程序设计实习

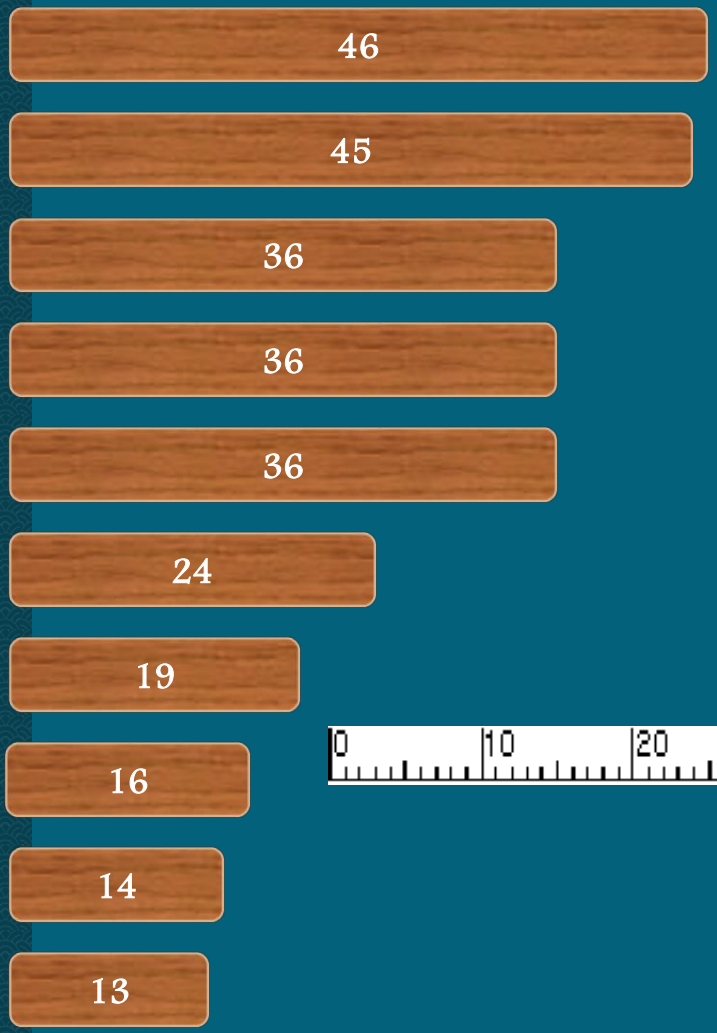
深度优先搜索

郭炜　刘家瑛

拯救少林神棍(POJ1011)

据说，少林寺的镇寺之宝，是救秦王李世民的十三棍僧留下的若干根一样长的棍子。

拯救少林神棍(POJ1011)



棍子长度：95

拯救少林神棍(POJ1011)

用程序解决：

输入：N节木棒的长度。
输出：能拼成的最小的棍子长度。

解题思路

尝试(枚举）什么？

枚举所有可能的棍子长度。从最长的那根木棒的长度一直枚举到木棒长度总和的一半，对每个假设的棍子长度，试试看能否拼齐若干根棍子。

真的要每个长度都试吗？

对于不是木棒总长度的因子的长度，可以直接否定，不需尝试。

解题思路

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子？

一根一根地拼棍子。如果拼好前i根棍子，结果发现第i+1根无论如何拼不成了，那么就要推翻第i根的拼法，重拼第i根.....直至有可能推翻第1根棍子的拼法。

解题思路

本题的“状态”是什么？

状态可以是一个二元组 (R, M)

R：还没被用掉的木棒数目。

M： 当前正在拼的棍子还缺少的长度。

初始状态和搜索的终止状态（解状态）是什么？

假设共有N节木棒，假定的棍子长度是L:

初始状态： (N, L)

终止状态： (0, 0)

所谓“成功拼出若干根长度为L的棍子”，就是要在状态空间中找到一条从(N,L)到(0,0)的路径

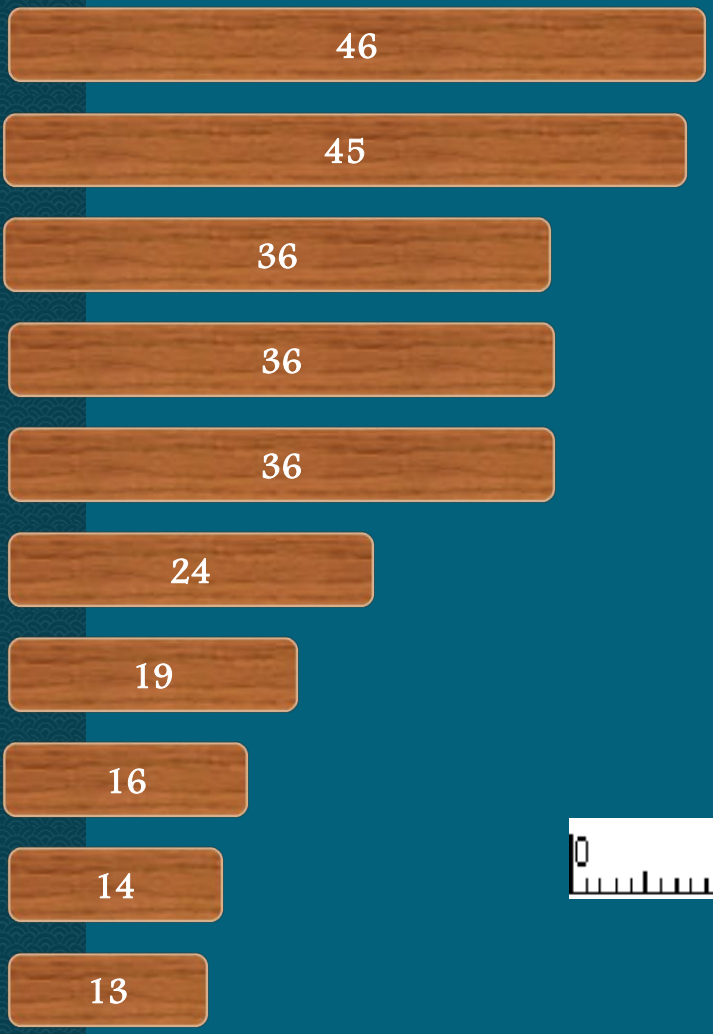
解题思路

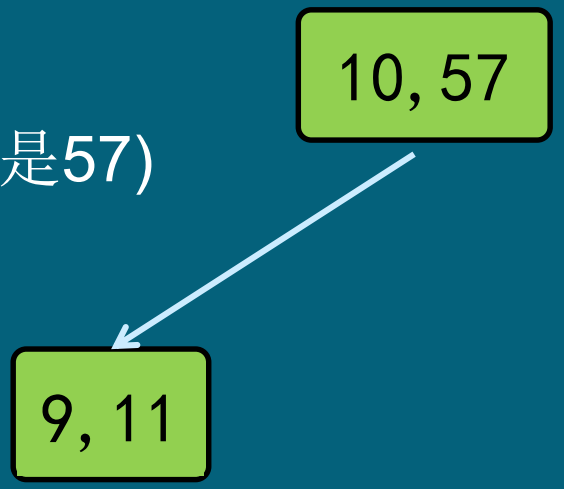
“状态”之间如何转移？

(R, M) → 拼接一节长度为S的木棒 → (R-1, M-S)

