## 绪论

计算模型: RAM

当有加减施加拳脚的地万,理性便有了容身之处;而在加减无所适从的地方,理性也就失去了容身之所。

三年之喪,二十五月而畢,哀痛未盡,思慕未忘,然而禮以是斷之者,豈不以送死有已,復生有節也哉!



### Random Access Machine: 组成 + 语言

# IR | R[0] | R[1] | R[2] | R[3] |

• • • • •

❖ 寄存器顺序编号,总数没有限制

//但愿如此

**❖ 可通过编号直接访问任意寄存器** 

//call-by-rank

❖ 每一基本操作仅需常数时间

//循环及子程序本身非基本操作

#### Random Access Machine: 效率

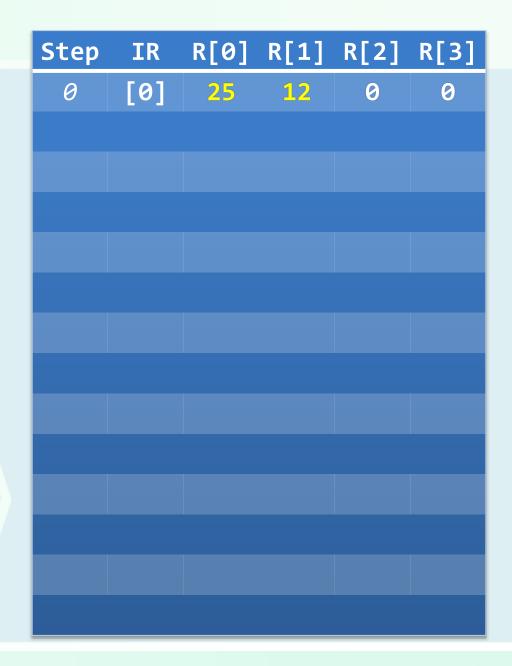
- ❖ 与TM模型一样,RAM模型也是一般计算工具的简化与抽象 使我们可以独立于具体的平台,对算法的效率做出可信的比较与评判
- ❖ 在这些模型中
  - 算法的运行时间 ∞ 算法需要执行的基本操作次数
  - T(n) = 算法为求解规模为n的问题,所需执行的基本操作次数
- ❖ 思考:在TM、RAM等模型中衡量算法效率,为何通常只需考查运行时间?空间呢?

## 实例: Ceiling Division: 思路

$$\Leftrightarrow \forall c \geq 0 \text{ and } d > 0$$
, define 
$$\lceil c/d \rceil = \min\{ x \mid c \leq d \cdot x \}$$
 
$$= \min\{ x \mid c - 1 < d \cdot x \}$$

\* 例如: 
$$\lceil 2/7 \rceil = 1$$
 
$$\lceil 35/5 \rceil = 7 \qquad \lceil 2021/43 \rceil = 47$$
 
$$\lceil 41/7 \rceil = 6 \qquad \lceil 2022/43 \rceil = 48$$

❖ 思路: 反复地从 R[0] = c 中, 减去 R[1] = d
统计在下溢之前,所做减法的次数x



## 实例: Ceiling Division: 算法

```
[0] R[3] <- 1 //constant increment
[1] GOTO 4 //in case R[0] = c == 0
[2] R[2] \leftarrow R[2] + R[3] //x++
[3] R[0] \leftarrow R[0] - R[1] //c -= d
[4] IF R[0] > 0 GOTO 2 //if c > 0 goto 2
[5] R[0] \leftarrow R[2] //else copy x to R[0] and
[6] STOP //return R[0] = x = c/d
```

## 时间成本 ~ 各条指令执行次数之总和

Step	IR	R[0]	R[1]	R[2]	R[3]
0	[0]	<b>25</b>	<b>12</b>	0	0
1	[1]	^	^	^	1
2	[4]	^	^	^	^
3	[2]	^	^	^	^
4	[3]	^	^	1	^
5	[4]	13	^	^	^
6	[2]	^	^	^	^
7	[3]	^	^	2	^
8	[4]	1	^	^	^
9	[2]	^	^	^	^
10	[3]	^	^	3	^
11	[4]	-11	^	^	^
12	[5]	^	^	^	^
13	[6]	3	^	^	^