知出口的大门会在T时间打开，判断在0时间从起点出发能否逃离迷宫。
数据范围n<=20, m <= 50, T <= 1000

输入
第一行3个数n,m,T

接下来是n*m的矩阵

输出
若能逃离输出YES，否则输出NO

样例输入
4 4 5
S.X.
..X.
..XD
....
样例输出
NO

这道题与棋盘、马的遍历类似
为了去除一些无效状态，可以根据步数的奇偶性剪枝

```
if ((sx + sy + ex + ey + T) % 2 != 0) {  //奇偶性剪枝
        cout << "NO" << endl;
}
   else {
       dfs(sx, sy, 0);
}
```

剪枝只是一个技巧，没固定的写法，需要自己练习、总结

有些题目可能需要4~5种剪枝，例：小木棍

Thanks

For Your Watching

小木棍剪枝参考

(1).优化搜索顺序：从大的木棍长度开始填能更快的逼近答案，也便于剪枝，所以我们要在搜索前先排序。

(2).排除等效冗余：限制先后加入一根切碎后的木棒的长度是递减的，因为先加入x后加入y和先加入y后加入x是一样的，只需其中一种即可。

(3).排除等效冗余：对于当前要填的木棍，如果前面搜的那个木棍与它长度相同，且前面那个木棍没有填，则这个木棍也没必要搜。如：3221，搜到第二个2，且第一个2也没找到填的位置，这时就可以跳过这个2了。

。

(4).排除等效冗余：如果在当前原始木棍中添加一根新木棍并恰好拼接完整，并且接下来的儿子分支递归失败，那么直接判定当前分支失败，原理类似于远远地望去是一个死胡同就立刻回头。

(5).排除等效冗余：如果长度为5的木棍已经填过，则没必要试2+3的可能了

(6).最开始，可以判断一下所有木棍的总和是否能整除x，来决定要不要搜索x