



# 【华为OD机考 统一考试机试C卷】伐木工 (C++ Java JavaScript Python C语言)

## 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

**真题目录：**华为OD机考机试 真题目录 (C卷 + D卷 + B卷 + A卷) + 考点说明

**专栏：**2023华为OD机试( B卷+C卷+D卷) (C++JavaJSPy)

**华为OD面试真题精选：**华为OD面试真题精选

**在线OJ：**点击立即刷题，模拟真实机考环境

## 题目描述

一根X米长的树木，伐木工切割成不同长度的木材后进行交易，交易价格为每根木头长度的乘积。规定切割后的每根木头长度都为正整数；也可以不切割，直接拿整根树木进行交易。

请问伐木工如何尽量少的切割，才能使收益最大化？

## 输入描述

木材的长度 ( $X \leq 50$ )

## 输出描述

输出最优收益时的各个树木长度，以空格分隔，按升序排列

## 用例

输入

1 | 10

输出

1 | 3 3 4

## 说明

一根2米长的树木，伐木工不切割，为 $2 * 1$ ，收益最大为2

一根4米长的树木，伐木工不需要切割为 $2 * 2$ ，省去切割成本，直接整根树木交易，为 $4 * 1$ ，收益最大为4

一根5米长的树木，伐木工切割为 $2 * 3$ ，收益最大为6

一根10米长的树木，伐木工可以切割方式一：3，4，4，也可以切割为方式二：3，2，2，3，但方式二伐木工多切割一次，增加切割成本却买了一样的价格，因此并不是最优收益。

## 解题思路

### C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6  // 定义函数，计算给定长度的最大乘积分割
7  vector<int> getMaxProfit(int length) {
8      // dp数组用于存储每个长度的最大乘积
9      vector<int> dp(length + 1);
10     // cutTimes数组用于存储每个长度的最佳切割次数
11     vector<int> cutTimes(length + 1);
12     // lastCut数组用于存储每个长度的最后一次切割长度
13     vector<int> lastCut(length + 1);
14
15     // 遍历每个长度
16     for (int i = 1; i <= length; ++i) {
17         // 初始化dp和lastCut数组
18         dp[i] = lastCut[i] = i;
19         // 遍历所有可能的切割长度
20         for (int j = 1; j < i; ++j) {
21             // 计算当前切割长度的乘积
22             int product = dp[i - j] * j;
23             // 如果当前乘积大于已知的最大乘积，更新最大乘积和最佳切割长度
24             if (product > dp[i]) {
```

```
25         lastCut[i] = j;
26         dp[i] = product;
27         cutTimes[i] = cutTimes[i - j] + 1;
28     }
29     // 如果当前乘积等于已知的最大乘积, 但切割次数更少, 更新最佳切割长度和切割次数
30     else if (product == dp[i] && cutTimes[i] > cutTimes[i - j] + 1) {
31         lastCut[i] = j;
32         cutTimes[i] = cutTimes[i - j] + 1;
33     }
34 }
35 }
36
37 // 创建一个vector来存储结果
38 vector<int> results;
39 // 从最大长度开始, 每次减去最佳切割长度, 直到长度为0
40 while (length > 0) {
41     // 将最佳切割长度添加到结果的开头
42     results.insert(results.begin(), lastCut[length]);
43     // 更新长度
44     length -= lastCut[length];
45 }
46 // 对结果进行排序
47 sort(results.begin(), results.end());
48 // 返回结果
49 return results;
50 }
51
52 int main() {
53     // 读取输入的长度
54     int length;
55     cin >> length;
56     // 调用getMaxProfit方法计算最大利润, 并将结果存储在一个vector中
57     vector<int> results = getMaxProfit(length);
58     // 遍历结果并打印
59     for (int i : results) {
60         cout << i << " ";
61     }
62     return 0;
63 }
```

## Java

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Scanner;
4
5 public class Main {
6
7     public static void main(String[] args) {
8         // 创建一个扫描器来读取用户输入
9         Scanner scanner = new Scanner(System.in);
10        // 读取输入的长度
11        int length = scanner.nextInt();
12        // 调用getMaxProfit方法计算最大利润, 并将结果存储在一个ArrayList中
13        ArrayList<Integer> results = getMaxProfit(length);
14        // 遍历结果并打印
15        for (int i : results) {
16            System.out.print(i + " ");
17        }
18    }
19
20    private static ArrayList<Integer> getMaxProfit(int length) {
21        // dp数组用于存储每个长度的最大乘积
22        int[] dp = new int[length + 1];
23        // cutTimes数组用于存储每个长度的最佳切割次数
24        int[] cutTimes = new int[length + 1];
25        // lastCut数组用于存储每个长度的最后一次切割长度
26        int[] lastCut = new int[length + 1];
27
28        // 遍历每个长度
29        for (int i = 1; i <= length; i++) {
30            // 初始化dp和lastCut数组
31            dp[i] = lastCut[i] = i;
32            // 遍历所有可能的切割长度
33            for (int j = 1; j < i; j++) {
34                // 计算当前切割长度的乘积
35                int product = dp[i - j] * j;
36                // 如果当前乘积大于已知的最大乘积, 更新最大乘积和最佳切割长度
37                if (product > dp[i]) {
38                    lastCut[i] = j;
39                }
40            }
41        }
42    }
43}
```

```
40         dp[i] = product;
41         cutTimes[i] = cutTimes[i - j] + 1;
42     }
43     // 如果当前乘积等于已知的最大乘积, 但切割次数更少, 更新最佳切割长度和切割次数
44     else if (product == dp[i] && cutTimes[i] > cutTimes[i - j] + 1) {
45         lastCut[i] = j;
46         cutTimes[i] = cutTimes[i - j] + 1;
47     }
48 }
49 }
50
51 // 创建一个ArrayList来存储结果
52 ArrayList<Integer> results = new ArrayList<>();
53 // 从最大长度开始, 每次减去最佳切割长度, 直到长度为0
54 while (length > 0) {
55     // 将最佳切割长度添加到结果的开头
56     results.add(0, lastCut[length]);
57     // 更新长度
58     length -= lastCut[length];
59 }
60 // 对结果进行排序
61 Collections.sort(results);
62
63 // 返回结果
64 return results;
65 }
```

