深搜的剪枝

信智学院　陈琰宏

教学内容

01 DFS 顺序剪枝 2773 3041 6348

02 DFS 剪枝优化 6336 1209 3652 3651

03 双向DFS 6401 1124 1128

1.1 剪枝

搜索算法的时间复杂度大多是**指数级**的，难以满足对程序运行时间的限制要求，为使降低时间复杂度，对深度优先搜索可以进行一种优化的基本方法——剪枝。

搜索的进程可以看做是从树根出发，遍历一颗倒置树（搜索树）的过程，所谓剪枝，就是**通过某些判断，避免一些不必要的遍历过程**，形象的说，就是减去搜索树中的某些枝条。

显而易见，应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条舍弃哪些枝条保留，设计出好的剪枝判断方法，可以使得程序运行时间大大缩短，否则会适得其反。

剪枝的原则：正确、准确、高效

1.1 剪枝

- **原则之一：正确性**

剪枝能优化程序的执行效率，是因为它“剪去”搜索树中的一些“枝条”。但如果在剪枝时，将所需要的解的枝条剪掉，一切优化也就没有意义。所以，剪枝的第一个原则是正确性，即必须保证不丢失正确的结果。

- **原则之二：准确性**

在保证了正确性的基础上，对剪枝判断的第二个要求就是准确性，即能够尽可能多的剪去不能通向正解的枝条。剪枝方法只有在具有了较高的准确性的时候，才能真正收到优化的效果。

- **原则之三：高效性**

利用出解的“必要条件”进行判断，会有不含正解的枝条没被剪枝。这些情况下的剪枝判断操作，对程序的效率的提高有副作用。为了减少剪枝判断的副作用，除了要改善判断的准确性外，还需要提高判断操作本身的时间效率。

1.1 剪枝的优化技巧

1. 优化搜索顺序

在不同的问题中，搜索树的各个层次、各个分支之间的顺序不是固定的，不同的搜索顺序会产生不同的搜索树形态，其规模大小也相差甚远，**优先搜索分支较少的节点**。

2. 排除等效冗余

在搜索过程中，若能判断从搜索树当前节点上沿某几条不同分支到达的子树是相同的，那么只需对其中一条分支执行搜索。

1.1 剪枝

3. 可行性剪枝

可行性剪枝也叫上下界剪枝，其是指在搜索过程中，及时对当前状态进行检查，**若发现分支已无法到达递归边界，就执行回溯**。

4. 最优性剪枝

在最优化问题的搜索过程中，**若当前花费的代价已超过当前搜索到的最优解**，那么无论采取多么优秀的策略到达递归边界，都不可能更新答案，**此时可以停止对当前分支的搜索进行回溯**。

5. 记忆化搜索

可以记录每个状态的搜索结果，在重复遍历一个状态是直接检索并返回。

1 [3650] 数的划分(可行性剪枝，上下界剪枝)

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。

1 1 5　　1 5 1　　5 1 1

问有多少种不同的分法。

输入n ，k (6<n<=200，2<=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

求把 n 无序的划分成k份的方案数，最直接的搜索方法是依次枚举x1，x2……xk的值，也就是求

x1+x2+...+xk=n的解数。

显然这么搜索，很容易TLE。所以我们需要剪枝，这道题用到的主要是可行性剪枝和上下界剪枝；

分析

①因为本题不考虑分解的顺序，所以我们可以规定分成的这k个数是从小到大分的，即a[i]<=a[i+1]。

②假设我们已经把n分解为i-1个数分别为a[1],a[2]……a[i-1]（其中a[1]<=a[2]<=……<=a[i-1]），那么**a[i]的最大值是将剩余k-i+1个数平均划分**，即设

m=n -（a[1]+a[2]+……+a[i-1]），

那么a[i]<=m/(k-i+1)，所以对于每个a[i]，扩展上一级的m/(k-i+1)；

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m,a[8],s=0;
void dfs(int k){//分第k分
    if(n==0)return;
    if(k==m){
        if(n>=a[k-1])
            s++;
        return;
    }
    for(int i=a[k-1];i<=n/(m-k+1);i++){
        //第k分的上下界
        a[k]=i;//第k份的值
        n-=i;
        dfs(k+1);
        n+=i;
    }

}
int main(){
    cin>>n>>m;
    a[0]=1;//初始条件
    dfs(1);
    cout<<s<<endl;
    return 0;
}
```

思路2：

设 f(n,m) 为整数 n 拆分成 m 个数字的方案数。

那么对于每一个情况一定可以分为以下两种情况，且不重不漏。

1.不选 1 的情况

如果不选择 1，我们把 n 拆分成 m 块的情况，可以等价于将每一块都减去1，然后分为m块，即 f(n-m,m)

2.选 1 的情况

那么就是其中一块肯定有一个 1，然后对n-1分成m-1块，即 f(n-1,m-1)。

所以总递推式为 f(n,m)=f(n-m,m)+f(n-1,m-1)

递归结束的条件是 n=0 或 n

```cpp
#include<iostream>
using namespace std;
int dfs(int n,int k)  //把n整数划分成k份
{
    if(n==0||n<k||k==0)return 0;  //无法继续划分
    if(k==1||n==k)return 1;       //只能划分成一项
    return dfs(n-1,k-1)+dfs(n-k,k);
}
int main()
{
    int n,k;  //把n整数划分成k份
    cin>>n>>k;
    int x=dfs(n,k);
    cout<<x<<endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,k,dp[205][8];
int main(){
    scanf("%d%d",&n,&k);
    dp[0][0]=1;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=i&&j<=k;j++)
            dp[i][j]=d[i-1][j-1]+d[i-j][j];
        printf("%d\n",dp[n][k]);
        return 0;
}
```

2 [3651]: 生日蛋糕

7月17日是Mr.W的生日，ACM-THU为此要制作一个体积为 N π 的**M层**生日蛋糕，每层都是一个圆柱体。

设从下往上数第i(1 <= i <= M)层蛋糕是半径为Ri，高度为Hi的圆柱。当i < M时，要求 $R_i > R_{i+1}$ 且 $H_i > H_{i+1}$。

由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积Q最小。

令Q = S π，请编程对给出的蛋糕的体积 N 和 M，找出**蛋糕的制作方案（适当的 Ri 和 Hi 的值），使S最小**。

（除Q外，以上所有数据皆为正整数）

[3651]: 生日蛋糕

输入有两行，第一行为N（N <= 10000），表示待制作的蛋糕的体积为N;第二行为M(M <= 20)，表示蛋糕的层数为M。

输出仅一行，是一个正整数S（若无解则S = 0）。

样例输入

100　//表示待制作的蛋糕的体积为N

2　　//表示蛋糕的层数为M

样例输出

68　　//最小的蛋糕外表面的面积Q

思路

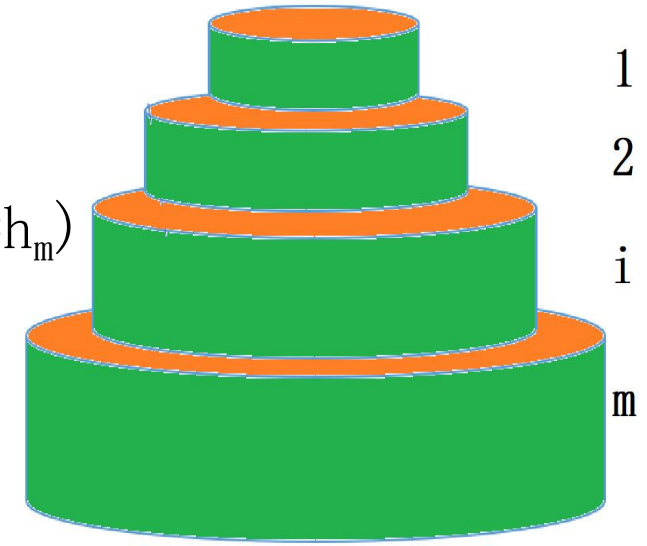
首先理解题意：题目要求在确定体积V和层数m情况下，求最小的表面积。即在约数条件V和m的情况下，求最小的半径ri和高度hi。可以用枚举或搜索求解。同时还要求：从下到上的半径越来越小，高也越来越小。

观察可的，所有的平面的表面积（红色部分）加起来就是底面的表面积，所以只要算侧面的表面积（绿色部分）就可以。

红色面积 $S1=\pi*r_m*r_m$

圆柱体侧面积=$2\pi*(r_1*h_1+r_2*h_2+...+r_m*h_m)$

圆柱体体积=$\pi r_1^2*h_1+...+\pi r_m^2*h_m$



分析1：优化搜索顺序

如何考虑搜索的顺序？

自下而上搜索时，先处理大的体积，再处理小的体积；先处理大的体积，表明剩余的体积更小，分支就会越少（即给剩余部分的选择余地越少越好），符合“**优化搜索顺序**”策略。同时在枚举半径r和高度h时，先枚举半径再枚举高度，因为半径是平方级别，占的比重更大，同样半径和高度越大，给剩余部分的选择余地就越少。所以枚举时，半径和高度都从大到小搜索。

- 层间：从下到上
- 层内：先枚举半径再枚举高（半径相对于高来说对体积的影响较大），半径由大到小，高度由大到小

```
//搜索顺序，先R后H，从大到小
for(int r = min(R[u + 1] - 1,(int)sqrt((n - v - minv[u - 1]) / u)); r >= u; r--)
    for(int h = min(H[u + 1] - 1, (n - v - minv[u - 1]) / r / r); h >= u; h--)
    {
        H[u] = h, R[u] = r;

        //最底层的时候需要加上r*r
        int t = u == m ? r * r : 0;

        dfs(u - 1, v + r * r * h, s + 2 * r * h + t);
    }
```

分析2：可行性剪枝

如何确定半径 r 的范围？

记总体积为n，当前层位u，第i层的高度为Hu，半径为Ru，体积为Vu，第m 层到第 u 层体积的累计值 V。对于 R，当前为第 u 层，第 u 层的体积为Vu。R 最小的取值应该是当前的层号u(上面还有u-1,u-2,..1层，第u层的最小半径为i），R的最大值应该由两部分决定

- u +1层的半径减1，即$R_{u+1}-1$
- 第u层体积的最大值除第u层高度的最小值u

这两者的最小值，故有以下等式成立

$$u \le R_u \le min\{R_{u+1}-1, \sqrt{\frac{n-min\sum_{i=1}^{u-1}V_i - V}{u}}\}$$


```
for(int r = min(R[u + 1] - 1,(int)sqrt((n - v - minv[u - 1]) / u)); r >= u; r--)
```

分析2：可行性剪枝

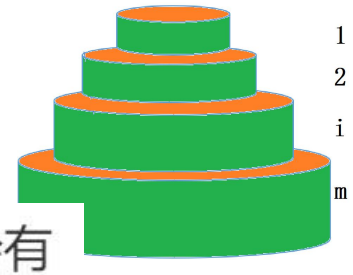
如何确定hi的范围？

对于第u层高度h的推导同理，高度h的取值的最小值应该大于等于层号u，高度的最小值由两部分决定

- $H_{u+1}-1$
- 第u层体积的最大值除第u层的底面积最小值

故同理可得出下列等式

$$u \le H_u \le min\{H_{u+1}-1, \frac{n-min\sum_{i=1}^{u-1}V_i - V}{R_u^2}\}$$



考虑体积的剪枝：预处理前u层的体积最小值$min\sum_{i=1}^{u-1}V_i$，会有

$V+min\sum_{i=1}^{u-1}V_i \le n$

```
for(int h = min(H[u + 1] - 1, (n - v - minv[u - 1]) / r / r); h >= u; h--)
```

分析2： 可行性剪枝

推表面积公式和体积公式的关系

第一层到第u层的表面积有（不考虑π）

$$S_{1-u} = 2\sum_{i=1}^{u} R_i H_i = \frac{2}{R_{u+1}} \sum_{i=1}^{u} R_{u+1} R_i H_i > \frac{2}{R_{u+1}} \sum_{i=1}^{u} R_i^2 H_i$$

第一层到第u层的体积有　（还未处理的部分体积）

$$n - V = \sum_{i=1}^{u} R_i^2 H_i$$

```
if(v + minv[u] > n) return;//超过最大体积
```

分析2： 可行性剪枝

$$S_{1-u} > \frac{2}{R_{u+1}} \sum_{i=1}^{u} R_i^2 H_i$$

$$n - V = \sum_{i=1}^{u} R_i^2 H_i$$

推出

$$S_{1-u} > \frac{2(n-V)}{R_{u+1}}$$

因此$S_{总} = S + S_{1-u} >= S_{ans}$，即$S + \frac{2(n-V)}{R_{u+1}} >= S_{ans}$时就可以剪枝掉

```
if (s + 2 * (n - v) / R[u + 1] >= res) return;
```

分析3： 最优性剪枝

记第m层到第u层表面积的累计值S, 第1到第$u-1$层表面积的最小值为$min\sum_{i=1}^{u-1}S_i$

则应该有$S+min\sum_{i=1}^{u-1}S_i<res$

```
//minv[u] 从第一层到第u层的最小体积
//mins[u] 从第一层到第u层的最小面积
if(v + minv[u] > n) return;//超过最大体积
if(s + mins[u] >= res) return;//超过现有的最优值

if (s + 2 * (n - v) / R[u + 1] >= res) return;
```

代码

```
#include<iostream>
#include<cmath>
using namespace std;

const int N = 24, INF = 1e9;

int n, m;
int minv[N];//从第一层到第u层的最小体积
int mins[N];////从第一层到第u层的最小面积
int res = INF;

//记录每层的半径和高，因为会用到上一层的高度
int R[N], H[N];
```

代码

```
42  int main()
43  {
44      cin >> n >> m;
45      for(int i = 1; i <= m; i++)
46      {//从上到下最小的半径和高度为i
47          minv[i] = minv[i - 1] + i * i * i;//初始最小体积 （不算pi）
48          mins[i] = mins[i - 1] + 2 * i * i;//初始最小面积
49      }
50      //m+1层不存在，作为哨兵，减少边界情况的判断
51      R[m + 1] = H[m + 1] = INF;// r = min(R[u + 1] - 1 有用到
52
53      //当前处理的面积和 s=0  //初始 当前处理的体积和v=0
54      dfs(m, 0, 0);//从最下面一层开始搜索
55
56      if(res == INF) res = 0;
57      cout << res << endl;
58      return 0;
59  }
```

代码

```
15  //u当前层次，v当前处理的体积和，s当前处理的面积和
16  void dfs(int u, int v, int s)
17  {
18      //minv[u] 从第一层到第u层的最小体积
19      //mins[u] 从第一层到第u层的最小面积
20      if(v + minv[u] > n) return;//超过最大体积
21      if(s + mins[u] >= res) return;//超过现有的最优值
22
23     if (s + 2 * (n - v) / R[u + 1] >= res) return;
24
25      if(u==0)//u==0表示搜索完所有层
26      {
27          if(v == n) res = s;
28          return;
29      }
```

代码

```
31  //搜索顺序，先R后H，从大到小
32  for(int r = min(R[u + 1] - 1,(int)sqrt((n - v - minv[u - 1]) / u)); r >= u; r--)
33      for(int h = min(H[u + 1] - 1, (n - v - minv[u - 1]) / r / r); h >= u; h--)
34      {
35          H[u] = h, R[u] = r;
36          int t = 0;  //最底层的时候需要加上r*r
37          if(u == m) t = r * r;
38          dfs(u - 1, v + r * r * h, s + 2 * r * h + t);
39      }
40  }
```

3 [2369] 数列分组

给你n个数，将这n个数分成总和相等的若干组，问最多可以分成多少组？

输入第一行为一个正整数n，第二行输出n个正整数。 输出一个整数，表示最多可以分成多少组。

9

5 2 1 5 2 1 5 2 1

输出

6

9个数，可以分成 [5 1] [5 1] [5 1] [2 2 2] 共四组。

样例输入

6

2 2 5 2 1 3

样例输出

6

4 [3652] 小木棍（木棒）

乔治有一些同样长的小木棍，他把这些木棍随意砍成几段，直到每段的长都不超过50。现在，他想把小木棍拼接成原来的样子，但是却忘记了自己开始时有多少根木棍和它们的长度。

给出每段小木棍的长度，编程帮他找出原始木棍的最小可能长度。

输入输入文件共有二行。第一行为一个单独的整数N表示砍过以后的小木棍的总数，其中N≤60。第二行为N个用空个隔开的正整数，表示N根小木棍的长度。

输出输出文件仅一行，表示要求的原始木棍的最小可能长度。

样例输入

9

5 2 1 5 2 1 5 2 1

样例输出

6

分析：

抽象题意，可将题意理解为： 给你n个数，将这n个数分成总和相等的若干组，问最多可以分成多少组？（组数越多即长度越小）

要得到原始最短木棍的可能长度，可以按照分段数的长度，依次枚举所有的可能长度len，**这里的长度len不是单调的，不能用二分**；可以确定原木棍的长度 len 在最长木棍的长度与 sum之间，

每次枚举len时，用深搜判断是否能用截断后的木棍拼合出整数个len，能用的话，找出最小的len即可。

对于1S的时间限制，用不加任何剪枝的深搜时，时间效率为指数级，效率非常低，程序运行将严重超时。对于此题，可以从可行性和最优性上加以剪枝

分析：

从最优性方面分析，可以做以下两种剪枝：

1） 设所有木棍的长度和是 sum,那么原长度（也就是需要输出的长度）一定能够被 sum整除，不然就没法拼了，即一定要拼出整数根

2）木棍原来的长度一定大于等于所有木棍中最长的那一根

综合上述两点，可以确定原木棍的长度 len 在最长木棍的长度与 sum之间，且 sum能被 len 整除。所以，在搜索原木棍的长度时，可以设定为从截断后所有木棍中最长的长度开始，每次增加长度后，必须能整除 sum。这样可以有效地优化程序

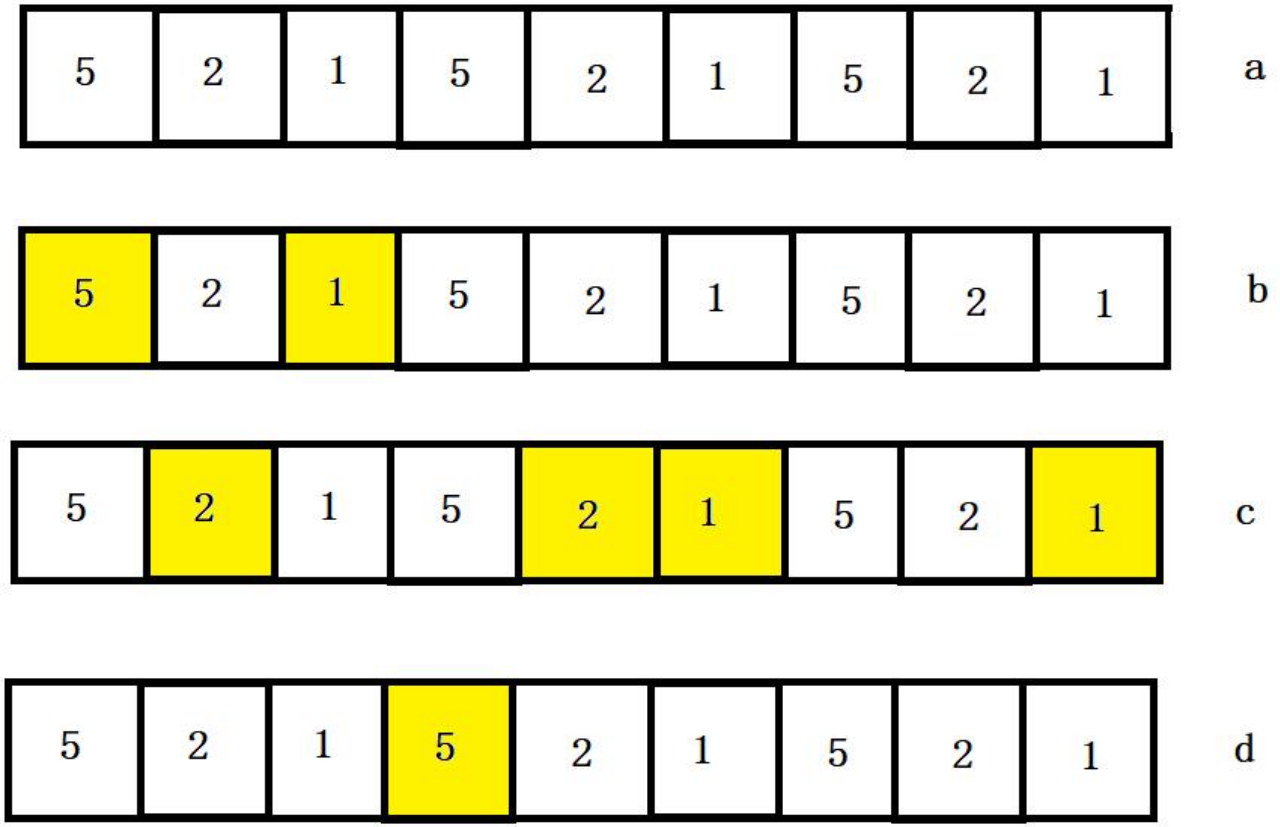
优化1：整除

拼接目标长度length，必然能被总长度sun整除，否则该进入下一个目标长度。

```
while(true)
{    //剪枝1
     if(sum%length==0&&dfs(0,0,0))
     {//如果k*length=sum,说明长度可以为length
          cout<<length<<endl;
          break;
     }
     length++; //枚举下一个长度
}
```

优化2：优化搜索顺序

无序的遍历



优化剪枝2：优化搜索顺序

有序的遍历


```
//剪枝2 优化搜索顺序
 //从大到小排序，从大的开始暴搜，方案数相对减少
sort(sticks,sticks+n,cmp);
```

优化剪枝3：排除等效冗余

3-1. 拼接一根木棒只与长度有关，跟木棍的顺序无关，所以这是组合问题。本题按照组合数的方式枚举。

```
//剪枝3-1 搜索从strat开始
for(int i=start;i<n;i++)
{ //枚举每一根木棍，判断当前木棍
```

3-2. 如果当前木棍加到当前棒中失败了，则直接略过后面相同长度的木棍。

```
//剪枝3-2 如果当前木棍失败，则跟它相等的木棍都跳过
int j=i;
while(j<n&&sticks[j]==l)j++;//跳过相同部分
i=j-1;     //因为这次循环后i要加一，所以先减一
```

优化剪枝3：排除等效冗余

3-3. 如果木棒的第一根木棍失败，则当前木棒一定失败。

```
//剪枝3-3
if(!cur)//cur=0表明 第一根木棍都不合适，表明这轮搜索失败，进入下一根木棒
    return false;
//所有木棒中最大的木棒(第一个木棒)放入失败，就一定失败
```

3-4. 最后一根木棒放入后可以达到length却没有成功，就一定失败（对结果没影响）

```
//剪枝3-4 最后一根木棒放入后可以达到length却没有成功，就一定失败
if(cur+l==length)return false;
```

综合上述分析，可以剪枝以下7点：

1）为保证未来分支更少，所以先从长木棍开始搜。同时短木棍比长木棍更灵活组合，所以可以对木棍按长度从大到小排序，。

2)在截断后的排好序的木棍中，当用木棍i拼合原始木棍时，可以从第i+1后的木棍开始搜。

3)用当前最长长度的木棍开始搜，如果拼不出当前设定的原木棍长度len，则直接返回，换一个原始木棍长度len。

4)相同长度的木棍不要搜索多次。用当前长度的木棍搜下去得不出结果时，可以提前返回。

5)判断搜到的几根木棍组成的长度是否大于原始长度len，如果大于，可以提前返回。

6)判断当前剩下的木棍根数是否能够拼的木棍根数，如果不够，肯定拼合不成功，直接返回。

7)找到结果后，在能返回的地方马上返回到上一层递归处。

代码（定义）

