搜索剪枝（一）

一、剪枝

在很多情况下，我们已经找到了一组比较好的解。但是计算机仍然会义无返顾地去搜索比它更“劣”的其他解，搜索到后也只能回溯。为了避免出现这种情况，我们需要灵活地去定制回溯搜索的边界

常用剪枝：

1、优化搜索顺序
2、排成等效冗余
3、可行性剪枝
4、最优性剪枝
5、记忆化搜索

剪枝注意：

1、正确性保证
2、尽可能挖掘剪枝条件
3、高效原则
4、边界处理

二、例题

【例一】生日蛋糕 H8002

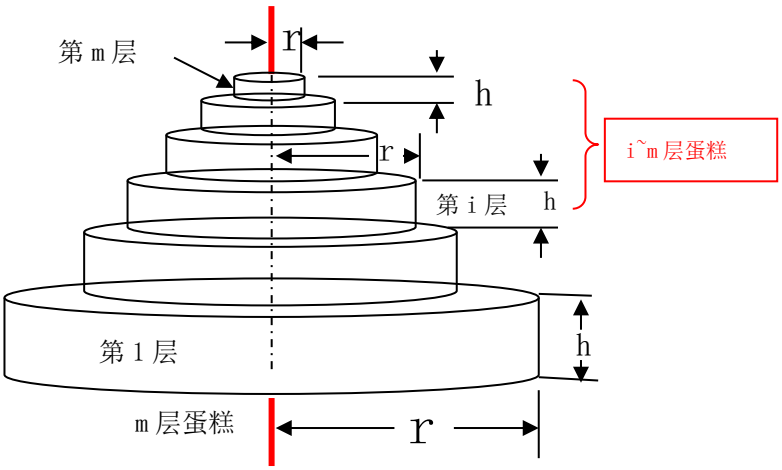
体积为 n 的 m 层的生日蛋糕，每层都是圆柱体，从下往上数第 i 层蛋糕的半径是 r_i，高度是 h_i。当 i<m 时，有 $r_i > r_{i+1}$，$h_i > h_{i+1}$。求表面积最小值 Q=πs （最下层的下底面积除外）

输入（两行）：

第一行 n(n<=10000)
第二行 m(m<=20)

输出 1 个正整数 s

(s=0 为无解) ：



【分析】

圆柱体公式： V＝r*r*h
侧面积：A1＝2*r*h
底面积： A2＝r*r
因 r,h 均为正整数
且有 $r_i > r_{i+1}$ $h_i > h_{i+1}$

2. 设定 i 层的最大蛋糕：

设 i 层的最大蛋糕的底层高度为 H，则各层高度为： H， H-1， H-2 ， …， H-(i-1)
设 i 层的最大蛋糕的底层半径为 R，则各层半径为： R， R -1， R-2， … ， R-(i-1)
则这 i 层蛋糕的最大体积为：$V_{maxi}=$ $\sum_{0 \le k \le i-1} (R-k)*(R-k)*(H-k)$

还需要的面积 S=2*ri*hi+2*ri-1*hi-1...>=2*ri*hi* ri/r + 2*ri-1*hi-1*ri-1/r.....=2*dv/r

dv 表示还剩体积，r 表示当前半径，因为递减所以 ri/r<1 最少需要面积 s=2*（n-v)/r.

3. 至下而上地计算各层蛋糕

⑴当前状态（i，vi，hi-1，ri-1，si-1）
i:即将做的层数，初值 i=1(做底层)
vi:还余体积，初值 vi=n
hi-1:前一层的高度
ri-1:前一层的半径
si-1:已产生的表面积，s 的初值为 0
蛋糕的表面积为：

s 外表= r1* r1 + Σ 2* rk *hk 1≤ k ≤i

⑵边界条件 i=m+1 蛋糕已做完
⑶目标状态 vi=0 刚好用完原料 记下此时的最优解
⑷搜索范围 ri ：m-i+1(即 rm≧1)～ri-1-1
hi： m-i+1 (即 hm≧1) ～ min{ hi-1-1,(vi-vmin)/ri2}
枚举 ri 和 hi 求出 si=si-1+2*ri*hi

