Introduzione all'Algoritmica per i Licei (C++).

3 – Ricerca binaria o dicotomica.

versione 1 febbraio 2015

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

Valutazione dell'andamento del tempo di esecuzione. Un esempio: cercare un elemento in un elenco.

Se l'elenco o sequenza di elementi in cui cercare non è ordinato (secondo il criterio di ricerca), si è costretti ad eseguire una ricerca sequenziale: a partire dal primo elemento della sequenza si esaminano uno dopo l'altro tutti gli elementi, finché o si trova l'elemento cercato o si arriva al termine della sequenza senza aver trovato ciò che si cerca.

Esempi:

- cercare una parola in una lunga lista non ordinata di parole;
- cercare in un lungo scontrino di ipermercato se si è acquistato un certo prodotto;
- cercare in un elenco telefonico a chi corrisponda un dato numero (nell'elenco gli elementi sono ordinati per nome, non per numero).

Ricerca sequenziale: tempo di calcolo.

Quanti passi si fanno in media, con la ricerca sequenziale, per trovare l'elemento in una sequenza di n elementi ?

Se si è molto fortunati lo si trova subito;

se si è sfortunati lo si trova in ultima posizione oppure non lo si trova, e in entrambi i casi si devono fare n passi.

In media si faranno circa n/2 passi, cioè un numero di passi proporzionale a n o, più precisamente, lineare in n.

Quindi anche il tempo medio necessario per effettuare la ricerca è proporzionale a n.

Se la lunghezza della lista raddoppia, il tempo raddoppia; se la lunghezza triplica, il tempo triplica, e così via.

Ciò vale tanto per gli esseri umani che per i computer.

Esempio

Sia `temperature` un array di numeri, ad esempio un array di temperature giornaliere in un mese; l'indice rappresenta il giorno del mese, ad es.:

`temperature[0]` contiene la temperatura del 1 gennaio;
`temperature[1]` contiene la temperatura del 2 gennaio;

… ecc, fino a:

`temperature[30]` contiene la temperatura del 31 gennaio.

Vogliamo sapere se c'è stato un giorno in cui la temperatura è stata esattamente 18 gradi, e che giorno è stato.

Oppure abbiamo un array che rappresenta una lista di nomi, e voglio sapere se un certo nome compare nella lista, e in che posizione.

Implementazione

Scriviamo una funzione che prende come argomenti un array di numeri, la sua lunghezza, e il numero da cercare, e dà come risultato l'indice dell'array a cui si trova quel numero.

Se il numero cercato non è presente nell'array, per segnalarlo la funzione dà come risultato un "indice impossibile", come -1.

```
int ricseq(float x, float a[], int n) {
  for(int i = 0; i < n; i++) {
    if(x == a[i]) return i;
  }
  return -1;
}
```


Versione con risultato booleano

Scriviamo una funzione che prende come argomenti un array di numeri, la sua lunghezza, e il numero da cercare, e dà come risultato true se il numero cercato è presente nell'array, false se non è presente.

```
bool ricseq(float x, float a[], int n) {
  for(int i = 0; i < n; i++) {
    if(x == a[i]) return true;
  }
  return false;
}
```

Elenco ordinato: ricerca binaria o dicotomica.

Se la sequenza di parole è ordinata alfabeticamente e il tempo di accesso a un suo elemento è indipendente dalla posizione dell'elemento nella sequenza stessa (ed è piccolo), allora si può usare la ricerca binaria. Ad esempio: un umano può aprire un volume circa alla pagina voluta senza dover sfogliare una per una tutte le pagine precedenti.

Lo stesso vale se si deve cercare un numero in una sequenza di numeri ordinata in modo crescente.

Precondizione

Precondizione necessaria per l'utilizzabilità della ricerca binaria è quindi che la sequenza in cui cercare sia:

- ordinata;
- ad *accesso diretto*
- Confronta: accesso (quasi) diretto in un lettore di CD
  rispetto a
  accesso sequenziale di un vecchio lettore di nastri.

Descrizione a parole, imprecisa, dell'algoritmo.

Precondizione: la sequenza (ad accesso diretto) è ordinata.

- Si accede all'elemento centrale della sequenza e lo si confronta con l'elemento da cercare;
- a seconda del risultato del confronto ci si restringe a considerare solo la prima o la seconda metà;
- si confronta l'elemento da cercare con l'elemento centrale della porzione-metà, e di conseguenza ci si restringe a una porzione di lunghezza dimezzata;
- ... e così via ...

