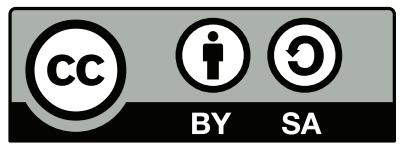
Algoritmi e Strutture Dati

Algoritmi probabilistici

Alberto Montresor

Università di Trento

2023/06/23



Sommario

Introduzione

"Audentes furtuna iuvat" - Virgilio

- Se non sapete quale strada prendere, fate una scelta casuale

Casualità negli algoritmi visti finora

- Analisi del caso medio
  - Si calcola la media su tutti i possibili dati di ingresso in base ad una distribuzione di probabilità
  - Esempio: caso medio Quicksort, si assume che tutte le permutazioni siano equiprobabili
- Hashing
  - Le funzioni hash equivalgono a "randomizzare" le chiavi
  - Distribuzione uniforme

Introduzione

Algoritmi probabilistici

Il calcolo delle probabilità è applicato non ai dati di input, ma ai dati di output

- Algoritmi la cui correttezza è probabilistica (Montecarlo)
- Algoritmi corretti, il cui tempo di funzionamento è probabilistico (Las Vegas)

Test di primalità (Primality test)

Test di primalità

Algoritmo per determinare se un numero in input n è primo.

Fattorizzazione

Algoritmo per listare i fattori che compongono un numero composto.

boolean isPrimeNaif(**int** n)

```
for i = 2 to ⌊√n⌋ do
    if n/i == ⌊n/i⌋ then
        return false
return true
```

Una soluzione inefficiente testa tutti gli interi fra 2 e $\lfloor\sqrt{n}\rfloor$

Test di primalità – Fermat

Piccolo teorema di Fermat

Se n è primo, allora:

$$\forall b, 2 \leq b < n : \qquad b^{n-1} \bmod n = 1$$

Test di primalità di Fermat

boolean isPrime1(**int** n)

$b = \mathsf{random}(2, n-1)$
if $b^{n-1} \bmod n \neq 1$ **then**
  return false
return true

Questo algoritmo è corretto?

Esistono numeri composti (pseudo-primi in base b) tali che:

$$\exists b, 2 \leq b < n-1 : b^{n-1} \bmod n = 1$$

Esempio :

$$n = 341 = 11 \times 13$$

$$2^{340} \bmod 341 = 1$$

Test di primalità – Fermat

Piccolo teorema di Fermat

Se n è primo, allora:

$$\forall b, 2 \leq b < n: \qquad b^{n-1} \bmod n = 1$$

Test di primalità di Fermat

Questo algoritmo è corretto?

boolean isPrime2(**int** n)

for $i = 1$ **to** k **do**
  $b = \mathsf{random}(2, n-1)$
  if $b^{n-1} \bmod n \neq 1$ **then**
    return false
return true

Output	n
false	sicuramente composto
true	possibilmente primo

Test di primalità – Fermat

Piccolo teorema di Fermat

Se n è primo, allora:

$$\forall b, 2 \leq b < n : \qquad b^{n-1} \bmod n = 1$$

Test di primalità – Miller-Rabin

Teorema

Se n è primo , allora per ogni intero b, con $2 \leq b < n$ valgono entrambe le seguenti condizioni:

1. $\text{mcd}(n, b) = 1$
2. $b^m \bmod n = 1 \vee \exists i, 0 \leq i < v : b^{m \cdot 2^i} \bmod n = n - 1$

con $n - 1 = m \cdot 2^v$, m dispari

Contrapposizione

Se esiste un intero b, con $2 \leq b < n$ tale che almeno una delle seguenti condizioni è vera:

1. $\text{mcd}(n, b) \neq 1$
2. $b^m \bmod n \neq 1 \wedge \forall i, 0 \leq i < v : b^{m \cdot 2^i} \bmod n \neq n - 1$

allora n è composto

Test di primalità – Miller-Rabin

Verifica primalità n – Esempio: $n = 221 = 13 \times 17$

Sia n un numero dispari	$n = 221$
Sia $n - 1 = m \cdot 2^v$, con m dispari	$220 = 55 \cdot 2^2$
$n-1$ in binario è pari a: m in binario seguito da v zeri	$\overbrace{110111}^{m=55}\underbrace{00}_{v=2}$
se $\exists b, 2 \leq b < n$ per cui una delle seguenti affermazioni è vera, n è composto:	$b = 137$ (Witness)
(1) $\mathrm{mcd}(n, b) \neq 1$	$\mathrm{mcd}(221, 137) = 1$
(2) $b^m \bmod n \neq 1$ **and**	$137^{55} \bmod 221 = 188 \neq 1$
$\forall i, 0 \leq i < v : b^{m \cdot 2^i} \bmod n \neq n - 1$	$137^{55 \cdot 2^0} \bmod 221 = 188 \neq 220$
	$137^{55 \cdot 2^1} \bmod 221 = 205 \neq 220$

