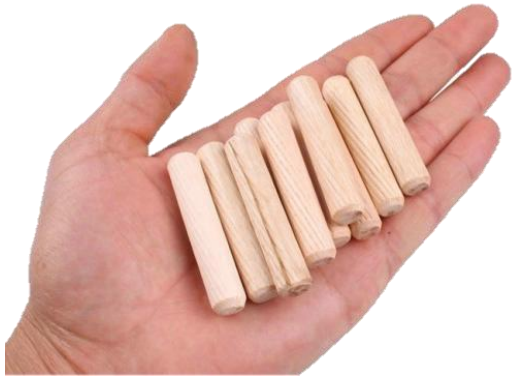
小木棍

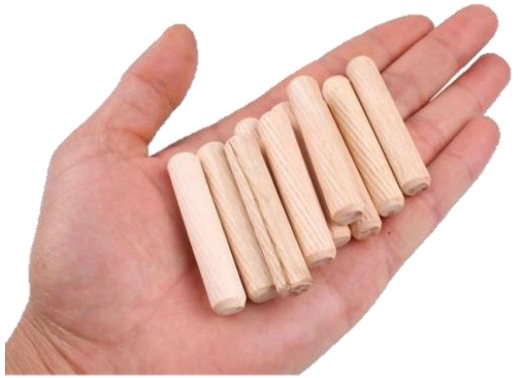


- 给出n根小木棍的长度L_i，已知这n根小木棍原本是由若干根长度相同的原木截断而来，求原木的最小可能长度

$n \le 65$， $L_i \le 50$

Sample input	Sample output
9 //n 5 2 1 5 2 1 5 2 1 //L_i	6
样例解释：4根长度最小为6的原木，可截断为9根给定长度的小木棒	

分析



- 我们研究这组数据：

  46、45、36、36、36、24、19、16、14、13（已排好序）

- 原木的最小可能长度

1. 小木棍中最长那根的长度

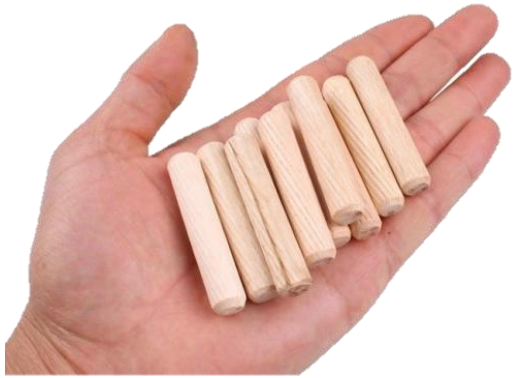
- 最大可能长度

2. 所有小木棍的长度和sum

- 因为原木长度相同

3. 原木的可能长度，只能是sum的因子数

分析



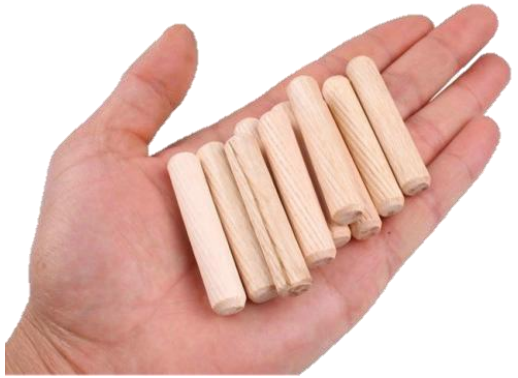
- 我们研究这组数据：

  46、45、36、36、36、24、19、16、14、13（已排好序）

- 我们总结一下目前的思路：

1. 先把所有小木棍按长度从大到小排序
2. 在[maxlen~sumlen]范围内，逐个尝试可能的原木长度（sumlen的因子数）

分析



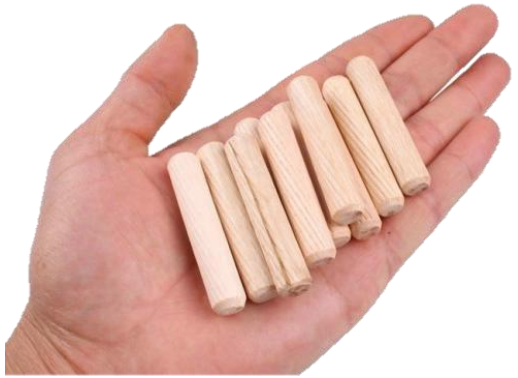
- 那么DFS需要记录几个信息？
- 因为我们要做的是：把n根小木棍拼接成m根原木
- 其中m=n/ans，ans是原木的长度，需要我们去逐个尝试