Tempo necessario per eseguire la procedura

- Quanti passi occorrono per cercare un elemento per mezzo della ricerca binaria in un elenco di n elementi ?
- Osserva che ad ogni passo la porzione di elenco in cui cercare si riduce all'incirca alla metà.
- Quindi, se ad es. è $n = 2^{10} = 1024$, al secondo passo la porzione in cui cercare è di circa 512, al terzo di circa 256, e così via. Nel caso peggiore dopo circa 10 passi si trova l'elemento, o si stabilisce che non è presente.
- Che cosa è 10 rispetto a 1024? è il logaritmo in base 2 !

$$2^{10} = 1024 \quad \rightarrow \quad 10 = \log_2 1024$$

Dietro alla grande velocità della ricerca binaria rispetto alla ric. sequenziale, riscontrabile anche dagli esseri umani nelle ricerche manuali o visive, c'è il diverso tasso di crescita delle funzioni-logaritmo rispetto alle funzioni lineari !

Realizzazione di un programma per la ricerca binaria

Come scrivere un ciclo partendo dal passo intermedio

Possibili esempi:

- ricerca binaria in array ordinato;
- problema della bandiera tricolore;
- altri algoritmi iterativi elementari;

Ricerca binaria: traduzione in programma

- Supponiamo che si tratti di una sequenza di stringhe ordinata alfabeticamente.
- Stabiliamo che la funzione di ricerca dia come risultato la posizione, cioè il valore dell'indice, dell'elemento cercato.
- Se nell'elenco non c'è nessuna stringa uguale a quella cercata, la funzione deve segnalarlo dando come risultato un valore impossibile di indice, come -1.

Ricerca binaria: traduzione in programma.

Bisogna *iterare* (cioè ripetere) più volte una certa sequenza di istruzioni, fino a trovare la risposta. Ciò si può ottenere per mezzo di un'istruzione di *ciclo*. Come si scrive un ciclo?

- Nota Bene: se si cerca di scrivere la procedura basandosi direttamente sulla precedente descrizione informale dell'algoritmo, si ottiene una procedura errata e/o brutta !
- Per scrivere un ciclo non banale è indispensabile pensare subito al passo generico, NON al passo iniziale.
- Cioè bisogna pensare non agli estremi (inizio o fine del ciclo), bensì ad un generico punto intermedio.

Situazione al passo generico.

Dalla descrizione a parole ricordata prima, si ricava che:

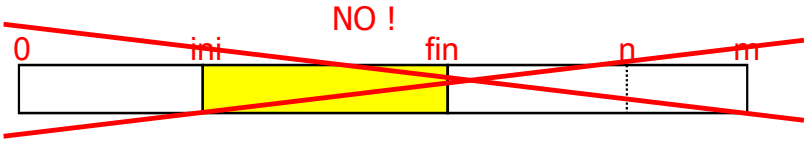
ad un generico passo intermedio si ha una porzione di array alla quale si è ristretta la ricerca.

Tale porzione non necessariamente si trova all'inizio o alla fine dell'array; in generale si troverà all'interno dell'array:

Per individuarla sono quindi necessari due indici.

Attenzione !

Quando, nella rappresentazione grafica di un array, si indica la posizione di un indice, tale indice deve essere scritto NON in corrispondenza della linea verticale di separazione fra due parti dell'array o della linea iniziale o della linea finale, bensì in corrispondenza di una cella dell'array (che poi, se è quella iniziale o finale di una porzione, può anche non essere esplicitamente disegnata).



Inoltre ...

Quando si deve indicare la prima o l'ultima cella di una porzione significativa di array, è meglio tracciare una linea tratteggiata invece di una linea continua, oppure non tracciarla affatto (a seconda di cosa si vuol evidenziare).

Invece di:

meglio:

Una abbreviazione utile.

Nella descrizione e giustificazione degli algoritmi avremo spesso bisogno di parlare di una porzione di array o di vettore compresa fra due indici, ad esempio: tutti gli elementi dell'array a da a[i] ad a[j] estremi inclusi, cioè la sequenza di elementi:

a[i], a[i+1], a[i+2], ..., a[j-1], a[j]

Per indicare tale porzione utilizzeremo la notazione abbreviata

a[i .. j]

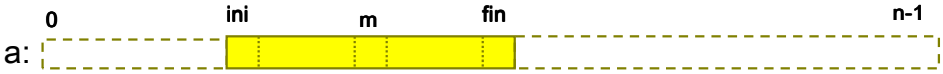
Naturalmente

- a[i .. i] coincide con a[i];
- se i > j la scrittura a[i .. j] indica la sequenza vuota.

Ma attenzione: questa non è una notazione del linguaggio C++ !
Non possiamo utilizzarla in un programma C++ !
Ci serve solo per "parlare di" un programma C++.

