计算机算法设计与分析

083500M01001H

Chap 5 课程作业解答

2022 年 10 月 22 号

Professor: 刘玉贵

学生: 周胤昌

学号: 202228018670052

学院: 网络空间安全学院

所属专业: 网络空间安全

方向: 安全协议理论与技术

Problem 1

最大子段和问题: 给定整数序列 $a_1, a_2, \cdots, a_n$, 求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值:

$$\max\left\{0, \max_{1\le i\le i\le n} \sum_{k=i}^{j} a_k\right\}$$

(1). 已知一个简单算法如下:

```
int Maxsum(int n, vector<int> a, int& besti, int& bestj) {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
        int suma = 0;
        for(int j = i; j <= n; j++) {
            suma += a[j];
            if(suma > sum) {
                sum = suma;
                besti = i;
                bestj = j;
            }
        }
    }
    return sum;
}
```

试分析该算法的时间复杂性;

(2). 试用分治算法解最大子段和问题, 并分析算法的时间复杂性;

(3). 试说明最大子段和问题具有最优子结构性质, 并设计一个动态规划算法求解最大子段和问题, 分析算法的时间复杂度. (提示: 可令 $b(j) = \max\limits_{1\le i\le j\le n} \sum_{k=i}^{j} a_k, j = 1, 2, \cdots, n$)

Solution: (1). 显然第 2 层 for 循环里面的操作都是常数次的 (记为 C), 所以算法总的关键操作数为

$$\sum_{i=1}^{n}\sum_{j=i}^{n} C = C\sum_{i=1}^{n}(n-i+1) = \frac{1}{2}C\left(n^2+n\right)$$

故显然时间复杂度为 $T(n) = O(n^2)$.

(2). 采用分治算法, 则考虑: 先将数组从中间 mid 切开. 此时, 最大和的子段可能出现在左半边, 也可能出现在右半边, 也有可能横跨左右两个子数组. 所以需要返回这三种情况下所分别对应的子问题解的最大值.

当最大和的子段出现在左半边 (右半边同理) 时, 继续分中点递归直至分解到只有一个数为止;

当最大和的子段横跨 mid 左右时, 只需分别求解左子数组的最优后缀和以及右子数组的最优前缀和. 这三种情形下的最大值即为整个数组的最大子段和. 具体分治算法见如下.

Algorithm 1 MaxSubSum(A, left, right)

Input: 数组 A, 左边界 left, 右边界 right

Output: A 的最大子段和 sum 及子段的前后边界

if left==right **then**
  return max(A[left], 0);
end if
k :=(left+right)/2;
leftsum = **MaxSubSum**(A,left,k);
rightsum = **MaxSubSum**(A, $k+1$,right);
类似 (1) 中的算法分别求得 S_1, S_2;
Sum:= $S_1 + S_2$;
return max(leftsum, rightsum, Sum);
end {MaxSubSum}

可以看出最坏情形下的时间复杂度的递推式和结果分别为 $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, 故根据主定理 ($\log_b(a) = \log_2(2) = 1 = d$) 可知 $T(n) = O(n\log n)$.

(3). 先证明此问题具有最优子结构性质: 依次考虑 ($1 \le i \le n$) 以 $a[i]$ **为结尾**的最大子段和 $C[i]$, 然后在这 n 个值当中取最大值即为原问题答案. 假设以 $a[i]$ 为结尾的最大 (和) 子段为 $\{a[k], \cdots, a[i]\}$, 那么 $\{a[k], \cdots, a[i-1]\}$ 一定是以 $a[i-1]$ 为结尾的最大 (和) 子段. 否则若 $\{a[m], \cdots, a[i-1]\}$ 为以 $a[i-1]$ 为结尾的最大 (和) 子段, 那么 $\{a[m], \cdots, a[i-1], a[i]\}$ 就是以 $a[i]$ 为结尾的最大 (和) 子段, 这显然与假设相矛盾, 也就是说该优化函数是满足优化原则的 (即此问题具有最优子结构性质).

