1.绪论

局限

字宽

God kisses the finite in his love and

man the infinite.

邓俊辉

deng@tsinghua.edu.cn

幂函数：$\mathcal{O}(2\^{}r)$

❖ 观察：同一问题的不同算法，复杂度不尽相同

❖ 问题：对任何给定的整数 $n > 0$，计算 a^n（a为常数）

❖ 平凡实现：

```
pow = 1; //O(1)
while ( 0 < n ) { //O(n)
  { pow *= a; n--; } //O(1) + O(1)
```

❖ 复杂度与n成正比，$T(n) = 1 + n*2 = \mathcal{O}(n)$

线性？$\boxed{\text{伪线性}}$！

❖ 所谓输入规模，准确地应定义为 $\boxed{\text{用以描述输入所需的空间的规模}}$

对于此类数值计算，即是n的二进制位数 $r = \log_2(n + 1)$

❖ 复杂度与r成指数关系，$T(r) = \mathcal{O}(2^r)$ //太高了

a^{98765}

$= a \wedge [9 \times 10^4 + 8 \times 10^3 + 7 \times 10^2 + 6 \times 10^1 + 5 \times 10^0]$

$= (a\wedge 10^4)\wedge 9 \times (a\wedge 10^3)\wedge 8 \times (a\wedge 10^2)\wedge 7 \times (a\wedge 10^1)\wedge 6 \times (a\wedge 10^0)\wedge 5$

= pow(pow(a, 10^4) , 9)	= pow(pow(pow(a, 10^3) , 10) , 9)
× pow(pow(a, 10^3) , 8)	× pow(pow(pow(a, 10^2) , 10) , 8)
× pow(pow(a, 10^2) , 7)	× pow(pow(pow(a, 10^1) , 10) , 7)
× pow(pow(a, 10^1) , 6)	× pow(pow(pow(a, 10^0) , 10) , 6)
× pow(pow(a, 10^0) , 5)	× pow(pow(a, 10^0) , 5)

❖ 若能在$O(1)$时间内得到pow(x, n)，$0 \le n \le 10$

则自右（低）向左（高），每个数位只需$O(1)$时间

a^{10110b}

$$= a \wedge [1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0]$$

$$= (a\wedge 2^4)\wedge 1 \times (a\wedge 2^3)\wedge 0 \times (a\wedge 2^2)\wedge 1 \times (a\wedge 2^1)\wedge 1 \times (a\wedge 2^0)\wedge 0$$

$$\begin{aligned} = \;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^4)}\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^3)}\ ,\ 0\ ) \\ \times\;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^2)}\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^1)}\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^0)}\ ,\ 0\ ) \end{aligned}$$

$$\begin{aligned} = \;& \texttt{pow}(\ \texttt{sqr}(\ \underline{\texttt{pow}(a,\ 2^3)}\ )\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \texttt{sqr}(\ \underline{\texttt{pow}(a,\ 2^2)}\ )\ ,\ 0\ ) \\ \times\;& \texttt{pow}(\ \texttt{sqr}(\ \underline{\texttt{pow}(a,\ 2^1)}\ )\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \texttt{sqr}(\ \underline{\texttt{pow}(a,\ 2^0)}\ )\ ,\ 1\ ) \\ \times\;& \texttt{pow}(\ \underline{\texttt{pow}(a,\ 2^0)}\ ,\ 0\ ) \end{aligned}$$

❖ pow(x, 0) = 1 //$\mathcal{O}(1)$

pow(x, 1) = x //$\mathcal{O}(1)$

pow(x, 2) = sqr(x) //$\mathcal{O}(1)$

❖ **故对应于每个数位，只需$\mathcal{O}(1)$时间！**

幂函数：$\mathcal{O}(r)$

- int power(int a, int n) {

```
int power( int a, int n ) {
    int pow = 1; int p = a; //O(1 + 1)
    while (0 < n) { //O(logn)
        if (n & 1) pow *= p; //O(1 + 1)
        n >>= 1; p *= p; //O(1 + 1)
    }
    return pow; //O(1)
}
```

- 输入规模 = n的二进制位数 = r = $\lceil \log_2(n+1) \rceil$
- 复杂度主要取决于$\boxed{\text{循环次数}}$，$T(r) = 1 + 1 + 4 \times r + 1 = \mathcal{O}(r)$
- 从$\mathcal{O}$(2^r)到$\mathcal{O}$(r)的改进，实际效果如何？

悖论？

- 观察：power(n) = a^n的二进制展开宽度，可以度量为$\Theta(n)$
- 根据以上算法，可在$\mathcal{O}(r = \log n)$时间内计算出power(n)
- 然而，即便是直接打印power(n)，也至少需要$\Omega(n)$时间

  ...哪里错了？

- RAM模型

  常数代价准则（uniform cost criterion）

  对数代价准则（logarithmic cost criterion）