Azzolini Riccardo 2019-02-26

Complessità computazionale

1 Complessità computazionale

La **complessità computazionale** di un algoritmo è la quantità di risorse utilizzate durante la computazione (nell'ambito di un *sistema formale generale* quale la macchina RAM).

Minore è l'uso di risorse, maggiore è l'efficienza dell'algoritmo.

Le risorse vengono definite come costi in termini di *tempo* e *spazio*, secondo due possibili criteri di costo: *uniforme* e *logaritmico*.

2 Criterio di costo uniforme

Il **criterio di costo uniforme** (**CCU**) assume che ogni operazione abbia costo unitario in tempo e spazio, e quindi che:

- ogni operando abbia dimensione unitaria;
- ogni registro occupi spazio unitario.

Secondo questo criterio, un programma RAM su input $\underline{x}$ richiede tempo di calcolo t e spazio di memoria s se la computazione di P su $\underline{x}$ esegue t istruzione e utilizza s registri:

$$T_P(\underline{x}) = t \qquad S_P(\underline{x}) = s$$

Per convenzione, si scrive:

- $t = \infty$ se la computazione non termina;
- $s = \infty$ se si utilizza un numero illimitato di registri.

Osservazione: $s = \infty \implies t = \infty$ (ma non viceversa), perché $s \leq t$.

2.1 Esempio

Calcolo del massimo tra n valori:

```
    READ     1
2   JBLANK   10
    LOAD     1
    READ     2
    SUB      2
    JGTZ     2
    LOAD     2
    STORE    1
    JUMP     2
10  WRITE    1
    HALT
```

Per ogni input $\underline{x} \in \mathbb{Z}^n$, si ha

- $S_P(\underline{x}) = 3 = \Theta(1)$ perché vengono sempre usati solo 3 registri;
- $T_P(\underline{x}) = 5(n-1) + 4 = \Theta(n)$ se il primo numero (x_1) è maggiore di tutti gli altri (*caso migliore*);
- $T_P(\underline{x}) = 8(n-1) + 4 = \Theta(n)$ se $\underline{x}$ è crescente (*caso peggiore*).

2.2 Limiti

Il criterio di costo uniforme non tiene conto delle dimensioni degli interi utilizzati: nella realtà, i registri possono contenere solo un numero limitato di bit, quindi le operazioni con numeri grandi *non* hanno costo costante.

Di conseguenza, questo criterio è realistico solo per algoritmi che non incrementano troppo le dimensioni dei valori in ingresso.

3 Criterio di costo logaritmico

Secondo il **criterio di costo logaritmico** (**CCL**), il costo di un'istruzione dipende dalla **dimensione** dell'operando, che per un intero è la sua **lunghezza**, cioè il numero di bit necessari alla sua memorizzazione:

$$l(k) = \lfloor \log_2(|k|) \rfloor + 1$$

Nota: per k grande, $l(k) \approx log_2|k|$.

3.1 Costo degli operandi

Il costo dell'operando a nello stato S è espresso dalla funzione $t_S(a)$:

Operando a	Costo $t_S(a)$
`=i`	$l(i)$
`i`	$l(i) + l(S(i))$
`*i`	$l(i) + l(S(i)) + l(S(S(i)))$

3.2 Costo delle istruzioni

Istruzione	Costo
`LOAD a`	$t_S(a)$
`STORE i`	$l(S(0)) + l(i)$
`STORE *i`	$l(S(0)) + l(i) + l(S(i))$
`ADD a`	$l(S(0)) + t_S(a)$
`SUB a`	$l(S(0)) + t_S(a)$
`MULT a`	$l(S(0)) + t_S(a)$
`DIV a`	$l(S(0)) + t_S(a)$
`READ k`	$l\left(x_{S(r)}\right) + l(k)$
`READ *k`	$l\left(x_{S(r)}\right) + l(k) + l(S(k))$
`WRITE a`	$t_S(a)$
`JUMP b`	1
`JGTZ b`	$l(S(0))$
`JZERO b`	$l(S(0))$
`JBLANK b`	1
`HALT`	1

3.3 Tempo di calcolo logaritmico

Il **tempo di calcolo logaritmico** $T_P^l(\underline{x})$ di un programma P su dati in ingresso $\underline{x}$ è la somma dei costi logaritmici delle istruzioni eseguite nella computazione di P su $\underline{x}$.

Osservazioni:

- $T_P(\underline{x}) \leq T_P^l(\underline{x}) \quad \forall x \in \mathbb{Z}^n$
- In alcuni casi, $T_P(\underline{x}) \ll T_P^l(\underline{x})$.

3.4 Spazio logaritmico

Allo stato S_i, lo spazio utilizzato è la somma delle lunghezze degli interi contenuti nei registri utilizzati.

Lo **spazio logaritmico** complessivo $S_P^l(\underline{x})$ è il massimo valore che raggiunge lo spazio utilizzato durante la computazione:

$$S_P^l(\underline{x}) = \max_{i \geq 0} \left\{ \sum_{j \geq 0} l(S_i(j)) \right\}$$

3.5 Esempio

Calcolo del massimo tra n valori:

```
    READ     1
2   JBLANK   10
    LOAD     1
    READ     2
    SUB      2
    JGTZ     2
    LOAD     2
    STORE    1
    JUMP     2
10  WRITE    1
    HALT
```

Si suppone che tutti gli interi in input siano compresi tra 1 e k, quindi la dimensione dell'intero più grande è $\log k$: si può quindi dire che la dimensione di un intero qualsiasi è $O(\log k)$.

Nel caso di questo programma, moltiplicando la dimensione di un intero

- per il numero di istruzioni eseguite, $O(n)$, si ricava il tempo di calcolo:

$$T_P^l(\underline{x}) = O(n \log k)$$

- per il numero di registri utilizzati, $O(1)$, si ottiene lo spazio utilizzato:

$$S_P^l(\underline{x}) = O(\log k)$$

4 Esempio di confronto tra CCU e CCL

Calcolo di 3^{2^n}, su input $n \in \mathbb{N}$:

```
            READ          1   read x
            LOAD         =3
            STORE         2   y ← 3
            LOAD          1
  while     JZERO  endwhile   while x ≠ 0 do
            LOAD          2
            MULT          2
            STORE         2     y ← y × y
            LOAD          1
            SUB          =1
            STORE         1     x ← x − 1
            JUMP      while   end while
endwhile    WRITE         2   write y
            HALT
```

4.1 CCU

Il ciclo `while` viene percorso n volte, quindi

$$T_P(n) = 8n + 7 = \Theta(n)$$

4.2 CCL

Dopo k iterazioni, R_2 contiene 3^{2^k}, quindi ciascuna delle istruzioni

```
LOAD   2
MULT   2
STORE  2
```

ha costo logaritmico

$$l\left(3^{2^k}\right) \approx 2^k \log_2 3$$

Trascurando i fattori costante $\log_2 3$ e 3 (il numero di istruzioni con questo costo), si ottiene il costo complessivo

$$T_P^l(n) = \Theta\left(\sum_{k=0}^{n} 2^k\right) = \Theta(2^n)$$

perché, in generale,

$$\sum_{k=0}^{r-1} 2^k = \underbrace{1111\ldots1}_{r}{}_2 = 1\underbrace{0000\ldots0}_{r}{}_2 - 1 = 2^r - 1$$

È evidentemente necessario utilizzare il criterio di costo logaritmico per ottenere una stima realistica della complessità di questo programma, che è molto superiore a quella ricavata con il CCU: $\Theta(2^n) \gg \Theta(n)$.