现在来推导 $C[i]$ 的递推表达式: 当 $C[i-1] \le 0$, 说明 $C[i-1]$ 对应的子段对于整体的贡献是没有的, 所以 $C[i] \leftarrow a[i]$; 当 $C[i-1] > 0$, 说明 $C[i-1]$ 对应的子段对于整体的是有贡献的, 于是 $C[i] \leftarrow a[i] + C[i-1]$. 两种可能情况 (对应两种决策) 取最大值即可:

$$\begin{cases} C[i] = \max\{a[i], C[i-1] + a[i]\}, 2 \le i \le n \\ C[1] = a[1] \end{cases}$$

最后返回数组 C 中的最大值 ($\max\limits_{1\le i\le n} C[i]$) 即可. 计算 $C[i]$ 的过程需要消耗 $O(n)$ 的时间, 找出数组最大值也需要 $O(n)$ 的时间 (一次遍历), 所以算法的总时间复杂度为 $T(n) = O(n)$. 并且我们可以给出 C++ 代码:

```
int mostvalue(vector<int>& a) { //时间复杂度为 O(n), 空间复杂度为 O(n)
    int n = a.size();
    vector<int> dp(n);
    dp[0] = a[0];
    for(int i = 1; i < n; i++) {
        dp[i] = max(dp[i - 1] + a[i], a[i]);
    }
    int index = max_element(dp.begin(), dp.end()) - dp.begin();
    return dp[index];
}
```

Problem 2

设 $A = \{x_1, x_2, \cdots, x_n\}$ 是 n 个不等的整数构成的序列, A 的一个单调递增子序列是指序列 $\{x_{i_1}, x_{i_2}, \cdots, x_{i_k}\}, i_1 < i_2 < \cdots i_k$ 且 $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$.(子序列包含 k 个整数). 例如, $A = \{1, 5, 3, 8, 10, 6, 4, 9\}$, 他的长度为 4 的递增子序列是: $\{1, 5, 8, 10\}, \{1, 5, 8, 9\}, \cdots$. 设计一个算法, 求 A 的最长的单调递增子序列, 分析算法的时间复杂度. 对于输入实例 $A = \{2, 8, 4, -4, 5, 9, 11\}$, 给出算法的计算过程和最后的解.

Solution: 定义 $dp[i]$ 是以 nums$[i]$ **为结尾** (且考虑前 i 个元素) 的最长单增子序列的长度. 于是可以写出如下转移方程 ($i \geq 1$) 和初始条件:

$$dp\left[i\right] = \begin{cases} \max\limits_{0 \leq j \leq i-1} dp\left[j\right] + 1, & \exists j \in \left[0, i-1\right], s.t. \, \text{nums}\left[j\right] < \text{nums}\left[i\right] \\ 1 & \forall j \in \left[0, i-1\right], s.t. \, \text{nums}\left[j\right] > \text{nums}\left[i\right] \end{cases}, dp\left[0\right] = 1$$

显然, 若 $dp[i]$ 的子序列是 $i_1 i_2 \cdots i_k i$, 则 $dp[i_k]$ 的子序列为 $i_1 i_2 \cdots i_k$, 即问题满足最优子结构性质. 需借助数组 m 来对解进行回溯 (即 $m[i]$ 记录 $dp[i]$ 是由哪个下标的状态转移而来的). 而要想算出**整个数组**的最长单增子序列长度, 则需要算好所有的 $dp[i]$ 值, 再对 dp 数组进行遍历, 由此得到最长单增子序列长度和对应下标. 最后使用数组 m 进行回溯以取得答案. 可以看出, 算法的空间复杂度为 $O(n)$, 而时间复杂度显然为

$$T\left(n\right) = O\left(\sum_{i=0}^{n-1}\sum_{j=0}^{i-1} 1\right) = O\left(\sum_{i=0}^{n-1} i\right) = O\left(\frac{n\left(n-1\right)}{2}\right) = O\left(n^2\right)$$

我们将上述的最优值求解过程和解的回溯过程写成 C++ 代码, 并且已完全通过**LeetCode-T300**的所有测试样例, 具体如下所示:

```cpp
vector<int> LIS(vector<int>& nums) {
    int n = nums.size();
    vector<int> dp(n, 0);
    dp[0] = 1;
    vector<int> m(n, 0);
    for (int i = 1; i <= n - 1; i++) {
        int prev = i, len = 1;
        for (int j = 0; j <= i - 1; j++) {
            if (nums[j] < nums[i]) {
                if (dp[j] + 1 > len) {
                    len = dp[j] + 1;
                    prev = j;
                }
            }
        }
        dp[i] = len, m[i] = prev;
    }
    int index = max_element(dp.begin(), dp.end()) - dp.begin();
    int MaxLen = dp[index];
    vector<int> res;
    while (res.size() != MaxLen) {
        res.push_back(nums[index]);
        index = m[index];
    }
    return res;
}
```