4、剪枝：

1、v-r*r*h< V_{i+1} 剩下的体积 v-r*r*h 不能造出 i+1 到 m 层的最小蛋糕（**可行性剪枝**，前边用料太多）

2、 s+2*(n-v)/r>best 剩下的体积能构成的最少面积加上已有的面积大于了最优，剪枝（**最优化剪枝**，前边用料太少）

3、 s+minS_{i+1}>best 在 s 的基础上做最小的蛋糕所有面积大于前面已做出的合理蛋糕的最小面积 best （**最优化剪枝**，即此种方案做出的表面积更大）

```
const int inf=1e9;
int n,best,m;
int mV[30]={},mS[30]={},maxV[50];
void dfs(int i,int v,int h,int r,int s){
    //i表示到第几层，v表示已经用的体积，h，r表示当前层的高度 半径，
    //s表示已经产生的面积
    if(i==0){
        if(v==n&&s<best)best=s;
        return;
    }
    if(v+mV[i-1]>n)return ;//
    //1已经搜索过的体积加上还未搜索过的最小体积不能比总体积n大
    if(s+mS[i-1]>best)return ;
    //已经搜索过的表面积加上还未搜索过的最小表面积不能比之前的最小总表面积best 大
    if(s+2*(n-v)/r>=best)return;
    //n-v是还剩的体积，还需要的最小表面积是2*（n-v)/r
    for(int j=r-1;j>=i;j--){
        if(i==m) s=j*j;//如果是最底层的面积是初始值
        int mxh=min(h-1, (n-v-mV[i-1])/(j*j));
        //最大高度
        for(int k=mxh;k>=i;k--)
            dfs(i-1,v+j*j*k ,k,j,s+2*j*k);
    }
}
```

```
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=20;i++){
        mV[i]=mV[i-1]+i*i*i;//最小体积
        mS[i]=mS[i-1]+2*i*i;//最小外表面

    }
    best=inf;
    dfs(m,0,n+1,n+1,0);
    if(best==inf)cout<<0;
    else
        cout<<best;
    return 0;
}
```

【例二】折半搜索：P1024 送礼物

作为惩罚，GY 被遣送去帮助某神牛给女生送礼物(GY：貌似是个好差事)但是在 GY 看到礼物之后，他就不这么认为了。某神牛有 N 个礼物，且异常沉重，但是 GY 的力气也异常的大(-_-b)，他一次可以搬动重量和在 w(w<=2^31-1)以下的任意多个物品。GY 希望一次搬掉尽量重的一些物品，请你告诉他在他的力气范围内一次性能搬动的最大重量是多少。

【解析】

根据题意，题目是一个大背包问题，即“子集和”问题，不过因为体积过大，无法用背包 dp 方法求解，这类问题还有直接求法，用搜索直接解决，效率是 2^N，搜索过程中可以适当用最优性剪枝，不过题目 N=45,复杂度还是过高。

