anno 2014-15

Introduzione all'Algoritmica per i Licei

6 – Ricerca binaria in un array (lista) ordinato.

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

versione 23 febbraio 2015

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.

Realizzazione in Python: esempio.

Scriviamo una funzione che prende come argomenti l'elemento da cercare e la lista in cui cercarlo, e dà come risultato la posizione (cioè l'indice) nella lista a cui si trova quell'elemento.

Se l'elemento cercato non è presente nell'array, per segnalarlo la funzione dà come risultato il valore speciale None, che in inglese vuol dire "nessuno".

```
def ricercaSeq(x, lis):
  for i in range(0, len(lis)):
    if x == lis[i]: return i
  return None
```

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.

Nota. 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 del programma

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.

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:

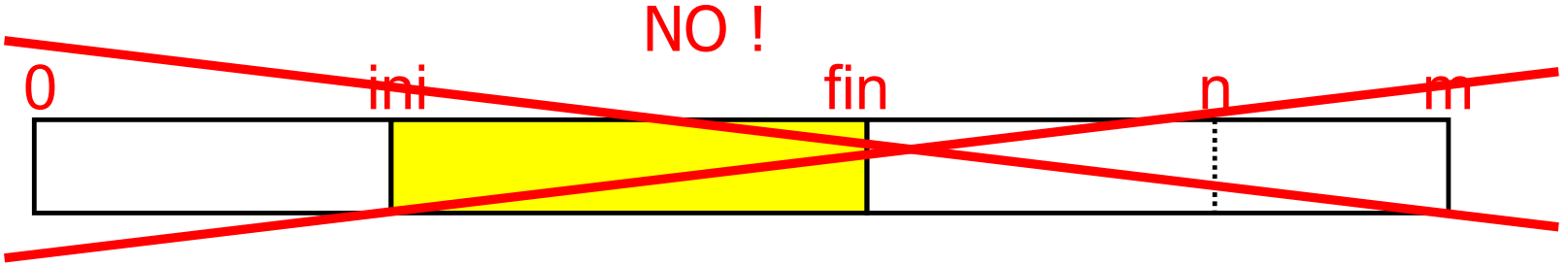
ad un generico passo intermedio si ha una porzione di array alla quale si è ristretta la ricerca.

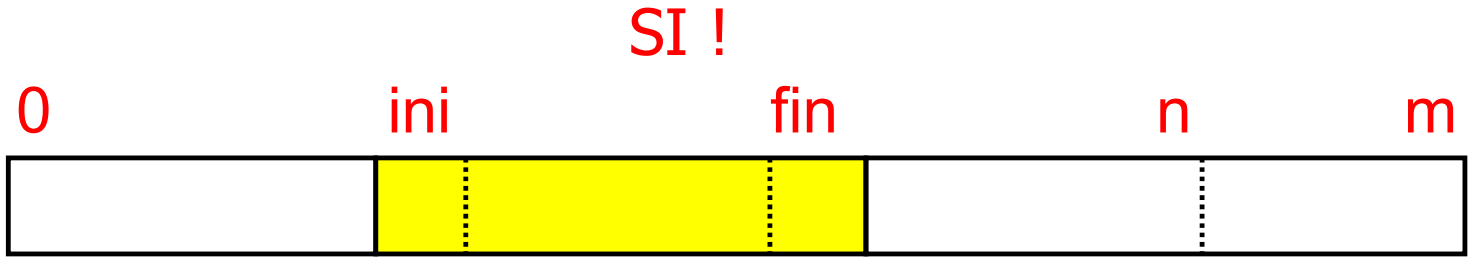
Tale porzione non necessariamente si trova all'inizio o alla fine dell'array; in generale si troverà all'interno dell'array:

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).

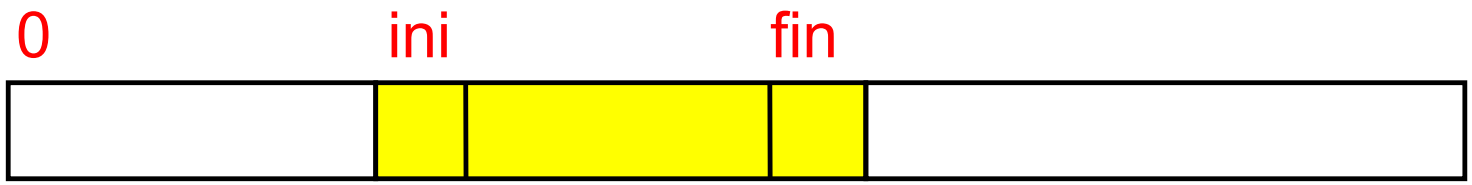




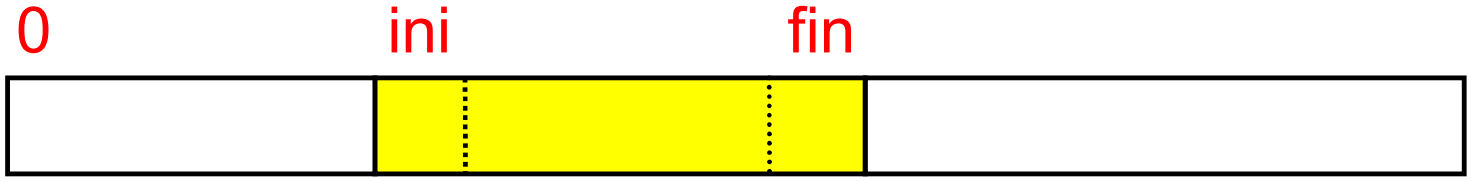
Inoltre ...

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:

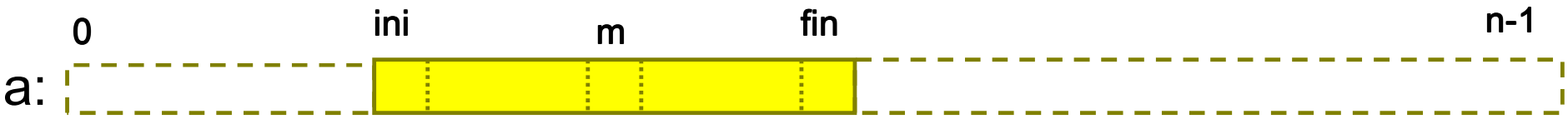


meglio:



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

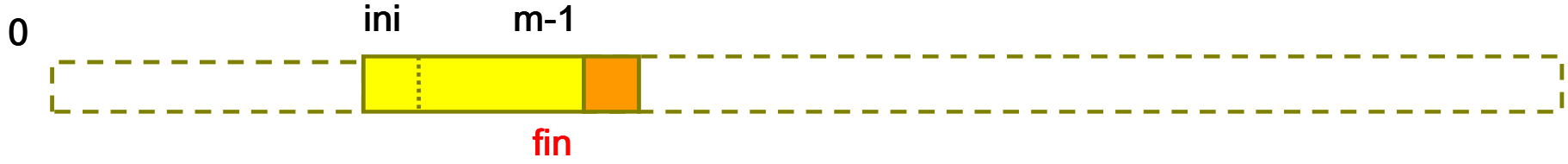
SITUAZIONE AL PASSO GENERICO

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.

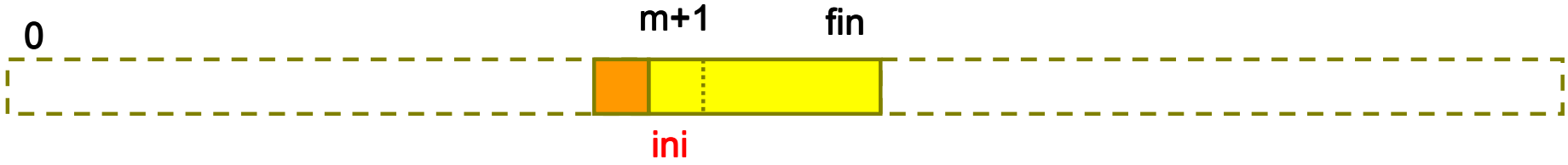
Casi 1 e 2

- **`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




```
elif x > a[m]: ini = m+1
```