对于具体实例, 计算过程如下:

$$C[1]=1; C[2]=2, k[2]=1; C[3]=2, k[3]=1; C[4]=1, k[4]=0;$$
$$C[5]=3, k[5]=3; C[6]=4, k[6]=5; C[7]=5, k[7]=6$$

显然在数组 C 中的最大值为 $C[7]=5$, 即最长递增子序列长度为 5 且追踪过程为:

$$x_7, k[7]=6 \Rightarrow x_6;\; k[6]=5 \Rightarrow x_5;\; k[5]=3 \Rightarrow x_3;\; k[3]=1 \Rightarrow x_1$$

故 $A=\{2,8,4,-4,5,9,11\}$ 的最长单调递增子序列为 $\{x_1,x_3,x_5,x_6,x_7\}=\{2,4,5,9,11\}$.

Problem 3

考虑下面特殊的整数线性规划问题

$$\max \sum_{i=1}^{n} c_i x_i$$
$$\sum_{i=1}^{n} a_i x_i \le b, x_i \in \{0,1,2\}, 1 \le i \le n$$

试设计一个解决此问题的动态规划算法, 并分析算法的时间复杂度.

Solution: 这是整数背包问题, 下证最优子结构性质: 设 $y_1, y_2, \cdots y_n$ 是原问题的最优解, 则 $y_1, y_2, \cdots y_{n-1}$ 是下述子问题的最优解:

$$\max \sum_{i=1}^{n-1} c_i x_i, \text{s.t.} \sum_{i=1}^{n-1} a_i x_i \le b - a_n y_n, x_i \in \{0,1,2\}, 1 \le i \le n-1$$

如若不然, 设 $y_1', y_2', \cdots y_{n-1}'$ 是子问题的最优解, 则

$$\sum_{i=1}^{n-1} c_i y_i' > \sum_{i=1}^{n-1} c_i y_i \text{且} \sum_{i=1}^{n-1} a_i y_i' \le b - a_n y_n \Rightarrow \sum_{i=1}^{n-1} c_i y_i' + c_n y_n > \sum_{i=1}^{n} c_i y_i \text{且} \sum_{i=1}^{n-1} a_i y_i' + a_n y_n \le b$$

于是 $y_1', y_2', \cdots y_{n-1}', y_n$ 为原问题的最优解, 与 $y_1, y_2, \cdots y_n$ 是最优解相矛盾! 设 $m[k][x]$ 表示容量约束 x, 可装入 $1, 2, \cdots, k$ 件物品的最优值, 则有递推公式:

$$m[k][x] = \max\{m[k-1][x], m[k-1][x-a_k]+c_k, m[k-1][x-2a_k]+2c_k\}, 0 \le x \le b$$
$$m[0][x]=0, x \ge 0; \quad m[0][x]=-\infty, x<0; \quad m[k][<0]=-\infty, k=1,\cdots,n$$

Problem 4

可靠性设计: 一个系统由 n 级设备串联而成, 为了增强可靠性, 每级都可能并联了不止一台同样的设备. 假设第 i 级设备 D_i 用了 m_i 台, 该级设备的可靠性 $g_i(m_i)$, 则这个系统的可靠性是 $\Pi g_i(m_i)$. 一般来说 $g_i(m_i)$ 都是递增函数, 所以每级用的设备越多系统的可靠性越高. 但是设备都是有成本的, 假定设备 D_i 的成本是 c_i, 设计该系统允许的投资不超过 c. 那么, 该如何设计该系统 (即各级采用多少设备) 使得这个系统的可靠性最高. 试设计一个动态规划算法求解可靠性设计问题.

Solution: 问题描述为

$$\max \prod_{i=1}^{n} g_i\left(m_i\right), \text{s.t.} \sum_{i=1}^{n} m_i c_i \leq C$$

证明问题具有最优子结构性质: 设 $m_1, m_2, \cdots, m_{n-1}, m_n$ 是原问题的最优解, 其可靠性为 $\prod_{i=1}^{n} g_i\left(m_i\right) = g_n\left(m_n\right) \prod_{i=1}^{n-1} g_i\left(m_i\right)$, 则 $m_1, m_2, \cdots, m_{n-1}$ 显然是下述子问题的最优解:

$$\max \prod_{i=1}^{n-1} g_i\left(m_i\right), \text{s.t.} \sum_{i=1}^{n-1} m_i c_i \leq C - m_n c_n$$

否则若 $m_1', m_2', \cdots, m_{n-1}'$ 是子问题的最优解, 则 $\prod_{i=1}^{n-1} g_i\left(m_i'\right) > \prod_{i=1}^{n-1} g_i\left(m_i\right)$ 且 $\sum_{i=1}^{n-1} m_i' c_i \leq C - m_n c_n$. 于是有 $g_n\left(m_n\right) \cdot \prod_{i=1}^{n-1} g_i\left(m_i'\right) > \prod_{i=1}^{n} g_i\left(m_i\right)$ 且 $\sum_{i=1}^{n-1} m_i' c_i + m_n c_n \leq C$, 即 $m_1', m_2', \cdots, m_{n-1}', m_n$ 则为原问题的最优解, 这显然与 $m_1, m_2, \cdots, m_{n-1}, m_n$ 是原问题最优解相矛盾! 因而 $m_1, m_2, \cdots, m_{n-1}$ 是子问题的最优解. 即此问题具有最优子结构性质. 设 $m[k][x]$ 为成本 x, 前 k 级设备串联所得最优可靠性值, 则有:

$$\begin{aligned} & m[k][x] = \max\left\{m[k-1]\left[x - m_k c_k\right] \times g_k\left(m_k\right)\right\}, 1 \leq m_k \leq x / c_k \\ & m[0][x] = 1, x \geq 0; \ m[k]\left[c_k\right] = 1 \end{aligned}$$

```
Safe(c[], g[], C) {
    int m[][], p[][];
    C = c - sum(c);
    for(j = 0 to C) m[0][j] = 1;
    for(i = 1 to n) {
        for(j = 0 to c) {
            m[i][j] = 0;
            for(k = 0 to j/c[i]) {
                t = m[i-1][j-k*c[i]]*g[i](k);
                if(m[i][j] < t) {
                    m[i][j] = t, p[i][j] = k;
                }
            }
        }
    }
}
```

最优解是 $m[n][C]$, 可以如下回溯求 m_i:

$$p[n][C] = m_n, C = C - m_n \cdot c[n]; p[n-1][C] = m_{n-1}; \cdots$$

Problem 5

(**双机调度问题**) 用两台处理机 A 和 B 处理 n 个作业. 设第 i 个作业交给机器 A 处理时所需要的时间是 a_i, 若由机器 B 来处理, 则所需要的时间是 b_i. 现在要求每个作业只能由一台机器处理, 每台机器都不能同时处理两个作业. 设计一个动态规划算法, 使得这两台机器处理完这 n 个作业的时间最短 (从任何一台机器开工到最后一台机器停工的总时间). 以下面的例子说明你的算法:

$$n = 6, (a_1, a_2, a_3, a_4, a_5, a_6) = (2, 5, 7, 10, 5, 2), (b_1, b_2, b_3, b_4, b_5, b_6) = (3, 8, 4, 11, 3, 4)$$

Solution: 在完成前 k 个作业时, 设机器 A 工作了 x 时间 (注意 x 限制为正整数), 则机器 B 此时最小的工作时间是 x 的函数. 设 $F[k][x]$ 表示完成前 k 个作业时, 机器 B 最小的工作时间, 则有

$$F[k][x] = \min\{F[k-1][x] + b_k, F[k-1][x-a_k]\}$$

其中 $F[k-1][x]+b_k$ 对应的是第 k 个作业由机器 B 来处理, 此时完成前 $k-1$ 个作业时机器 A 的工作时间仍是 x, 则 B 在 $k-1$ 阶段用时为 $F[k-1][x]$; 而 $F[k-1][x-a_k]$ 对应第 k 个作业由机器 A 处理 (完成 $k-1$ 个作业, 机器 A 工作时间时 $x-a[k]$, 而 B 完成 k 阶段与完成 $k-1$ 阶段用时都为 $F[k-1][x-a_k]$). 于是完成前 k 个作业所需要的时间为 $T = \max\{x, F[k][x]\}$. 根据上述递推关系很容易证得问题满足最优子结构性质. 并且通过下述算法代码可知时间复杂度为 $O\left(n \cdot \min\left\{\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i\right\}\right)$.

```
int schedule() {
    int sumA = a[1], time[n];
    //k = 1 的情况
    for(int x = 0; x < a[1]; x++) {
        F[1][x] = b[1];
    }
    F[1][a[1]] = min(b[1],a[1]);
    //初始化
    for(int i = 2; i <= n; i++) {
        for(int j = 0; j <= n; j++) {
            F[i][j] = INT_MAX;
        }
    }
    //k >= 2 的情况
    for(int k = 2; k <= n; k++) {
        sumA += a[k];
        time[k] = INT_MAX;
        for(int x = 0; x <= sumA; x++) {
            if(x < a[k]) {
                F[k][x] = F[k-1][x] + b[k];
            } else {
                F[k][x] = min(F[k-1][x] + b[k], F[k-1][x-a[k]]);
            }
            //判断完成作业 k 时, 到底是机器 B 所需最小时间小, 还是 A 所需时间小
            time[k] = min(time[k],max(x,F[k][x]));
        }
    }
    return time[n];
}
```

Problem 6

有 n 项作业的集合 $J=\{1,2,\cdots,n\}$, 每项作业 i 有加工时间 $t(i)\in\mathbb{Z}^+, t(1)\le t(2)\le\cdots\le t(n)$, 效益值 $v(i)$, 任务的结束时间 $D\in\mathbb{Z}^+$, 其中 $\mathbb{Z}^+$ 表示正整数集合. 一个可行调度是对 J 的子集 A 中任务的一种安排, 对于 $i\in A$, $f(i)$ 是开始时间, 且满足下述条件:

$$\begin{cases} f(i)+t(i)\le f(j) \text{或} f(j)+t(j)\le f(i), \text{其中} j\ne i \text{且} i,j\in A \\ \sum\limits_{k\in A} t(k)\le D \end{cases}$$

设机器从 0 时刻开始启动, 只要有作业就不闲置, 求具有最大总效益的调度. 给出算法并分析其时间复杂度.

Solution: 与 0-1 背包问题相类似, 使用 DP 算法, 令 $N_j(d)$ 表示考虑作业集 $\{1,2,\cdots,j\}$、结束时间为 d 的最优调度的效益, 那么有递推方程

$$N_j(d)=\begin{cases}\max\left\{N_{j-1}(d), N_{j-1}(d-t(j))+v_j\right\}, & d\ge t(j)\\ N_{j-1}(d), & d<t(j)\end{cases}$$

并且边界 (初始) 条件为

$$N_1(d)=\begin{cases} v_1, & d\ge t(1)\\ 0, & d<t(1)\end{cases},\quad N_j(0)=0,\quad N_j(d)=-\infty\left(\text{其中} d<0\right)$$

自底向上计算, 存储使用备忘录 (以存代算), 可以使用标记函数 $B(j)$ 记录使得 $N_j(d)$ 达到最大时是否

$$N_{j-1}(d-t(j))+v_j>N_{j-1}(d)$$

如果是, 则 $B(j)=j$; 否则 $B(j)=B(j-1)$.(换句话说, 如果装了作业 j, 那么就追踪其下标; 否则就不追踪更新)

伪代码如后页算法2中所示, 由此我们可以分析出时间复杂度: 得到最大效益 $N[n,D]$ 后, 通过对 $B[n,D]$ 的追踪就可以得到问题的解, 算法的主要工作在于第 7 行到第 16 行的 for 循环, 需要执行 $O(nD)$ 次, 循环体内的工作量是常数时间, 因此算法的总时间复杂度为 $O(nD)$. 显然该算法是**伪多项式时间**的算法[1].

[1] 问题就在于如果 D 过大, 即 D 的 2 进制表示会很长, 且 $O(n\cdot D)=O\left(n\cdot 2^{\log(D)}\right)=O\left(n\cdot 2^{\text{输入长度}}\right)$, 是与输入长度相关的指数表达式, 这种复杂度形式的算法称之为**伪多项式时间算法**.