Test di primalità – Miller-Rabin

Verifica primalità n – Esempio: $n = 229$ numero primo

Sia n un numero dispari	$n = 229$
Sia $n - 1 = m \cdot 2^v$, con m dispari	$228 = 57 \cdot 2^2$
$n-1$ in binario è pari a: m in binario seguito da v zeri	$\overbrace{111001}^{m=57}\underbrace{00}_{v=2}$
se $\exists b, 2 \leq b < n$ per cui una delle seguenti affermazioni è vera, n è composto:	Proviamo $b = 137$
(1) $\mathrm{mcd}(n, b) \neq 1$	$\mathrm{mcd}(229, 137) = 1$
(2) $b^m \bmod n \neq 1$ **and**	$137^{57} \bmod 229 = 188 \neq 1$
$\forall i, 0 \leq i < v : b^{m \cdot 2^i} \bmod n \neq n - 1$	$137^{57 \cdot 2^0} \bmod 229 = 122 \neq 228$
	$137^{57 \cdot 2^1} \bmod 229 = 228$

Test di primalità – Miller-Rabin

Verifica primalità n – Esempio: $n = 221 = 13 \times 17$

Sia n un numero dispari	$n = 221$
Sia $n - 1 = m \cdot 2^v$, con m dispari	$220 = 55 \cdot 2^2$
$n-1$ in binario è pari a: m in binario seguito da v zeri	$\overbrace{110111}^{m=55}\underbrace{00}_{v=2}$
se $\exists b, 2 \leq b < n$ per cui una delle seguenti affermazioni è vera, n è composto:	$b = 174$ (Strong liar)
(1) $\mathrm{mcd}(n, b) \neq 1$	$\mathrm{mcd}(221, 174) = 1$
(2) $b^m \bmod n \neq 1$ **and**	$174^{55} \bmod 221 = 47 \neq 1$
$\forall i, 0 \leq i < v : b^{m \cdot 2^i} \bmod n \neq n - 1$	$174^{55 \cdot 2^0} \bmod 221 = 47 \neq 220$
	$174^{55 \cdot 2^1} \bmod 221 = 220$

Test di primalità – Miller-Rabin

- Rabin ha dimostrato che se n è composto, allora ci sono almeno $3/4\,(n-1)$ testimoni in $[2,\ldots,n-1]$
- Il test di compostezza ha una probabilità inferiore a $1/4$ di rispondere erroneamente

---

boolean isPrime(**int** n)

```
for i = 1 to k do
    b = random(2, n − 1)
    if isComposite(n, b) then
        return false
return true
```

Test di Miller-Rabin

Riassunto

- Complessità: $O(k \log^2 n \log\log n \log\log\log n)$
- Probabilità di errore: $(1/4)^k$
- Algoritmo di tipo Montecarlo

Algoritmi probabilistici vs algoritmi deterministici

- Dal 2002, esiste l'algoritmo deterministico AKS di complessità $O(\log^{6+\epsilon} n)$
- I fattori moltiplicativi coinvolti sono molto alti
- Si preferisce quindi l'algoritmo di Miller-Rabin

Esempio – Espressione polinomiale nulla

Problema

Data un'espressione algebrica polinomiale $p(x_1, \ldots, x_n)$ in n variabili, determinare se p è identicamente nulla oppure no.

