搜索-DFS之剪枝与优化

一、AcWing 165. 小猫爬山

【题目描述】

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$C_1, C_2, \ldots, C_N$。

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？

【输入格式】

第1行：包含两个用空格隔开的整数，N和W。

第$2 \sim N+1$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C_i。

【输出格式】

输出一个整数，表示最少需要多少美元，也就是最少需要多少辆缆车。

【数据范围】

$1 \leq N \leq 18$

$1 \leq C_i \leq W \leq 10^8$

【输入样例】

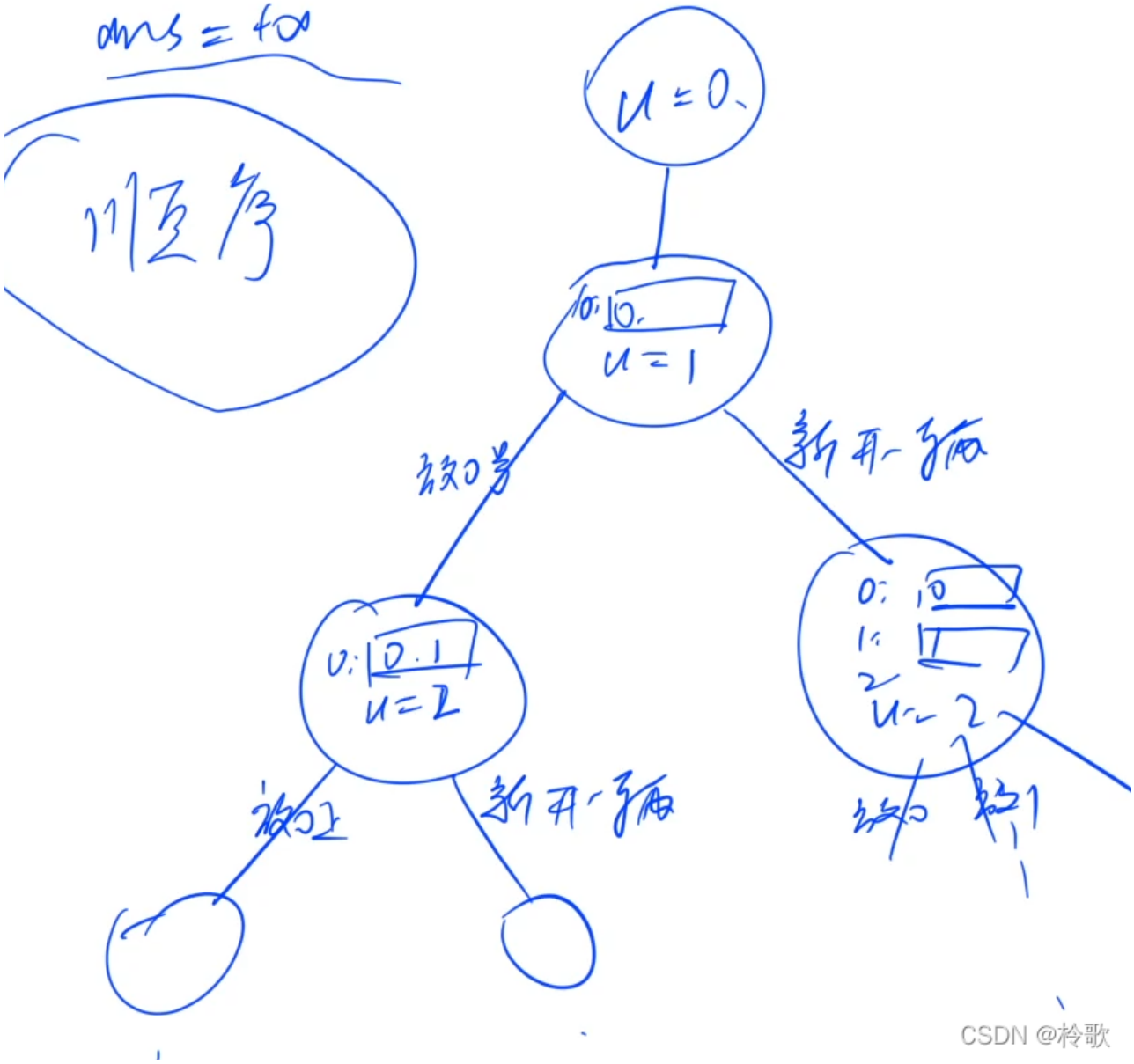
```
5 1996
1
2
1994
12
29
```

【输出样例】

```
2
```

【分析】

这道题目我们肯定是搜索了，我们枚举每只小猫，然后枚举现有的每辆车，如果能放进去那么就放进去然后继续搜索下一只，如果所有车都不能放进去则新开一辆车，其搜索树如下图所示：



我们发现这道题目有两个可以剪枝的部分，一个是如果当前的答案已经大于了我们已知的最小答案，不用说直接剪枝即可。第二个剪枝则是我们可以将小猫的体重从大到小排序，这样我们的搜索树就会缩短许多，至于为什么，因为我们的剩余空间就变小了，然后可选择的猫也就少了。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 20;
int n, m, res = N;
int w[N], sum[N];//sum表示每辆车当前的重量和

//当前搜索到第u只猫,车的数量为k
void dfs(int u, int k)
{
    if (k >= res) return;//最优性剪枝
    if (u == n) { res = k; return; }//搜索完全部小猫
    for (int i = 0; i < k; i++)//枚举判断是否有车能放入当前小猫
        if (sum[i] + w[u] <= m)//可行性剪枝
        {
            sum[i] += w[u];//将第u只猫放进第i辆车
            dfs(u + 1, k);//车的数量不变
            sum[i] -= w[u];//恢复现场
        }
    sum[k] = w[u];//现有的k辆车都放不了那么就开新的
    dfs(u + 1, k + 1);//车的数量+1
    sum[k] = 0;//恢复现场
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++) cin >> w[i];
    sort(w, w + n, greater<int>());//优化搜索顺序
    dfs(0, 0);
    cout << res << endl;
    return 0;
}
```

二、AcWing 166. 数独

【题目描述】

数独是一种传统益智游戏，你需要把一个9×9的数独补充完整，使得图中每行、每列、每个3×3的九宫格内数字$1 \sim 9$均恰好出现一次。

请编写一个程序填写数独。

【输入格式】

输入包含多组测试用例。

每个测试用例占一行，包含81个字符，代表数独的81个格内数据（顺序总体由上到下，同行由左到右）。

每个字符都是一个数字（$1 \sim 9$）或一个`.`（表示尚未填充）。

您可以假设输入中的每个谜题都只有一个解决方案。

文件结尾处为包含单词`end`的单行，表示输入结束。

【输出格式】

每个测试用例，输出一行数据，代表填充完全后的数独。

【输入样例】

```
4.....8.5.3.........7......2.....6.....8.4......1.......6.3.7.5..2.....1.4......
......52..8.4......3...9...5.1...6..2..7........3.....6...1.........7.4.....3.
end
```

【输出样例】

```
417369825632158947958724316825437169791586432346912758289643571573291684164875293
416837529982465371735129468571298643293746185864351297647913852359682714128574936
```

【分析】

搜索不用说，相信你一眼就可以看到是搜索算法，问题是这道题目纯搜索明显是要时间爆炸的，所以我们得剪枝。

- 优化搜索顺序：很明显，我们肯定是从当前能填合法数字最少的位置开始填数字。
- 排除等效冗余：任意一个状态下，我们只需要找一个位置填数即可，而不是找所有的位置和可填的数字。

- 位运算：很明显这里面$check$判定很多，我们必须优化这个$check$，所以我们可以对于每一行、每一列以及每一个九宫格，都利用一个九位二进制数保存状态，表示当前还有哪些数字可以填写，若第i位为1则表示可以填第i个数，由于i是$0\sim9$的，因此需要注意转换映射。
- $lowbit$：我们这道题目当前得需要用$lowbit$运算取出当前可以能填的数字。

其余细节详见代码部分。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;

const int N = 9;
char g[100];
int row[N], col[N], cell[3][3];//表示行,列与小方格中能填的数的状态
int ones[1 << N];//表示每种状态中1的数量

void init()
{
    //初始9个数全为1,表示都可以用
    for (int i = 0; i < N; i++) row[i] = col[i] = (1 << N) - 1;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cell[i][j] = (1 << N) - 1;
}

//在(x,y)处操作,即填上t或删除t,is_set标记填上还是删除
void op(int x, int y, int t, bool is_set)
{
    if (is_set) g[x * N + y] = t + '1';//0~8转换为1~9
    else g[x * N + y] = '.';
    int v = 1 << t;//v为t的二进制表示的状态
    row[x] ^= v, col[y] ^= v, cell[x / 3][y / 3] ^= v;//操作完后这个数的状态一定会和之前相反
}

//返回该位置可填数的状态,也就是行列与小方块状态的与
int get(int x, int y)
{
```

```
33      return row[x] & col[y] & cell[x / 3][y / 3];
34  }
35
36  int lowbit(int x)
37  {
38      return x & -x;
39  }
40
41  bool dfs(int cnt)
42  {
43      if (cnt == 0) return true;
44      int minv = 10, x, y;
45      //先找出搜索分支最少的点,即状态中1最少的空位
46      for (int i = 0; i < N; i++)
47          for (int j = 0; j < N; j++)
48              if (g[i * N + j] == '.')
49              {
50                  int s = get(i, j);
51                  if (ones[s] < minv) minv = ones[s], x = i, y = j;
52              }
53      int state = get(x, y);//(x,y)的状态中1最少
54      for (int i = state; i; i -= lowbit(i))//枚举state的每一位1
55      {
56          //lowbit(i)为只有i的最低位1时表示的数,因此使用log2转换成这个1是第几个
    数
57          op(x, y, log2(lowbit(i)), true);
58          if (dfs(cnt - 1)) return true;
59          op(x, y, log2(lowbit(i)), false);
60      }
61      return false;
62  }
63
64  int main()
65  {
66      //预处理ones数组
67      for (int i = 0; i < 1 << N; i++)
68          for (int j = 0; j < N; j++)
69              ones[i] += i >> j & 1;
70      while (cin >> g, g[0] != 'e')
71      {
72          init();//初始化row,col,cell的状态
73          int cnt = 0;//表示空位的数量
74
74          //将已知的数填上且统计出空位的数量
```

```
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (g[i * N + j] != '.') op(i, j, g[i * N + j] - '1', true);
                else cnt++;
        dfs(cnt);
        puts(g);
    }
    return 0;
}
```

三、AcWing 167. 木棒

【题目描述】

乔治拿来一组等长的木棒，将它们随机地砍断，使得每一节木棍的长度都不超过50个长度单位。

然后他又想把这些木棍恢复到为裁截前的状态，但忘记了初始时有多少木棒以及木棒的初始长度。

请你设计一个程序，帮助乔治计算木棒的可能最小长度。

每一节木棍的长度都用大于零的整数表示。

【输入格式】

输入包含多组数据，每组数据包括两行。

第一行是一个不超过64的整数，表示砍断之后共有多少节木棍。

第二行是截断以后，所得到的各节木棍的长度。

在最后一组数据之后，是一个零。

【输出格式】

为每组数据，分别输出原始木棒的可能最小长度，每组数据占一行。

【数据范围】

数据保证每一节木棍的长度均不大于50。

【输入样例】

```
9
5 2 1 5 2 1 5 2 1
4
1 2 3 4
0
```

【输出样例】

```
6
5
```

【分析】

直接上剪枝思路：

- 可行性剪枝：木棒长度$length$必须能整除木棍总长度sum。
- 优化搜索顺序：将木棍长度从大到小排序，优先枚举长度较大的，这样搜索分支较少。
- 排除等效冗余：