(R-1, M-S) → 拆掉一节长度为S的木棒 → (R, M)

以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

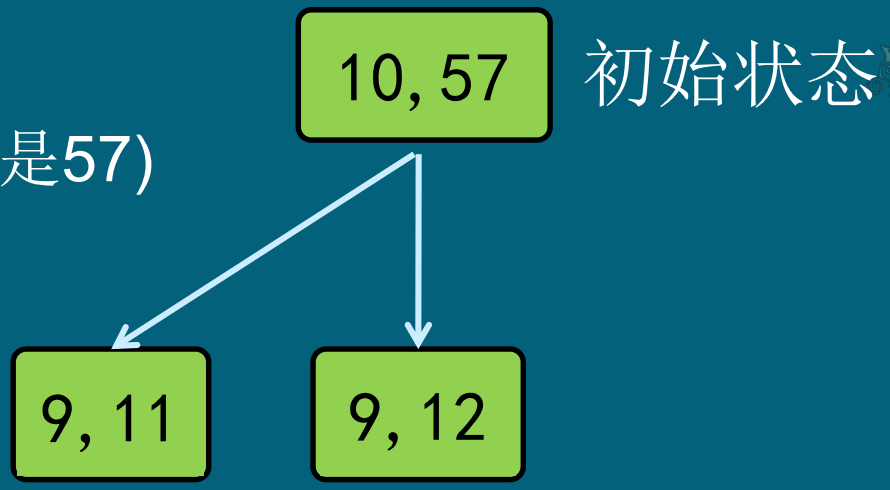


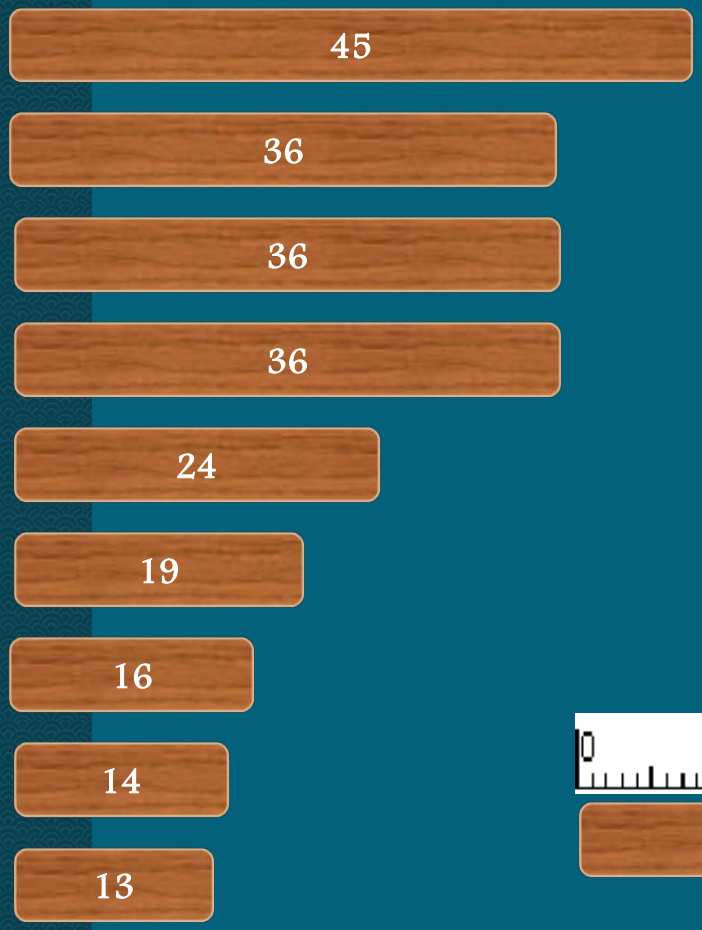


初始状态

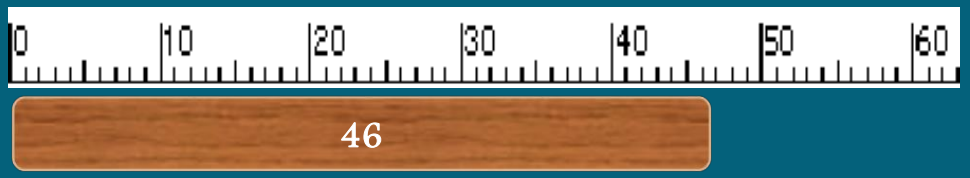
棍子长度 = 57

以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

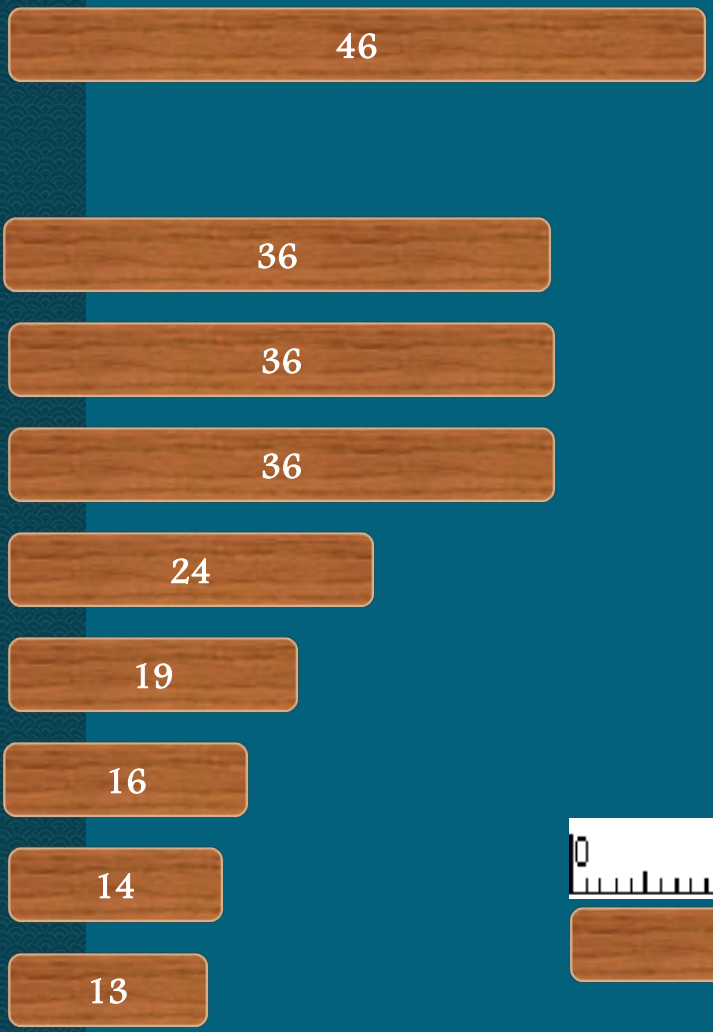


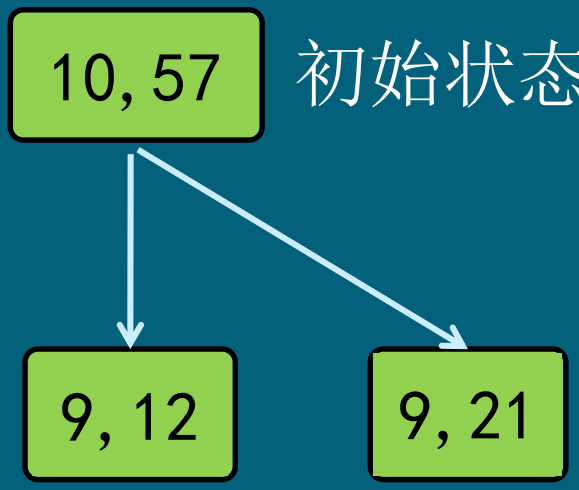


棍子长度 = 57

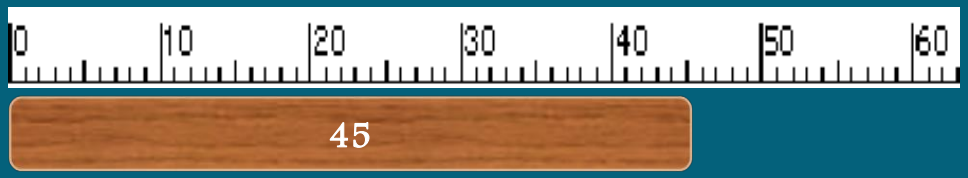


以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

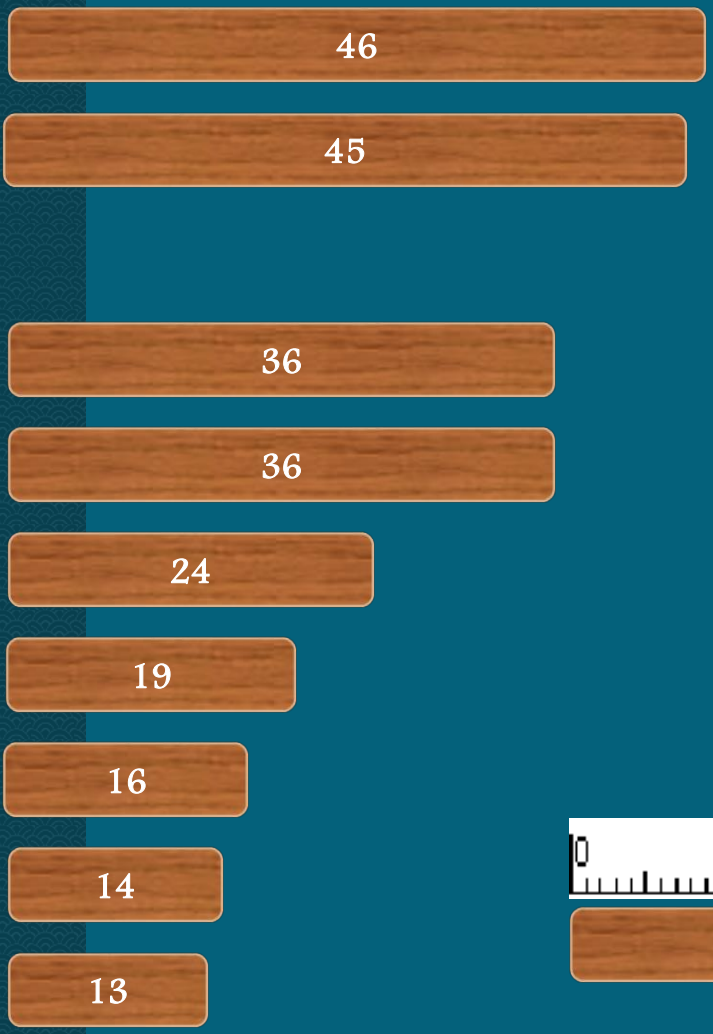


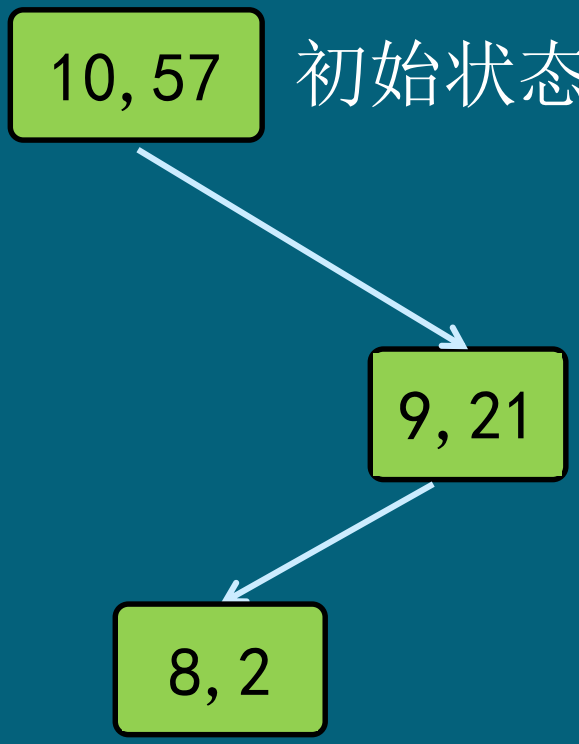


棍子长度 = 57



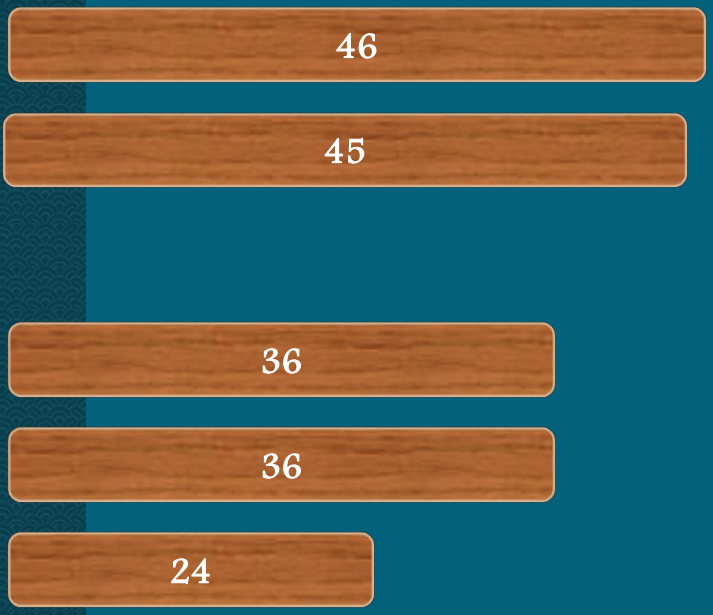
以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

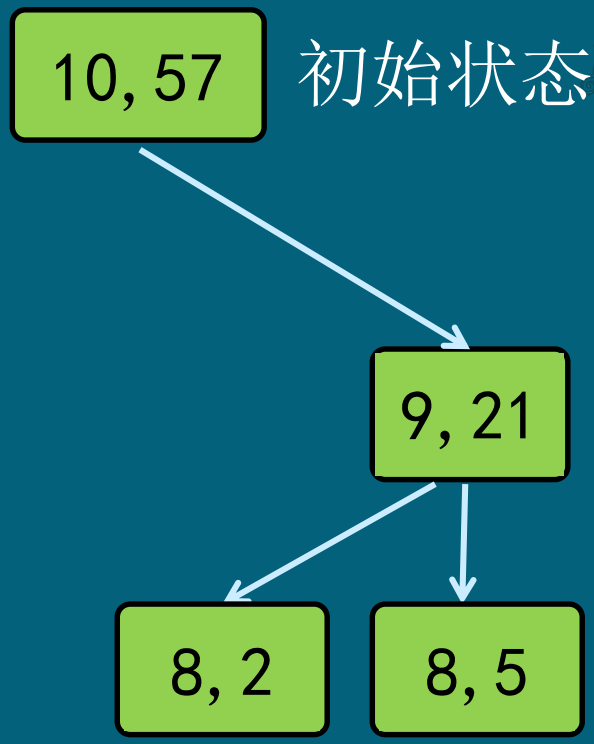




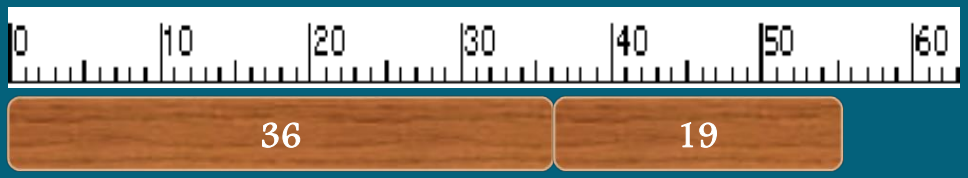
棍子长度 = 57

以N=10,L=57为例:
（10节木棒，假设棍子长度是57)





棍子长度 = 57



bool Dfs(int R, int M)

表示: 当前有R根未用木棒，而且当前正在拼的那根棍子比假定的棍子长度还少M, 求在这种情况下能全部否拼成功。

Dfs的基本递推关系：

```
bool  Dfs(int R, int M) {
   if( R == 0 &&  M == 0)
                return true; //拼接任务完成
   如果能找到一根长度不超过M的木棒, 假设长为S，
   拼在当前棍子上，然后
        Dfs(R – 1,M - S);
   如果找不到：