Discussione

- Assumiamo che non sia in forma di monomi - altrimenti è banale
- Gli algoritmi basati su semplificazioni sono molto complessi

Esempio – Espressione polinomiale nulla

Algoritmo

- Si genera una n-pla di valori $v_1, \ldots, v_n$
- Si calcola $x = p(v_1, \ldots, v_n)$
  - Se $x \neq 0$, p non è identicamente nulla
  - Se $x = 0$, p potrebbe essere identicamente nulla
- Se ogni v_i è un valore intero compreso casuale fra 1 e $2d$, dove d è il grado del polinomio, allora la probabilità di errore non supera $1/2$.
- Si ripete k volte, riducendo la probabilità di errore a $(1/2)^k$

BitSet + Tabelle Hash = Bloom Filters

BitSet

Vantaggi

- 1 bit/oggetto

Svantaggi

- Elenco prefissato di oggetti

Tabelle Hash

Vantaggi

- Struttura dati dinamica

Svantaggi

- Alta occupazione di memoria

Bloom filters

Vantaggi

- Struttura dati dinamica
- Bassa occupazione di memoria (10 bit/oggetto)

Svantaggi

- Niente cancellazioni
- Risposta probabilistica
- No memorizzazione

Bloom filter

Specifica

- insert(key): Inserisce l'elemento key nel bloom filter
- **boolean** contains(key)
  - Se restituisce **false**, l'elemento key è sicuramente non presente nell'insieme
  - Se restituisce **true**, l'elemento key può essere presente oppure no (falsi positivi)

Bloom filter

Trade-off fra occupazione di memoria e probabilità di falso positivo

- Sia ϵ la probabilità di falso positivo
- I bloom filter richiedono $1.44 \log_2(1/\epsilon)$ bit per elemento inserito

ϵ	Bit
10^{-1}	4.78
10^{-2}	9.57
10^{-3}	14.35
10^{-4}	19.13

Applicazioni dei Bloom Filter

Chrome Safe Browsing

- Chrome contiene un database delle URL associate a siti con malware, costantemente aggiornato
- Fino al 2012, memorizzato con un Bloom Filter
- Chrome verifica l'appartenenza di ogni URL al database
  - Se la risposta è **false**, non appartiene
  - Se la risposta è **true**, potrebbe appartenere e viene fatta una verifica tramite un servizio centralizzato di Google

Qualche dato (da prendere cum grano salis)

- Nel 2011, 650k URL memorizzati in 1.94MB
- 25 bit per URL, $\epsilon \approx 10^{-5}$

Applicazioni dei Bloom Filter

Ogni qual volta una verifica locale permette di evitare un'operazione più costosa, quali operazioni di I/O e comunicazioni di rete

They say...

- Medium uses Bloom filters to avoid recommending articles a user has previously read.
- Apache HBase uses bloom filter to boost read speed by filtering out unnecessary disk reads of HFile blocks which do not contain a particular row or column.
- Ethereum uses Bloom filters for quickly finding logs on the Ethereum blockchain.

