# A 钢管切割 1

## 题目描述

给定一段长度为n英寸的钢管和一个价格表，该价格表表示长度为 i（i=1,2,...,n）英寸的钢管的价格为pi。求钢管切割方案，使得总销售价格最大，注意**钢管的长度必须为整英寸**。

## 题目分析

这是一道经典的动态规划问题。动态规划（dynamic programming）与分治法类似。分治策略将问题划分为互不相交的子问题，递归求解子问题，再将子问题进行组合，求解原问题。动态规划应用于子问题重叠的情况，在这种情况下，分治法将会对重叠问题进行多次重复求解，而动态规划对每个子问题只求解一次。

现在对于一根长度为n的钢条，求它能获得的最大效益r。

注意到，每一次切割后将得到两根钢铁，下一步，又将这两根钢条看作两个独立的钢铁切割问题。通过组合子问题的最优解，选取组合收益最大者，构成原问题的最优解。钢条切割问题满足最优子结构性质：问题的最优解由相关子问题的最优解组合而成，而这些子问题都可以独立求解。
对钢条问题，我们可以再进行简化。对于每次切割后得到的两段钢条，我们只对第二段进行再次切割，对第一段不再切割。

这里面注意状态转移方程的书写是关键。

## 代码实现

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define inf 0x3f3f3f3f
using namespace std;
long long  p[10010]; //注意题中数据范围
long long  r[10010];
long long  s[10010];
int n;

int bottom_up_cut_rod(long long p[], int n) {
    for (int i = 0; i <= n; i++)
        r[i] = 0; //记录每个长度的效益最大值
    long long q;
    for (int j = 1; j <= n; j++) {
        q = -inf;
        for (int i = 1; i <= j; i++) {
            if (q < p[i] + r[j - i]) { //状态转移方程
                q = p[i] + r[j - i];
                s[j] = i;
            }
        }
        r[j] = q;
    }
    return r[n];
}
```

```
int print_cut(long long p[], int n, long long s[]) {
    int sum = 0;
    long long k[10010];
    while (n > 0) {

        k[sum] = s[n];
        sum += 1;
        n = n - s[n];
    }
    printf("%d\n", sum); //共切割成几段
    for (int i = 0; i < sum; i++) {
        printf("%lld ", k[i]); //每段的长度
    }
    cout << endl;
}

int main() {
    int num;
    cin >> num;
    for (int i = 1; i <= num; i++)
        scanf("%lld", &p[i]);
    cout  << bottom_up_cut_rod(p, num) << endl;
    print_cut(p, num, s);

    cout << endl;
    return 0;
}
```