Introduzione agli algoritmi

November 4, 2022

Contents

https://mega.nz/folder/4osyiZAI##2I2lpukRbJ-n7-HsmHLxhA/folder/x1szTAhI

0.1 RAM

La RAM è la componente del computer che si occupa di memorizzare i bits. I byte, struttura minima indirizzabile, con l'indirizzo che corrisponde allo scostamento in bytes dall'inizio del memory block del programma, dalla CPU, sono raggruppati a loro volta in **words**, l'unità massima quali la CPU può operare con una **singola** istruzione.
Il tempo impiegato dalla RAM per accedere a un byte è $\Theta(1)$.
Il numero massimo di indirizzi usabili è detto **spazio di memorizzazione**, e corrisponde alla lunghezza di una word nel sistema preso in considerazione.

0.2 Il modello random access machine

Il modello usato nell'analisi della complessità computazionale di un algoritmo.
Le caratteristiche sono:

- possiede un singolo processore che effettua operazioni sequenzialmente
- possiede un insieme di **operazioni elementari** dal costo $\Theta(1)$
- possiede un **limite** massimo di grandezza per ogni valore memorizzato e per lo spazio di indirizzamento

0.3 Criterio di misurazione del costo

0.3.1 Criterio di misurazione del costo uniforme

Data la dimensione d di una word, se tutti i dati hanno grandezza $< d$, le operazioni elementari su essi avranno costo $\Theta(1)$.
Non molto realistico in quanto molto spesso i dati hanno grandezza $> d$ ma spesso usato in quanto più semplice

0.3.2 Criterio di misurazione del costo logaritmico

Criterio più realistico ma che rende le misurazioni più complicate, afferma che il costo di una singola operazione su un dato di grandezza n è $\log_2 n$.

0.4 Notazione asintotica

La notazione asintotica è un modello astratto che descrive, in termini di tempo d'esecuzione e/o di memoria usata, il **tasso di crescita** del costo di una funzione in base alla grandezza dell'input.

0.4.1 Notazione O

Prese $f(n), g(n) \geq 0$ si dice che $f(n)$ è un O(g(n)) se

$$\exists c,\, n_0 \text{ t.c. } c * g(n) \geq f(n) \geq 0 \, \forall n \geq n_0$$

Di infinite funzioni $g(n)$ a noi interessa quella che meglio approssima $f(n)$ dall'alto

0.4.2 Notazione Ω

Come la notazione O, solo che approssimiamo dal basso.

0.4.3 Notazione Θ

f(n) è un Θ(g(n)) quando f(n) è O(g(n)) $\wedge \Omega$g(n)

0.4.4 Algebra della notazione asintotica

$$\forall k > 0,\, f(n) \geq 0,\, f(n) \text{ è un } O/\Omega/\Theta(g(n)) \Longrightarrow k * f(n) \text{ è un } O/\Omega/\Theta(g(n))$$

$$\forall f(n), d(n) \geq 0, f(n) \text{ è un } O/\Omega/\Theta(g(n)) \wedge d(n) \text{ è un } O/\Omega/\Theta(h(n)) \Longrightarrow$$
$$f(n) + d(n) \text{ è un } O/\Omega/\Theta(\max(g(n), h(n)))$$

$$\forall f(n), d(n) \geq 0, f(n) \text{ è un } O/\Omega/\Theta(g(n)) \wedge d(n) \text{ è un } O/\Omega/\Theta(h(n)) \Longrightarrow$$
$$f(n) * d(n) \text{ è un } O/\Omega/\Theta(g(n) * h(n))$$

0.4.5 Calcolo della notazione asintotica tramite limiti

$\lim_{n\to\infty} \frac{f(n)}{g(n)} =$

- $k > 0 \Longrightarrow f(n) = \Theta$(g(n))
- $\infty \Longrightarrow f(n) = \Omega(g(n)) \neq \Theta(g(n))$
- $0 \Longrightarrow f(n) = O(g(n)) \neq \Theta(g(n))$

0.5 Ccosto computazionale di un algoritmo

0.5.1 Calcolo del costo

1. Individuare la **dimensione dell'input**
2. Formulare in modo chiaro l'algoritmo in **pseudocodice**
3. Seguire le regole base:
   - le operazioni elementari hanno costo $\Theta(1)$
   - l'istruzione `if cond then st1 else st2` ha come costo la somma fra il costo di verifica della condizione e il massimo dei costi di st1 ed st2
   - l'istruzioni iterative hanno come costo la somma fra la verifica della condizione * n e la somma dei costi massimi di ogni singola interazione (spesso uguale)
   - Il costo dell'algoritmo è la somma dei costi di tutte le istruzioni
4. Quando un algoritmo potrebbe avere costi diversi in base all'input, occorre valutare anche il **caso migliore** ed il **caso peggiore**
5. Quando si valuta il costo di un algoritmo senza conoscere l'input bisogna sempre considerare il caso peggiore

0.5.2 I problemi intrattabili e l'importanza dell'efficienza

Si chiamano problemi intrattabili quei problemi che, data una dimensione realistica dell'input, il costo computazionale è proibitivo.
Per questo, quando si progetta un algoritmo oltre alla correttezza dello stesso è fondamentale risolverlo in modo efficiente

0.6 Problema della ricerca

Problema molto ricorrente, si analizza nella versione con I = A[] ed output l'indice dell'elemento x cercato in A[].

0.6.1 Soluzione sequenziale

- Semplice
- Caso peggiore: Θ(n)
- Caso migliore: $\Theta(1)$
- Analisi del costo: considerando ogni possibile posizione di x come equiprobabile, il numero medio di iterazione è

$$\Sigma_{i=1}^{n} i\frac{1}{n} = \frac{1}{n} * \frac{n*(n+1)}{2} = \Theta(\text{n})$$

0.6.2 Ricerca binaria

- Prerequisito: A[] è ordinato
- Caso peggiore: $\Theta(\log_2 n)$
- Caso migliore: $\Theta(1)$
- Analisi del costo: ad ogni iterazione i escludiamo $2^{i-1} - 1$ posizioni e ne confrontiamo 1. Quindi, il numero medio di iterazioni è

$$\Sigma_{i=1}^{\log_2 n} i\frac{2^{i-1}-1+1}{n} = \frac{1}{n}\Sigma_{i=1}^{\log_2 n} i2^{i-1} =$$

$$\frac{1}{n}\left(\left(\log_2 n - 1\right)2^{\log_2 n} + 1\right) = \log_2 n - 1 + \frac{1}{n}$$

0.7 Ricorsione

Un algoritmo **ricorsivo** è espresso in termini di se stesso.
La successione dei sottoproblemi che dividono la soluzione deve convergere verso uno o più **casi base** che termina la ricorsione.
Di norma, una soluzione ricorsiva ha **costi maggiori**, in termini di memoria e tempo di esecuzione, rispetto a una soluzione iterativa, dati dai costi intrinsechi dell'uso delle funzioni.
Una soluzione ricorsiva può sempre essere espressa anche come soluzione iterativa.
La ricorsione è detta **diretta** se f chiama f, mentre è **indiretta** se f chiama g che chiama f

0.7.1 Calcolare il costo di una funzione ricorsiva

La prima cosa da fare è impostare l'**equazione di ricorrenza**, costituita dalla formulazione ricorsiva e dal caso base.
Ad esempio, poniamo di avere un algoritmo che fa un test su $x \in A[]$ che, se soddisfatto interrompe la ricorsione, in caso contrario effettua una chiamata ricorsiva su $A[1:]$. Il costo computazionale dell'algoritmo A è quindi:

- T(n): T(n-1) + Θ(n)
- T(1): Θ(n)

Metodo di sostituzione

1. si ipotizza una soluzione per l'equazione di ricorrenza
2. per **induzione**, si verifica la correttezza della soluzione

La vera difficoltà nel metodo sta nel trovare O ed Ω che meglio approssimano f.
Prendendo l'algoritmo A, proviamo a calcolarne il costo col metodo di sostituzione.

1. Sostituiamo le notazioni asintotiche con costanti:
   - T(n) = T(n-1) + c (c > 0)
   - T(1) = d (d > 0)
2. Ipotizziamo T(n) = O(n), con T(n) $\leq k * n$ con k da determinare
3. Nel caso base abbiamo quindi T(1) $\leq k * n = k * 1 \implies d \leq k$
4. Assumiamo quindi T(r) $\leq k * r \forall r < n \implies$
5. T(n) = T(n-1) + c $\leq kn \implies$ T(n) $\leq k(n-1) + c \leq kn \implies k \geq c$
6. L'ipotesi è quindi vera $\forall k \geq c$, $d \implies$ T(n) è un O(n)
7. Ipotizziamo ora T(n) = Ω(n), con T(n) $\geq hn$ con h da determinare
8. Analogamente a prima, T(1) $\geq hn = h \implies d \geq h$
9. Sempre analogamente a prima assumiamo T(r) $\geq hr \forall r < n \implies$
10. T(n) = T(n-1) + c $\geq hn \implies$ T(n) $\geq h(n-1) + c \geq hn \implies h \leq c$
11. L'ipotesi è quindi vera $\forall h \leq c$, $d \implies$ T(n) è un Ω(n)
12. Essendo T(n) un Ω(n) ed un O(n) $\implies$ T(n) è un Θ(n)