Scriviamo il corpo del ciclo

SITUAZIONE AL PASSO GENERICO



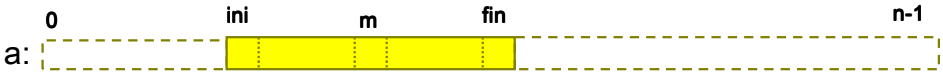
L'elemento da cercare, se è presente nell'array, si trova nella porzione di array compresa fra gl'indici ini e fin (inclusi).

Che cosa devo fare nel corpo del ciclo ?

Per prima cosa, è necessario calcolare il valore dell'indice m dell'elemento centrale (o mediano) della porzione di array.
Come si fa ?

Scriviamo il corpo del ciclo

SITUAZIONE AL PASSO GENERICO



Attenzione:

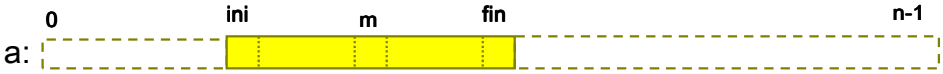
Se si pensa alla situazione iniziale, in cui la parte "gialla" è tutto l'array, l'indice m dell' el. centrale è semplicemente n/2.

Ma NON bisogna pensare alla situazione iniziale!
Bisogna mettersi mentalmente al passo generico!
Che cosa è m in tal caso? (fin – ini)/2 ?
No ! Supponiamo ini = 20 e fin = 30: l'indice m non è 5 !
È la media aritmetica: (20 + 30)/2 = 25.
La prima istruzione del corpo del ciclo è quindi:

```
m = (ini + fin)/2
```
(divisione intera)

Scriviamo il corpo del ciclo

SITUAZIONE AL PASSO GENERICO



Dopo aver calcolato l'indice m dell'elemento centrale, confronto la chiave da cercare con quella di elemento a[m]; sono possibili tre casi:

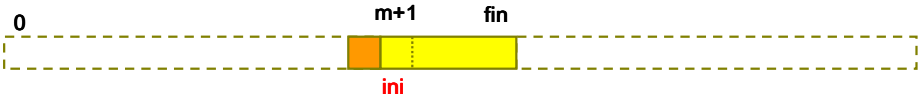
- x < a[m]: x, se c'è, si trova in a[ini .. m-1], cioè nella porzione compresa fra ini e m-1 (inclusi);
- x > a[m]: x, se c'è, si trova in a[m+1 .. fin], cioè nella porzione compresa fra m+1 e fin (inclusi);
- x == a[m]: x c'è, e si trova nella posizione m.

Casi 1 e 2

- `x < a[m]:` l'elemento, se c'è, si trova nella porzione compresa fra ini e m-1 (inclusi): quindi


```
if(x < a[m]) fin = m-1;
```

- `x > a[m]:` l'elemento, se c'è, si trova nella porzione compresa fra m+1 e fin (inclusi): quindi




```
else if(x > a[m]) ini = m+1;
```


Terzo caso

- `x == a[m]` ho trovato il valore cercato, quindi termino la funzione dando come risultato l'indice dell'elemento cercato:

```
else return m;
```

Il corpo del ciclo, cioè la sequenza di istruzioni che si deve ripetere, è quindi:

```
int m = (ini + fin)/2; // divisione intera
if(x < a[m]) fin = m-1;
else if(x > a[m]) ini = m+1;
else return m;
```


Qual è la condizione per cui il ciclo deve continuare?

La porzione di array su cui fare la ricerca non deve essere vuota; cioè

ini **<=** fin

NOTA: se ini = fin la porzione non è vuota, ma contiene un elemento.

Quali sono i valori iniziali di ini e fin ?

Inizialmente la porzione di array su cui effettuare la ricerca è l'intero array.

ini **=** 0 fin **=** indice dell'ultimo elemento;

Che cosa succede se il ciclo termina normalmente, cioè senza interrompersi per dare il risultato ?

Vuol dire che l'elemento cercato non c'è: quindi diamo come risultato -1.

Ricerca binaria (in C++)

```
int ricbin(string x, string a[], int n) {
  int ini = 0, fin = n-1;
  while(ini <= fin) {
    int m = (ini + fin)/2;
    if(x < a[m]) fin = m-1;
    else if(x > a[m]) ini = m+1;
    else return m;
  }
  return -1;
}
```

uscita

Anche qui sarebbe conveniente qualificare come `const` tutti e tre i parametri.

Per i "puristi" ..., una versione con flag booleana.

```
int ricbin(string x, string a[], int n) {
  int ini = 0, fin = n-1, m;
  bool trovato = false;
  while(ini <= fin && !trovato) {
    m = (ini + fin)/2;
    if(x < a[m]) fin = m-1;
    else if(x > a[m]) ini = m+1;
    else trovato = true;
  }
  if(trovato) return m;
  else return -1;
}
```

Se l'elemento cercato non c'è, il ciclo termina sempre ?

- Osserva: a ogni ripetizione del ciclo l'intervallo di ricerca si restringe almeno di un elemento, perché viene escluso l'elemento centrale.
- Quindi, se l'elemento cercato non viene trovato, prima o poi i due indici ini e fin si incrociano, e il ciclo termina.

Ricerca binaria con vector

Il numero di elementi è un attributo del vettore e quindi non deve essere passato come argomento separato.

```
int ricbin(string x, const vector<string> & v){
  int n = v.size();
  int ini = 0, fin = n-1;
  while(ini <= fin) {
  int m = (ini + fin)/2;
    if(x < v[m]) fin = m-1;
    else if(x > v[m]) ini = m+1;
    else return m;
  }
  return -1;
}
```

Nota

Se l'intestazione della funzione di ricerca binaria su un vector viene scritta semplicemente come:

```
int ricbin(string x, vector<string> v)
```

la funzione dà ancora sempre il risultato corretto.

Tuttavia, poiché in C++ i vector, se non si indica esplicitamente con & il passaggio per riferimento, sono passati per valore, la funzione si crea una propria copia locale del vettore passatole, cioè si copia tutti gli elementi: ma per far ciò ci mette un tempo proporzionale alla lunghezza del vettore, rendendo inutile la "maggiore velocità" della ricerca binaria rispetto alla ricerca sequenziale.

In realtà nelle ultime implementazioni di C++ la copia locale non viene effettuata se non è necessario; tuttavia è sempre buona norma che i vectors vengano passati per riferimento.

Esercizio

Definisci un template di funzione di ricerca binaria, la quale, preso un array di elementi di tipo generico (in cui la chiave di ricerca è l'elemento stesso), dia come risultato l'indice a cui si trova l'elemento nell'array, oppure -1 se l'elemento non è presente.

Oppure, in alternativa:

Definisci un template di funzione di ricerca binaria, la quale, preso un vector di elementi di tipo generico (in cui la chiave di ricerca è l'elemento stesso), dia come risultato l'indice a cui si trova l'elemento nel vector, oppure -1 se l'elemento non è presente.

Problema: una possibile ottimizzazione.

- L'algoritmo di ricerca binaria illustrato nelle slides precedenti, nei casi in cui l'elemento cercato non sia presente, compie sempre circa log n passi.
- C'è tuttavia un caso (anzi due) in cui "ci si potrebbe accorgere subito" che l'elemento non è presente.
- Quali sono tali casi?
- Modifica la funzione ricbin in modo da renderla più efficiente nei suddetti casi.

Test per stabilire se un array o un vettore è ordinato

Un array `a` di n elementi, indiciato da 0 a n-1, è ordinato se:

```
a[0] <= a[1] <= a[2] … <= a[n-1]
```

Scriviamo una funzione che, preso come argomento un array e la sua lunghezza, restituisce true se l'array è ordinato, false se non lo è.

```
template<typename E> bool isSorted(E a[], int n)
```

Scriviamo una funzione che, presi come argomenti un array e la sua lunghezza, restituisce true se l'array è ordinato, false se non lo è.

Evidentemente bisogna eseguire il confronto fra `a[i]` e `a[i+1]` per tutti gli `i` e `i+1` possibili, oppure, equivalentemente, il confronto `a[i-1]` e `a[i]` per tutti gli `i-1` e `i` possibili.

Attenzione ai valori iniziale e finale dell'indice `i` !

Una soluzione

```
template<typename E> bool isSorted(E a[], int n) {
  for(int i = 1; i < n; i++) {
    if(a[i] < a[i-1]) return false;
  }
  return true;
}
```

oppure

```
template<typename E> bool isSorted(E a[], int n) {
  for(int i = 0; i < n-1; i++) {
    if(a[i+1] < a[i]) return false;
  }
  return true;
}
```


Esercizio

Scrivi una funzione che, dato un array di lunghezza ≥ 2, trovi gl'indici di due elementi (di valori non necessariamente distinti, ma di indici diversi) che siano i due valori più grandi che compaiono nell'array, e li scriva sullo schermo; ad es.:

```
void iDueMaggiori(string a[], int n)
```

Attenzione: per scrivere la procedura corretta pensa a qual è la situazione al passo generico, e a cosa devi fare in esso!

Esercizio

Scrivi una funzione:

```
void partizione(double x, double a[], int n)
```

la quale, presi come argomenti

un numero x, un array di numeri, e la lunghezza dell'array,

modifica l'array

spostando tutti i numeri < x a sinistra di tutti i numeri >= x.

Ragiona considerando la situazione al passo generico, ad es.: