

CoderFarm - Corso base

Lezione 3

Carlo Collodel, Francesco Cerroni

15 novembre 2022

Ripasso della scorsa lezione

Array

```
#include <iostream>

using namespace std;

int main()
{
    int array[5] = {6, 9, 4, 2, 0};

    cout << array[0] << array[1] << endl;

    array[0] = 777;
    array[3] = 101010;

    for (int i = 0; i < 5; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

Ripasso della scorsa lezione

std::vector



```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> vec(10, 5); /* 10 elementi con valore iniziale 5 */

    vec[1] = 314;
    vec[9] = 271828;

    cout << "{" << vec[0] << ", " << vec[9] << "}\n";
}
```

std::vector

Altri metodi



```
vector<int> vec(3, 2); /* 3 elementi con valore iniziale 2 */

vec.push_back(7);
vec.insert(vec.begin(), 1);
vec.insert(vec.begin(), 2);

for(auto &el : vec) {
    cout << el << endl;
}

vec.erase(vec.rbegin()); /* elimino il primo elemento */

for(auto &el : vec) {
    cout << el << endl;
}

vec.clear(); /* svuoto il vettore */
```

std::vector

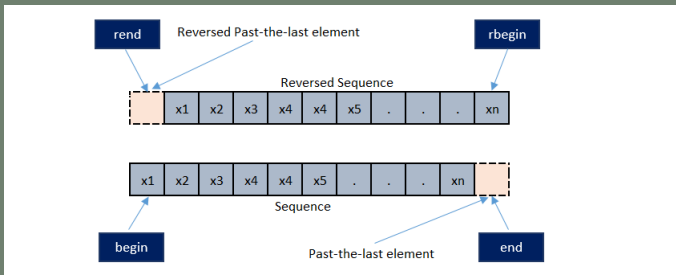
Altri metodi

Iteratori

Ci sono diversi tipi di *iteratori* nei vari contenitori di C++.

`std::vector::begin()` indica l'inizio di un vettore,

`std::vector::end()` punta a **dopo** la fine di un vettore.



Fonte: <https://alphacodingskills.com/>

Complessità asintotica

Definizione

Come misurare l'efficienza di un algoritmo?

Spesso in informatica (e durante le gare) è necessario misurare approssimativamente quante operazioni o quanta memoria un algoritmo userebbe qualora venisse eseguito da un computer. Un metodo usato molto spesso è quello basato sulla complessità asintotica: questo permette di trovare delle stime "abbastanza buone" per calcolare velocemente il numero di operazioni che richiede un algoritmo.

Complessità asintotica

O-grande

La Notazione

Il simbolo più usato per indicare questo tipo di stima è detto O-grande: $\mathcal{O}(\text{complessità})$.

Questo simbolo viene spesso usato per indicare il "caso peggiore" o il "caso medio" di un algoritmo.

Esempio

- ▶ Per stampare un array di dimensione N : $\mathcal{O}(N)$.
- ▶ Per accedere a un elemento di un array: $\mathcal{O}(1)$.
- ▶ Ricerca binaria: $\mathcal{O}(\log_2 N)$.

Complessità asintotica

O-grande

Combinare funzioni in notazione O-grande

Per combinare delle funzioni, per le somme si "scartano" le funzioni che crescono più lentamente e si tiene la funzione che cresce più velocemente per ogni variabile.

In caso di prodotti, semplicemente si moltiplicano le funzioni nella notazione.

Spesso si possono omettere fattori moltiplicativi dall'O-grande ($\mathcal{O}(N) = \mathcal{O}(100 \cdot N)$) e somme costanti ($\mathcal{O}(N) = \mathcal{O}(N + 1000)$). Noi di solito ignoreremo le somme di valori costanti, ma presteremo più attenzione alle moltiplicazioni!

Complessità asintotica

Funzioni utili

Esponenziale

La funzione:

$$a^n = \underbrace{a \cdot a \cdot \dots \cdot a}_{n \text{ volte}}$$

è detta **esponenziale** con base a ed esponente n . Noi useremo principalmente l'esponenziale con base 2 (2^n), la complessità asintotica di "stampare tutti i sottoinsiemi di un insieme di cardinalità N " è pari a $\mathcal{O}(2^N)$.

Una funzione esponenziale cresce molto rapidamente, ad esempio $2^{32} \approx 4'000'000'000$.

Complessità asintotica

Funzioni utili

Forse non tutti siete familiari con 2 funzioni matematiche che spesso useremo nello studio delle complessità: il logaritmo e l'esponenziale.

Logaritmo

Si definisce "**logaritmo** in base a di n " il numero c ($\log_a n = c$) tale che $a^c = n$. Noi useremo in particolare il logaritmo in base 2: $\log_2 n = c \Rightarrow 2^c = n$.

Il logaritmo è una funzione che cresce molto lentamente, ad esempio $\log_2 1'000'000 \approx 20$. Nella notazione O-grande si tende ad omettere la base del logaritmo.

Complessità asintotica

O-grande

Esercizi

- ▶ $\mathcal{O}(N^2 + 70000 \cdot N)$.
- ▶ $\mathcal{O}(77 + 2^N \cdot N^2 + N^3 - 4 \cdot N)$.
- ▶ $\mathcal{O}(1 + 2)$.
- ▶ $\mathcal{O}(\log_2 N + 14 \cdot N + N!! + N!)$.

Complessità asintotica

Applicazioni reali

`std::vector`

- ▶ `std::vector.push_back(elemento)`: $\mathcal{O}(1)$ ammortizzato.
- ▶ `std::vector.pop_back(elemento)`: $\mathcal{O}(1)$ ammortizzato.
- ▶ `std::vector.assign(N, x)`: $\mathcal{O}(N)$.
- ▶ `std::vector.insert(it, x)`: $\mathcal{O}(N)$.
- ▶ `std::vector.erase(it, x)`: $\mathcal{O}(N)$.

Complessità asintotica

Tabella delle complessità


n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
≤ 100	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 400	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Fonte: <https://cpbook.net/>

Array multidimensionali

Matrici

Per creare un array bidimensionale (una matrice) in C++, è sufficiente usare la seguente notazione:



```
int mat[7][20]; /* definisco una matrice 10x10 */  
  
/* con il primo indice accedo a un array (con 20 elementi) */  
/* con il secondo indice accedo agli elementi singoli */  
mat[0][1] = 10;  
mat[6][9] = 42;  
mat[6][19] = 999; /* ultimo elemento */
```

Array multidimensionali

Matrici

Come stampo una matrice?

Itero con due indici i e j sulla matrice e stampo gli elementi.

Qual è la complessità asintotica?

Dati N il numero di righe e M il numero di colonne, la complessità dell'algoritmo è $\mathcal{O}(N \cdot M)$.

Se valesse $N = M$ (nel caso di una matrice quadrata), la complessità si potrebbe riscrivere come $\mathcal{O}(N \cdot N) = \mathcal{O}(N^2)$

Array multidimensionali

Matrici




```
/* inizializzo una matrice con dei valori */  
int mat[3][2] = {{1, 2}, {3, 4}, {5, 6}};  
  
for (int i = 0; i < 3; ++i) {  
    for (int j = 0; j < 2; ++j) {  
        cout << mat[i][j] << " ";  
    }  
    cout << endl; /* vado a capo a fine riga */  
}
```


Array multidimensionali

Più di due dimensioni

In generale, per creare un array multidimensionale in C++, posso concatenare i [] nella dichiarazione:



```
/* dichiaro un array di quattro dimensioni */  
int multidim[11][4][7][100];  
  
multidim[1][0][4][55] = -7;  
multidim[10][3][6][99] = 1000; /* ultimo elemento */
```

std::vector multidimensionali

Posso creare anche `vector` multidimensionali, in generale un *contenitore* permette di inserire tra le parentesi angolari `< >` altri contenitori (!).



```
/* dichiaro un vector di tre dimensioni */  
vector<vector<vector<int>>> multidim;  
  
/*  
 * equivalente di multidim[5][4][3],  
 * tutti i valori inizializzati a 100  
 */  
multidim.assign(5, vector<vector<int>>(4, vector<int>(3, 100)));
```

Complessità generalizzata per array multidimensionali

Come stampo un array con K dimensioni?

Itero con K indici i_0, i_1, \dots, i_{K-1} sull'array e stampo gli elementi.
Qual è la complessità asintotica?

Dati N_0, N_1, \dots, N_{K-1} il numero di elementi per ogni dimensione, la complessità dell'algoritmo è $\mathcal{O}(\prod N_i) = \mathcal{O}(N_0 \cdot N_1 \cdot \dots \cdot N_{K-1})$.
Se valesse $N_0 = N_1 = \dots = N_{K-1}$, la complessità si potrebbe riscrivere come $\mathcal{O}(N^K)$

Complessità asintotica

Ricerca sequenziale

Problema

Supponiamo di avere una lista di elementi e dobbiamo verificare se un dato elemento è presente, come fare?

Complessità asintotica

Ricerca sequenziale

Soluzione

Scorro tutti gli elementi della lista finché trovo l'elemento cercato oppure raggiungo la fine. Per quanto sembri ovvia la soluzione non ne esiste una migliore.

Ma qual è la complessità asintotica? Distinguiamo 3 casi:

- ▶ *caso migliore*: l'elemento si trova in posizione 0. L'algoritmo termina subito e la complessità è costante ($O(1)$).
- ▶ *caso medio*: l'elemento si trova più o meno a metà. L'algoritmo impiega quindi circa $\frac{N}{2}$ operazioni, perciò la soluzione ha complessità $O(\frac{N}{2}) = O(N)$
- ▶ *caso pessimo*: l'elemento si trova alla fine o non è presente, il numero di operazioni richieste è N . La complessità è $O(N)$.

Complessità asintotica

Ricerca sequenziale

```
bool cerca(vector<int> &v, int val) {  
    for (int i = 0; i < v.size(); i++) {  
        if (v[i] == val) {  
            return true;  
        }  
    }  
    return false;  
}  
  
bool cerca(vector<int> &v, int val) {  
    for (int &i: v) {  
        if (i == val) {  
            return true;  
        }  
    }  
    return false;  
}
```

Complessità asintotica

Ricerca sequenziale

```
bool cerca(vector<vector<int>> &v, int val) {  
    for (int i = 0; i < v.size(); i++) {  
        for (int j = 0; j < v[i].size(); j++) {  
            if (v[i][j] == val) {  
                return true;  
            }  
        }  
    }  
    return false;  
}
```

```
bool cerca(vector<vector<int>> &v, int val) {  
    for (int &i: v) {  
        for (int &j: i) {  
            if (j == val) {  
                return true;  
            }  
        }  
    }  
    return false;  
}
```

Complessità asintotica

Ricerca binaria

Problema

Riconsideriamo il problema della ricerca sequenziale, se la lista di valori fosse ordinata, sarebbe possibile trovare un algoritmo più efficiente?

Complessità asintotica

Ricerca binaria

Ricerca binaria

Idea: supponiamo che x sia un valore da cercare in una lista ordinata L e che $L[m]$ sia il valore in posizione centrale. Nel caso in cui $x > L[m]$ possiamo affermare che nessun elemento nella metà sinistra della lista è uguale a x perché la lista è ordinata. Il ragionamento è analogo nel caso $x < L[m]$. A questo punto possiamo ripetere lo stesso ragionamento considerando come lista la metà non scartata e procedere finché il valore centrale non sarà uguale a x oppure si giungerà ad una lista di dimensione 0.

Complessità asintotica

Ricerca binaria

Consideriamo la seguente lista:

$$L = [1, 4, 6, 10, 11, 25, 30, 37, 70]$$

Supponiamo di voler cercare il valore 6, e indichiamo con l, r gli estremi della sottolista considerata durante la ricerca binaria.

- ▶ $l = 0, r = 8$. Calcoliamo $m = \frac{l+r}{2} = 4$, quindi $L[m] = l[4] = 11 > 6$.
- ▶ $l = 0, r = 3$. Calcoliamo $m = \frac{l+r}{2} = 1$, quindi $L[m] = l[1] = 4 < 6$.
- ▶ $l = 2, r = 3$. Calcoliamo $m = \frac{l+r}{2} = 2$, quindi $L[m] = l[2] = 6 == 6$.

Abbiamo quindi trovato che il valore 6 è contenuto nella lista e si trova in posizione 2.

Complessità asintotica

Ricerca binaria

Qual è la complessità della ricerca binaria?

Ad ogni passaggio della ricerca binaria eliminiamo metà degli elementi nello spazio di ricerca, quindi chiamando N la lunghezza della lista, dopo p passaggi la lunghezza del sotto-intervallo considerato sarà $\frac{N}{2^p}$, perciò il numero di passaggi necessario a raggiungere un intervallo di lunghezza di 1 è:

$$\frac{N}{2^p} = 1 \Rightarrow 2^p = N \Rightarrow p = \log_2 N$$

Segue quindi che il numero di passaggi è circa $\log_2 N$, la complessità della ricerca binaria è $O(\log(N))$.

Complessità asintotica

Ricerca binaria

```
bool cerca(vector<int> v, int val) { // supponiamo v ordinato
    int l = 0, r = v.size() - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (v[m] == val) {
            return true;
        } else if (v[m] < val) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    return false;
}
```

Esercizi!

- ▶ Sand Buckets

https://training.olinfo.it/#/task/ois_buckets

- ▶ Flappy Bird

https://training.olinfo.it/#/task/ois_flappybird

- ▶ Kalindrome Strings

https://training.olinfo.it/#/task/ois_kalindrome



Fine

Ci vediamo alla prossima lezione!

- ▶ **E-Mail:** `base@coderfarm.it`
- ▶ **Telegram:** T.B.D.