Università degli Studi di Ferrara

Corso di Laurea in Matematica - A.A. 2021 - 2022

Programmazione
Lezione 9 – Ordinamento e Matrici

Docente: Michele Ferrari - michele.ferrari@unife.it

Nelle lezioni precedenti

- Tipi di dato fondamentali, limiti e caratteristiche: int (interi) float (reali) double (reali a doppia precisione) char (caratteri)
- Conversioni di tipo
- Come creare e manipolare array monodimensionali

In questa lezione

- L’Operatore ternario
- Operazioni base su un array
- Ordinamento di un array
- Ricerca di un elemento in un array
- Array multidimensionali

Operatore Ternario

Il C prevede un operatore speciale detto operatore ternario che consente di effettuare una selezione fra 2 espressioni:

Sintassi:

```
<espressione1> ? <espressione2> : <espressione3>
```

Lo possiamo interpretare come:

SE espressione1 è vera,

ALLORA considera espressione2,

ALTRIMENTI considera espressione3

Operatore Ternario

```
<espressione1> ? <espressione2> : <espressione3>
```

Esempio:

```
maggiore = a>b ? a : b;
```

Equivale a scrivere:

```
if (a>b) {
    maggiore = a;
} else {
    maggiore = b;
}
```

Operazioni sugli Array

- Lettura e Scrittura (già visto)
- Shift
- Rotazione
- Determinazione del minimo (o massimo) e posizione
- Ricerca
- Ordinamento

Altre OPERAZIONI sono ottenute per mezzo delle precedenti:

- **Fusione (Merge)**: Combinazione di due o più array in uno
- **Separazione (Split)**: opposto della precedente

Shift di un Array

Shift completo verso destra: Tutti gli elementi sono spostati verso destra di una posizione.

Si perde l'ultimo elemento mentre il primo viene ripetuto due volte.

Shift completo a sinistra: Tutti gli elementi sono spostati verso sinistra di una posizione, perdendo il primo elemento.

Si perde il primo elemento mentre l'ultimo viene ripetuto due volte.

Shift di un Array

```
char vett[5] = {‘a’,’b’,’c’,’d’,’e’};
int i;

for (i=0; i<(5-1); i++) {
    vett[i] = vett[i+1];
}
for (i=0; i<5; i++) {
    printf("%c ", vett[i]);
}
```

Rotazione di un Array

La rotazione, è un particolare caso dello Shift in cui tutti gli elementi sono spostati verso destra (o sinistra) di una posizione, compreso l'ultimo elemento.

Ricerca del minimo

Si tratta di scorrere l'array alla ricerca del valore minimo (o massimo) memorizzato all'interno dell'array.

```
int vett[5] = {3,6,8,2,5};
int i, minimo;
minimo = vett[0];

for (i=1; i<N; i++) {
  if (vett[i] < minimo) {
    minimo = vett[i];
  }
}
printf("%d \n", minimo);
```

Ricerca di un valore

La ricerca di un valore all'interno di un array NON ORDINATO è un caso molto simile alla ricerca del minimo, in questo caso però, il valore da ricercare è inserito dall'utente.

```
int vett[N] = {3,6,8,2,5};
int i, val;

printf("Inserisci il valore da ricercare: ");
scanf("%d", &val);

for (i=0; i<N; i++) {
  if (vett[i] == val) {
    printf("Ho trovato %d in posizione %d \n", val, i);
    break; // se voglio fermarmi dopo la prima occorrenza
  }
}
```

NOTA: La parola chiave `break;` all'interno di un ciclo, causa l'interruzione dello stesso.

Il problema dell’ordinamento

Problema

Data una sequenza di elementi in ordine qualsiasi, ordinarla.

Si tratta di classico problema algoritmico utilizzato per l’analisi dei diversi approcci alla risoluzione di un problema.

Insertion Sort

L'insertion sort, (ordinamento per inserimento), è un algoritmo di ordinamento sul posto, (cioè ordina l'array senza doverne creare una copia), si tratta di un modo relativamente semplice per ordinare un array.

