

Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano



- Lezioni: Martedì 8.30 - 10.30 in aula 8 Mercoledì 10.30 - 13.30 in aula 2
Giovedì 15.30 - 18.30 in aula 2 Venerdì 10.30 - 12.30 in aula 3
- Ricevimento: su appuntamento (Dipartimento di Informatica)
- E-mail: roberto.cordone@unimi.it
- Pagina web: <http://homes.di.unimi.it/~cordone/courses/2023-algo/2023-algo.html>
- Sito Ariel: <https://mgoldwurma.ariel.ctu.unimi.it>

Strutture dati astratte e implementazioni

La memoria dei processori è una semplice sequenza di celle elementari

Una **struttura dati** è una **organizzazione delle celle della memoria** che **consenta di operare sui dati in modo semplice ed efficiente**

Distingueremo tra

- **struttura dati astratta** (**che cosa si fa e su quali dati**), che **consiste in**
 - **uno o più insiemi di dati elementari**
 - **una o più operazioni eseguibili su tali insiemi**

senza specificare l'organizzazione e la tecnologia adoperate

- **implementazione** (**come si fa**), che **consiste nel modo specifico in cui**
 - **i dati vengono distribuiti nella memoria**
 - **le operazioni vengono eseguite sui dati**

Questo consente di dividere l'analisi di un algoritmo in due fasi

- ① **descrizione dell'algoritmo in termini di dati, operazioni e risultati**
- ② **realizzazione specifica di ciascuna operazione**

La prima fase è più semplice, la seconda condivisa fra più algoritmi

Strutture dati astratte e implementazioni

Per operazioni molto comuni, si arriva al progetto di librerie generiche

Per esempio, conservare un insieme di oggetti potendo estrarne il minimo è utile in diverse situazioni

- ordinare l'insieme di oggetti
- costruire sottoinsiemi di peso minimo
- ...

Ogni diversa implementazione di una struttura corrisponde a:

- un **costo spaziale**, misurato dal **numero di celle che richiede per rappresentare una struttura con n elementi**
- un **costo temporale per ogni operazione**, misurato dal **numero di passi che richiede per eseguirla su una struttura con n elementi**

Vettori: struttura dati astratta

Un **vettore** V di **dimensione** n su un insieme U è definito come una n -upla ordinata (v_1, \dots, v_n) di elementi di U

V associa a ogni intero fra 1 e n un elemento di U

La struttura dati astratta “vettore di dimensione n su U ” è definita come

- l'insieme $\mathcal{V}_{n,U}$ di tutti i possibili vettori di dimensione n su U

$$\mathcal{V}_{n,U} = U^n = U \times \dots \times U$$

- le due operazioni fondamentali di

- 1 **proiezione** $\pi_i(V)$, che associa a un vettore V e un numero $i \in \{1, \dots, n\}$ un elemento di U , generalmente indicato come v_i

$$\pi : U^n \times \{1, \dots, n\} \rightarrow U$$

Restituisce l'elemento di indice dato del vettore dato

- 2 **sostituzione** $\sigma_i(V, u)$, che associa a un vettore V , un numero $i \in \{1, \dots, n\}$ e un elemento $u \in U$ il vettore ottenuto sostituendo v_i con u in V

$$\sigma : U^n \times \{1, \dots, n\} \times U \rightarrow U^n$$

Cambia l'elemento d'indice dato del vettore dato con l'elemento dato

Vettori: implementazione in C

In C **vettore** è realizzato con una **sequenza di celle consecutive** contenente un dato numero N di elementi dello stesso tipo

Il tipo in C determina l'insieme dei possibili valori (ad es., int)

L'indice d'un elemento è la sua posizione nella sequenza: va da 0 a $N-1$ (non da 1 a $N!$)

Si dichiara un vettore specificando

tipo variabile [numero];

- il **tipo** degli elementi: predefinito o definito da utente, semplice o “strutturato” (cioè vettore o record)
- il **numero** degli elementi: **un'espressione costante positiva**

Esempio:

```
#define N 10
int V[N], A[100];
int B[10*N+4];
```

Vettori: costi

Adottiamo il **criterio di costo uniforme**

- ogni elemento di U è rappresentato da un numero $d_U = \log_{|\mathcal{A}|} |U|$ **costante** (superiormente limitato) di simboli di un alfabeto \mathcal{A}

Il **costo spaziale** per un vettore di dimensione n su U è **lineare** ($\Theta(n)$), dato che si usano $n \cdot d_U$ **celle contigue** (n blocchi da d_U)

Il **costo temporale della proiezione** $\pi_i(V)$ è **costante** ($O(1)$):

- **determinazione della prima cella occupata dall'elemento v_i**

$$\text{Ind}(V) + i \cdot d_U$$

dove $\text{Ind}(V)$ è l'indirizzo della prima cella occupata dal vettore V

- **lettura delle d_U celle che rappresentano v_i**

Il **costo temporale della sostituzione** $\sigma_i(V, u)$ è **costante** ($O(1)$):

- **determinazione della prima cella occupata da v_i**
dove $\text{Ind}(V)$ è l'indirizzo della prima cella occupata dal vettore V
- **copia delle d_U celle che contengono u in quelle che contengono v_i**

Con il criterio di costo logaritmico, i costi sono diversi!

Indicizzazione: proiezione e sostituzione

L'operazione di **proiezione** si rappresenta con
il vettore, seguito dall'indice dell'elemento fra parentesi quadre

```
Esempio: #define N 10
          int V[N];
          i = V[4];
```

Per definire vettori con estremi diversi (positivi), si allarga il vettore:
un vettore con estremi S e D tali che $0 \leq S \leq D$ si dichiara con

```
int V[D+1];
```

cioè lasciando i primi S elementi inutilizzati

L'operazione di **sostituzione** combina **parentesi quadre e assegnamento**

```
Esempio: V[3] = 7;
```

In C non c'è controllo che l'indice cada entro l'intervallo dichiarato:
si può scrivere in aree di memoria incontrollate (tipica causa di errori)

Vettori multidimensionali

Un vettore può avere qualsiasi numero di dimensioni, cioè i suoi elementi possono essere identificati da qualsiasi numero di indici

Matrice è un vettore a due o più dimensioni

La dichiarazione specifica il numero di valori per ciascun indice

tipo variabile [numero1] [numero2] [numero3];

Gli elementi dei vettori multidimensionali sono in sequenza lessicografica: sono ordinati prima per righe, poi per colonne; se l'indice di colonna eccede N, si accede alla riga seguente

Per una matrice di M righe, da 0 a M-1, e N colonne, da 0 a N-1:

```
#define M 5
#define N 10
int V[M][N];
```

V[0][0]	...	V[0][9]	V[1][0]	...	V[1][9]	...	V[4][0]	...	V[4][9]
---------	-----	---------	---------	-----	---------	-----	---------	-----	---------

I vettori non si possono copiare con il semplice assegnamento:
vanno copiati elemento per elemento

Sbagliato	Corretto
<pre>#define M 5 #define N 10 int A[M+1][N+1]; int B[M+1][N+1]; B = A;</pre>	<pre>#define M 5 #define N 10 int A[M+1][N+1]; int B[M+1][N+1]; int i, j; for (i = 0; i <= M; i++) for (j = 0; j <= N; j++) B[i][j] = A[i][j];</pre>

Record: struttura dati astratta

Un **record** R sui **campi** U_a è definito come un **insieme finito** $\{x_a\}_{a \in A}$ che associa un elemento x_a dell'insieme U_a a ogni simbolo a di un alfabeto A

R associa a ogni simbolo $a \in A$ un elemento del campo associato U_a

La struttura dati astratta “record sui campi U_a ” è definita come

- la **collezione** $\mathcal{R}_{\{U_a\}}$ degli **insiemi di elementi singoli tratti dagli** U_a
- le due operazioni fondamentali di
 - ① **proiezione** $\pi_a(R)$, che **associa a un record** R e a un simbolo $a \in A$ un **elemento di** U_a , generalmente indicato come $R \cdot a$

$$\pi : \mathcal{R}_{\{U_a\}} \times A \rightarrow U_a$$

Restituisce il campo dato del record dato

- ② **sostituzione** $\sigma_a(R, u)$, che **associa a un record** R , un simbolo $a \in A$ e un elemento $u \in U_a$ il record ottenuto da R sostituendo $R \cdot a$ con u

$$\sigma : \mathcal{R}_{\{U_a\}} \times A \times U_a \rightarrow \mathcal{R}_{\{U_a\}}$$