https://en.wikipedia.org/wiki/Bloom_filter

Applicazioni dei Bloom Filter

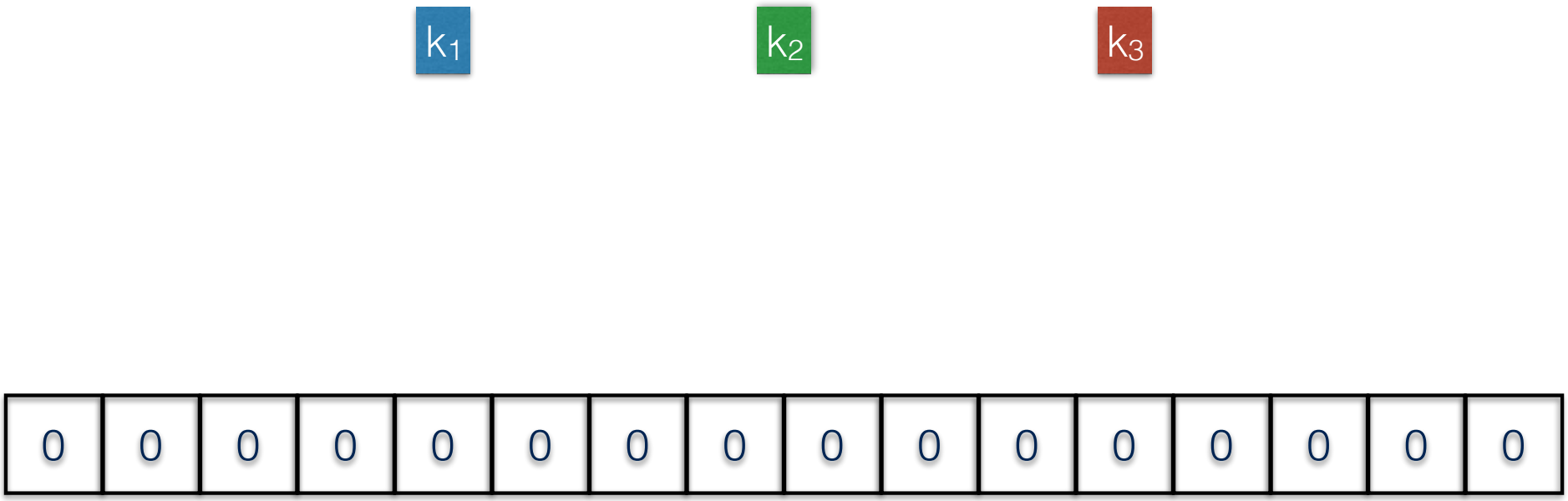
Firefox CRLite

CRLite is a technology proposed by a group of researchers at the IEEE Symposium on Security and Privacy 2017 that compresses revocation information so effectively that 6.7 GB of revocation data can become 1.3 MB. It accomplishes this by combining Certificate Transparency data and Internet scan results with cascading Bloom filters, building a data structure that is reliable, easy to verify, and easy to update.

https://blog.mozilla.org/security/2020/01/09/crlite-part-1-all-web-pki-revocations-compressed/
https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design/

Implementazione

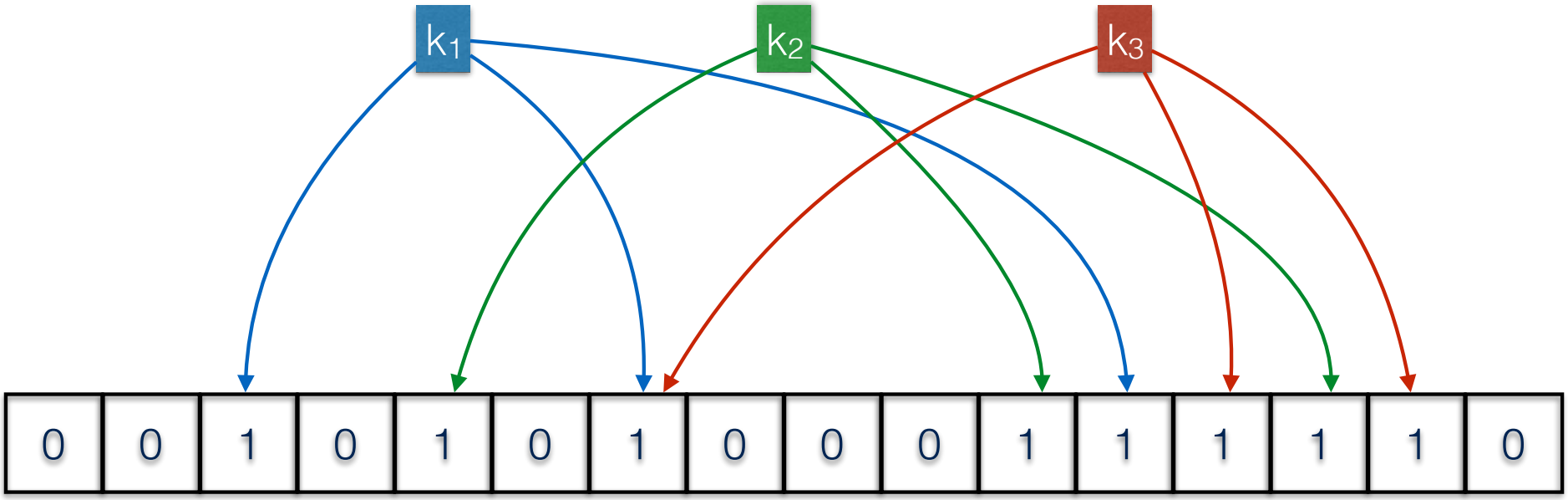
- Un vettore booleano A di m bit, inizializzato a **false**
- k funzioni hash $h_1, h_2, \ldots, h_k : U \to [0, m-1]$