这个时候可以采取折半搜索的思想。

先搜从前一半的礼物搜索选出若干个，可能达到的 0~W 的所以可能值，放入数组 A，然后排序

然后第二次搜索，在后一半中搜索出可能的值 T，在数组 A 中二分查找 W-T 的下界，用二者和更新答案。

同时还可以加入剪枝：

1、**可行性剪枝**：第二次搜索时，如果当前重量 T 加上 A 数组中最小的已经大于 W，剪枝

2、**优化搜索顺序**：将物品降序后再分 2 半搜索

```
inline int cmp(int a,int b){return a>b;}
inline void dfs1(int l,int r,ll now){
    if(now>w)return;
    f[tp1++]=now;MIN=min(now,MIN);
    for(int i=l;i<=r;i++)dfs1(i+1,r,(ll)now+a[i]);
}
void dfs2(int idx,int last,ll now){
    if(now+MIN>w)return;
    if(idx>last){
        int tmp=w-now;//最大需求
        int pos=lower_bound(f,f+tp1,tmp)-f;//
        if(pos==0&&f[pos]>tmp)return;//
        if(pos==0&&f[pos]==tmp){
            ans=w;return;
        }
        ans=max(ans,f[pos-1]+now);
        return ;
    }
    dfs2(idx+1,last,now);//不选
    dfs2(idx+1,last,now+a[idx]);
}

int main(){
    w=in;n=in;
    for(int i=1;i<=n;i++)a[i]=in;
    sort(a+1,a+n+1,cmp);
    int mid=n/2;
    dfs1(1,mid,0);
    sort(f,f+tp1);
    dfs2(mid+1,n,0);
    cout<<ans;
```

双向搜索

对于一些问题具备“初始状态”和“终止状态”，并且从初态搜索和终态搜索都可以产生整个状态空间，这样的情况下可以采取双向搜索：

从初态和终态同时出发各搜一般的状态，在中间交会组合出答案。

【例三】双向搜索 ：1893 字串变换

分析：

本题明确初始状态，最终状态，可以双向搜索，掌握好字符串与 STL 是解决本题的制胜法宝

```
struct Node{
    string  s;int dep;
};
string a[50],b[50];
int n=0;
map<string ,int >f1,f2;
queue<Node>q1,q2;
Node t1,t2;
string ss,s1,s2;;
```

```
void double_bfs(){
    q1.push((Node){s1,0});q2.push((Node){s2,0});
    f1[s1]=0;f2[s2]=0;
    while(!q1.empty()&&!q2.empty()){
        if(q1.front().dep+q2.front().dep>10){
            printf("NO ANSWER!\n");return;
        }
        t1=q1.front();q1.pop();
        int len=t1.s.size();
        for(int i=0;i<len;i++)//枚举每个字串
            for(int j=0;j<n;j++){
                int tmp=a[j].size();
                ss=t1.s;
                if(i+tmp-1>=len)continue;//不够这个规则的长度
                if(ss.substr(i,tmp)!=a[j])continue;//子串与规则不同
                if(f1.count(ss.replace(i,tmp,b[j])))continue;//该字符串出现过
                t2.s=ss;
                t2.dep=t1.dep+1;//下个状态
                f1[t2.s]=t2.dep;//标记
                q1.push(t2);//入队
                if(f2.count(t2.s)){
                    printf("%d\n",t2.dep+f2[t2.s]);return;//交会
                }
            }
        t1=q2.front();q2.pop();
        len=t1.s.size();
        for(int i=0;i<len;i++)
            for(int j=0;j<n;j++){
                int tmp=b[j].size();
                ss=t1.s;
                if(i+tmp-1>=len)continue;
                if(ss.substr(i,tmp)!=b[j])continue;
                if(f2.count(ss.replace(i,tmp,a[j])))continue;
                t2.s=ss;
                t2.dep=t1.dep+1;
                f2[t2.s]=t2.dep;
                q2.push(t2);
                if(f1.count(t2.s)){
                    printf("%d\n",t2.dep+f1[t2.s]);return;
                }
            }
    }
    printf("NO ANSWER!\n");
}
```

```
int main(){
    cin>>s1>>s2;
    while(cin>>a[n]>>b[n])n++;
    double_bfs();
    return 0;
}
```

【习题一】 cost 数

给你一个有 n 个正整数的数列{an}。一个正整数 x 若满足在数列{an}中存在一个正整数 ai，使 x≡17（mod ai），那么 x 就是一个‘cost 数’。请问 1 到 m 的正整数中，有多少个‘cost 数’

【分析】

将式子转化一下：

x≡17（mod ai）=> x mod ai =17 mod ai => (x-17) mod ai=0（注意已保证 ai >17，所以 17 mod ai =17），于是**问题变为求 0 到 m-17 的整数中至少能被{an}中一个数整除的有多少个。**