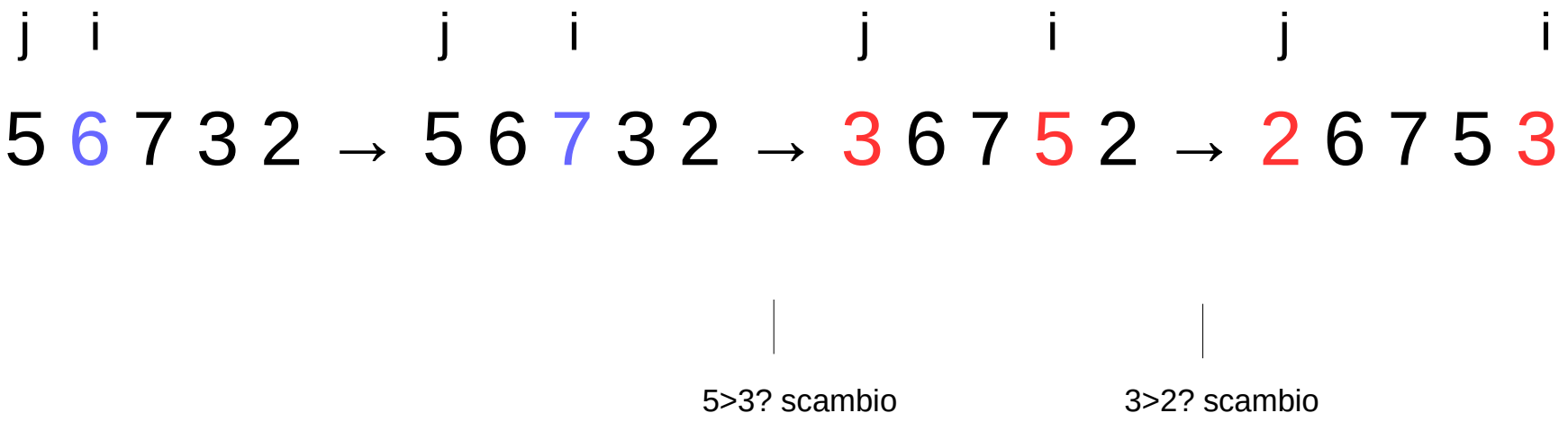
Strategia

L'algoritmo utilizza due indici: i e j, i quali scorrono l’array a due “velocità” diverse:

- L’indice più lento (nel nostro caso j) punta inizialmente al primo elemento dell’array, mentre l’indice più “veloce” (nel nostro caso i), punta inizialmente al secondo elemento dell'array
- L’elemento puntato dall’indice j, viene confrontato con tutti i restanti elementi dell’array puntati dall’indice i
- Ogni volta che l'elemento puntato da j è maggiore di quello a cui punta i, gli elementi vengono scambiati di posto

Insertion Sort

Spostiamo man mano gli elementi maggiori verso destra.



Insertion sort

I due indici scorrono l'array a due "velocità" diverse, cioè l'indice più lento avanza di una posizione quando l'indice più veloce ha già finito di scorrere tutti gli elementi rimanenti.

```
int vett[N] = {3, 6, 7, 5, 2};
int i, j, tmp;

for (j=0; j<N; j++) {
  for (i=j+1; i<N; i++) {
    if (vett[j] > vett[i]) {
      tmp = vett[j];
      vett[j] = vett[i];
      vett[i] = tmp;
    }
  }
}
```

Selection Sort

Anche l'ordinamento selection sort (ordinamento per selezione) è un algoritmo che opera sul posto.

Strategia

In questo ordinamento, la sequenza da ordinare viene suddivisa in due parti:

1. La sottosequenza ordinata, che occupa le prime posizioni dell'array
2. La sottosequenza da ordinare, che occupa la parte restante dell'array

Selection Sort

L’algoritmo seleziona il numero minore nella sequenza di partenza e lo sposta nella sequenza ordinata.

1. Si inizializza un indice i che va da 1 a N
2. Si cerca il più piccolo elemento dell'array
3. Scambia l'elemento più piccolo con l'elemento alla posizione i
4. Si incrementa l'indice i e si torna al passo 1

Selection Sort

```
int vett[N] = {3, 6, 7, 5, 2};
int i, j, k, tmp;

for (j=0; j < N-1; j++) {
  k = j;
  for (i=j+1; i<N; i++) {
    if (vett[k] > vett[i]) {
      k = i;
    }
  }
  if (k != j) {
    tmp = vett[j];
    vett[j] = vett[k];
    vett[k] = tmp;
  }
}
```

Bubblesort

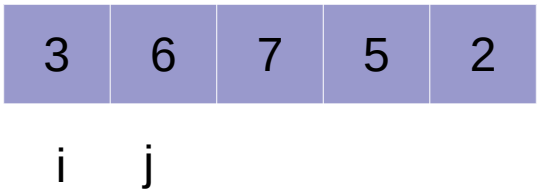
Il bubblesort è un algoritmo iterativo, ovvero basato sulla ripetizione di un procedimento fondamentale.

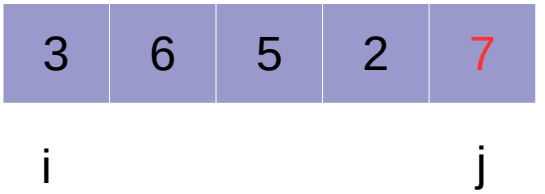
La singola iterazione dell'algoritmo prevede che gli elementi dell'array siano confrontati a due a due, procedendo in un verso stabilito.