Implementazione

insert(key)

for $i = 1$ **to** k **do**
  $A[h_i(key)] = \textbf{true}$

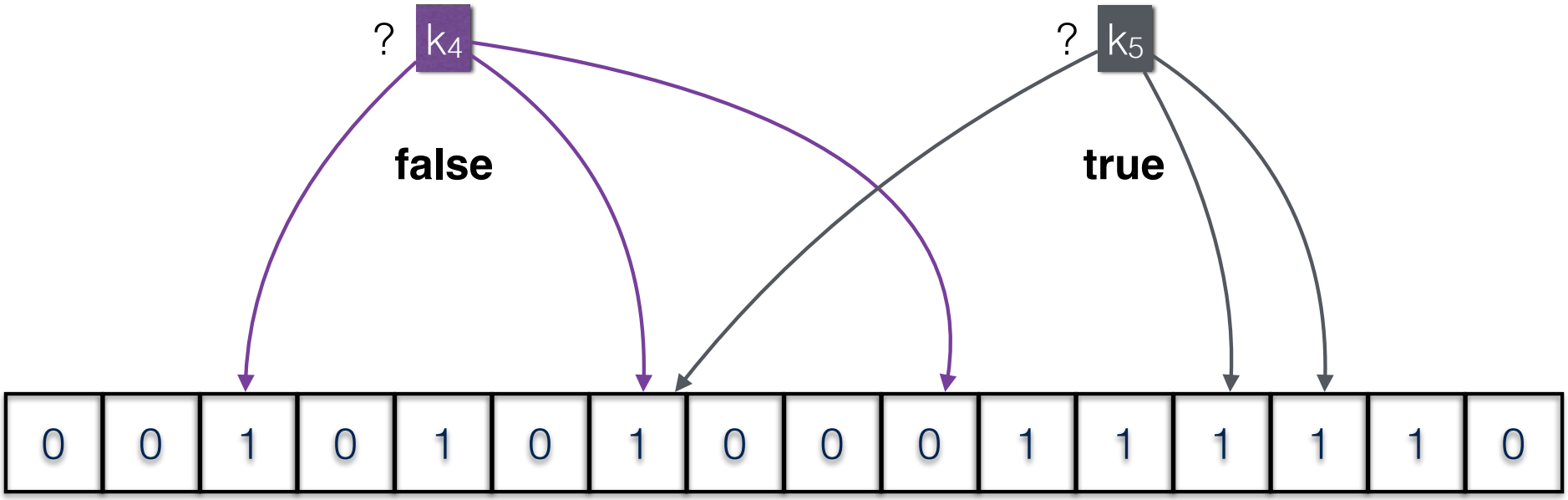


Implementazione

boolean contains(key)

for $i = 1$ **to** k **do**
  if $A[h_i(key)]$ == **false then return false**

return true



Qualche formula (senza dimostrazione)

- Dati n oggetti, m bit, k funzioni hash, la probabilità di un falso positivo è pari a:

$$\epsilon = \left(1 - e^{-kn/m}\right)^k$$

- Dati n oggetti e m bit, il valore ottimale per k è pari a

$$k = \frac{m}{n} \ln 2$$

- Dati n oggetti e una probabilità di falsi positivi ϵ, il numero di bit m richiesti è pari a:

$$m = -\frac{n \ln \epsilon}{(\ln 2)^2}$$

Statistica

Algoritmi statistici su vettori

Estraggono alcune caratteristiche statisticamente rilevanti da un vettore numerico

Esempi

- Media: $\mu = \frac{1}{n}\sum_{i=1}^{n} A[i]$
- Varianza: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(A[i] - \mu)^2$
- Moda: il valore (o i valori) più frequenti

Statistiche d'ordine

Selezione

Dato un array A contenente n valori e un valore $1 \leq k \leq n$, trovare l'elemento che occuperebbe la posizione k se il vettore fosse ordinato

Mediana

Il problema del calcolo della mediana è un sottoproblema del problema della selezione con $k = \lceil n/2 \rceil$.

Come risolvereste il problema?

Ovviamente, possiamo ordinare i valori e andare a cercare il valore in posizione k, con costo $O(n \log n)$. Possiamo fare meglio di così?