Algorithm 2 Homework 算法

Input: 加工时间 $t[1,\cdots,n]$, 效益 $v[1,\cdots,n]$, 结束时间 D

Output: 最优效益 $N[i,j]$, 标记函数 $B[i,j], i=1,2,\cdots,n, j=1,2,\cdots,D$

for $d=1; d\le t[1]-1; d{+}{+}$ **do**
    $N[1,d]\leftarrow 0, B[1]\leftarrow 0;$
end for
for $d=t[1]; d\le D; d{+}{+}$ **do**
    $N[1,d]\leftarrow v[1], B[1]\leftarrow 1;$
end for
for $j=2; j\le n; j{+}{+}$ **do**
    for $d=1; d\le D; d{+}{+}$ **do**
        $N[j,d]\leftarrow N[j-1,d];$
        $B[j,d]\leftarrow B[j-1,d];$
        if $d\ge t[j]$ && $N[j-1,d-t[j]]+v[j]>N[j-1,d]$ **then**
            $N[j,d]\leftarrow N[j-1,d-t[j]]+v[j];$
            $B[j,d]\leftarrow j;$
        end if
    end for
end for
end {Homework}

Problem 7

设 A 是顶点为 $1,2,\cdots,n$ 的凸多边形, 可以用不在内部相交的 $n-3$ 条对角线将 A 划分成三角形, 下图中就是 5 边形的所有划分方案. 假设凸 n 边形的边及对角线的长度 d_{ij} 都是给定的正整数, 其中 $1\le i<j\le n$. 划分后三角形 ijk 的权值等于其周长, 求具有最小权值的划分方案. 设计一个动态规划算法求解该问题, 并说明其时间复杂度 (提示: 参考矩阵连乘问题).

如下图1所示, n 边形的顶点是 $1,2,\cdots,n$. 顶点 $i-1,i,\cdots,j$ 构成的凸多边形记作 $A[i,j]$, 于是原始问题就是 $A[2,n]$.

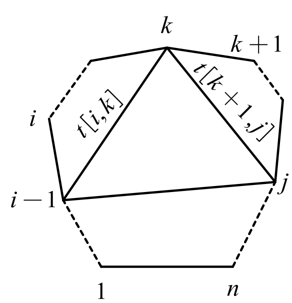

图 1: 子问题归约图

考虑子问题 $A[i,j]$ 的划分, 假设它的所有划分方案中最小权值为 $t[i,j]$. 从 $i,i+1,\cdots,j-1$ 中任选顶点 k, 它与底边 $(i-1)j$ 构成一个三角形 (图1中的三角形). 这个三角形将 $A[i,j]$ 划分成两个凸多边形: $A[i,k]$ 和 $A[k+1,j]$, 从而产生了两个子问题. 这两个凸多边形的划分方案的最小权值分别为 $t[i,k]$

和 $t[k+1,j]$. 根据 DP 思想, $A[i,j]$ 相对于这个顶点 k 的划分方案的最小权值是

$$t\left[i,k\right]+t\left[k+1,j\right]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}$$

其中 $d_{(i-1)k}+d_{kj}+d_{(i-1)j}$ 是三角形 $(i-1)kj$ 的周长, 于是得到递推关系

$$t\left[i,j\right]=\begin{cases}0, & i=j\\ \min\limits_{i\le k\le j-1}\left\{t\left[i,k\right]+t\left[k+1,j\right]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}\right\}, & i<j\end{cases}$$

根据上述递推关系可知, 若最优划分 $t[i,j]$ 与 $(i-1)j$ 相连的三角形第三顶是 k, 则这个划分也是 $A[i,k]$ 和 $A[k+1,j]$ 的最优划分, 即最优子结构性质得证. 可以通过标记函数来得到最小权值对应顶点 k 的位置, 于是该划分算法最坏情况下的时间复杂度为 $O\left(n^3\right)$.

```
void MatrixChain(int **d, int n, int **t, int **s) {
    for (int i = 1; i <= n; i++) t[i][i] = 0;
    for (int r = 2; r <= n; r++){
        for (int i = 1; i <= n - r + 1; i++){
            int j = i + r - 1;
            t[i][j] = t[i + 1][j] + d[i - 1][i] + d[i][j] + d[i - 1][j];
            s[i][j] = i;
            for (int k = i + 1; k < j; k++){
                int T = t[i][k] + t[k + 1][j] + d[i - 1][k] + d[k][j] + d[i - 1][j];
                if(t < t[i][j]) {
                    t[i][j] = t, s[i][j] = k;
                }
            }
        }
    }
}
```