        return false;
}
```

```
#include <iostream>
#include <memory.h>
#include <stdlib.h>
#include <vector>
#include <algorithm>
using namespace std;
int N;
int L;
vector<int> anLength;
int anUsed[65];//是否用过的标记
int i,j,k;
int Dfs(int R, int M);
```

```
int main()
{
        while(1) {
                cin >> N;
                if( N == 0 )
                        break;
                int nTotalLen = 0;
                anLength.clear();
                for( int i = 0; i < N; i ++ ) {
                        int n;
                        cin >> n;
                        anLength.push_back(n);
                        nTotalLen += anLength[i];
                }
                sort(anLength.begin(),anLength.end(),
                greater<int>()); //要从长到短进行尝试
```

```
        for( L = anLength[0]; L <= nTotalLen / 2; L ++ ) {
                if( nTotalLen % L)
                        continue;
                memset( anUsed, 0,sizeof(anUsed));
                if( Dfs( S,L)) {
                        cout << L << endl;
                        break;
                }
        }
        if( L > nTotalLen / 2 )
                cout << nTotalLen << endl;
    } // while
    return 0;
}
```

```
int Dfs( int R, int M)  {
// M表示当前正在拼的棍子和 L 比还缺的长度
        if( R == 0 && M == 0 )
                return true;
        if( M == 0 ) //一根刚刚拼完
                M = L;  //开始拼新的一根
        for( int i = 0;i < N;i ++) {
                if( !anUsed[i] && anLength[i] <= M) {
                        anUsed[i] = 1;
                        if ( Dfs( R - 1,
                                M - anLength[i]))
                                return true;
                        else
                            anUsed[i] = 0;//说明本次不能用第i根
                                        //第i根以后还有用
                }
        }
        return false;           }
```

敢问施主，要多久，
才能拼好呢？
有100多节木棒呢！

怎么也得…
10000年吧

咣当！！

搜索题，要解决一个问题：如何剪枝。

即尽可能快地发现没有希望的状态，避免从没希望的状态往下继续尝试。

第一种剪枝方案：

不要在同一个位置多次尝试相同长度的木棒。

即：

如果某次拼接选择长度为S 的木棒，导致最终失败，则在同一位置尝试下一根木棒时，要跳过所有长度为S 的木棒。

第二种剪枝方案：

如果由于以后的拼接失败，需要重新调整第i根棍子的拼法，则不会考虑替换第i根棍子中的第一根木棒（换了也没用）。如果在不替换第一根木棒的情况下怎么都无法成功，那么就要推翻第i-1根棍子的拼法。如果不存在第i-1根棍子，那么就推翻本次假设的棍子长度，尝试下一个长度。

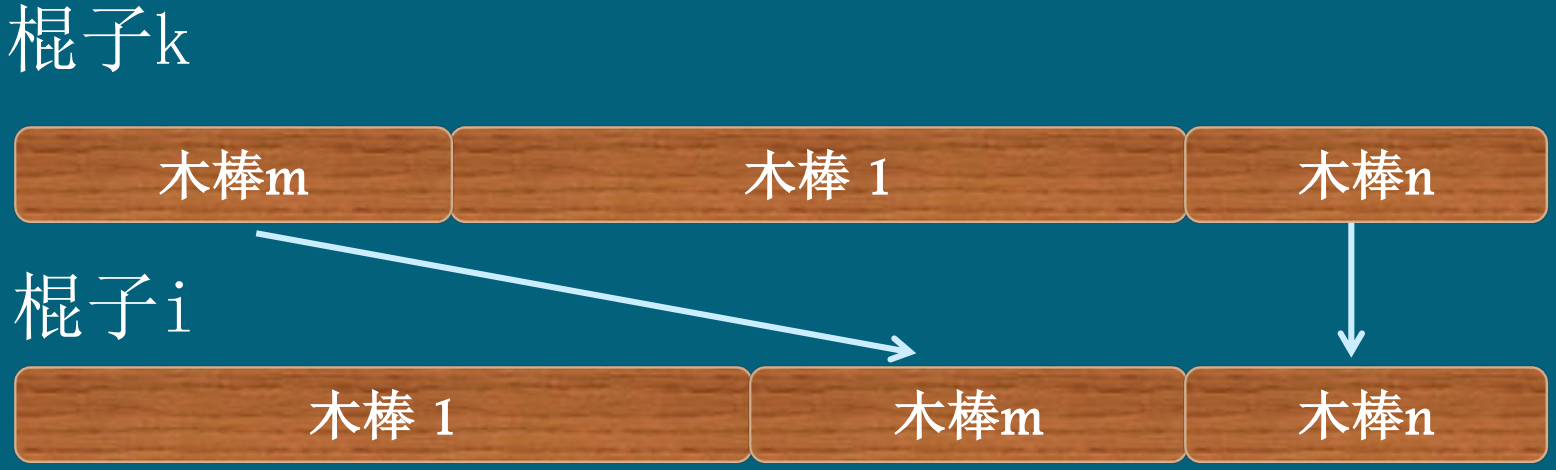
若棍子i如下拼法导致最后不能成功：

可以考虑把木棒2,3换掉重拼棍子i,但是把2，3都去掉后，换1是没有意义的。

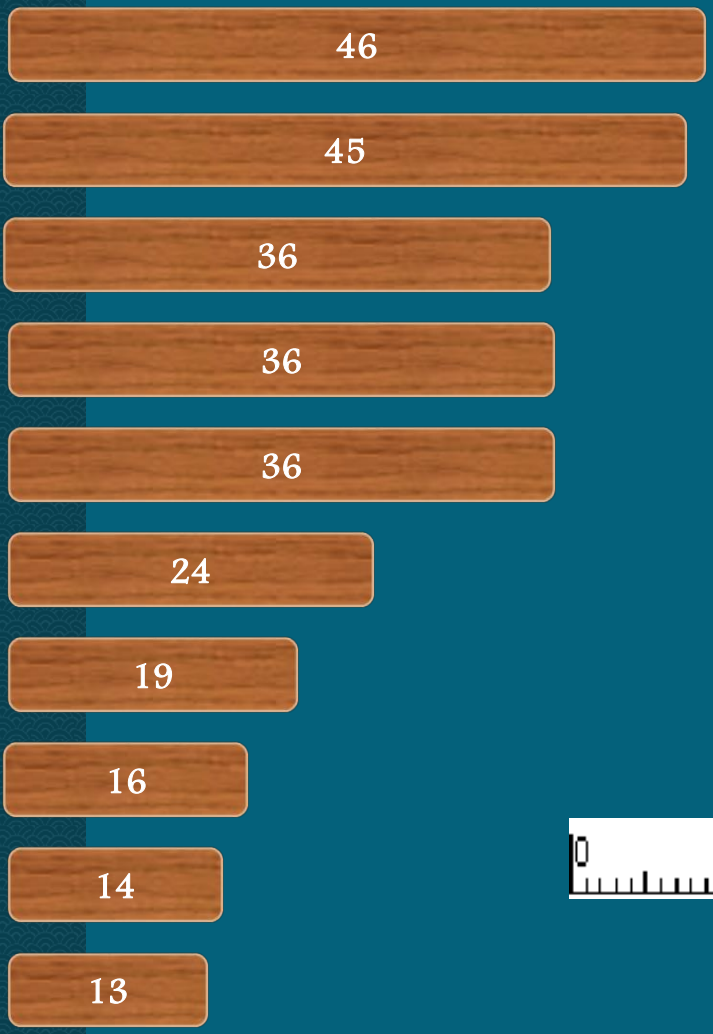
第二种剪枝方案：

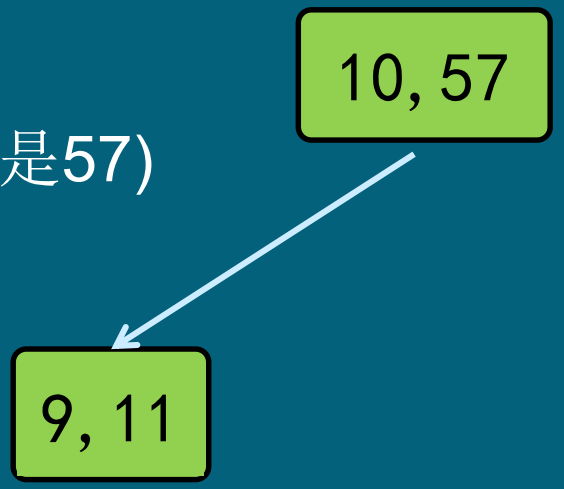
为什么替换第i根棍子的第一根木棒是没用的？

因为假设替换后能全部拼成功，那么这被换下来的第一根木棒，必然会出现在以后拼好的某根棍子k中。那么我们原先拼第i根棍子时，就可以用和棍子k同样的构成法来拼，照这种构成法拼好第i根棍子，继续下去最终也应该能够全部拼成功。



