2022

Algoritmi

E STRUTTURE DATI

MICHELE VANTAGGI

Algoritmi

E STRUTTURE DATI

Algoritmo

Un algoritmo è una procedura passo a passo per eseguire delle mansioni una finita quantità di volte

Algoritmo deterministico

Un algoritmo che dato lo stesso input esegue sempre gli stessi passi.

Questa materia si occupa di scegliere per ogni problema l'algoritmo più efficiente in tempo o spazio.

La maggior parte di questi algoritmi trasforma un input in un output, e il suo tempo di esecuzione aumenta in modo direttamente proporzionale con la dimensione dell'input.

Il tempo di esecuzione medio è spesso molto difficile da determinare, e quindi ci concentriamo sul ***peggior tempo di esecuzione*** che è più facile da analizzare e cruciale per la maggior parte delle applicazioni.

Esempi

Ho 2 array:

A: array [1: n] of integers

A is permutation of B?

Input

A = 5, 7, 3, 1

B = 3, 1, 7, 5

Risposta: si

Input

A = 5, 7, 3, 1

B = 2, 7, 3, 1

Risposta: no

Questo fa schifo:

```
bool permute(int* A, int* B){
    int count = 0;
    for( int i = 0; i<n; i++){
        int j = 0;
        while(j<n && A[i] == B[j] )
```

```
            j++;
        if(B[j] == A[i]) count++;
    }
    return count == n;
}
```

Questo è meglio:

```
int permut(int* A, int* B){
    int count = 0;
    for( int i = 0; i<n; i++){
        bool trovato = false;
        for(int j = 0; j<n && !trovato; j++){
            if(B[j] == A[i]){
                trovato = true;
                count++;
            }

        }
        if(!trovato)
            return false;

    }
    return count == n;
}
```

La terza opzione implica l'utilizzo di un algoritmo di sorting per ordinare i vettori e controllare che nelle posizioni ci sia lo stesso elemento, la complessità dell'algoritmo dipende quindi dalla complessità degli algoritmi di sorting utilizzati.

Algoritmo randomizzato

È un algoritmo che, dato uno stesso input, l'output sarà sempre diverso

Scalabilità

Poter risolvere istanze dello stesso problema di dimensioni estremamente variabili gli scienziati devono trovare delle soluzioni che siano scalabili.

Esempio

Molte compagnie nei colloqui di lavoro valutano la capacità di problem solving chiedendo domande circa algoritmi e strutture di dati da utilizzare anche in atti pratici.

Struttura di dati

Una struttura di dati è un modo sistematico di organizzare ed accedere ai dati

Studi sperimentali

Per studiare il comportamento di un algoritmo bisogna:

- Scrivere un programma che implementi un algoritmo
- Eseguire il programma molteplici volte con input di dimensioni e composizioni diverse
- Fare il grafico di esecuzione

Limitazioni

- Implementare l'algoritmo potrebbe essere complicato
- I risultati potrebbero non comprendere il tempo di esecuzione su altri input non sperimentati
- Per comparare due algoritmi bisogna utilizzare lo stesso ambiente software e hardware

Analisi teorica

- Usa una descrizione ad alto livello dell'algoritmo al posto della sua implementazione
- Mostra il tempo di esecuzione come una funzione di input la dimensione n
- Prende in considerazione tutti gli input
- Permette di valutare la velocità di un algoritmo indipendentemente dall'ambiente

Pseudocodice

- Descrizione ad alto livello di un algoritmo
- Più strutturato del linguaggio comune
- Meno dettagliato di un programma
- Notazione preferita per descrivere un algoritmo
- Nasconde i problemi di design di un programma

Tipi di funzioni

1. Costante -> l
2. Logaritmica -> log n
3. Lineare -> nn
4. N Log n -> n log n
5. Quadratica -> n^2
6. Cubica -> n^3
7. Esponenziale -> 2^n

In una funzione ***log(log(n))*** la pendenza (***derivata***) corrisponde al rateo di crescita.

Operazioni primitive

- Calcoli base eseguiti da un algoritmo
- Identificabili in pseoudocodice
- Indipendenti dai linguaggi di programmazione
- La definizione esatta non è importante
- Si assume l'impiego di un tempo costante nel modello RAM

Rateo di crescita del tempo di esecuzione

- Cambiando software e hardware
    - Cambia T(n) di un fattore costante

    - Non altera il rateo di crescita di T(n)
- Il rateo di crescita lineare del tempo di esecuzione T(n) è una proprietà intrinseca dell'algoritmo arrayMax (ricerca del valore massimo di un vettore)

$\Omega(n)$

- Rappresenta il caso migliore in riferimento a uno specifico algoritmo.
- Rappresenta la complessità intrinseca del problema. (indipendente dall'algoritmo pensato)

Un algoritmo è ottimo quando il caso migliore è uguale al caso peggiore

Analisi di un algoritmo ricorsivo

La funzione T(n) deriva da una **relazione di ricorrenza** che caratterizza il tempo di esecuzione sui valori più piccoli di n

```
Algorithm recursiveMax(A, n):
      input: array A of n >= 1 integers
      output: maximum element in A
if n = 1 then
      return A[0]
return max{recursiveMax(A, n-1), A[n-1]}
```

$$T(n) = \begin{cases} 3, & \text{n} = 1 \\ \text{T(n} - 1) + 7, & \text{altrimenti} \end{cases}$$

Fattori costanti

Il rateo di crescita è minimamente affetto da fattori costanti o da termini di ordine minore

Notazione Big-Oh

Date due funzioni f(n) e g(n), diciamo che f(n) è O(g(n)) se ci sono delle costanti c positive e n_0 tali che

$$f(n) \leq cg(n)\ for\ n \geq n_0$$

- Se f(n) è un polinomio di grado D, allora f(n) è $O(n^D)$
- Usa la classe di funzioni più piccola possibile
- Usa la forma più semplice di espressione della classe

Compagni di Big-Oh

Big Omega

F(n) è $\Omega(g(n))$ se esiste una costante c > 0 e una costante intera $n_0 \geq 1$ tali che

$$f(n) \geq cg(n) \forall n \geq n_0$$

Big Theta

F(n) è $\Theta(g(n))$ se ci sono costanti c' > 0 e c'' > 0 e una costante intera $n_0 \geq 1$ tali che

$$c'g(n) \leq f(n) \leq c''g(n), \forall n \geq n_0$$

ω

È l'inverso di O, quindi f(n) è $\omega(g(n))$ per ogni costante c > 0 se esiste una costante $n_0 > 0$ tale che

$$0 \leq cg(n) < f(n), \forall n \geq n_0$$

Insertion sort

INSERTION-SORT(A, n)

```
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

L'insertion sort ha un tempo di esecuzione O(n^2)

- Il for esterno viene eseguito n-1 volte
- Il for interno viene eseguito al massimo i-1 volte
- Il numero esatto di iterazione del while dipende dai valori su cui itera, che variano tra 0 e i-1
- Dato che i al massimo arriva ad n, il numero totale di iterazioni del ciclo interno è (n-1)(n-1), minore di n^2
- Ogni iterazione del loop interno impiega un tempo costante, con un totale di al più cn^2 per qualche costante c, oppure O(n^2)

Il tempo peggiore di esecuzione è quindi $\Omega(n^2)$

- Si osserva che un valore che viene spostato di k posizioni, la riga 6 (a[j+1] = a[j]) deve essere eseguita k volte
- Assumendo che n sia multiplo di 3, possiamo dividere l'array in 3 gruppi di n/3 posizioni
- Dato che almeno n/3 valori devono essere spostati di n/3 posizioni, la riga 6 viene eseguita almeno (n/3)(n/3) volte che sarebbe $\Omega(n^2)$

Dato che abbiamo mostrato che l'insertion sort ha tempo di esecuzione $O(n^2)$ e $\Omega(n^2)$, possiamo concludere che il peggior tempo di esecuzione è di $\Theta(n^2)$

Logaritmi

$$log_d(n) = \frac{log_a(n)}{log_a(d)} = c$$

$$
\begin{aligned}
a &= b^{\log_b a}\,, \\
\log_c(ab) &= \log_c a + \log_c b\,, \\
\log_b a^n &= n \log_b a\,, \\
\log_b a &= \frac{\log_c a}{\log_c b}\,, \\
\log_b(1/a) &= -\log_b a\,, \\
\log_b a &= \frac{1}{\log_a b}\,, \\
a^{\log_b c} &= c^{\log_b a}\,.
\end{aligned}
$$

Modello RAM (Random Access Machine)

Si il nome confonde.

La RAM è composta da

- CPU
- Un potenziale banco di celle di memoria sconnesse, ognuna dei quali può contenere un numero arbitrario di numeri o caratteri
- Celle di memoria numerate il cui accesso impiega delle unità di tempo

Ammortizzazione

Il **tempo di esecuzione ammortizzato** di un'operazione all'interno di una serie di operazioni è il tempo di esecuzione nel caso pessimo diviso per il numero di operazioni.

Esempio

l'estensione della dimensione di un array.

Ho un array di dimensione C, e voglio farlo arrivare a dimensione $N = kc, k \in Z$

Ogni volta che voglio ingrandire l'array di altre C dimensioni, copio la dimensione dell'array (1C, 2C,..ecc) e esegui C operazioni O(1).

Merge sort

Moltiplicazione tra matrici

Input

$$A = [a_{ij}], B = [b_{ij}]$$

Output

$$C = [c_{ij}] = A \cdot B$$

```
for i=1:n
    for j=1:n
             c[ij]=0
```