```
5   const int N=64;
6   int sticks[N];//没跟木棍的长度
7   bool st[N];//标记每根木棍是否用过
8   int n,sum,length;
9
10  int cmp(int a, int b){
11      return a>b;
12  }
13
14  //排到第u个木棍，该木棍长度为cur，从start开始枚举
15  bool dfs(int u,int cur,int start)
16  {
52
53  int main()
54  {
```

代码（main）

```
52  int main()
53  {
54      cin>>n;
55      sum=length=0;                    //清空
56      for(int i=0;i<n;i++)             //读入
57      {
58          cin>>sticks[i];
59          sum+=sticks[i];                          //统计长度
60          length=max(length,sticks[i]);    //统计最大长度
61      }
62
63      //剪枝2 优化搜索顺序
64       //从大到小排序，从大的开始暴搜，方案数相对减少
65      sort(sticks,sticks+n,cmp);
66      memset(st,0,sizeof st);  //清空
```

代码（main）

```
68          while(true)
69          {    //剪枝1
70              if(sum%length==0&&dfs(0,0,0))
71              {//如果k*length=sum,说明长度可以为length
72                  cout<<length<<endl;
73                  break;
74              }
75              length++; //枚举下一个长度
76          }
77          return 0;
78      }
```

代码(dfs1 普通写法)

```
14  //排到第u个木棍，该木棍长度为cur，从start开始枚举
15  bool dfs(int u,int cur,int start)
16  {
17      //当前木棒的数量*木棒的长度等于总和
18      if(u*length==sum)return true;
19      //当前木棒处理完，开始新的一根木棒
20      if(cur==length)
21          return dfs(u+1,0,0);
22      for(int i=start;i<n;i++)  //搜索从strat开始
23      { //枚举每一根木棍，判断当前木棍是否可用于组成木棒
24          if(st[i])continue; //已经被访问过了 可行性剪枝
25          if(sticks[i]+cur>length)continue;
26              //可行性剪枝 ,如果加上后超出木棍长度
27
28          st[i]=1;     //标记为已用
29          if(dfs(u,cur+sticks[i],i+1))
30              return true;//当前木棍搜索成功，继续下一根
31          st[i]=0;//复原现场
32      }
33      return false;  //失败
34  }
```

代码(dfs1 各种剪枝)

```
35  //剪枝3-3
36  if(!cur)//cur=0表明 第一根木棍都不合适，表明这轮搜索失败，进入下一
37      return false;
38  //所有木棒中最大的木棒(第一个木棒)放入失败，就一定失败
39
40  //剪枝3-4 最后一根木棒放入后可以达到length却没有成功，就一定失败
41  if(cur+sticks[i]==length)return false;
42
43
44    //剪枝3-2 如果当前木棍失败，则跟它相等的木棍都跳过
45      int j=i;
46      while(j<n&&sticks[j]==sticks[i])j++;//跳过相同部分
47      i=j-1;     //因为这次循环后i要加一，所以先减一
48
49  }
50  return false;  //失败
51  }
```

5 [3653] Addition Chains 加成序列

已知一个数列a0,a1.....am（其中a0 = 1，am = n，a0 < a1 < a2 < ... < am-1 < am）。对于每个k(1<=k<=m)，需要满足ak=ai+aj（0 <= i，j <= k-1，这里i与j可以相等）。
现给定n的值，要求m的最小值（并不要求输出），及这个数列的值(可能存在多个数列,只输出任一个满足条件的就可以了)。

【输入格式】

多组数据，每行给定一个正整数n。输入以0结束。

【输出格式】

对于每组数据，输出满足条件的长度最小的数列。

[3653] Addition Chains 加成序列

Sample Input
5
7
12
15
77
0
Sample Output

1 2 4 5
1 2 4 6 7
1 2 4 8 12
1 2 4 5 10 15
1 2 4 8 9 17 34 68 77
Source

分析

先理解题意。题目要求输出这样的一个序列：

1. 序列第一个数是1, 最后一个元素为输入的n
2. 序列是严格单调递增的
3. 序列中的每个数等于序列中前面的 两个元素的和，或者某个元素的2倍。
4. 构造一个长度最短的合法方案

考虑DFS求解，题目n=100, 理论上要搜索100层，但实际上最优情况下 log100, 约8层就可以，如[1, 2, 4, 8 , 16, 32, 64, 128] 8层就可以，预估题目15层也就差不多。

这就是迭代加深的精髓所在, 就是我们确定答案在一个比较浅的位置。

分析

迭代加深是指，在深度优先搜索时限制搜索的深度，如果当前深度限制下搜不到答案，就把深度限制增加，重新在进行一次搜索。虽然该过程在深度限制为d是，会重复搜索1~d-1层的节点，但当搜索树节点分子数目较多时，随着层数深度，每层节点会呈现指数级增长，这点重复搜素与深层子树的规模相比，实在是小巫见大巫。

总而言之，但搜索树规模随着层次的深入增加很快，并且我们能够确保答案在一个较浅层的节点时，就可以采用迭代加深的DFS方法来解决。

搜索框架

依次搜索序列中的每个位置k，枚举和作为分支，x[i]和x[j]的和填到x[k]上，然后递归填写下一个位置。

加入以下剪枝

1. 优化搜索顺序

为了让序列中的数尽快逼近n，在枚举i和j时从大到小枚举.

2. 排除等效冗余

对于不同的和j，x[i]+x[j] 可能是相等的。我们可以在枚举时用一个bool数组对x[i]+x[j]进行判重，避免重复搜索某一个和.

经过观察分析可以发现，m的值（序列长度）不对太大（≤10），而每次枚举两个数的和，分支很多。所以我们采用迭代加深的搜索方式，从1开始限制搜索深度，若搜索失败就增加深度限制重新搜索，知道找到一组解时即可输出答案

搜索框架