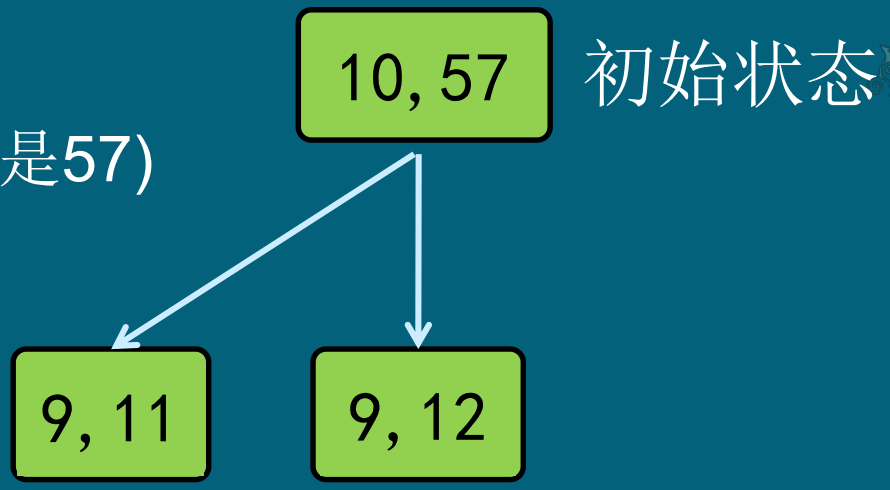
以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