利用 n 个集合的容斥原理 优化

这时我们需要用个 dfs 枚举每个 ai 选与不选，若选出 k 个数，且 k 为奇数，则 ans 加上 m/LCM(选出的数)，若为偶数 ans 减去 m/LCM(选出的数)。当然，还要加上优化，包括可行性剪枝（选出的数的 LCM 要小于等于 m。由于 ai >17，该剪枝十分有效）、优化搜索顺序（先搜大数）。时间复杂度：O(2^nlogm)（实际小得多）。

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  #define LL long long
4.  LL a[32];LL n,m,ans=0;
5.  inline LL gcd(LL a,LL b){return (!b)?a:gcd(b,a%b);}
6.  inline LL lcm(LL a,LL b){return a/gcd(a,b)*b;}
7.  bool cmp(LL a,LL b){return a>b;}
8.  void dfs(LL tmp,LL s,LL dep){
9.      if(tmp==n+1||s>m)return;
10.     LL w=lcm(s,a[tmp]);
11.     if(dep%2)ans+=m/w;
12.     else ans-=m/w;
13.     dfs(tmp+1,s,dep);
14.     dfs(tmp+1,w,dep+1);
15. }
16. int main(){
17.     freopen("count.in","r",stdin);freopen("count.out","w",stdout);
18.     cin>>n>>m;
19.     if(m<17){cout<<0;return 0;}
20.     else m-=17;
21.     for(LL i=1;i<=n;i++)cin>>a[i];
22.     sort(a+1,a+n+1,cmp);
23.     dfs(1,1,1);
```

```
    cout<<ans+1;
    return 0;
}
```

【练习二】P1032 木棒分组

有 n 根木棒，你需要将它们分成尽量多的组， 满足每组木棒长度和相等。你不能 将木棒砍断。 输出最优方案下一组中木棒长度之和。

【分析】

70 分：直接搜索

100 分：剪枝

1、长度的范围明显在 max{ai}~sum 之间

2、长度从大到小开始搜，长度小的比较灵活，到后面容易拼凑（深度大的状态多）

3、在某组搜索中长度一样的不用重复枚举，因为排序过，之前枚举不合适现在选择也不会合适。

4、重要剪枝：对于某个目标 Len，在每次构建新的长度为 Len 的原始棒时，检查新棒的第一根 stick[i]，若在搜索完所有 stick[]后都无法组合，则说明 stick[i]无法在当前组合方式下组合，不用往下搜索(往下搜索会令 stick[i]被舍弃)，直接返回上一层

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <functional>
using std::sort;

int a[99];
bool v[99];
int len, sum, n, i;

bool dfs(int p, int l, int s) {
    //p 表示当前枚举木棍，l 表示当前产生的长度，s 表示已经用过的长度
    if ( l>len ) return false;//剪枝
    if ( s==sum ) return true;
    if ( l==len ) {//重新从最大的开始搜索一组
        if ( s+len==sum ) return true;
        for (p=n; p && v[p] ;--p) ;//找到最大的没使用的木棍
        v[p] = true;
        if ( dfs(p-1, a[p], s+len) ) return true;
        v[p] = false;//回溯
        return false;//第一根都不合适，后续不在搜
    }
    for (; p ;--p)
        if ( !v[p] && (a[p]!=a[p+1] || v[p+1]) ) {
            //长度一样的不在搜索
        v[p] = true;
        if ( dfs(p-1, l+a[p], s) ) return true;
```

```
28.         v[p] = false;
29.     }
30.     return false;
31. }
32.
33. int main() {
34.     scanf("%d", &n);
35.     len = 0; sum = 0;
36.     for (i=1; i<=n ;++i) {
37.         scanf("%d", a+i);
38.         sum += a[i];
39.         len=max(a[i],len);
40.     }
41.
42.     sort(a+1, a+1+n);
43.     memset(v, 0, sizeof v); a[n+1] = 0;
44.     for (; len<=sum ;++len)
45.     if ( sum%len==0 ) {
46.         memset(v, 0, sizeof v);
47.         v[n] = true;
48.         if ( dfs(n-1, a[n], 0) ) {printf("%d\n", len); break;}
49.     }
50. }
```

【推荐试题】

P1033