Analizziamo ora un altro caso definito da:

- $T(n) = 2T(\frac{n}{2}) + \Theta(1)$
- $T(1) = \Theta(1)$

1. Sostituiamo le notazione asintotiche con costanti:
   - $T(n) = 2T\frac{n}{2} + c$ $(c > 0)$
   - $T(1) = d$ $(d > 0)$
2. Ipotizziamo $T(n) = O(n)$, con $T(n) \leq kn$
3. Nel caso base abbiamo quindi $T(1) \leq kn = k \implies d \leq k$
4. Analogamente a prima, $T(n) = 2T(\frac{n}{2}) + c \leq kn \implies T(n) \leq 2^*(k * \frac{n}{2}) + c \leq kn \implies$
   $T(n) \leq kn + c \leq kn \implies c \leq 0$, impossibile in quanto c è per definizione > 0.
5. La funzione quindi non è limitata superiormente da kn
6. Proviamo con $T(n) \leq kn - h$
7. Caso base: $T(1) \leq kn - h = k - h \implies d + h \leq k$
8. $T(n) = 2(k * \frac{n}{2} - h) + c \leq kn - h \implies kn - 2h + c \leq kn - h \implies$
   $c - h \leq 0 \implies c \leq h \implies$
9. $T(n)$ è un $O(n)$
10. Analogamente possiamo dimostrare che $T(n)$ è un $\Omega(n)$

Metodo iterativo

L'idea di base è di sviluppare l'equazione di ricorrenza come **somma fra caso base e termini dipendenti da n**.
Prendendo la funzione di prima:

1. $T(n) = T(n-1) + \Theta(1)$, $T(n-1) = T(n-2) + \Theta(1) \implies$
2. $T(n) = T(n-2) + \Theta(1) + \Theta(1)$, $T(n-2) = ... \implies$
3. $T(n) = T(1) + (n-1)\Theta(1)$, $T(1) = \Theta(1) \implies$
4. $T(n) = n\Theta(1) = \Theta(n)$

Non tutte le equazione di ricorrenza sono però efficacemente risolvibili col metodo classico, per esempio l'algoritmo della sequenza di fibonacci:

- $T(n) = T(n-1) + T(n-2) + \Theta(1)$
- $T(0) = T(1) = \Theta(1)$

1. Possiamo però notare che $2T(n-2)+\Theta(1) \leq T(n) \leq 2T(n-1) + \Theta(1)$

2. Risolviamo il limite superiore:
   $T(n) \leq 2T(n\text{-}1) + \Theta(1) \leq 2^2 T(n\text{-}2) + 3\Theta(1)\ leq 2^3 T(n\text{-}3) + 7\Theta(1) \Longrightarrow$
   $\leq 2^k T(n\text{-}k) + \left(2^k - 1\right)\Theta(1)$, che si ferma con $k = n-1$, per cui otteniamo:
   $T(n) \leq 2^{n-1}\Theta(1) + \left(2^{n-1} - 1\right)\Theta(1) = (2^n - 1)\Theta(1) \Longrightarrow$

3. T(n) è un $O(2^n)$

4. Analogamente risolviamo il limite inferiore:
   $T(n) \geq 2T(n\text{-}2) + \Theta(1) \geq 2^2 T(n-4) + 3\Theta(1) \Longrightarrow$
   $\geq 2^k T(n\text{-}2k) + \left(2^k - 1\right)\Theta(1)$, che si ferma con $k = \frac{n}{2}$ per cui otteniamo:
   $T(n) \geq 2^{\frac{n}{2}}\Theta(1) + \left(2^{\frac{n}{2}} - 1\right)\Theta(1) = \left(2^{\frac{n+1}{2}} - 1\right)\Theta(1) \Longrightarrow$ T(n) è un $\Omega\left(2^{\frac{n}{2}}\right)$

5. Quindi, $k_1 2^{\frac{n}{2}} \leq T(n) \leq k_2 2^n$

Metodo dell'albero

0.8 Esercizi

https://mega.nz/folder/4osyiZAI##2I2lpukRbJ-n7-HsmHLxhA/folder/l1tXEIrS

0.8.1 1

Il caso peggiore è $\Theta(n)$ in quanto scorre tutti i numeri da 1 ad n per poi effettuare un'operazione $\Theta(()1)$.
Esiste una versione $\Theta(()1)$ per effettuare il calcolo col metodo di Gauss: $\frac{n(n)}{2}$

0.8.2 2