Selezione per piccoli valori di k

Intuizione

- Si può utilizzare uno heap
- L'algoritmo può essere generalizzato a valori generici di $k > 2$

int heapSelect(ITEM [] A, **int** n, **int** k)

```
buildHeap(A)
for i = 1 to k − 1 do
    deleteMin(A, n)
return deleteMin(A, n)
```

Complessità

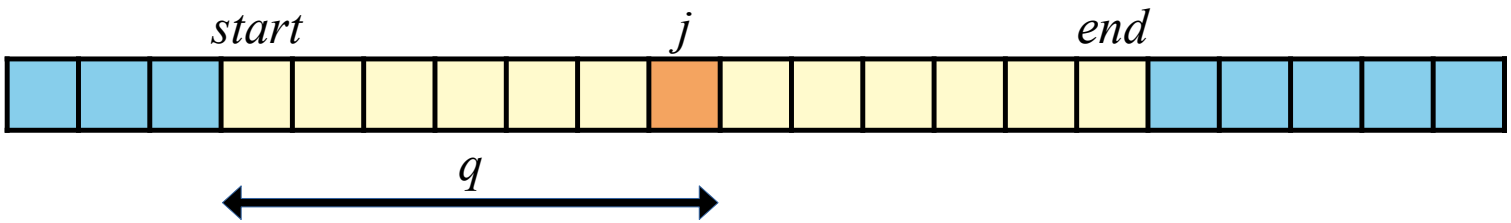
- $O(n + k \log n)$
- Se $k = O(n/\log n)$, il costo è $O(n)$
- Se $k = n/2$, non va bene

Idea

- Approccio divide-et-impera simile al Quicksort
- Essendo un problema di ricerca, non è necessario cercare in entrambe le partizioni, basta cercare in una sola di esse
- Bisogna fare attenzione agli indici

Algoritmo di selezione

```
ITEM selection(ITEM[] A, int start, int end, int k)
if start == end then
    return A[start]
else
    int j = pivot(A, start, end)
    int q = j − start + 1
    if k == q then
        return A[j]
    else if k < q then
        return selection(A, start, j − 1, k)
    else
        return selection(A, j + 1, end, k − q)
```



Complessità

Caso pessimo

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n-1) + n & n > 1 \end{cases}$$

$$T(n) = O(n^2)$$

Caso ottimo

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n/2) + n & n > 1 \end{cases}$$

$$T(n) = O(n)$$

Complessità

Caso medio

Assumiamo che pivot() restituisca con la stessa probabilità una qualsiasi posizione j del vettore A

$$T(n) = n + \frac{1}{n}\sum_{q=1}^{n} T\left(\max\{q-1, n-q\}\right) \qquad \text{Media su } n \text{ casi}$$

$$\leq n + \frac{1}{n}\sum_{q=\lfloor n/2 \rfloor}^{n-1} 2T(q), \qquad \text{per } n > 1$$

Esempi

- $n = 4$: $\max\{0,3\} + \max\{1,2\} + \max\{2,1\} + \max\{3,0\}$
- $n = 5$: $\max\{0,4\} + \max\{1,3\} + \max\{2,2\} + \max\{3,1\} + \max\{4,0\}$

Complessità

$$T(n) \leq n + \frac{1}{n} \sum_{q=\lfloor n/2 \rfloor}^{n-1} 2 \cdot cq \leq n + \frac{2c}{n} \sum_{q=\lfloor n/2 \rfloor}^{n-1} q$$ Sostituzione, raccolgo $2c$

$$= n + \frac{2c}{n} \left(\sum_{q=1}^{n-1} q - \sum_{q=1}^{\lfloor n/2 \rfloor - 1} q \right)$$ Sottrazione prima parte

$$= n + \frac{2c}{n} \cdot \left(\frac{n(n-1)}{2} - \frac{(\lfloor n/2 \rfloor)(\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{2c}{n} \cdot \left(\frac{n(n-1)}{2} - \frac{(n/2 - 1)(n/2 - 2)}{2} \right)$$ Rimozione limite inferiore

$$= n + \frac{\cancel{2}c}{n} \cdot \frac{\left(n^2 - n - \left(1/4\, n^2 - 3/2\, n + 2\right)\right)}{\cancel{2}}$$

$$= n + c/n \cdot \left(3/4\, n^2 + 1/2\, n - 2\right)$$

$$\leq n + c/n \cdot \left(3/4\, n^2 + 1/2\, n\right) = n + 3/4\, cn + 1/2 \overset{?}{\leq} cn$$ Vera per $c \geq 6$ e $n \geq 1$

Complessità

- Siamo partiti dall'assunzione
  - j assume equiprobabilisticamente tutti i valori compresi fra 1 e n
- E se non fosse vero?
- Lo forziamo noi!
  - $A[\mathsf{random}(start, end)] \leftrightarrow A[start]$
- Questo accorgimento vale anche per QuickSort
- La complessità nel caso medio:
  - $O(n)$ nel caso della Selezione
  - $O(n \log n)$ nel caso dell'Ordinamento

Selezione deterministica

Algoritmo black-box

Supponiamo di avere un algoritmo "black box" che mi ritorni il mediano di n valori in tempo $O(n)$

Domande

- Potrei utilizzarlo per ottimizzare il problema della selezione?
- Che complessità otterrei?