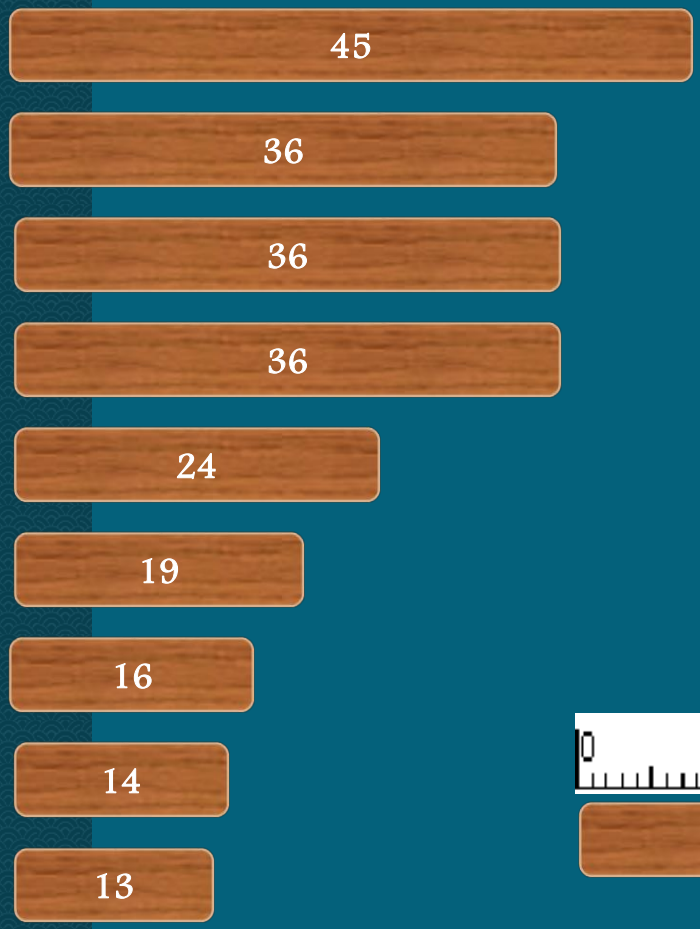




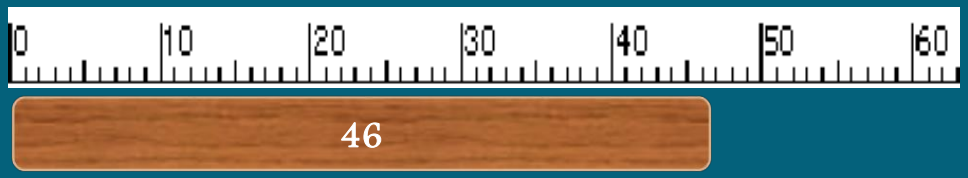
棍子长度 = 57

以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

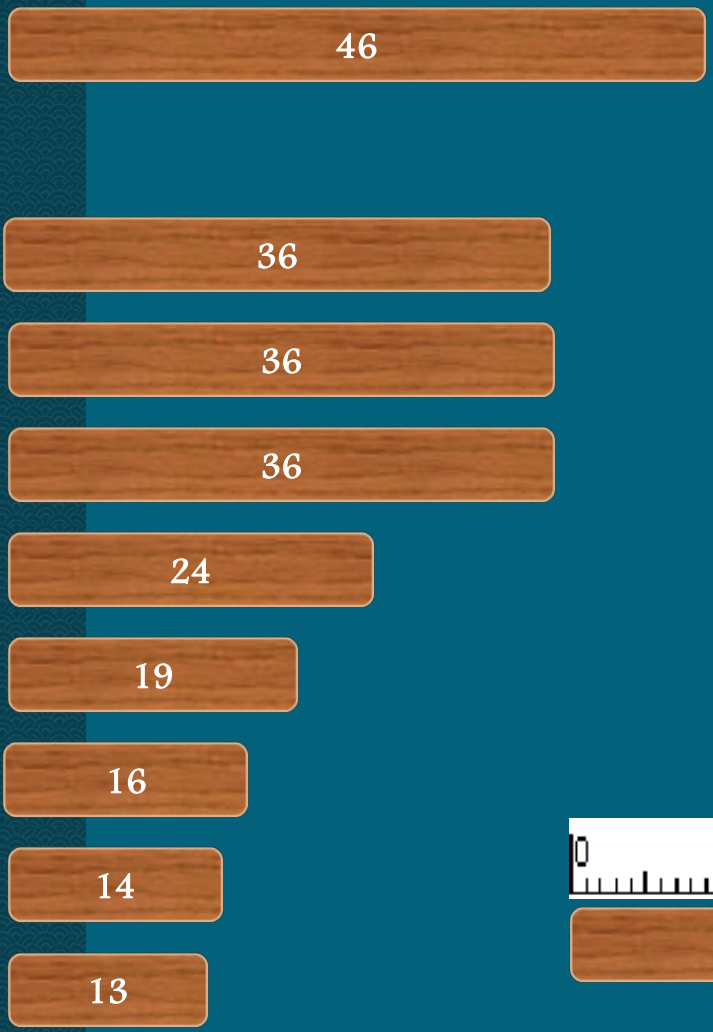


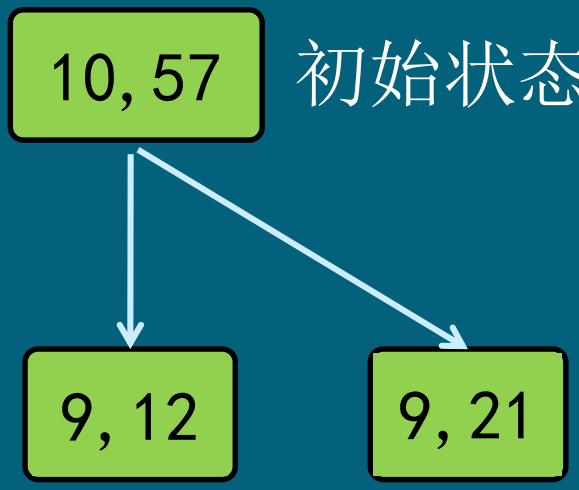


棍子长度 = 57

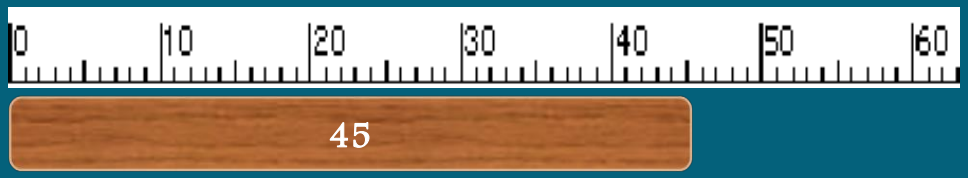


以N=10,L=57为例:
（10节木棒，假设棍子长度是57)

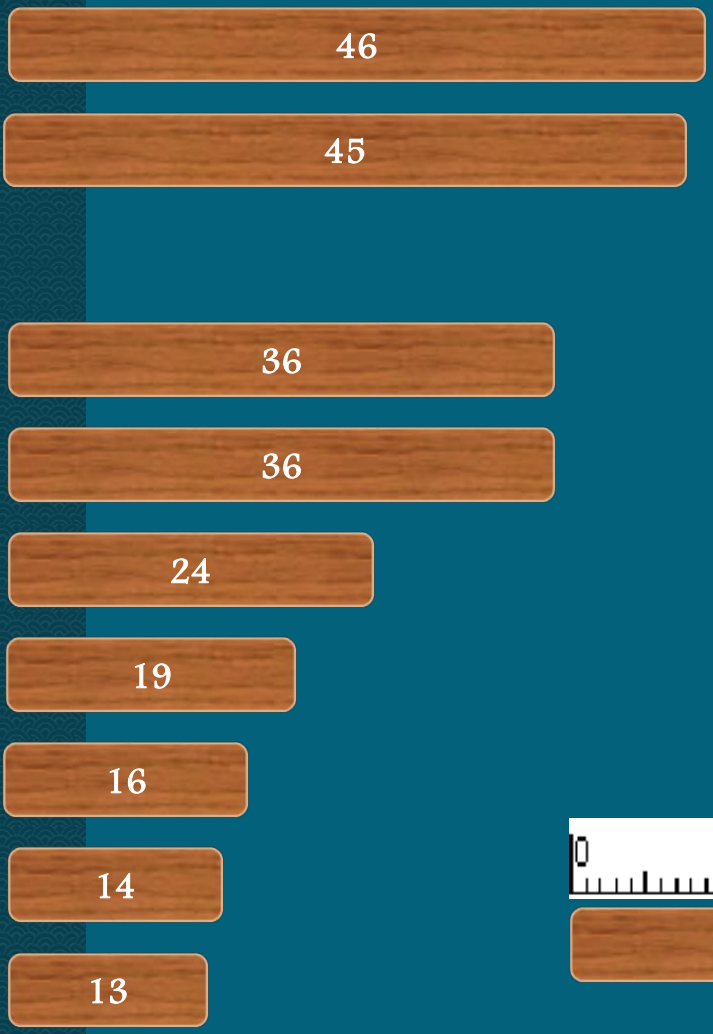


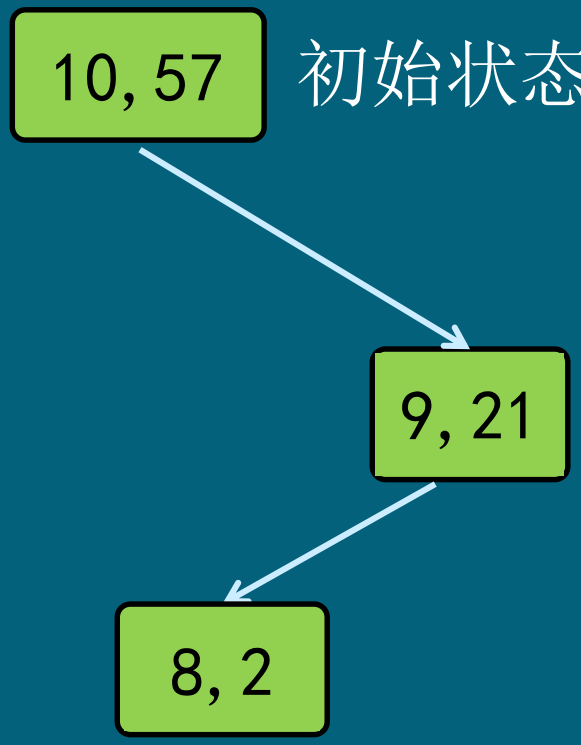


棍子长度 = 57



以N=10,L=57为例:
（10节木棒，假设棍子长度是57)





棍子长度 = 57

```
int Dfs( int R, int M) {
// M表示当前正在拼的棍子和 L 比还缺的长度
        if( R == 0 && M == 0 )
                return true;
        if( M == 0 ) //一根刚刚拼完
                M = L;  //开始拼新的一根
        for( int i = 0;i < N;i ++) {
                if( !anUsed[i] && anLength[i] <= M) {
                        if( i > 0 ) {
                                if( anUsed[i-1] == false
                                  && anLength[i] == anLength[i-1])
                                        continue; //剪枝1
                        }
                        anUsed[i] = 1;
```

```
                    if ( Dfs( R - 1,
                             M - anLength[i]))
                            return true;
                    else {
                            anUsed[i] = 0;//说明本次不能用第i根
                                          //第i根以后还有用
                            if( M == L)
                                    return false;//剪枝2
                    }
            }
    }
    return false;
}
```

剪枝3：

不要希望通过仅仅替换已拼好棍子的最后一根木棒就能够改变失败的局面。

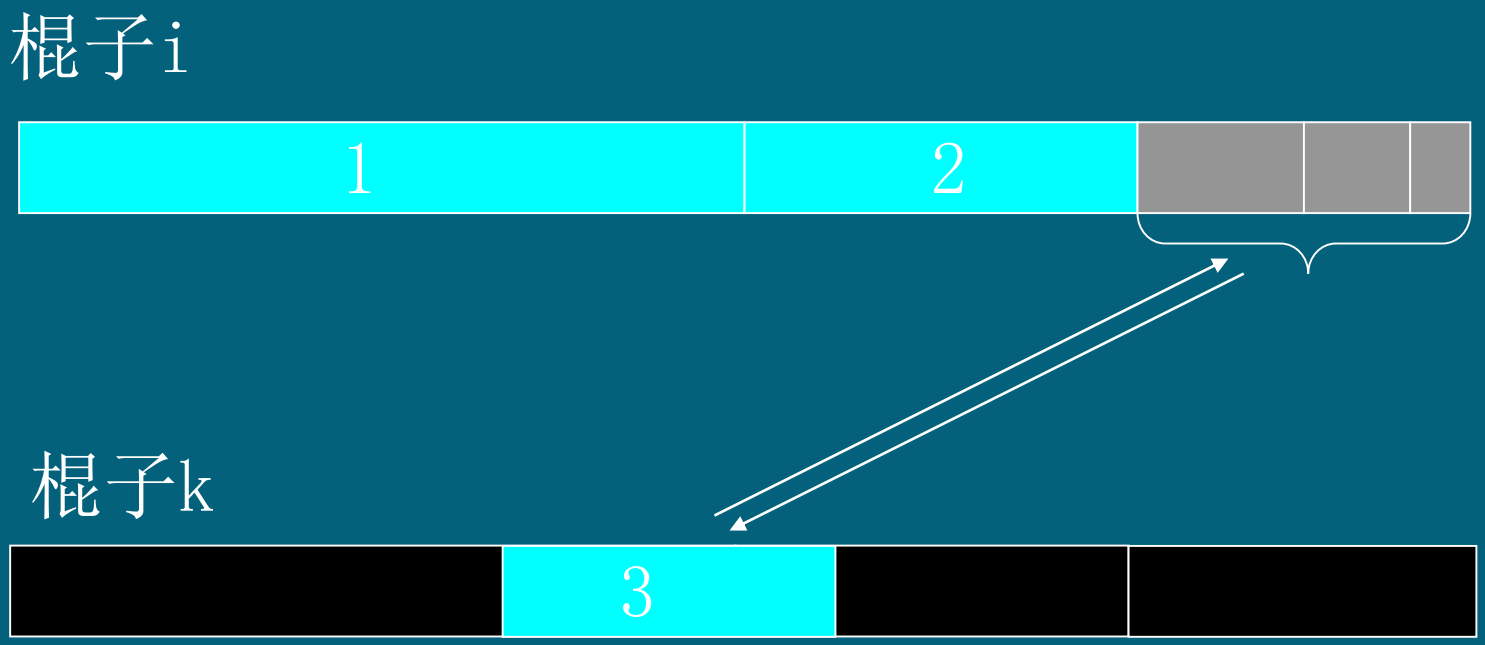
假设由于后续拼接无法成功，导致准备拆除的某根棍子如下：

棍子i

1	2	3

将 3 拆掉，留下的空用其他短木棒来填，是徒劳的

剪枝3：

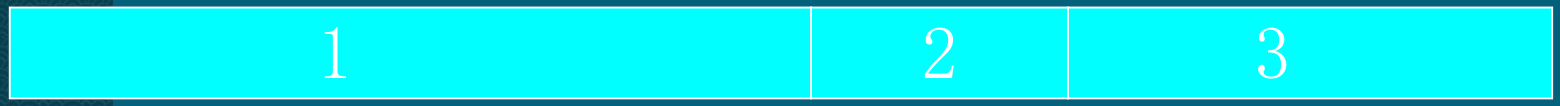


假设替换3后最终能够成功，那么3必然出现在后面的某个棍子k里。将棍子k中的3和棍子i中用来替换3的几根木棒对调，结果当然一样是成功的。这就和i原来的拼法会导致不成功矛盾

剪枝 4：

拼每一根棍子的时候，应该确保已经拼好的部分，长度是从长到短排列的，即拼的过程中要排除类似下面这种情况：

未完成的棍子i



木棒3 比木棒2长，这种情况的出现是一种浪费。因为要是这样往下能成功，那么2，3 对调的拼法肯定也能成功。由于取木棒是从长到短的，所以能走到这一步，就意味着当初将3放在2的位置时，是不成功的

剪枝 4：

排除办法：每次找一根木棒的时候，只要这不是一根棍子的第一条木棒，就不应该从下标为0的木棒开始找，而应该从刚刚(最近）接上去的那条木棒的下一条开始找。这样，就不会往2后面接更长的3了

为此，要设置一个全局变量 nLastStickNo ，记住最近拼上去的那条木棒的下标。

```
int Dfs( int nUnusedSticks, int nLeft)
// nLeft表示当前正在拼的棍子和 L 比还缺的长度
{
        if( nUnusedSticks == 0 && nLeft == 0 )
                return true;
        if( nLeft == 0 ) //一根刚刚拼完
                nLeft = L;  //开始拼新的一根
        int nStartNo = 0;
        if( nLeft != L ) //剪枝4
                nStartNo = nLastStickNo + 1;
        for( int i = nStartNo;i < S;i ++) {
                if( !anUsed[i] && anLength[i] <= nLeft) {
                        if( i > 0 ) {
                                if( anUsed[i-1] == false
                                   && anLength[i] == anLength[i-1])
                                        continue; //剪枝1
                        }
                        anUsed[i] = 1;  nLastStickNo = i;
```

```
                    if ( Dfs( nUnusedSticks - 1,
                              nLeft - anLength[i]))
                        return true;
                    else {
                        anUsed[i] = 0;//说明本次不能用第i根
                                      //第i根以后还有用

                        if( anLength[i] == nLeft || nLeft == L)
                            return false;//剪枝3、2
                    }
                }
            }
        return false;
}
```

总结:

1) 要选择合适的搜索顺序

如果一个任务分为A、B、C.....等步骤，要优先尝试可能性少的步骤。

2) 要发现表面上不同，实质相同的重复状态，避免重复的搜索

3) 要根据实际问题发掘剪枝方案