算法作业

朱一帆-58-智科

1-3 按照渐进阶从低到高的顺序排列以下表达式：$4n^2, \log n, 3^n, 20n, 2, n^{\frac{2}{3}}$. 又$n!$应该排在哪一位?

解：常数阶：2

对数阶：$\log n$

多项式阶：$4n^2$, $n^{2/3}$, $20n$

指数阶：3^n

故由低到高排列顺序为：

$2, \log n, n^{2/3}, 20n, 4n^2, 3^n$

由Stirling公式可知，$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$

故 $n! = O(n^n)$，$n! = \Omega(2^n)$

所以$n!$应排在3^n后面.

1-4 （1）假设某算法在输入规模为n时的计算时间为$T(n) = 3 \times 2^n$。
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，那么在这台新机器上用同一算法在t秒内能解输入规模为多大的问题?

解：设原来机器能解决问题规模为n_1，

新机器为n_2. 由题意知.

$t = 3\times 2^{n_1}$，$t = \frac{1}{64}\times 3\times 2^{n_2}$

联立解得 $n_2 = n_1 + 6$

（2）若上述算法的计算时间改进为$T(n) = n^2$，其余条件不变，则在新机器上用t秒时间能解输入规模为多大的问题?

解：同(1)设的未知数. 可得方程

$t = n_1^2$，$t_2 = \frac{1}{64}n_2^2$

解得 $n_2 = 8n_1$

（3）若上述算法的时间进一步改进为$T(n) = 8$，其余条件不变，则在新机器上用t秒时间能解输入规模为多大的问题?

解：因为所需时间与问题规模n无关. 故n_2可以取任意值

1-6 对于下列各组函数$f(n)$和$g(n)$，确定$f(n) = O(g(n))$或$f(n) = \Omega(g(n))$或$f(n) = \theta(g(n))$，并简述理由。

解：(1) $f(n) = \log n^2$，$g(n) = \log n + 5$

$\because f(n)=\log n^2=2\log n=\Theta(\log n)$

$g(n)=\log n+5=\Theta(\log n)$

$\therefore f(n)=\Theta(\log n)$

(2) $f(n)=\log n^2,\ g(n)=\sqrt{n}$

$f(n)$为对数阶，$g(n)$为多项式阶.

$\therefore f(n)=O(g(n))$

(3) $f(n)=n,\ g(n)=\log^2 n$

$\because f(n)$为多项式阶，$g(n)$为对数阶.

$\therefore f(n)=\Omega(g(n))$

(4) $f(n)=n\log n+n,\ g(n)=\log n$

$\because f(n)=\Theta(n\log n),\ g(n)=\Theta(\log n)$

$\therefore f(n)=\Omega(g(n))$

(5) $f(n)=10,\ g(n)=\log 10$

$\because f(n)$为常数阶，$g(n)$为常数阶

$\therefore f(n)=\Theta(g(n))$

(6) $f(n)=\log^2 n,\ g(n)=\log n.$

$n\geqslant 2$时，$f(n)\geqslant cg(n)$

$\therefore f(n)=\Omega(g(n))$

(7) $f(n)=2^n,\ g(n)=100n^2$

$\because f(n)$为指数阶，$g(n)$为多项式阶.

$\therefore f(n)=\Omega(g(n))$

(8) $f(n)=2^n,\ g(n)=3^n$

当 $n \geq 1$ 时，$f(n) \leq cg(n)$

$\therefore f(n) = O(g(n))$

算法实现1-1（题略，课本P7）

解：通过遍历法，从1到n统计每个数字出现的次数，代码如下.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int page[10] = { 0 };
    int pages;
    int i;
    cout << "Pages:";
    cin >> pages;
    for (i = 1; i <= pages; i++)
    {
        int t = i;
        while (t) {
            page[t % 10]++;
            t /= 10;
        }
    }


    for (i = 0; i < 10; i++)
        cout << page[i] << endl;
    return 0;
}
```

$O(\lg n)$ 和位数有关

$O(n)$

时间复杂度为 $O(n\lg n)$

运行结果如下

```
C:\WINDOWS\system32\
Pages:11
1
4
1
1
1
1
1
1
1
1
请按任意键继续. . .
```

通过查找资料，找到一种时间复杂度为 $O(\lg n)$ 的算法.

通过查找资料，找到一种时间复杂度O(logn)的算法.

https://blog.csdn.net/qq_40889820/article/details/88092340

https://blog.csdn.net/acm17773729889/article/details/78090184

分析与解答：考察由 0,1,2,…,9 组成的所有 n 位数。从 n 个 0 到 n 个 9 共有 10^n 个 n 位数。在这 10^n 个 n 位数中，0,1,2,…,9 每个数字使用次数相同，设为 $f(n)$。$f(n)$满足如下递归式：

$$f(n)=\begin{cases}10f(n-1)+10^{n-1} & n>1\\ 1 & n=1\end{cases}$$

由此可知，$f(n)=n10^{n-1}$。

据此，可从高位向低位进行统计，再减去多余的 0 的个数即可。

算法过程如下图例子所示、

5678。

1）[0,999]间0~9出现的次数可以直接由公式得出。f[3]=3*100=300。

2）[1000,1999]、[2000,2999]、[3000,3999]、[4000,4999]间0~9出现的次数由一开始对公式的推导可以得到启发：分为两部分计算，**最高位**和**非最高位**，非最高位好算，不就是复制的4份[0,999]吗，公式解决，最高位也好算，不就是4份3个0到3个9组成的10^3的个数吗。

3）[5000,5678]，同样分为两部分，**最高位**和**非最高位**，最高位是679个5，最低位就是计算[0,678]，嗯？嗯，问题就这样缩小规模了。