Cambia il campo dato del record dato con l'elemento dato

Record: implementazione in C

In C **record** o **struttura** è una **sequenza di celle consecutive** contenente **elementi eterogenei accessibili attraverso un nome simbolico (campo)**

Una variabile di tipo struttura si dichiara specificando

- il **tipo** di ciascun campo
- il **nome** di ciascun campo
- il **nome** dell'intera variabile

```
struct {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
} variabile;
```

La dichiarazione ha la solita struttura (*tipo variabile;*)
ma il **tipo** è composto da più parole: `struct {...}`

Dichiarazione di strutture

Usando l'istruzione `typedef` è opportuno separare

- la **dichiarazione della variabile** (`meteo_oggi`)
- la **dichiarazione del tipo** (`dati_meteo`)

```
struct {  
    char data[17+1];  
    float temperatura;  
    int pressione;  
    double umidita;  
} meteo_oggi;
```

```
typedef struct {  
    char data[17+1];  
    float temperatura;  
    int pressione;  
    double umidita;  
} dati_meteo;  
  
dati_meteo meteo_oggi;
```

In questo modo

- si evita di ripetere una lunga dichiarazione per ogni variabile
- **si comunica al compilatore che due variabili sono dello stesso tipo**
(il compilatore non saprebbe riconoscere l'uguaglianza!)

La **proiezione** si rappresenta con
la **struttura seguita da un punto ('.') e dal campo**

```
Esempio: printf("%s\n",meteo_oggi.data);  
          t = meteo_oggi.temperatura;
```

La **sostituzione** combina **l'operatore punto con l'assegnamento**

```
Esempio: meteo_oggi.pressione = 1020;
```

Annidamento (1)

Strutture e vettori possono contenere strutture e vettori ricorsivamente, con qualsiasi numero di livelli

```
typedef struct {
    long id;
    char nome[LUNGHEZZA+1];
    char cognome[LUNGHEZZA+1];
} persona;

typedef struct {
    long matricola;
    persona identita;
} studente;
```

In questo modo è più facile modularizzare il codice

- costruire funzioni che operano su strutture
- combinando funzioni che operano su sottostrutture

Annidamento (2)

Per accedere a un campo di una sottostruttura, si specificano la struttura, la sottostruttura e il campo, separati da punti ('.')

```
strcpy(studente1.identita.nome,"");
```

Per accedere a un elemento di un vettore che è campo di una struttura, si specifica la struttura, il campo vettore e la posizione

```
iniziale1 = studente1.identita.nome[0];  
iniziale2 = studente1.identita.cognome[0];
```

Per accedere a un campo di una struttura elemento di un vettore, si specifica il vettore, la posizione e il campo

```
studente classe[100];  
m = classe[12].matricola;
```

Si può applicare l'operatore di assegnamento (=) a intere strutture

```
dati_meteo meteo_ieri, meteo_oggi;
```

```
meteo_oggi = meteo_ieri;
```

equivale a

```
strcpy(meteo_oggi.data,meteo_ieri.data);
```

```
meteo_oggi.temperatura = meteo_ieri.temperatura;
```

```
meteo_oggi.pressione = meteo_ieri.pressione;
```

```
meteo_oggi.umidita = meteo_ieri.umidita;
```

Copia i campi della struttura a destra in quelli della struttura a sinistra

- copia i campi di tipo elementare
- copia i campi di tipo struttura, sottocampo per sottocampo
- copia i campi di tipo vettore statico (strano!)
- copia i campi di tipo puntatore, ma non duplica l'oggetto puntato (vettori dinamici!); questo può essere molto pericoloso