Terzo caso

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

```
else: return m
```

Il corpo del ciclo è quindi:

```
m = (ini + fin)//2
if   x < a[m]: fin = m-1
elif 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 inf e sup ?

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

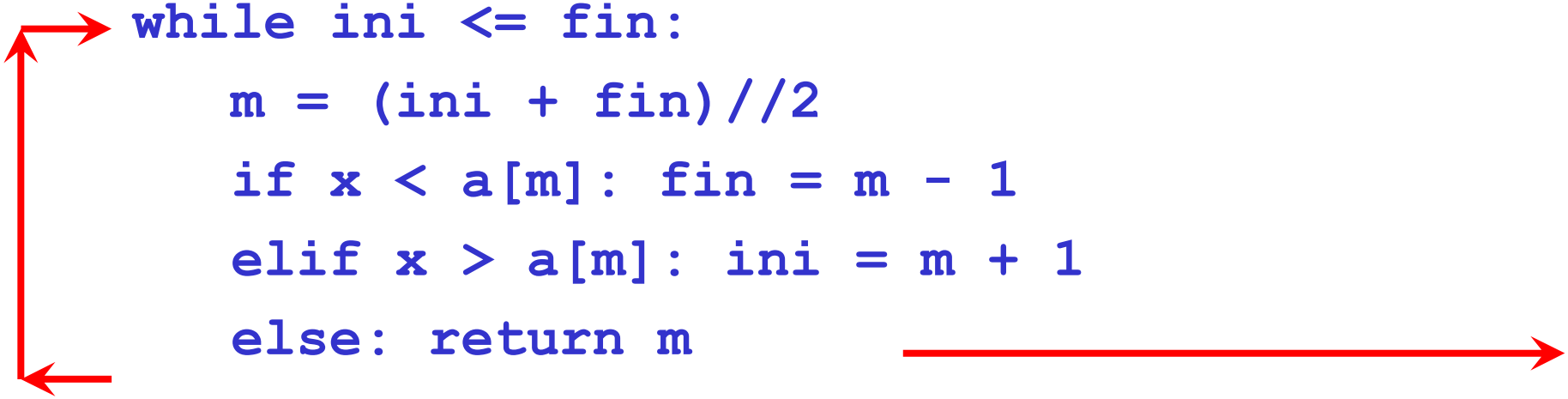
ini = 0 fin = indice dell'ultimo elemento;

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 si deve dare come risultato None (nessuno).

Ricerca binaria (in Python)

```
def ricercaBin(x, a):
    n = len(a)
    ini = 0
    fin = n-1
    while ini <= fin:
        m = (ini + fin)//2
        if x < a[m]: fin = m - 1
        elif x > a[m]: ini = m + 1
        else: return m
    return None
```



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 inf e sup si incrociano, e il ciclo termina.

Una versione più realistica, con gli "oggetti".

Si ha una lista di studenti ordinata per numero di matricola, dove ogni elemento della lista riporta le informazioni relative allo studente individuato da quel numero di matricola.

Vogliamo definire delle funzioni che, dato un numero di matr., diano come risultato non l'indice, bensì l'elemento-studente avente quella matricola.

Realizzazione in Python: esempio.

```
class Studente:
  def __init__(self, matr, nome, cognome):
    self.matr = matr
    self.nome = nome
    self.cognome = cognome

def ricercaSeq(x, lis):
  for stud in lis:
    if stud.matr == x: return stud
  return None
```

oppure

```
def ricercaSeq(x, lis):
  for i in range(0, len(lis)):
    if x == lis[i].matr: return lis[i]
  return None
```

Prova

```
listaStudenti = [Studente(2531, 'Ada', 'Rossi'),
Studente(4127, 'Edo', 'Bruni'), Studente(1384,
'Zoe', 'Bianchi'), Studente(3422, "Aldo", "Verde")]

ris = ricercaSequen(1384, listaStudenti)

if ris == None: print("non trovato")
else:
  print(ris.nome, ris.cognome)
```

La funzione di ricerca binaria per matricola

```
def ricercaBin(x, a):
    n = len(a)
    ini = 0
    fin = n-1
    while ini <= fin:
        med = (ini + fin)//2
        if x < a[med].matr: fin = med - 1
        elif x > a[med].matr: ini = med + 1
        else: return a[med]
    return None
```

Osserva: viene dato come risultato proprio l'elemento, non il suo indice!