Algoritmi in C++

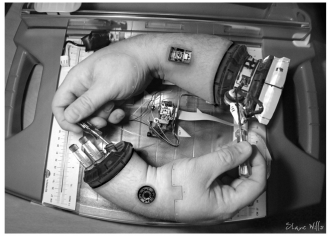
Ricorsione

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia

Algoritmi ricorsivi

Cos'è un "algoritmo ricorsivo" ?

"In informatica viene detto algoritmo **ricorsivo** un algoritmo espresso in termini di se stesso, ovvero in cui l'esecuzione dell'algoritmo su un insieme di dati comporta la **semplificazione** o **suddivisione** dell'insieme di dati e l'applicazione dello **stesso** algoritmo agli insiemi di dati semplificati"[1]



Il concetto fondamentale è di cercare di **RIDURRE** il problema che dobbiamo risolvere ad uno dello stesso tipo con **meno dati** da elaborare

La Ricorsione si basa sul concetto matematico di principio di induzione[2]

$$(\forall P)[P(0) \land (\forall k \in \mathbb{N})(P(k) \Rightarrow P(k+1))] \Rightarrow (\forall n \in \mathbb{N})[P(n)]$$

1) rif: https://it.wikipedia.org/wiki/Algoritmo_ricorsivo
2) rif: https://it.wikipedia.org/wiki/Principio_d%27induzione

Esempi classici di applicazione

- Calcolo della successione di Fibonacci
- Ricerca dicotomica su un insieme ordinato
- Ricerca di sottosequenze simili su stringhe
- Calcolo di percorsi in una mappa
- Soluzione a giochi come dama, scacchi, tetris, tictactoe, etc...

REGOLA AUREA

La **soluzione** dell'algoritmo è di solito la sua stessa **definizione**

corollario

Devi avere fede nella definizione del problema stesso

Successione di Fibonacci

Nell'immagine sottostante la definizione della nota **successione di Fibonacci**.

Tale successione è definita già in maniera ricorsiva, nel senso che la definizione generica di un numero è descritta utilizzando lo stesso concetto applicato a numeri più piccoli

$$F_0 = 0,$$
$$F_1 = 1,$$
$$F_n = F_{n-1} + F_{n-2} \text{ (per ogni n>1)}$$

Implementare una funzione **ricorsiva** che calcoli un numero di questa successione significa solamente seguire tale **definizione** (e tradurla in C++)

C++

```
int F(int n) {
    if (n==0) return 0;
    if (n==1) return 1;
    return F(n-1) + F(n-2);
}
```

Ricerca di un valore in un array

Assumiamo di voler cercare un valore in un array di numeri.

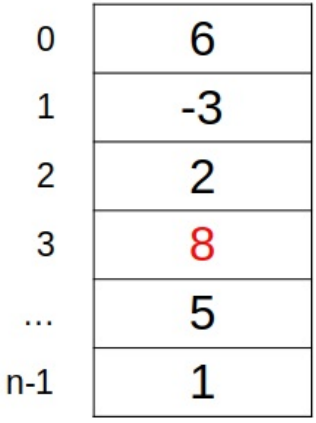
E di voler utilizzare una procedura ricorsiva per questo scopo

(n.b. soluzione assolutamente sconsigliata perché onerosa e inutilmente complessa)

L'idea di base è che posso avere pochi casi:

- Se l'array ha 0 elementi il numero che cerco NON c'è
- Altrimenti se il numero N che cerco è il primo l'ho trovato
- Altimenti se c'è è negli elementi successivi

5 minuti per provare a risolvere il problema...



Ricerca di un valore in un array - Soluzione

Abbiamo i seguenti casi:

- Se l'array ha 0 elementi il numero che cerco NON c'è
- Altrimenti se il numero N che cerco è il primo l'ho trovato
- Altimenti se c'è è negli elementi successivi

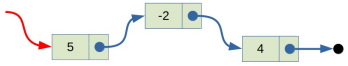
Posso implementare esattamente questa definizione:

```
bool cerca(int V[], int n, int inizio, int fine) {
    if (inizio > fine) return false; // l'array è vuoto
    if (V[inizio] == n) return true; // elemento trovato in posizione "inizio"
    return cerca(V, n, inizio+1, fine); // cercalo da qui in poi
}
```

0	6
1	-3
2	2
3	8
…	5
n-1	1

Ricerca di un valore in una lista

Assumiamo di voler cercare un valore in una lista di numeri interi e di voler utilizzare una procedura ricorsiva per questo scopo



(n.b. anche qui soluzione sconsigliata perché è sufficiente una scansione lineare della struttura)

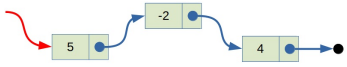
L'idea di base è che posso avere pochi casi:

- Se la lista è vuota il numero che cerco NON c'è
- Altrimenti se il numero N che cerco è il primo l'ho trovato
- Altimenti, se c'è, è negli elementi successivi

10 minuti per provare a risolvere il problema...

Ricerca di un valore in una lista - Soluzione

L'idea di base è che posso avere pochi casi:



- Se la lista è vuota il numero che cerco NON c'è
- Altrimenti se il numero N che cerco è il primo l'ho trovato
- Altimenti, se c'è, è negli elementi successivi

Posso implementare esattamente questa definizione:

C++

```
bool cercaLista(nodo* p, int n) {   // ricevo il puntatore al nodo dal quale iniziare
    if (p == nullptr) return false;     // lista vuota, niente da trovare
    if (p->info == n) return true;  // elemento trovato nel nodo puntato da p
    return cercaLista(p->next, n);  // cercalo a partire dal nodo successivo (puntatore successivo)
}
```

oppure, che è anche più bello...

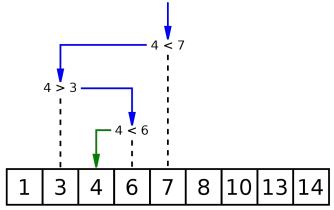
C++

```
bool cercaLista(nodo* p, int n) {
    if (p == nullptr) return false;
    else return ((p->info == n) || (cercaLista(p->next, n)));
}
```

Ricerca dicotomica in un array ordinato

Assumiamo di voler cercare un valore in un array **ordinato** di numeri interi e di voler utilizzare una procedura ricorsiva per questo scopo

(n.b. qui la soluzione è fortemente consigliata perché, in caso di numerosità molto alta, abbatterà i tempi di ricerca in maniera esponenziale



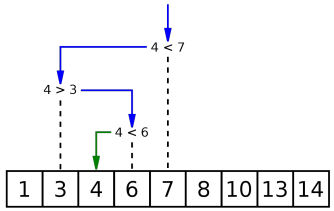
L'idea di base è che posso avere pochi casi:

- Se l'intervallo di ricerca è nullo il numero non c'è
- Altrimenti se il numero N che cerco è nell'indice che sto considerando, allora l'ho trovato
- Altimenti, se è maggiore di quello che sto guardando, lo cerco nell'intervallo superiore
- Altimenti, se è maggiore di quello che sto guardando, lo cerco nell'intervallo inferiore

Ricerca dicotomica in un array ordinato - Soluzione

L'idea di base è che posso avere pochi casi:

- Se l'intervallo di ricerca è nullo il numero non c'è
- Altrimenti se il numero N che cerco è nell'indice che sto considerando, allora l'ho trovato
- Altimenti, se è maggiore di quello che sto guardando, lo cerco nell'intervallo superiore
- Altimenti, se è maggiore di quello che sto guardando, lo cerco nell'intervallo inferiore



Posso implementare esattamente questa definizione:

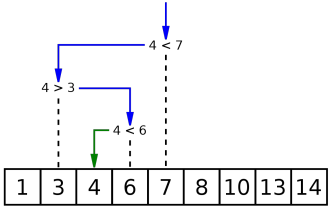
C++

```
bool binarysearch(int V[], int n, int inizio, int fine) {
    if (inizio > fine) return false; // l'intervallo specificato non è più valido
    // cerco elemento medio nell'intervallo specificato da inizio e fine
    int medio = (fine+inizio)/2;
    if (V[medio] == n) return true; // elemento trovato in posizione "medio"
    if (V[medio] > n) return binarysearch(V, n, inizio, medio-1); // lo cerco da inizio a medio-1
    if (V[medio] < n) return binarysearch(V, n, medio+1, fine); // lo cerco da medio+1 a fine
    return false;   // risultato di default
}
```

binary search vs linear search

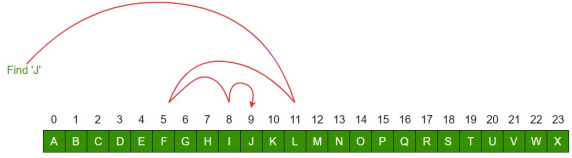
A binary search however, cut down your search to half as soon as you find middle of a sorted list.

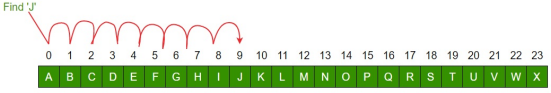
The middle element is looked to check if it is greater than or less than the value to be searched. Accordingly, search is done to either half of the given list



Important Differences

Input data needs to be sorted in Binary Search and not in Linear Search Linear search does the sequential access whereas Binary search access data randomly.
Time complexity of **linear** search is **O(n)** : **binary** search has time complexity **O(log n)**
Linear search performs equality comparisons and Binary search performs ordering comparisons





Rif: https://www.geeksforgeeks.org/linear-search-vs-binary-search
Standard search finita in 31.3967 millisecondi vs Binary search finita in 0.006964 millisecondi

esercizi...

risolvere i seguenti problemi con funzioni ricorsive

- Visualizza gli N elementi di un vettore
- Acquisire una sequenza di elementi finché l'utente digita '*'e inserirli in una lista
- Visualizzare una lista di elementi avente puntatore iniziale P0
  - Dal primo all'ultimo
  - In ordine inverso
- Calcola la somma degli elementi di un lista avente puntatore iniziale P0
- Calcola la somma degli elementi negativi di una lista avente puntatore iniziale P0
- Calcola il prodotto degli elementi pari di una lista avente puntatore iniziale P0
- Restituisce la posizione che un elemento (indicato da utente) occupa all'interno di una lista avente puntatore iniziale P0
- Dato il puntatore iniziale di una lista caricata di numeri reali, dire quanti di questi sono positivi
- Siano date due stringhe A e B di lunghezza N. Si scriva una funzione ricorsiva "subseq" che riceve in ingresso le due stringhe e restituisce la sottosequenza comune più lunga
  - es: A = 'AG**GTAB**' B = '**G**X**T**X**A**Y**B**'
  - ris: 'GTAB'

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia