

Lavoriamo con gli Array

Programmazione

Corso di Laurea in Informatica

Corso di Laurea in Informatica per la Comunicazione Digitale

Università di Camerino

Prof. Michele Loreti

Anno Accademico 2024/25

Ricerca ed Ordinamento

I problemi di **ricerca** e **ordinamento** sono particolarmente importanti in informatica, e per questo motivo è fondamentale individuare algoritmi **efficienti** per la risoluzione di questi problemi

Ricerca sequenziale o “ingenua”

Problema

Data una sequenza a_1, a_2, \dots, a_n di elementi, trovare, se esiste, l'indice dell'elemento di valore x

Soluzione

```
int ricerca_ingenua(int[] a, int x) {  
    for (int i = 0; i < a.length; i++)  
        if (a[i] == x) return i;  
    return -1;  
}
```

Note

- per convenzione, il metodo restituisce -1 se l'elemento non è presente (-1 non può essere confuso con un indice valido che è sempre non negativo)
- nel **caso pessimo** (se l'elemento cercato non c'è) si fanno n confronti

Ricerca ordinata

Problema

Data una sequenza **ordinata** $a_1 \leq a_2 \leq \dots \leq a_n$ di elementi, trovare, se esiste, l'indice dell'elemento di valore x

Soluzione

```
int ricerca_ordinata(int[] a, int x) {  
    int i = 0;  
    while (i < a.length && a[i] < x) i++;  
    return i < a.length && a[i] == x ? i : -1;  
}
```

Note

- la ricerca termina non appena si trova un elemento uguale o più grande
- nel **caso pessimo** (se l'elemento cercato è **l'ultimo** nella sequenza) si fanno $n + 1$ confronti
- è fondamentale che il controllo $i < a.length$ sia fatto **per primo**

Sulla valutazione delle espressioni booleane

a	b	a && b	a b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Osservazioni

- true è l'elemento **assorbente** di ||
- false è l'elemento **assorbente** di &&

Valutazione cortocircuitata

Java valuta un'espressione booleana **da sinistra verso destra** e si **interrompe** non appena il suo valore è determinato

a	b	a && b	b valutato
false	false	false	NO
false	true	false	NO
true	false	false	SÌ
true	true	true	Sì

a	b	a b	b valutato
false	false	false	SÌ
false	true	true	SÌ
true	false	true	NO
true	true	true	NO

In Java (e in molti altri linguaggi) i connettivi && e || **non sono commutativi**

Espressione	Valore
false && 1 / 0 == 0	false
true 1 / 0 == 0	true
1 / 0 == 0 && false	divisione per zero
1 / 0 == 0 true	divisione per zero

Ricerca binaria

Problema

Data una sequenza **ordinata** $a_1 \leq a_2 \leq \dots \leq a_n$ di elementi, trovare, se esiste, l'indice dell'elemento di valore x

Idea

- si usano due indici i e j per individuare lo **spazio di ricerca**, ovvero l'intervallo entro il quale si trova (se c'è) l'elemento cercato
- inizialmente si imposta i a 1 e j a n
- si calcola $k = (i + j)/2$ il punto medio dello spazio di ricerca
- se $x = a_k$, la ricerca termina con successo
- se $x < a_k$, allora x si trova (se c'è) nell'intervallo compreso tra i e $k - 1$
- se $x > a_k$, allora x si trova (se c'è) nell'intervallo compreso tra $k + 1$ e j

Realizzazione della ricerca binaria

```
int ricerca_binaria(int[] a, int x) {  
    int i = 0;  
    int j = a.length - 1;  
    while (i <= j) {  
        int k = (i + j) / 2;  
        if (x == a[k]) return k;  
        else if (x < a[k]) j = k - 1;  
        else i = k + 1;  
    }  
    return -1;  
}
```

Osservazioni

- nel **caso pessimo** (se l'elemento cercato non c'è) si fanno $2 \log_2 n$ confronti
- per n "grande" la ricerca binaria è (molto) più efficiente di quella ordinata
- la ricerca binaria presuppone di poter accedere efficientemente a ogni elemento della sequenza a prescindere dalla sua posizione