```
4   const int N = 110;
5   int n, path[N];
6
7   bool dfs(int u, int k)
8   {// u表示当前层数，k表示最大的层数
33
34  int main()
35  {
36      path[0] = 1;
37      while (cin >> n && n!=0) {
38          int k = 1;
39          while (!dfs(1, k)) {  // 不断扩大范围
40              k++;//
41          }
42          for (int i = 0; i < k; i++) {
43              cout << path[i] << " ";
44          }
45          cout << endl;
46      }
47      return 0;
48  }
```

```
bool dfs(int u, int k)
{// u表示当前层数，k表示最大的层数
    if (u == k) {
        return path[u - 1] == n;
    }
    bool st[N] = {0}; // 通过 bool数组排除等效冗余
    for (int i = u - 1; i >= 0; i--) {
        for (int j = i; j >= 0; j--) {
            int s = path[i] + path[j];
            if (s > n || s <= path[u - 1] || st[s]) { //
              //超过范围，序列要严格递增 ， s已经出现过
                continue;
            }
            st[s] = true;//标记
            path[u] = s;
            if (dfs(u + 1, k)) {
                return true;
            }
        }
    }
    return false;
}
```

6 [1292] weight （搜索对象的选择）

已知原数列a_1，a_2，…，a_n中前1项，前2项，前3项……前n项的和，以及后1项，后2项，后3项……后n项的和，但是所有的数据都已经被打乱了顺序，还知道数列中的数存在于集合S中，求原数列。当存在多组可能数列的时候求左边的数最小的数列。

【输入格式】

第一行一个整数n。

第二行2n个整数，注意：数据已被打乱。

第三行一个整数m，表示S集合的大小。

第四行m个整数，表示S集合中的元素。

【输出格式】

输出满足条件的最小数列。

[1292] weight

输出满足条件的最小数列。
样例输入：
5
1 2 5 7 7 9 12 13 14 14
4　//集合的大小
1 2 4 5　//S集合中的元素
输出：
1 1 5 2 5 //

样例解释

从左往右求和	从右往左求和
1=1	5=5
2=1+1	7=2+5
7=1+1+5	12=5+2+5
9=1+1+5+2	13=1+5+2+5
14=1+1+5+2+5	14=1+1+5+2+5

结果序列（1,1,5,2,4）中的值都来自集合S，该样例选取集合中的{1,2,5}。

分析

因为题目中的S∈{1..500}，最坏的情况下每个数可以取到的值有500种，从数学方面很难找到较好方法，而采用搜索方法是一种很好的解决办法，根据数列从左往右依次搜索原数列每个数可能的值，然后与所知道的值进行比较。这样，得到了一个最简单的搜索方法A。

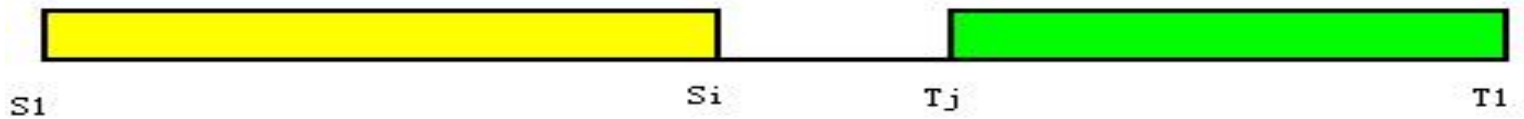
但方法A最坏的情况下扩展的节点为500^{1000}，运算速度过慢。

在这个算法中，对数列中的每个数分别进行了500次搜索，而导致了搜索量大。如何有效的减少搜索量是提高本题算法效率的关键。而运用搜索顺序的方法在本题中由于规定了左边的数最小而无法运用。

