# 习题一

202328015926048-丁力

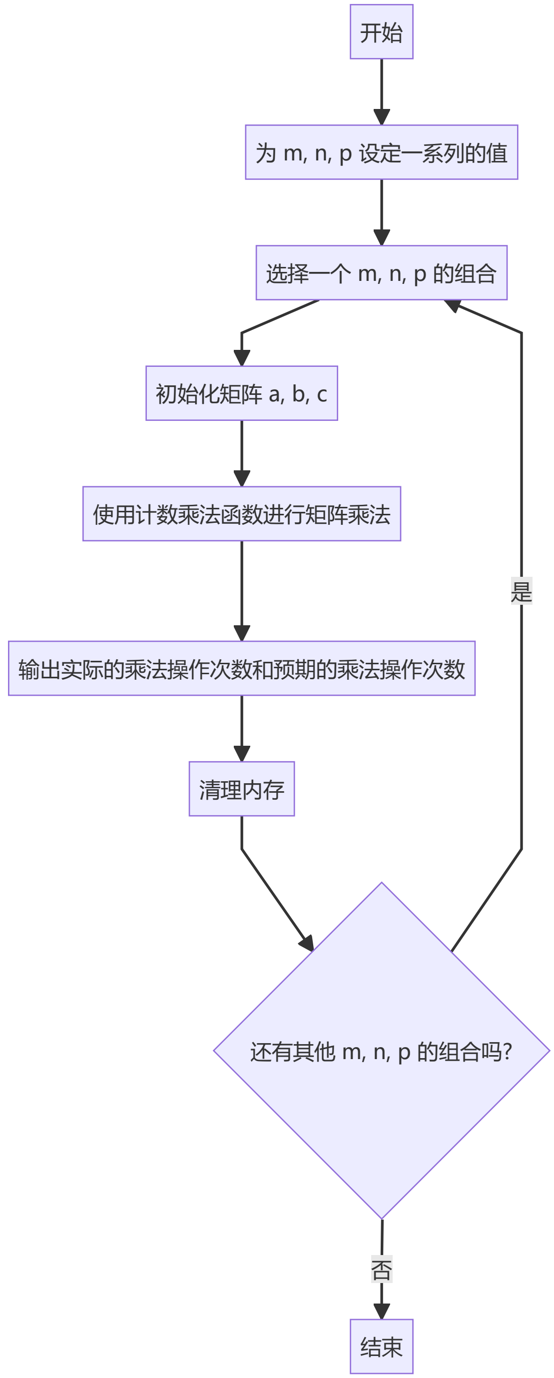
---

1. **矩阵乘法运算**

```
template <class T>
void Mult(T** a, T** b, int m, int n, int p) {
    // m x n 矩阵 a 与 n x p 矩阵 b 相乘得到 m x p 矩阵 c
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++) {
            T sum = 0;
            for (int k = 0; k < n; k++)
                sum += a[i][k] * b[k][j];
            c[i][j] = sum;
        }
}
```

试确定上述程序的执行步数。

> 首先，我们需要纠正代码中的一个小错误。在给出的代码中，矩阵 `c` 并没有被声明。我们需要将函数的参数列表修改为包括 `c`。
>
> 然后，为了确定上述矩阵乘法运算的运行次数， 我们这里将重新给出所有的代码，并在代码的注释部分给出该行的运行次数，如下：
>
> ```
> template <class T>
> void Mult(T** a, T** b, T** c, int m, int n, int p) {
>  // m x n 矩阵 a 与 n x p 矩阵 b 相乘得到 m x p 矩阵 c
>  for (int i = 0; i < m; i++) // m次
>      for (int j = 0; j < p; j++) { // p次
>          T sum = 0;
>          for (int k = 0; k < n; k++) // k次
>              sum += a[i][k] * b[k][j];
>          c[i][j] = sum;
>      }
> }
> ```
>
> 所以该程序的运行次数为$m \times p \times k = mpk$，也就是 `mpk` 次。
>
> 为了验证我们结论的准确性，我们按照如下流程进行检验：




为此，我们编写如下的 `C++` 代码进行检验：

```
#include <iostream>

template <class T>
void Mult(T** a, T** b, T** c, int m, int n, int p) {
 for (int i = 0; i < m; i++)
     for (int j = 0; j < p; j++) {
         T sum = 0;
         for (int k = 0; k < n; k++)
             sum += a[i][k] * b[k][j];
         c[i][j] = sum;
     }
}

int counter = 0;

template <class T>
void CountedMult(T** a, T** b, T** c, int m, int n, int p) {
```

```
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++) {
            T sum = 0;
            for (int k = 0; k < n; k++) {
                sum += a[i][k] * b[k][j];
                counter++;
            }
            c[i][j] = sum;
        }
}

int main() {
    int sizes[] = {2, 3, 4}; // 你可以根据需要增加更多的大小
    int num_sizes = sizeof(sizes) / sizeof(sizes[0]);

    for (int m_idx = 0; m_idx < num_sizes; m_idx++)
        for (int n_idx = 0; n_idx < num_sizes; n_idx++)
            for (int p_idx = 0; p_idx < num_sizes; p_idx++) {
                int m = sizes[m_idx];
                int n = sizes[n_idx];
                int p = sizes[p_idx];

                // 初始化矩阵
                int** a = new int*[m];
                int** b = new int*[n];
                int** c = new int*[m];
                for (int i = 0; i < m; i++) {
                    a[i] = new int[n];
                    c[i] = new int[p];
                }
                for (int i = 0; i < n; i++) {
                    b[i] = new int[p];
                }

                // 填充矩阵
                for (int i = 0; i < m; i++)
                    for (int j = 0; j < n; j++)
                        a[i][j] = 1;
                for (int i = 0; i < n; i++)
                    for (int j = 0; j < p; j++)
                        b[i][j] = 1;

                counter = 0;
                CountedMult(a, b, c, m, n, p);
                std::cout << "For m=" << m << ", n=" << n << ", p=" << p << ":\n";
                std::cout << "Total multiplications: " << counter << std::endl;
                std::cout << "Expected multiplications: " << m * n * p << "\n\n";

                // 清理内存
                for (int i = 0; i < m; i++) {
                    delete[] a[i];
                    delete[] c[i];
                }
                for (int i = 0; i < n; i++) {
                    delete[] b[i];
                }
                delete[] a;
                delete[] b;
                delete[] c;
            }

    return 0;
}
```

> 其运行结果如下：
>
> ```
> (base) PS C:\Users\23174\Desktop\GitHub Project\Algorithm-Analysis\HW\week3\code> g++ .\q1.cpp
> (base) PS C:\Users\23174\Desktop\GitHub Project\Algorithm-Analysis\HW\week3\code> .\a.exe
> For m=2, n=2, p=2:
> Total multiplications: 8
> Expected multiplications: 8
>
> For m=2, n=2, p=3:
> Total multiplications: 12
> Expected multiplications: 12
>
> For m=2, n=2, p=4:
> Total multiplications: 16
> Expected multiplications: 16
>
> For m=2, n=3, p=2:
> Total multiplications: 12
> Expected multiplications: 12
>
> For m=2, n=3, p=3:
> Total multiplications: 18
> Expected multiplications: 18
>
> For m=2, n=3, p=4:
> Total multiplications: 24
> Expected multiplications: 24
>
> For m=2, n=4, p=2:
> Total multiplications: 16
> Expected multiplications: 16
>
> For m=2, n=4, p=3:
> Total multiplications: 24
> Expected multiplications: 24
>
> For m=2, n=4, p=4:
> Total multiplications: 32
> Expected multiplications: 32
>
> For m=3, n=2, p=2:
> Total multiplications: 12
> Expected multiplications: 12
> ```
>
> 通过不同的排列组合验证，基本可以证明，我们的结论是正确的。

2. **找最大最小元素**

   函数 $MinMax$ 用来查找数组 $a[0:n-1]$中的最大元素和最小元素，以下给出两个程 序。令 $n$ 为实例特征。

   - 方法一：

```cpp
template <class T>
bool MinMax(T a[], int n, int& Min, int& Max) {
    // 寻找 a[0: n-1] 中的最小元素与最大元素
    if (n < 1) return false;
    Min = Max = 0; // 初始化
    for (int i = 1; i < n; i++) {
        if (a[Min] > a[i]) Min = i;
        if (a[Max] < a[i]) Max = i;
    }
    return true;
}
```

   - 方法二：

```cpp
template <class T>
bool MinMax(T a[], int n, int& Min, int& Max) {
    // 寻找 a[0: n-1] 中的最小元素与最大元素
    if (n < 1) return false;
    Min = Max = 0; // 初始化
    for (int i = 1; i < n; i++) {
        if (a[Min] > a[i]) Min = i;
        else if (a[Max] < a[i]) Max = i;
    }
    return true;
}
```

试问：在各个程序中，$a$ 中元素之间的比较次数在最坏情况下各是多 少?

> - **方法一：**
>
>   在方法一种，主要是程序是在一个执行 `n-1` 次的for循环，而在循环体内，进行了两次比较，所以在最坏的情况下，$a$元素之间的比较次数是:
>
>   $$2(n-1)$$
>
> - **方法二：**
>
>   在方法二中，举一个极端的例子，该数组的所有元素全部相同，那么也将会在循环体内进行 `2` 次比较，所以最坏情况下时，方法二的比较次数也是:
>
>   $$2(n-1)$$

3. **证明以下关系式不成立**：
   1. $10n^2+9=O(n)$

> 证明：
> 如果$10n^2+9=O(n)$成立，那么根据定义：
> 如果存在一个常数$C>0$和一个常数$n_0 \geq 0$ 使得对所有的 $n \geq n_0$，都有
>
> $$10n^2+9 \leq C \cdot n$$
>
> 也就是：
>
> $$C \cdot n - 10n^2 - 9 \geq 0$$
>
> 成立。
> 假设原式成立，那么存在$C>0, n_0 \geq 0$满足上式子， 此时，我们取$n=n_1(n_1>C)$,那么有:
>
> $$C \cdot n - 10n^2 - 9 = n \cdot (C - 10 \times n) - 9 < 0$$
>
> 与原式矛盾，故关系式不成立。

   1. $n^2 \log n = \Theta(n^2)$

> - 证伪 $n^2 \log n = \Theta(n^2)$
>
> 由定义，我们知道：
>
> $$n^2 \log n = \Theta(n^2) \iff n^2 \log n = O(n^2), n^2 \log n = \Omega(n^2)$$
>
> 为了证伪 $n^2 \log n = \Theta(n^2)$，我们需要证明 $n^2 \log n$ 不能同时满足 $n^2 \log n = O(n^2)$ 和 $n^2 \log n = \Omega(n^2)$。
>
> - 步骤1: 尝试证明 $n^2 \log n$ 不是 $O(n^2)$
>
> 假设存在常数 $C>0$ 和 $n_0 \geq 0$ 使得对所有的 $n \geq n_0$，都有
>
> $$n^2 \log n \leq C \cdot n^2$$
>
> 这意味着
>
> $$\log n \leq C$$
>
> 但是我们知道 $\log n$ 是一个随 $n$ 增长的函数，因此对于足够大的 $n$，$\log n$ 将超过任何给定的常数 $C$，这导致上述不等式不成立。
>
> - 步骤2: 证明 $n^2 \log n$ 是 $\Omega(n^2)$
>
> 我们需要找到常数 $C>0$ 和 $n_0 \geq 0$ 使得对所有的 $n \geq n_0$，都有
>
> $$n^2 \log n \geq C \cdot n^2$$
>
> 我们可以选择 $C=1$ 和 $n_0=2$ 来证明这一点，因为对于所有 $n \geq 2$，我们有
>
> $$n^2 \log n \geq n^2 \log 2 = n^2 \cdot 1 > n^2 \cdot 0 = 0$$
>
> 因此，我们可以得出结论，$n^2 \log n$ 是 $\Omega(n^2)$ 但不是 $O(n^2)$。
>
> - 结论

> 由于 $n^2 \log n$ 不能同时满足 $n^2 \log n = O(n^2)$ 和 $n^2 \log n = \Omega(n^2)$，我们可以得出结论 $n^2 \log n$ 不是 $\Theta(n^2)$。
>
> ■ Q.E.D.

4. 证明：当且仅当 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ 时，$f(n) = o(g(n))$.

> 我们要证明：
>
> $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \iff f(n) = o(g(n))$$
>
> 我们需要证明两个内容：
>
> > **1:**
> >
> > $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \to f(n) = o(g(n))$$
> >
> > **2:**
> >
> > $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \leftarrow f(n) = o(g(n))$$
>
> 首先，我们先给出这两个表达式的定义。
>
> 给定两个函数 $f(n)$ 和 $g(n)$，我们说"$f(n)$ 是 $g(n)$ 的小o"，记作 $f(n) = o(g(n))$，如果对于任意的正常数 $C > 0$，存在一个 $n_0$ 使得对所有 $n > n_0$ 有：
>
> $$f(n) < C \cdot g(n)$$
>
> 给定公式：
>
> $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$
>
> 这可以被数学地表达为：
>
> 对于任意的 $\epsilon > 0$，存在某个 $n_0$ 使得当 $n > n_0$ 时，有
>
> $$0 \leq \frac{f(n)}{g(n)} < \epsilon$$
>
> 也就是说：
>
> $$f(n) < \epsilon g(n)$$
>
> 令$C = \epsilon$,那么我们可以看到，从定义来看，两者是完全等价的，也就是：
>
> $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \leftarrow f(n) = o(g(n)), \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \to f(n) = o(g(n))$$
>
> 也就是：
>
> $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \iff f(n) = o(g(n))$$
>
> ■ Q.E.D.

5. **下面哪些规则是正确的？为什么？**
   1. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = O\left(\frac{F(n)}{G(n)}\right)$
   2. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = \Omega\left(\frac{F(n)}{G(n)}\right)$
   3. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = \Theta\left(\frac{F(n)}{G(n)}\right)$
   4. $\{f(n) = \Omega(F(n)), g(n) = \Omega(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = \Omega\left(\frac{F(n)}{G(n)}\right)$
   5. $\{f(n) = \Omega(F(n)), g(n) = \Omega(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = O\left(\frac{F(n)}{G(n)}\right)$
   6. $\{f(n) = \Theta(F(n)), g(n) = \Theta(G(n))\} \Rightarrow \frac{f(n)}{g(n)} = \Theta\left(\frac{F(n)}{G(n)}\right)$

> 首先，我们给出$O, \Omega, \Theta$符号的定义：
>
> 1. $f(n) = O(F(n))$ 表示存在常数 $c_1$ 和 $n_0$ 使得对于所有 $n \ge n_0$，有 $f(n) \le c_1 \times F(n)$。
> 2. $f(n) = \Omega(F(n))$ 表示存在常数 $c_2$ 和 $n_0$ 使得对于所有 $n \ge n_0$，有 $f(n) \ge c_2 \times F(n)$。
> 3. $f(n) = \Theta(F(n))$ 表示 $f(n) = O(F(n))$ 且 $f(n) = \Omega(F(n))$。
>
> 现在我们来分析每个规则：
>
> 1. 如果 $f(n) = O(F(n))$ 和 $g(n) = O(G(n))$，那么存在常数 $c_1$ 和 $c_2$ 使得 $f(n) \le c_1 \times F(n)$ 和 $g(n) \le c_2 \times G(n)$。但是，我们不能直接得出 $\frac{f(n)}{g(n)} = O\left(\frac{F(n)}{G(n)}\right)$。因为当 $g(n)$ 非常小的时候，分数可能会变得非常大。所以，这个规则是不正确的。
> 2. 同样的原因，我们不能得出 $\frac{f(n)}{g(n)} = \Omega\left(\frac{F(n)}{G(n)}\right)$。所以，这个规则也是不正确的。
> 3. 由于前两个规则都是不正确的，这个规则也是不正确的。
> 4. 如果 $f(n) = \Omega(F(n))$ 和 $g(n) = \Omega(G(n))$，那么存在常数 $c_1$ 和 $c_2$ 使得 $f(n) \ge c_1 \times F(n)$ 和 $g(n) \ge c_2 \times G(n)$。从这里我们可以得出 $\frac{f(n)}{g(n)} \ge \frac{c_1}{c_2} \times \frac{F(n)}{G(n)}$。所以，这个规则是正确的。
> 5. 同样的，我们可以得出 $\frac{f(n)}{g(n)} \le \frac{c_2}{c_1} \times \frac{F(n)}{G(n)}$。所以，这个规则也是正确的。
> 6. 如果 $f(n) = \Theta(F(n))$ 和 $g(n) = \Theta(G(n))$，那么 $f(n) = O(F(n))$、$f(n) = \Omega(F(n))$、$g(n) = O(G(n))$ 和 $g(n) = \Omega(G(n))$ 都是成立的。但是，由于我们在前三个规则中已经证明了这种关系不总是成立的，所以这个规则是不正确的。
>
> 总结：
>
> - 规则 `1、2、3和6是不正确的`。
> - 规则 `4和5是正确的`。

6. **按照渐近阶从低到高的顺序排列以下表达式**：
   $4n^2, \log n, 3^n, 20n, n^{2/3}, n!$

> 首先分析上述渐进阶：
>
> 1. $\log n$: 对数函数增长非常慢。
> 2. $n^{2/3}$: 比对数函数增长快，但比线性函数慢。
> 3. $20n$: 线性函数。
> 4. $4n^2$: 二次函数，比线性函数增长快。
> 5. $3^n$: 指数函数，增长速度非常快，比多项式函数（如$n^2$）增长得更快。
> 6. $n!$: 阶乘函数，增长速度非常快，比指数函数还要快。
>
> 可以得到如下的排序：
>
> $$\log n, \quad n^{2/3}, \quad 20n, \quad 4n^2, \quad 3^n, \quad n!$$

7. 1. 假设某算法在输入规模是 $n$ 时为 $T(n) = 3 \times 2^n$. 在某台计算机上实现并完成该算法的时间是 $t$ 秒. 现有另一台计算机, 其运行速度为第一台的 64 倍, 那么, 在这台计算机上用同一算法在 $t$ 秒内能解决规模为多大的问题?

   > 计算速度在这里可以表示为：
   >
   > $$v = \frac{3 \times 2^n}{t}$$
   >
   > 假设速度变为 `64` 倍，能解决规模为 `m` 的问题，那么我们有：
   >
   > $$64v = \frac{3 \times 2^m}{t}$$
   >
   > 带入上式，可以得到:
   >
   > $$m = 6 + n$$

2. 若上述算法改进后的新算法的时间复杂度为 $T(n)=n^2$, 则在新机器上用 $t$ 秒时间能解决输入规模为多大的问题?

> 同1., 我们可以得到如下式子:
>
> $$64v=\frac{m^2}{t}$$
>
> 得到:
>
> $$m=8\sqrt{3}\times 2^{\frac{n}{2}}$$

3. 若进一步改进算法, 最新的算法的时间复杂度为 $T(n)=8$, 其余条件不变, 在新机器上运行, 在 $t$ 秒内能够解决输入规模为多大的问题?

> 新算法的时间复杂度为$T(n)=8$，这意味着无论输入的规模是多少，算法都需要8单位时间来完成。
>
> 所以，新机器在$t$秒内可以处理任意大的输入规模，因为算法的执行时间是常数，与输入规模无关。

8. **Fibonacci 数的递推关系**:

$$F(n)=\begin{cases}1 & \text{if } n=0\\ 1 & \text{if } n=1\\ F(n-1)+F(n-2) & \text{if } n>1\end{cases}$$

试求出 $F(n)$ 的表达式。

---

> 从Fibonacci数列的递推关系出发:
>
> $$F(n)=F(n-1)+F(n-2)$$
>
> 我们可以将其视为一个线性齐次递推关系。为了解这个关系，我们尝试找到一个形如 $r^n$ 的解，其中 $r$ 是一个常数。代入递推关系，我们得到:
>
> $$r^n=r^{n-1}+r^{n-2}$$
>
> 除以 $r^{n-2}$，我们得到:
>
> $$r^2=r+1$$
>
> 这是一个二次方程，解之，我们得到:
>
> $$r_1=\frac{1+\sqrt{5}}{2}=\phi$$
> $$r_2=\frac{1-\sqrt{5}}{2}=-\phi^{-1}$$
>
> 因此，Fibonacci数列的通解可以写成:
>
> $$F(n)=A\phi^n+B(-\phi^{-1})^n$$
>
> 其中，A和B是待定的常数。使用初始条件 $F(0)=0$ 和 $F(1)=1$，我们可以解出A和B。
>
> 当 $n=0$ 时:
>
> $$F(0)=A+B=0$$
>
> 当 $n=1$ 时:
>
> $$F(1)=A\phi+B(-\phi^{-1})=1$$

解这个线性方程组，我们得到：

$$A = \frac{1}{\sqrt{5}}$$
$$B = -\frac{1}{\sqrt{5}}$$

代入上面的通解，我们得到：

$$F(n) = \frac{\phi^n - (-\phi^{-1})^n}{\sqrt{5}} \$\$$$

这就完成了$F(n)$的推导。