Vengono confrontati il primo elemento con il secondo elemento, poi il secondo con il terzo e così via fino al confronto fra penultimo e ultimo elemento.

Bubblesort

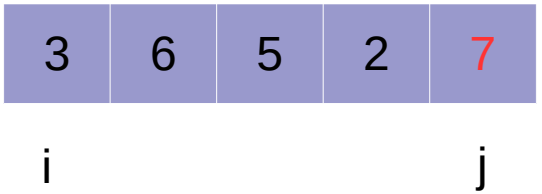
Per ogni confronto, se i due elementi confrontati non sono ordinati, essi vengono scambiati. Durante ogni iterazione, almeno un valore viene spostato rapidamente fino a raggiungere la sua collocazione definitiva; in particolare, alla prima iterazione il numero più grande raggiunge l'ultima posizione dell'array.





Bubblesort

- Se i numeri sono in tutto N, dopo N-1 iterazioni si avrà la garanzia che l'array sia ordinato
- La sequenza di confronti può essere terminata col confronto dei valori alle posizioni N-1-i e N-i



Bubblesort

```
int vett[N] = {3, 6, 7, 5, 2};
int i, j, tmp;

for (i=0; i < (N - 1); i++) {
  for (j=0; j < (N - i - 1); j++) {
    if (vett[j] > vett[j+1]) {
      tmp = vett[j];
      vett[j] = vett[j+1];
      vett[j+1] = tmp;
    }
  }
}
```

Ricerca binaria

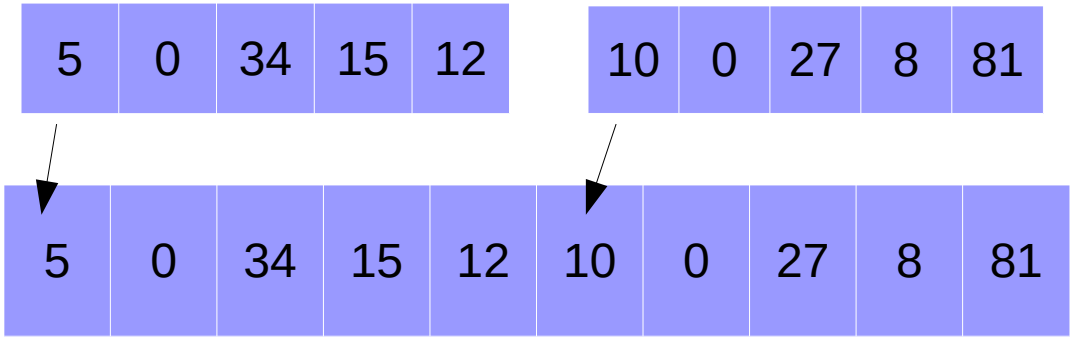
La ricerca binaria **prevede che l'array sia ordinato** e procede, in un certo senso come la ricerca di una parola in un dizionario.

- Si confronta l'elemento da cercare con l'elemento in posizione centrale
- Se sono uguali si termina la ricerca con successo

Altrimenti si prosegue con la prima metà del vettore o l'ultima metà a seconda che l'elemento da cercare sia inferiore o superiore all'elemento del vettore con cui si è fatto il confronto.

Fusione (merge)

Si scorrono gli N array con indici diversi e si inseriscono in un N+1-esimo array.



L’array risultante deve (ovviamente) avere dimensione sufficiente a contenere il nuovo numero di elementi

Fusione (merge)

```
int main(){
  int vetta[N] = {4,7,2,9,3};
  int vettb[M] = {8,1,6};
  int vettc[M+N];
  int i;

  i=0;
  while(i<N) {
    vettc[i] = vetta[i];
    i++;
  }
  i=0;
  while(i<M) {
    vettc[N+i] = vettb[i];
    i++;
  }
  return 0;
}
```

Complessità computazionale

Per risolvere un problema computabile esistono molti algoritmi risolutivi. Gli algoritmi vengono valutati secondo diversi criteri, che fanno di un certo algoritmo la soluzione migliore in alcuni casi ma non in altri.

I principali criteri di valutazione d'un algoritmo sono:

- Tempo di calcolo occorrente (complessità temporale)
- Occupazione di memoria (complessità spaziale)

Solitamente ottimizzare rispetto a un criterio significa perdere prestazioni rispetto all'altro.

L’operazione più costosa è il CONFRONTO

Algoritmi di ordinamento: analisi

Algoritmo	Caso Migliore	Caso Peggiore
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$

Array bidimensionali

In un array bidimensionale i dati sono organizzati per righe e per colonne.

Per la memorizzazione si usa una variabile come per l'array specificando il numero di elementi per ciascuna delle due dimensioni che la costituiscono:

```
int mat[N][M];
```

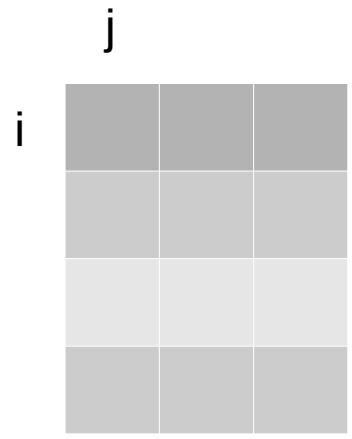
Dove N è il numero di righe ed M è il numero di colonne.

Array bidimensionali

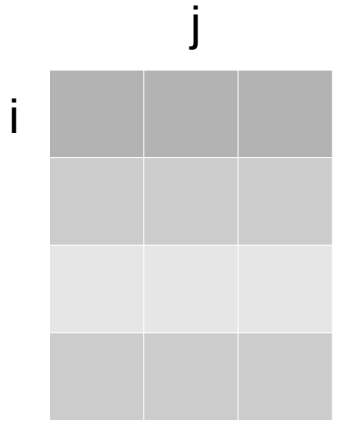
```
int mat[4][3];
```

La matrice mat è composta da 4 righe e 3 colonne, e per potervi accedere è necessario utilizzare 2 indici:

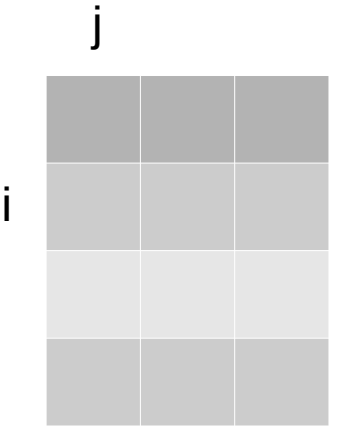
```
mat[i][j]
```



`mat[0][0]`



`mat[0][1]`



`mat[1][0]`

Scorrimento di una matrice

```
#include <stdio.h>

int main() {
int i,j;
int mat[4][3]; // 4 righe, 3 colonne

for(i=0; i<4; i++) { // scorro le righe
    for(j=0; j<3; j++) { // scorro le colonne
        // uso mat[i][j] per accedere alla matrice
    }
}
}
```

I due cicli annidati ci consentono di scorrere la matrice mat per righe

Domanda

Domanda: E’ più veloce scorrere una matrice per righe, per colonne oppure è indifferente?

Array multidimensionali

L’array bidimensionale (o matrice) è un caso particolare dei più generici array multidimensionali.

E’ possibile dichiarare array a più dimensioni con la seguente sintassi:

```
<tipo> nome_array[dim1][dim2]...[dimN]
```

Per poter accedere agli elementi di un array ad N dimensioni è necessario utilizzare N indici.

Funzioni rand e srand

Le funzioni rand e srand sono utilizzate per la generazione di numeri casuali, in particolare la funzione rand restituisce un intero compreso fra **0** e **RAND_MAX**.

I numeri restituiti da rand **non sono del tutto casuali**, ma sono generati a partire da un “seme” (seed).

La chiamata alla funzione srand fornisce il seme per la funzione rand

Entrambe le funzioni sono definite dalla libreria stdlib.h

Funzioni rand e srand

Se un programma utilizza sempre lo stesso seme, dalla funzione rand otterrà sempre la stessa sequenza di numeri casuali.

Per rendere sempre diverse le sequenze di numeri casuali generate, è quindi necessario cambiare sempre il valore del seme ed il modo più semplice per rendere il suo valore sempre diverso è utilizzare la funzione time che restituisce un valore che codifica la data e l'ora corrente.

Passare il valore restituito dalla funzione time alla funzione srand farà in modo che la funzione rand restituisca sequenze sempre diverse da una esecuzione all'altra.

```
srand((unsigned) time(NULL));
numero_casuale = rand() % 10;
```

La funzione time è definita nella libreria time.h

Grazie per l’attenzione

Riferimenti

Il corso di programmazione per il primo anno della Laurea Triennale in Matematica nasce con l'intento di unire ai principi di programmazione una conoscenza basilare di uno degli strumenti software più diffusi nell'ambito matematico: Matlab.

La prima parte del corso pertanto è una rielaborazione del programma di Programmazione per la Laurea Triennale in Informatica 15/16 (in particolare) e 16/17.

Parte del materiale originale da cui ho ricavato il percorso didattico, alcuni approfondimenti ed integrazioni possono essere trovati in:

- Lezioni di Programmazione 15/16 CdS Informatica - Giacomo Piva
- Lezioni di Programmazione 16/17 CdS Informatica - Marco Alberti