换个角度思考，搜索方法B：回过头来看看题目提供给我们的约束条件，我们用S_i表示前i项的和，用T_i表示后i项的和。
题目中的数据应是数列中的$S_1, S_2 \cdots\cdots S_n$，以及$T_1, T_2 \cdots\cdots T_n$。其中的任意$S_{i+1}-S_i$和$T_{i+1}-T_i$都属于集合S。另一个比较容易发现的约束条件是对于任意的i，有$S_n=T_n=S_i+T_{i+1}$。同样，在搜索的过程中最大化这些约束条件是提高程序效率的关键。
当任意从已知的数据中取出两个数的时候，会出现两种情况：

1、两个数同属于S_i，或者T_i

2、两数分别属于T_i和S_i



当两数同属于S_i或者T_i时，两个数之差，就是图中Sj-Si那一段，而当j=i+1时，Sj-Si必然属于题目给出的集合S。由此，当每次得到一个数Si或者Ti时，若已知S_{i-1}或者T_{i-1}，便能判断此时的S_i或T_i是否合法。所以在搜索中尽可能利用S_{i-1}和T_{i-1}推得S_i和T_i的可能取值，便能尽可能利用题目的约束条件。

因为题目的约束条件集中在S_i和T_i中，我们改变搜索的对象，不再搜索原数列中每个数的值，而是搜索给出的数中出现在S_i或者T_i中的位置。又由于约束条件中得出的S_{i+1}与S_i的约束关系，所以可在搜索中按照S_i和T_i递增或者递减的顺序进行搜索。

- 例如，对于数据：1 1 5 2 5，由它得到的值为
  1 2 7 9 14 5 7 12 13 14
  排序后为：
  1 2 5 7 7 9 12 13 14 14
  最大的两个数为所有数的和，在搜索中不用考虑，去掉14：
  1 2 5 7 7 9 12 13
- 观察发现，数列中的最小数1，只可能出现在所求数列的头部或者尾部。再假设1的位置已经得到了，去掉它以后，我们再观察剩下的数中最小的数2，显然也只可能在当前状态的头部或者尾部加上一个数得到2。这样，每搜索一个数，都只会将它放在头部和尾部，也就是放入S_i中或者T_i中。

- 推而广之，由小到大对排序的数进行搜索，判断每个数出现在头部还是尾部。此时由原数列的两头向中间搜索。由之前的分析可知，每个数只可能属于S_i和T_i中。当已经搜索出原数列的$S_1, S_2 \cdots S_i$和$T_1, T_2 \cdots T_j$，对于正在搜索的数K，只有两种可能：S_{i+1}和T_{j+1}，分别搜索这两个可能，即判断$K-S_i$和$K-T_j$是否属于已知集合S。并且在每搜索出一个数K的时候，我们将排序后的数列中S_n-k去掉。这样，当$K-S_i$(T_i)不属于集合S或者S_n-k不存在于排序后的数列时，就回溯。
- 这样得到的算法在最坏情况下扩展的节点为2^{1000}，并且由于在搜索过程中充分利用了题目约束条件，运行结果是相当可观的。
- 在这道题目中，原始的搜索方法搜索量巨大，但通过分析，选择适当的搜索对象，在搜索量减少的同时充分利用了题目的约束条件，从而成为了程序的一个有利的剪枝。

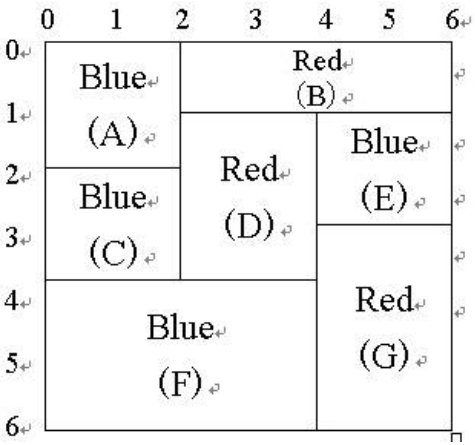
7 [3655] 平板涂色

CE 数码公司开发了一种名为自动涂色机（APM）的产品。它能用预定的颜色给一块由不同尺寸且互不覆盖的矩形构成的平板涂色。 为了涂色，APM 需要使用一组刷子，每个刷子涂一种不同的颜色。

APM 拿起一把有颜色 C 的刷子，并给所有颜色为 C 且符合下面限制的矩形涂色：为了避免颜料渗漏导致颜色混合，一个矩形只能在所有紧靠它上方的矩形涂色后，才能涂色。

例如下图中矩形 F 必须在 C 和 D 涂色后才能涂色。 注意，每一个矩形必须立刻涂满，不能只涂一部分。

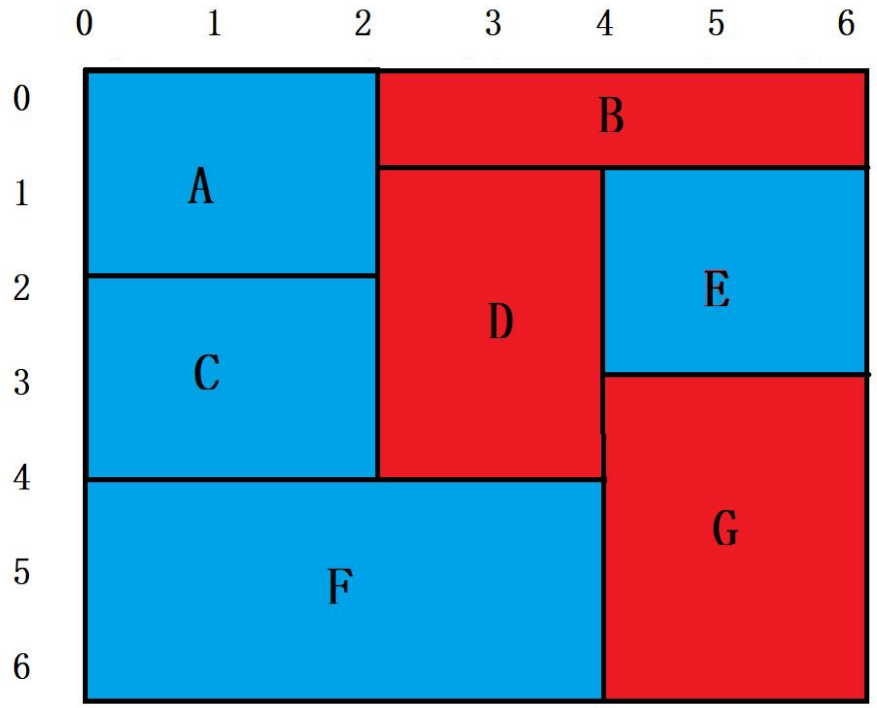
写程序求一个使 APM 拿起刷子次数最少的涂色方案。 注意，如果一把刷子被拿起超过一次，则每一次都必须记入总数。

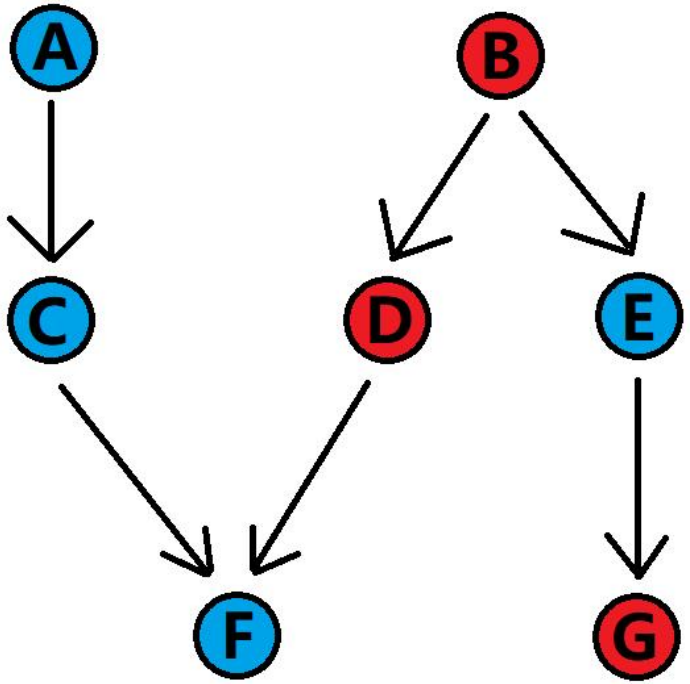


7 [3655] 平板涂色