Ordinamento per selezione

Problema

Ordinare una sequenza a_1, a_2, \dots, a_n

Idea

- il **primo elemento** della sequenza ordinata sarà il **minimo** della sequenza iniziale
- il **secondo elemento** della sequenza ordinata sarà il **minimo** della sequenza iniziale escluso il primo elemento della sequenza ordinata
- ...
- per ogni posizione i da 1 a n si scambia l'elemento in posizione i con il minimo elemento trovato nell'intervallo da i a n

Realizzazione dell'ordinamento per selezione

```
void selection_sort(int[] a) {  
    for (int i = 0; i < a.length; i++) {  
        int m = i;  
        for (int j = i + 1; j < a.length; j++)  
            if (a[j] < a[m]) m = j;  
        int t = a[i];  
        a[i] = a[m];  
        a[m] = t;  
    }  
}
```

Osservazioni

- si eseguono sempre $n(n - 1)/2$ confronti indipendentemente dal contenuto dell'array
- l'ordinamento per selezione è **quadratico**

Realizzazione dell'ordinamento per selezione

```
int trovaMinimoDopo(int[] a, int i) {  
    int m = i;  
    for (int j = i + 1; j < a.length; j++)  
        if (a[j] < a[m]) m = j;  
    return m;  
}  
  
void scambia(int[] a, int i, int j) {  
    int t = a[i];  
    a[i] = a[j];  
    a[j] = t;  
}  
  
void selection_sort(int[] a) {  
    for (int i = 0; i < a.length; i++) {  
        int m = trovaMinimoDopo(a, i);  
        scambia(a, i, m);  
    }  
}
```

Ordinamento per fusione

Problema

Ordinare una sequenza a_1, a_2, \dots, a_n

Strategia “divide et impera”

- se $n \leq 1$ il problema è banale (la sequenza è già ordinata)
- se $n > 1$ procedo in questo modo:
 1. divido la sequenza in due sotto-sequenze di lunghezza (quasi) uguale, a_1, \dots, a_k da una parte e a_{k+1}, \dots, a_n dall'altra, dove $k = \lfloor \frac{n}{2} \rfloor$
*(**divide**: scompongo il problema in sotto-problemi più semplici)*
 2. osservo che ciascuna di queste sotto-sequenze è **più corta** della sequenza iniziale, pertanto posso ordinarle ricorsivamente ottenendo due sequenze corrispondenti b_1, \dots, b_k e c_1, \dots, c_{n-k}
(questo passaggio non richiede codice grazie alla ricorsione)
 3. fondo le due sequenze ottenute in una nuova sequenza ordinata
*(**impera**: la fusione di sequenze ordinate è relativamente semplice)*

Realizzazione dell'ordinamento per fusione

```
int[] merge_sort(int[] a, int i, int j) {  
    if (i > j) return new int[] { };  
    else if (i == j) return new int[] { a[i] };  
    else {  
        int k = (i + j) / 2;  
        int[] b = merge_sort(a, i, k);  
        int[] c = merge_sort(a, k + 1, j);  
        return merge(b, c);  
    }  
}
```

Note

- gli argomenti *i* e *j* indicano l'intervallo di *a* da ordinare
- l'espressione `new int[] { }` crea un **array vuoto**
- l'espressione `new int[] { x }` crea un **array con un solo elemento x**

Fusione di array ordinati

```
int[] merge(int[] a, int[] b) {  
    int[] c = new int[a.length + b.length];  
    int i = 0;  
    int j = 0;  
    int k = 0;  
    while (i < a.length && j < b.length)  
        if (a[i] <= b[j]) c[k++] = a[i++];  
        else c[k++] = b[j++];  
    while (i < a.length) c[k++] = a[i++];  
    while (j < b.length) c[k++] = b[j++];  
    return c;  
}
```

Nota

- l'uso dei post-incrementi è fondamentale per la correttezza del metodo

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Dobbiamo inoltre comprendere il flusso di computazione che fa il programma.