1. 当前尝试拼接第j根原木
- 那意味着当j>m时搜索成功

2. 当前尝试的原木（已经拼接完成的）长度len
- 那意味着当len=ans时，我们就拼接好了一根原木，此时可以dfs(下一根原木)

3. 然后我们在拼接每一根原木的过程中，势必要记录使用的每一根小木棍序号

分析



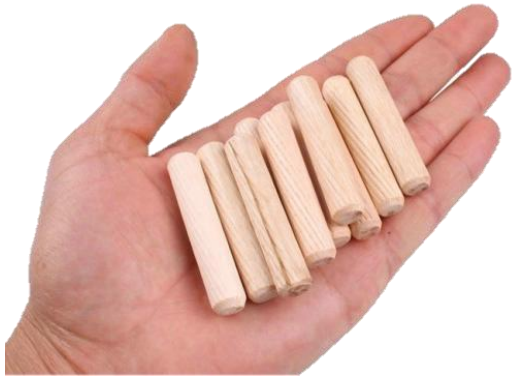
- 目前已知的DFS框架

```
bool dfs(int j, int len, int last)
//尝试拼好第 j根原木，其已经拼接好的长度为 len，  上一根使用的小木棍编号 last
{
    if (j > m) return true;    //原木已经全部拼完，搜索成功

    if (len == ans) return dfs(j + 1, 0, 1);  //第 j根原木已经拼好，去拼下一根原木

    尝试拼接;
}
```

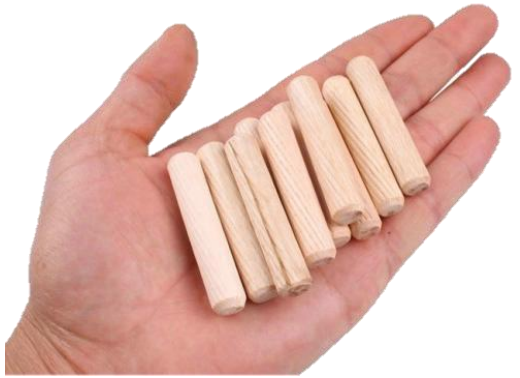
分析



- 尝试拼接

```
for (int i = last; i <= n; i ++)  //从 last开始枚举每一根小木棍
    if (如果第i根小木棍可以拼接上)
    {
        标记第i根小木棍已被尝试;
        尝试下一根小木棍i+1;
        释放i的标记;
    }
return false; //循环结束时所有情况都尝试过，搜索失败
```

分析



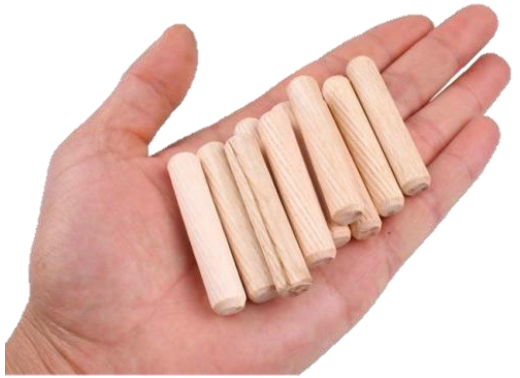
- 还是这组数据：

  46、45、36、36、36、24、19、16、14、13（已排好序）

- 假设当前j=1，len=0，我们会拼接46
- 然后如果len(46+45)≤ans（目前的ans是我们枚举假定的），会拼接继续45
- 反之则需要跳过45，尝试36
- 所以我们这里的尝试下一根小木棍i+1：

```
dfs(j, len + stick[i], i + 1);
//尝试下一根小木棍 i+1;
```

分析



- 还是这组数据：

  46、45、36、36、36、24、19、16、14、13（已排好序）

- 假设当前j=1，len=0，我们会拼接46
- 然后如果len(46+45)≤ans（目前的ans是我们枚举假定的），会拼接继续45
- 反之则需要跳过45，尝试36
- 所以我们这里的尝试下一根小木棍i+1：

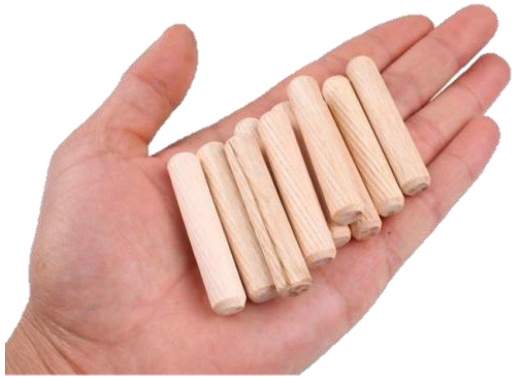
```
if (dfs(j, len + stick[i], i + 1)) return true;
//尝试下一根小木棍 i+1;
```

√

目前的代码

```
bool dfs(int j, int len, int last)
{
    if (j > m) return true;
    if (len == ans) return dfs(j + 1, 0, 1);
    for (int i = last; i <= n; i ++)
        if (!flag[i] && len + stick[i] <= ans)
        {
            flag[i] = 1;
            if (dfs(j, len + stick[i], i + 1)) return true;
            flag[i] = 0;
        }
    return false;
}
```

继续分析



- 还是这组数据：

  46、45、36、36、36、24、19、16、14、13（已排好序）

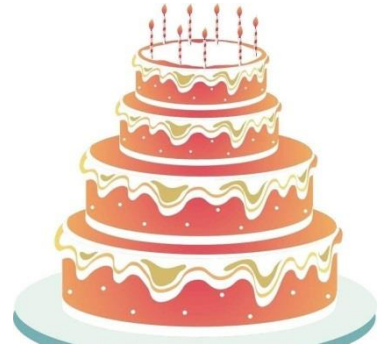
- 然后我们还可以加几个剪枝进来：

1. 一旦发现36失败，下一次可以直接跳过36，就是相同的长度只试一次
2. 如果当前还剩的原木长度小于要尝试的小木棍长度，就直接返回
3. 每次拼原木时，只要当前尝试的不是第一根小木棍，就不要从头开始尝试，而是应该从刚才所试小木棍的下一条开始。因此应该增加一个变量用于记录当前已尝试的最后一根小木棍的长度
4. 如果发现len<0，也可以直接返回
5. ……

参考代码

- 略

生日蛋糕

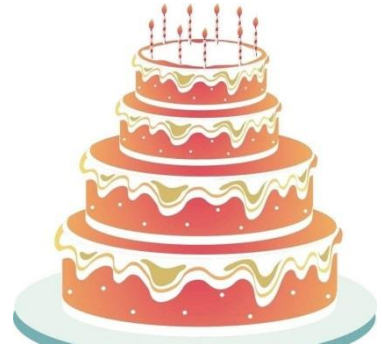


- 制作一个体积为nπ的m层生日蛋糕，每层都是一个圆柱体
- 设从下往上数第i层蛋糕是半径为r_i，高度为h_i的圆柱体。当i<m时，要求$r_i>r_{i+1}$且$h_i>h_{i+1}$
- 由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕侧面积sπ最小。对给出的n和m，找出蛋糕的制作方案（适当的r_i和h_i的值），使sπ最小

n≤20,000，m≤15

Sample input	Sample output
10 2 //n，m	68 //s

分析



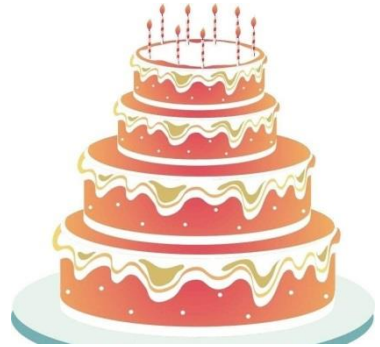
- 因为下面的蛋糕大小制约了上面的，所以从下往上搜索

- 考虑对于每一个状态需要记录的变量：还剩余i层没有制作完成，前一层蛋糕的半径为r，高度为h，当前的表面积之和s，剩余的体积v

```
void dfs(int x, int r, int h, int s, int v)
//还剩下 x层，前一层半径 r，高度 h，当前已有面积 S，剩余体积 V
{


}
```