  （1）按照组合数方式枚举，因为一根木棒如果由$1,2,3$号木棍拼接而成，那么$2,1,3$等其它组合方式也同样能拼成；

  （2）如果当前木棍拼到当前木棒里失败了，那么跳过所有与之等长的木棍，因为换任何一根等长的木棍效果也是一样的；

  （3）如果当前木棍i拼到当前木棒的第一个位置失败了，那么整个方案失败。可以使用反证法，如果最后方案能成功，那么木棍i一定在后面的某根木棒j中，由于顺序不影响结果，可以将木棍i放到木棒j的第一个位置，然后将木棒j与之前i作为开头的那根木棒进行交换，则得到i作为第一根木棍的合法方案，产生矛盾；

  （4）如果当前木棍i拼到当前木棒的最后一个位置**且将当前木棒成功拼完**，而拼之后的木棒失败了，那么整个方案失败。同样使用反证法，如果最后方案能成功，那么之前那根木棒会使用其它的几根木棍比如j,k，这几根木棍一定也得填满这根木棒的末尾，也就是长度之和等于i的长度，然后i会在之后的某根木棒中，那么可以交换i和j,k，这样整个方案也是成功的，而且i在之前那根木棒的最后一个位置，产生矛盾。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

```

```cpp
const int N = 70;
int w[N];
int sum, length;//sum表示木棍总长度,length表示当前枚举的木棒长度
bool st[N];
int n;

//当前已经拼好u根木棒,当前木棒长度为s,需要从第start根木棍开始选
bool dfs(int u, int s, int start)
{
    if (u * length == sum) return true;
    if (s == length) return dfs(u + 1, 0, 0);
    for (int i = start; i < n; i++)
    {
        if (st[i] || s + w[i] > length) continue;//已经用过或者拼上后长度过长则剪枝
        st[i] = true;
        if (dfs(u, s + w[i], i + 1)) return true;
        st[i] = false;
        //如果当前木棍放在木棒的第一根或最后一根(当前木棒已拼完)失败了,那么整个方案就失败了
        if (s == 0 || s + w[i] == length) return false;
        //如果当前木棍失败了,那么用与之等长的木棍也一定会失败,因此跳过等长的木棍
        while (i + 1 < n && w[i + 1] == w[i]) i++;
    }
    return false;
}

int main()
{
    while (cin >> n, n)
    {
        memset(st, false, sizeof st);
        sum = 0;
        for (int i = 0; i < n; i++) { cin >> w[i]; sum += w[i]; }
        sort(w, w + n, greater<int>());
        for (length = 1; length <= sum; length++)
            if (sum % length == 0 && dfs(0, 0, 0))//木棒长度必须能被总长度整除
            {
                cout << length << endl;
                break;

            }
```

```
    }
    return 0;
}
```

四、AcWing 168. 生日蛋糕

【题目描述】

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

设从下往上数第i层蛋糕是半径为R_i，高度为H_i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$Q = S\pi$，请编程对给出的N和M，找出蛋糕的制作方案（适当的R_i和H_i的值），使S最小。

除Q外，以上所有数据皆为正整数。

【输入格式】

输入包含两行，第一行为整数N，表示待制作的蛋糕的体积为$N\pi$。

第二行为整数M，表示蛋糕的层数为M。

【输出格式】

输出仅一行，是一个正整数S（若无解则$S = 0$）。

【数据范围】

$1 \leq N \leq 10000$

$1 \leq M \leq 20$

【输入样例】

```
100
2
```

【输出样例】

```
68
```

【分析】

本题统一不考虑π，因此我们分析的时候就无需将π带入计算了。本题的剪枝有如下几种：

- 假设蛋糕从上到下分别为第$1,2,\ldots,m$层，那么第$u+1$层的半径和高度都必须大于第u层的，因此我们从最下面那层开始枚举，从下往上搜，然后对于每一层，我们从大到小枚举半径r和高度h。
- 假设当前搜索到第u层，已经搜索过的第$u+1\sim m$层的体积之和为v，表面积之和为s，那么当前这层的r_u,h_u的最小值只能是u，因为前面还有$u-1$层，最大值分别为$r_{u+1}-1,h_{u+1}-1$，因为必须比后一层的小。当前还剩余的可用体积为$(n-v)\geq r_u^2h_u$，则$r_u\leq\sqrt{(n-v)/h_u},h_u\leq(n-v)/r_u^2$，当$h_u$取值为$u$时，$r_u$最大，因此$r_u\leq\sqrt{(n-v)/u}$，同理$h_u\leq(n-v)/u^2$。综上，$r_u$的取值范围为$[u,min(r_{u+1}-1,\sqrt{(n-v)/u})]$，$h_u$的取值范围为$[u,min(h_{u+1}-1,(n-v)/u^2)]$。
- 预处理出$minv[u],mins[u]$分别表示$1\sim u$层的最小体积和最小表面积，要求出最小值那么第i层的半径和高度就为i。当$v+minv[u]>n$时说明目前的体积之和加上剩下u层的最小体积也超过要求了，那么直接剪枝；当$s+mins[u]>=res$时说明目前的表面积之和加上剩下u层的最小表面积也不可能比当前最优解更优了，那么直接剪枝。
- 第$1\sim u$层的表面积$S_{1\sim u}=\Sigma_{k=1}^{u}2*r_k*h_k$，体积为$(n-v)=\Sigma_{k=1}^{u}r_k^2*h_k$。对表面积的公式进行转换：

  $S_{1\sim u}=2/r_{u+1}\Sigma_{k=1}^{u}r_k*h_k*r_{u+1}>2/r_{u+1}\Sigma_{k=1}^{u}r_k*h_k*r_k$（根据$r_{u+1}>r_k$）

  因此可以推出公式：$S_{1\sim u}>2(n-v)/r_{u+1}$。所以当$s+2(n-v)/r_{u+1}\geq res$时直接剪枝。

【代码】

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

const int N = 25;
int R[N], H[N];
int minv[N], mins[N];//分别表示前1~u层的最小体积和最小侧面积
int n, m, res = 0x3f3f3f3f;

void dfs(int u, int v, int s)//当前搜索第u层,u+1~m层的总体积为v,总表面积为s
{

    if (v + minv[u] > n) return;//体积不合法
```

```
    if (s + mins[u] >= res) return;//表面积一定无法构成更优解
    if (s + 2 * (n - v) / R[u + 1] >= res) return;//推公式剪枝
    if (!u)//已经搜完m层了
    {
        if (v == n) res = s;//是一个合法解
        return;
    }
    for (int r = min(R[u + 1] - 1, (int)sqrt((n - v) / u)); r >= u; r--)
        for (int h = min(H[u + 1] - 1, (n - v) / (u * u)); h >= u; h--)
        {
            R[u] = r, H[u] = h;
            int t = 0;
            if (u == m) t = r * r;//如果是最下面那层的话那么算上顶部的表面积之和
            dfs(u - 1, v + r * r * h, s + 2 * r * h + t);
        }
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)//第i层的最小值就是半径和高都是i
    {
        minv[i] = minv[i - 1] + i * i * i;//v = r * r * h
        mins[i] = mins[i - 1] + 2 * i * i;//s = 2 * r *h
    }
    R[m + 1] = H[m + 1] = 0x3f3f3f3f;//哨兵
    dfs(m, 0, 0);//从最下面那层开始枚举
    if (res == 0x3f3f3f3f) cout << 0 << endl;
    else cout << res << endl;
    return 0;
}
```