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Dobbiamo inoltre comprendere il flusso di computazione che fa il programma.

Dobbiamo per prima cosa decidere come **rappresentare** i dati rilevanti per il gioco:

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Dobbiamo inoltre comprendere il flusso di computazione che fa il programma.

Dobbiamo per prima cosa decidere come **rappresentare** i dati rilevanti per il gioco:

- la sequenza;
- i suggerimenti;

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Dobbiamo inoltre comprendere il flusso di computazione che fa il programma.

Dobbiamo per prima cosa decidere come **rappresentare** i dati rilevanti per il gioco:

- la sequenza;
- i suggerimenti;

In entrambi i casi possiamo usare degli array!

Il Mastermind

Usiamo gli array per implementare il gioco del Mastermind

Dobbiamo inoltre comprendere il flusso di computazione che fa il programma.

Dobbiamo per prima cosa decidere come **rappresentare** i dati rilevanti per il gioco:

- la sequenza;
- i suggerimenti;

In entrambi i casi possiamo usare degli array!

Nella scrittura del codice seguiamo un approccio *bottom up*: partiamo dal `main` ed andiamo via via ad individuare i metodi di cui abbiamo bisogno.

Il Mastermind

```
public static void main(String[] args) {
    int n = scegliLunghezzaSequenza();
    int[] segreto = inserisciSequenza(n);
    int[] risultato = {0, 0};
    while (risultato[0] < n) {
        int[] guess = inserisciSequenza(n);
        risultato = confronta(segreto, guess);
    }
    StdOut.println("Sequenza indovinata!");
}

public static int[] inserisciSequenza(int n) {
    int[] guess = new int[n];
    for(int i = 0; i < n; i++) {
        StdOut.print("Inserire l'elemento in posizione "+i+":");
        guess[i] = StdIn.readInt();
    }
    return guess;
}

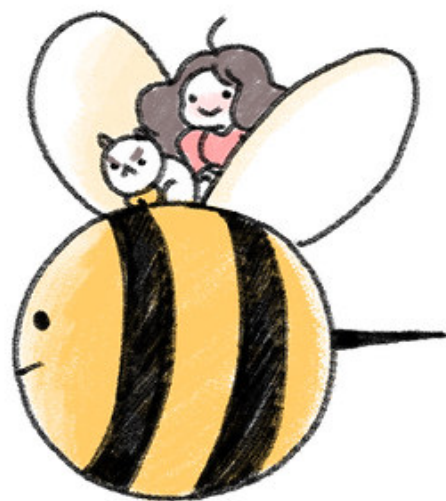
public static int scegliLunghezzaSequenza() {
    StdOut.print("Inserisci un valore intero (maggiore di 0): ");
    return StdIn.readInt();
}
```

Il Mastermind

```
public static int[] confronta(int[] segreto, int[] guess) {  
    int[] risultato = {0, 0};  
    risultato[0] = contaCorretti(segreto, guess);  
    risultato[1] = contaPosizioneSbagliata(segreto, guess);  
    return risultato;  
  
    public static int contaCorretti(int[] segreto, int[] guess) {  
        int c=0;  
        for(int i=0; i<segreto.length; i++) {  
            if(segreto[i] == guess[i]) {  
                c++;  
            }  
        }  
        return c;  
    }  
}
```

Il Mastermind

```
private static int contaPosizioneSbagliata(int[] segreto, int[] guess) {  
    boolean[] elementiUsati = new boolean[segreto.length];  
    int c = 0;  
    for (int i = 0; i < guess.length; i++) {  
        boolean found = false;  
        for(int j = 0; (j < segreto.length)&&!found; j++) {  
            if (!elementiUsati[j]  
                &&(i!=j)  
                &&(guess[i] == segreto[j])  
                &&(guess[j] != segreto[j])) {  
                c++;  
                elementiUsati[j] = true;  
                found = true;  
            }  
        }  
    }  
    return c;  
}
```



TO
BE
CONTINUED!