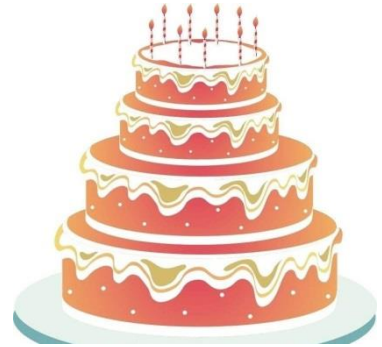
分析



- 然后转移枚举下一层的半径和高

```
for(int i = x; i < r; i ++)          //枚举当前层的半径
    for(int j = x; j < h; j ++)      //枚举当前层的高度
        dfs(x - 1, i, j, s + 2 * i * j, v - i * i * j);
```

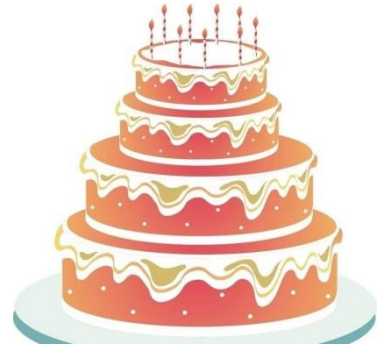
分析



- 初始的时候，从最底层开始搜索

```
for(int i = m; i * i * m <= n; i ++)  //枚举半径
    for(int j = m; i * i * j <= n; j ++)  //枚举高
        dfs(m - 1, i, j, i * i + 2 * i * j, n - i * i * j);
        //从最底下一层的半径和高度开始 dfs
```

可行性剪枝



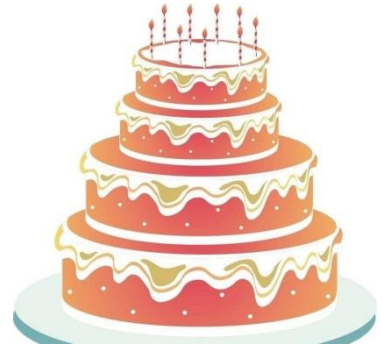
1. 可行性剪枝

当搜索到一个状态时，如果可以判断这个状态之后的状态都不合法，则直接退出当前状态

- 如果当前的侧面积+剩余部分侧面积的下限≥当前最优解，应该返回

```
if (s + 2 * v / i > ans) continue;   //可行性剪枝
```

最优性剪枝



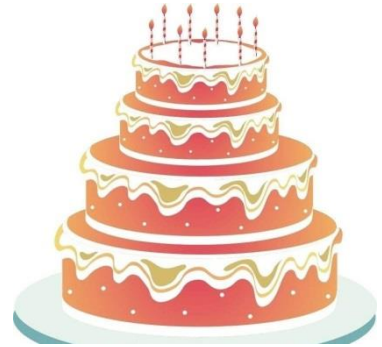
2. 最优性剪枝

当搜索到一个状态时，如果可以判断这个状态之后的状态都不会比当前的最优状态更优，则直接退出当前状态

- 剩余的体积太少，剩余部分最小化也会超过nπ
- 我们先预处理好每层的体积，保存在a[ ]中

```
for(int i = 1; i <= m; i ++)
    a[i] = a[i-1] + i * i * i;
```

最优性剪枝



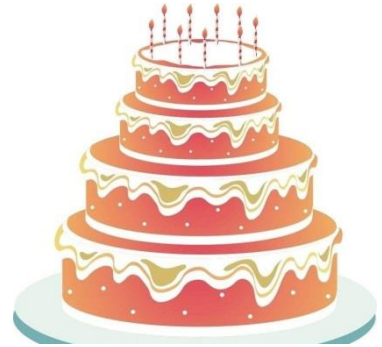
2. 最优性剪枝

当搜索到一个状态时，如果可以判断这个状态之后的状态都不会比当前的最优状态更优，则直接退出当前状态

- 剩余的体积太少，剩余部分最小化也会超过nπ

```
if (a[x] > v) return;
//剩余体积太小，最小化也会超过
```

最优性剪枝



2. 最优性剪枝

当搜索到一个状态时，如果可以判断这个状态之后的状态都不会比当前的最优状态更优，则直接退出当前状态

- 剩余的体积太多，剩余部分最大化也达不到nπ

```
if (x * (r - 1) * (r - 1) * (h - 1) < v) return;
//剩余体积太大，最大化也达不到
```