## javaScript

```
1 // 引入readline模块用于读取输入
2 const readline = require('readline');
3
4 // 创建readline接口
5 const rl = readline.createInterface({
6     input: process.stdin,
7     output: process.stdout
8 });
9
10 // 定义函数getMaxProfit, 计算给定长度的最大乘积分割
11
```

```
11 const getMaxProfit = (length) => {
12     // dp数组用于存储每个长度的最大乘积
13     const dp = new Array(length + 1).fill(0);
14     // cutTimes数组用于存储每个长度的最佳切割次数
15     const cutTimes = new Array(length + 1).fill(0);
16     // lastCut数组用于存储每个长度的最后一次切割长度
17     const lastCut = new Array(length + 1).fill(0);
18
19     // 遍历每个长度, 从1到length
20     for (let i = 1; i <= length; i++) {
21         // 初始化dp和lastCut数组
22         dp[i] = lastCut[i] = i;
23         // 遍历所有可能的切割长度, 从1到i
24         for (let j = 1; j < i; j++) {
25             // 计算当前切割长度的乘积
26             const product = dp[i - j] * j;
27             // 如果当前乘积大于已知的最大乘积, 更新最大乘积和最佳切割长度
28             if (product > dp[i]) {
29                 dp[i] = product;
30                 lastCut[i] = j;
31                 cutTimes[i] = cutTimes[i - j] + 1;
32             }
33             // 如果当前乘积等于已知的最大乘积, 但切割次数更少, 更新最佳切割长度和切割次数
34             else if (product === dp[i] && cutTimes[i] > cutTimes[i - j] + 1) {
35                 lastCut[i] = j;
36                 cutTimes[i] = cutTimes[i - j] + 1;
37             }
38         }
39     }
40
41     // 创建一个数组来存储结果
42     const results = [];
43     // 从最大长度开始, 每次减去最佳切割长度, 直到长度为0
44     while (length > 0) {
45         // 将最佳切割长度添加到结果的开头
46         results.unshift(lastCut[length]);
47         // 更新长度
48         length -= lastCut[length];
49     }
50     // 对结果进行排序
51     --
```

```
52     results.sort((a, b) => a - b);
53     // 返回结果
54     return results;
55 };
56
57 rl.on('line', (input) => {
58     // 将输入转换为整数
59     const length = parseInt(input);
60     // 调用getMaxProfit方法计算最大利润, 并将结果存储在一个数组中
61     const results = getMaxProfit(length);
62     // 打印结果
63     console.log(results.join(' '));
64     // 关闭readLine接口
65     rl.close();
66 });
```