```
7     //矩形的个数N
0 0 2 2 1    左上角的 y坐标和 x坐标，右下角的 y坐标和 x坐标，以及预定颜色。
0 2 1 6 2
2 0 4 2 1
1 2 4 4 2
1 4 3 6 1
4 0 6 4 1
3 4 6 6 2
```

分析





首先，对于一张有依赖关系的涂色图可以转化为一张拓扑图，这样看起来更加直观一点(只能先涂入度为0的点)。本题最优策略是先涂B->D，再涂A->C->F和E，最后涂G，所以答案是3。

显然易见我们可以用搜索来完成本题，因为数据范围很小。

分析

第一个搜索的点应该是以哪一个为起点开始搜索？

本例中可以以A作为起点，或者以B作为起点，那么得到的涂色次数是不一定相同的。比如说以A为起点，第一步：A->C->F，第二步：B->D，第三步：E，第四步：G，总共需要4步，比最优解3步还多了一步。

所以说，起点要从所有一开始入度为0的点都枚举一遍，最后在所有情况中取最小值即是答案。

分析

下一步，**拓扑排序**了。把起点的连接点的入度减一，**第二个搜索的点**是枚举所有没有被涂过色的点，如果说，连接点的入度为0，分为两种情况：

$$\begin{cases} \text{如果所用画笔颜色和矩阵颜色一致，则继续用此画笔涂色，即涂色次数不变} \\ \text{如果所用画笔颜色和矩阵颜色不同，则改用其他画笔涂色，即涂色次数加一} \end{cases}$$

两步搜索就可以枚举所有的情况。

下面来处理**如何从一个点向另一个点连边**？
很显然，从直观上来看，就是第二个矩形在第一个矩形的笼罩下。
用坐标来表示(按照题目所给坐标系，x轴向右，y轴向下，记矩形左上角为

分析

下面处理**如何从一个点向另一个点连边？**

很显然，从直观上来看，就是第二个矩形在第一个矩形的笼罩下。

用坐标来表示(按照题目所给坐标系，x 轴向右，y 轴向下，记矩形左上角为P_1，矩形右下角为P_2)，那么即为：

①矩形 1. P_2. y<=矩形 2. P_1. y

②矩形 1. P_1. x<=矩形 2. P_1. x

③矩形 1. P_2. x>=矩形 2. P_2. x

如果都满足，矩形 1 向矩形 2 连一条有向边。

```cpp
#include<iostream>
#include<vector>
using namespace std;
int n;//矩形数量
int y1[30],x1[30],y2[30],x2[30],colour[30];//坐标和所用颜色
int in[30];//入度
bool f[30];//标记数组，true表示已经被涂色过了
int mint=2e9;//答案次数
vector<int>s[30];//链接表vector实现
//dfs:
//第一维u表示当前在哪个点涂色
//第二维cnt表示涂色次数
//第三维表num示已经涂了多少个矩形
//第四维color表示当前的画笔颜色
void dfs(int u,int cnt,int num,int color)
{
    //减枝，如果本方案的涂色次数已经溢出了就return
    if(cnt>=mint)return;
    if(num==n){//达到终点，画完了n个矩形
        mint=min(mint,cnt);
        return;
    }
```

```
23      //减少连接点的入度，拓扑排序
24      for(auto it:s[u]){
25          in[it]--;
26      }
27      //【第二个搜索的点】 枚举剩下没有被涂过色的点
28      for(int i=1;i<=n;i++){
29          //如果该点没有被涂过色并且入度为0，表示可以深入搜索
30          if(!f[i]&&!in[i]){
31              f[i]=true;//标记已被涂色
32              //按颜色进行对应的递归
33              if(colour[i]!=color)dfs(i,cnt+1,num+1,colour[i]);
34              else dfs(i,cnt,num+1,colour[i]);
35              f[i]=false;//恢复现场
36          }
37      }
38      //恢复现场
39      for(auto it:s[u]){
40          in[it]++;
41      }
42  }
```

[6336] 小猫爬山

翰翰和达达饲养了 N 只小猫，这天，小猫们要去爬山。
经历了千辛万苦，小猫们终于爬上了山顶，但是疲倦的它们再也不想徒步走下山了（呜咕>_<）。
翰翰和达达只好花钱让它们坐索道下山。
索道上的缆车最大承重量为 W，而 N 只小猫的重量分别是 C1、C2……CN。
当然，每辆缆车上的小猫的重量之和不能超过 W。
每租用一辆缆车，翰翰和达达就要付 1 美元，所以他们想知道，最少需要付多少美元才能把这 N 只小猫都运送下山？

[6336] 小猫爬山

翰翰和达达饲养了 N 只小猫，这天，小猫们要去爬山。
经历了千辛万苦，小猫们终于爬上了山顶，但是疲倦的它们再也不想徒步走下山了（呜咕>_<）。
翰翰和达达只好花钱让它们坐索道下山。
索道上的缆车最大承重量为 W，而 N 只小猫的重量分别是 C1、C2……CN。
当然，每辆缆车上的小猫的重量之和不能超过 W。
每租用一辆缆车，翰翰和达达就要付 1 美元，所以他们想知道，最少**需要付多少美元才能把这 N 只小猫都运送下山**？

[6336] 小猫爬山

输入格式

第 1 行：包含两个用空格隔开的整数，N 和 W。

第 2..N+1 行：每行一个整数，其中第 i+1 行的整数表示第 i 只小猫的重量 Ci。 ？

输入格式

第 1 行：包含两个用空格隔开的整数，N 和 W。

第 2..N+1 行：每行一个整数，其中第 i+1 行的整数表示第 i 只小猫的重量 Ci。

$1 \leqslant N \leqslant 18$,

$1 \leqslant Ci \leqslant W \leqslant 10^8$

[6336] 小猫爬山

5 1996// n只猫 每辆缆车上的小猫 2
缆车的重量之和不能超过 W
1 2 199 12 29 //第 i 只小猫的重量 Ci

$1 \leqslant N \leqslant 18$,
$1 \leqslant Ci \leqslant W \leqslant 10^8$

5 1996

1

2

1994

12

29

今天的课程结束啦……

下课了…
同学们再见！