Selezione deterministica

Se conoscessi tale algoritmo

- il problema della selezione sarebbe quindi risolto...
- ... ma dove lo trovo un simile algoritmo?

Rilassiamo le nostre pretese

- Supponiamo di avere un algoritmo "black box" che mi ritorni un valore che dista al più $\frac{3}{10}n$ dal mediano (nell'ordinamento)
- Potrei utilizzarlo per ottimizzare il problema della selezione?
- Che complessità otterrei?

Selezione deterministica

Idea

- Suddividi i valori in gruppi di 5. Chiameremo l'i-esimo gruppo S_i, con $i \in [1, \lceil n/5 \rceil]$
- Trova il mediano M_i di ogni gruppo S_i
- Tramite una chiamata ricorsiva, trova il mediano m delle mediane $[M_1, M_2, \ldots, M_{\lceil n/5 \rceil}]$
- Usa m come pivot e richiama l'algoritmo ricorsivamente sull'array opportuno, come nella selection() randomizzata
- Quando la dimensione scende sotto una certa dimensione, possiamo utilizzare un algoritmo di ordinamento per trovare il mediano

Selezione deterministica

ITEM select(ITEM[] A, **int** $start$, **int** end, **int** k)

% Se la dimensione è inferiore ad una soglia (10), ordina il vettore e
% restituisci il k-esimo elemento di $A[start \ldots end]$
if $end - start + 1 \leq 10$ **then**
    InsertionSort(A, $start$, end) % Versione con indici inizio/fine
    return $A[start + k - 1]$

% Divide A in $\lceil n/5 \rceil$ sottovettori di dim. 5 e ne calcola la mediana
$M =$ **new int**$[1 \ldots \lceil n/5 \rceil]$
for $i = 1$ **to** $\lceil n/5 \rceil$ **do**
    $M[i] =$ median5(A, $start + (i-1) \cdot 5$, end)

% Individua la mediana delle mediane e usala come perno
ITEM $m =$ select($M, 1, \lceil n/5 \rceil, \lceil \lceil n/5 \rceil / 2 \rceil$)
int $j =$ pivot(A, $start$, end, m) % Versione con m in input
[...]

Selezione deterministica

ITEM **select**(ITEM[ ] A, **int** $start$, **int** end, **int** k)

```
[...]
% Calcola l'indice q di m in [start ... end]
% Confronta q con l'indice cercato e ritorna il valore conseguente
int q = j − start + 1
if q == k then
    return m
else if q < k then
    return select(A, start, q − 1, k)
else
    return select(A, q + 1, end, k − q)
```

Selezione deterministica

- Il calcolo dei mediani $M[\,]$ richiede al più $6\lceil n/5 \rceil$ confronti.
- La prima chiamata ricorsiva dell'algoritmo select() viene effettuata su $\lceil n/5 \rceil$ elementi
- La seconda chiamata ricorsiva dell'algoritmo select() viene effettuata al massimo su $7n/10$ elementi (esattamente $n - 3\lceil \lceil n/5 \rceil /2 \rceil$)
- L'algoritmo select() esegue nel caso pessimo $O(n)$ confronti

$$T(n) = T(n/5) + T(7n/10) + 11/5n$$

1	4	**7**	10	13	2	5	**8**	11	14	3	6	**9**	12	15

Conclusioni

Quale preferire?

- Algoritmo probabilistico Las Vegas in tempo atteso $O(n)$
- Algoritmo deterministico in tempo $O(n)$, con fattori moltiplicativi più alti

Note storiche

- Nel 1883 Lewis Carroll (!) notò che il secondo premio nei tornei di tennis non veniva assegnato in maniera equa.
- Nel 1932, Schreier dimostrò che $n + \log n - 2$ incontri sono sempre sufficienti per trovare il secondo posto
- Nel 1973, a opera di Blum, Floyd, Pratt, Rivest e Tarjan, appare il primo algoritmo deterministico