## Python

```
1  # 定义函数, 计算给定长度的最大乘积分割
2  def get_max_profit(length):
3      # dp数组用于存储每个长度的最大乘积
4      dp = [0] * (length + 1)
5      # cutTimes数组用于存储每个长度的最佳切割次数
6      cut_times = [0] * (length + 1)
7      # lastCut数组用于存储每个长度的最后一次切割长度
8      last_cut = [0] * (length + 1)
9
10     # 遍历每个长度
11     for i in range(1, length + 1):
12         # 初始化dp和lastCut数组
13         dp[i] = last_cut[i] = i
14         # 遍历所有可能的切割长度
15         for j in range(1, i):
16             # 计算当前切割长度的乘积
17             product = dp[i - j] * j
18             # 如果当前乘积大于已知的最大乘积, 更新最大乘积和最佳切割长度
19             if product > dp[i]:
20                 last_cut[i] = j
21                 dp[i] = product
22                 cut_times[i] = cut_times[i - j] + 1
```



```

43     # 如果当前乘积等于已知的最大乘积, 但切割次数更少, 更新最佳切割长度和切割次数
44     elif product == dp[i] and cut_times[i] > cut_times[i - j] + 1:
45         last_cut[i] = j
46         cut_times[i] = cut_times[i - j] + 1
47
48     # 创建一个列表来存储结果
49     results = []
50     # 从最大长度开始, 每次减去最佳切割长度, 直到长度为0
51     while length > 0:
52         # 将最佳切割长度添加到结果的开头
53         results.insert(0, last_cut[length])
54         # 更新长度
55         length -= last_cut[length]
56     # 对结果进行排序
57     results.sort()
58     # 返回结果
59     return results
60
61 # 读取输入的长度
62 length = int(input(""))
63 # 调用get_max_profit方法计算最大利润, 并将结果存储在一个列表中
64 results = get_max_profit(length)
65 # 打印结果
66 print(' '.join(map(str, results)))

```

## C语言

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // 定义函数, 计算给定长度的最大乘积分割
5  void getMaxProfit(int length, int* results, int* resultsSize) {
6      // dp数组用于存储每个长度的最大乘积
7      int* dp = (int*)malloc((length + 1) * sizeof(int));
8      // cutTimes数组用于存储每个长度的最佳切割次数
9      int* cutTimes = (int*)malloc((length + 1) * sizeof(int));
10     // lastCut数组用于存储每个长度的最后一次切割长度
11     int* lastCut = (int*)malloc((length + 1) * sizeof(int));
12
13     // 遍历每个长度
14

```

```
14 for (int i = 1; i <= length; ++i) {
15     // 初始化dp和lastCut数组
16     dp[i] = lastCut[i] = i;
17     // 遍历所有可能的切割长度
18     for (int j = 1; j < i; ++j) {
19         // 计算当前切割长度的乘积
20         int product = dp[i - j] * j;
21         // 如果当前乘积大于已知的最大乘积, 更新最大乘积和最佳切割长度
22         if (product > dp[i]) {
23             lastCut[i] = j;
24             dp[i] = product;
25             cutTimes[i] = cutTimes[i - j] + 1;
26         }
27         // 如果当前乘积等于已知的最大乘积, 但切割次数更少, 更新最佳切割长度和切割次数
28         else if (product == dp[i] && cutTimes[i] > cutTimes[i - j] + 1) {
29             lastCut[i] = j;
30             cutTimes[i] = cutTimes[i - j] + 1;
31         }
32     }
33 }
34
35 // 从最大长度开始, 每次减去最佳切割长度, 直到长度为0
36 while (length > 0) {
37     // 将最佳切割长度添加到结果的开头
38     results[*resultsSize] = lastCut[length];
39     (*resultsSize)++;
40     // 更新长度
41     length -= lastCut[length];
42 }
43
44 // 对结果进行排序
45 for (int i = 0; i < *resultsSize; i++) {
46     for (int j = i + 1; j < *resultsSize; j++) {
47         if (results[i] > results[j]) {
48             int temp = results[i];
49             results[i] = results[j];
50             results[j] = temp;
51         }
52     }
53 }
54
--
```

```
55
56 // 释放动态分配的内存
57 free(dp);
58 free(cutTimes);
59 free(lastCut);
60 }
61
62 int main() {
63 // 读取输入的长度
64 int length;
65 scanf("%d", &length);
66
67 // 创建一个数组来存储结果
68 int* results = (int*)malloc(length * sizeof(int));
69 int resultsSize = 0;
70
71 // 调用getMaxProfit方法计算最大利润, 并将结果存储在一个数组中
72 getMaxProfit(length, results, &resultsSize);
73
74 // 遍历结果并打印
75 for (int i = 0; i < resultsSize; i++) {
76     printf("%d ", results[i]);
77 }
78
79 // 释放动态分配的内存
80 free(results);
81
82 return 0;
}
```

## 文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷  
题目描述  
输入描述  
输出描述  
用例  
解题思路

C++

Java

JavaScript

Python

C语言

# 机考真题 华为OD



CSDN @算法大师