POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Capitolo 4: Problem-solving con uso di vettori

DAL PROBLEMA AL PROGRAMMA: INTRODUZIONE AL PROBLEM-SOLVING IN LINGUAGGIO C

I vettori nel problem solving

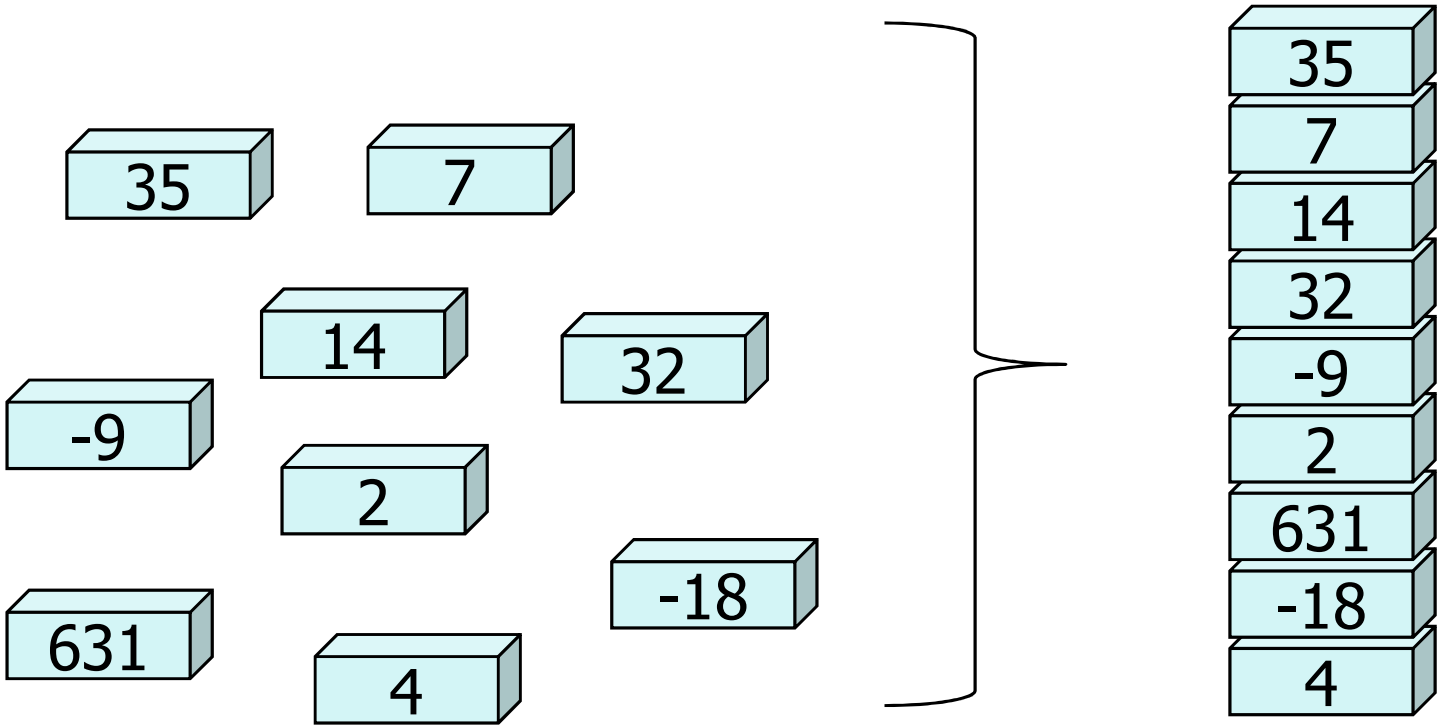
A COSA SERVONO I VETTORI: INSIEMI NON ORDINATI. SEQUENZE ORDINATE, CORRISPONDENZA INDICE-DATO

I vettori nel problem solving

- I vettori sono insieme di dati dello stesso tipo
- Un vettore può essere utilizzato come contenitore di dati (omogenei)
  - Senza alcun criterio di ordine
  - Con un criterio di ordine
  - Sfruttando la corrispondenza dato-indice

Vettore come contenitore (non ordinato)

- Il vettore è un semplice contenitore di dati
- L'ordine non interessa



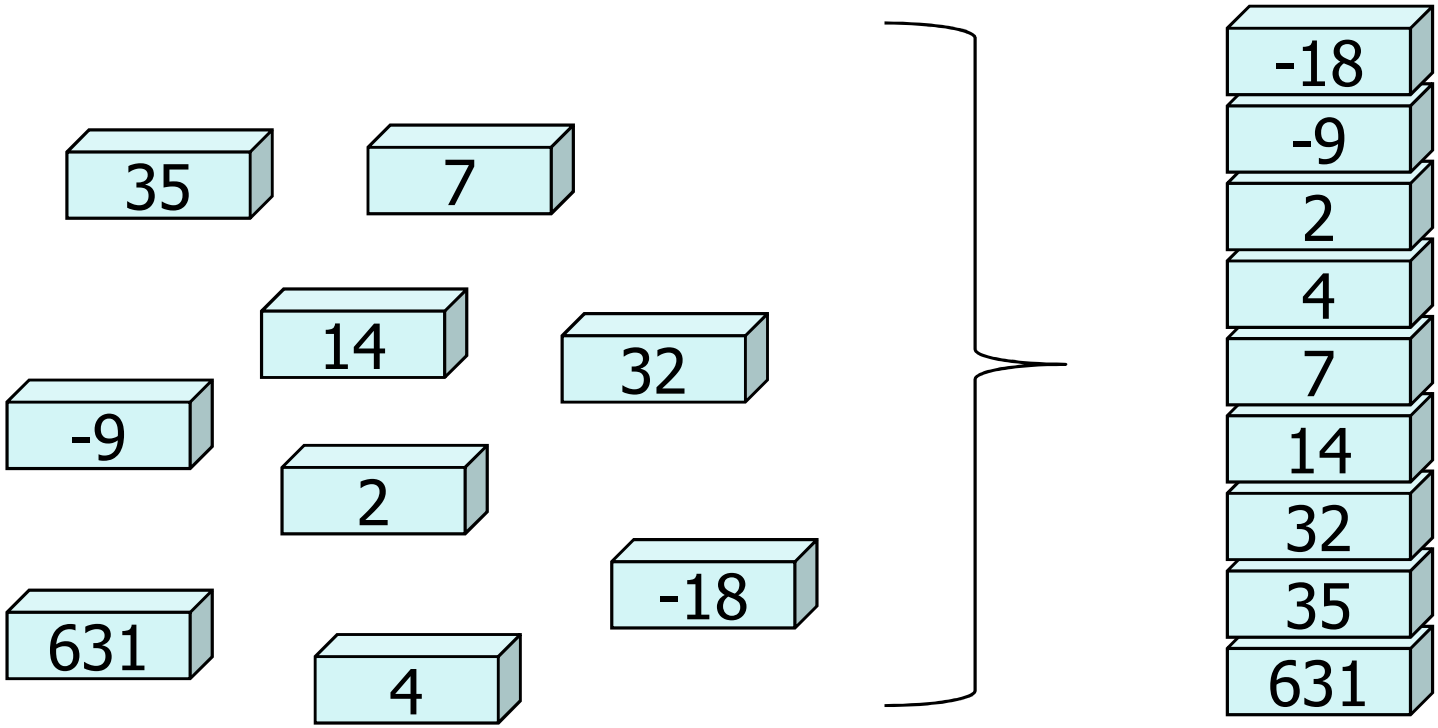
Il vettore come contenitore

- Utilizzo:
  - Problemi in cui è sufficiente raccogliere un insieme di dati, senza relazioni di ordine
    - Un dato può essere in qualunque posizione nel vettore
    - Accesso ai dati mediante generazione (iterativa) di tutti gli indici

    Esempi:
    - Collezionare dati in input, per elaborazioni successive, o preparare dati per l'output
    - Operazioni su insiemi di dati

Vettore ordinato

- I dati nel vettore sono organizzati secondo un criterio di ordine (es. valori crescenti)



Vettore ordinato

Utilizzo:

- Problemi in cui occorre ordinare e/o mantenere ordinato un insieme di dati, secondo un certo criterio
  - Il caso più frequente è il vettore mono-dimensionale: ordine lineare (totale o parziale)
  - L'indice di un dato indica la posizione nell'ordinamento

Esempi:

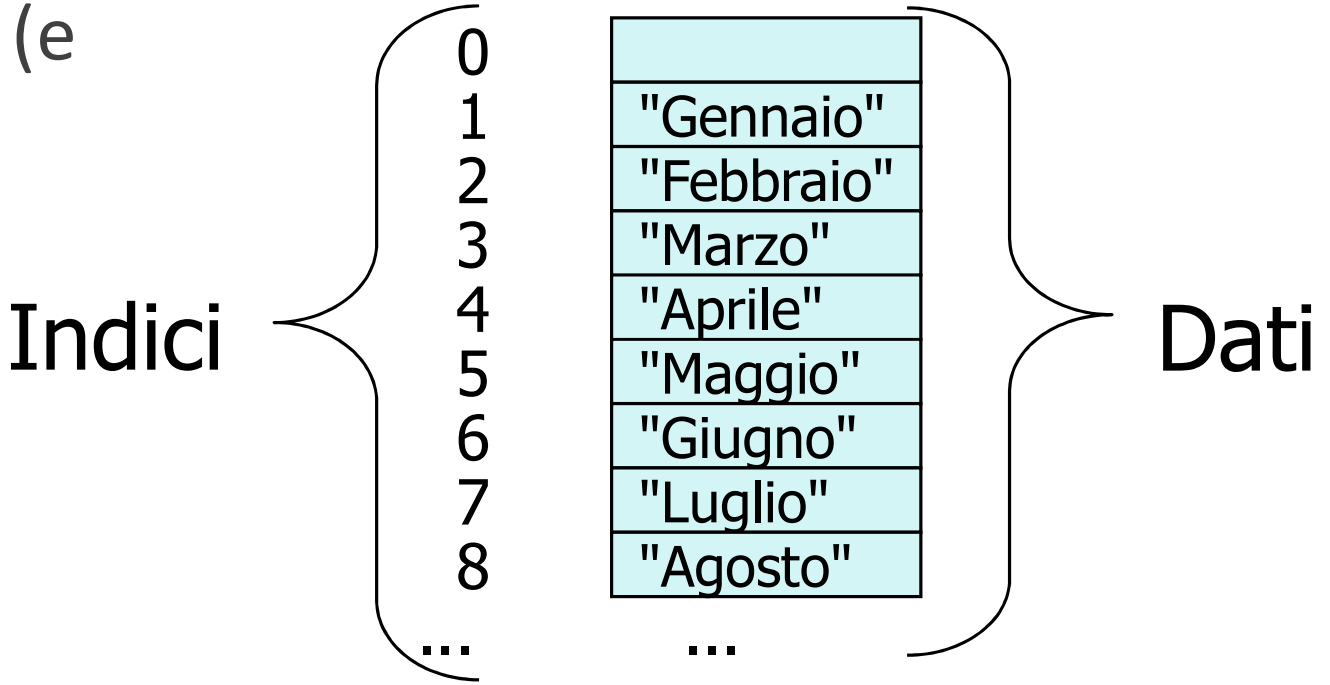
- Ordinare dati mediante criterio cronologico (inverso all'input) o secondo valori (crescenti/decrescenti)
- Sequenze di campioni (numeri) di grandezze fisiche
- Calcoli matematici e/o statistici su successioni di numeri

Corrispondenza indice ↔ dato

Ad ogni indice (intero nell'intervallo 0..NDATI-1) corrisponde un dato (e viceversa):

Corrispondenza indice ↔ dato

Ad ogni indice (intero nell'intervallo 0..NDATI-1) corrisponde un dato (e viceversa):



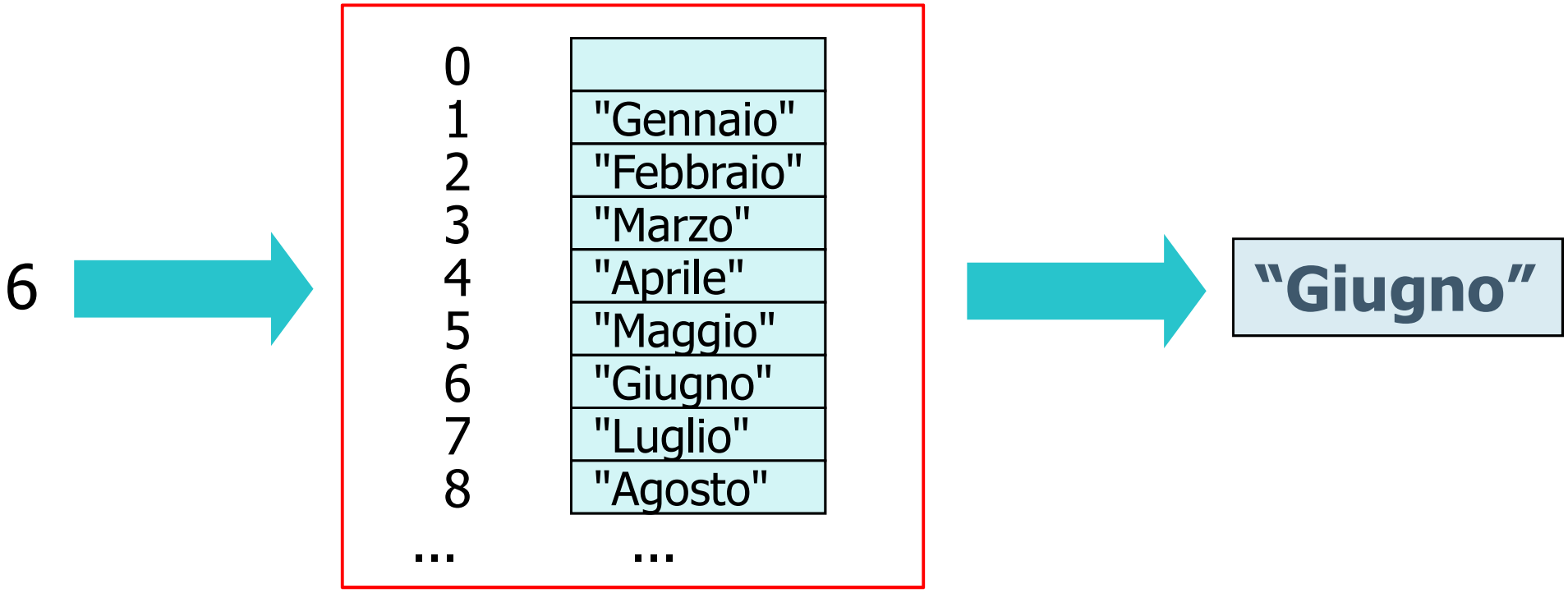
Indici e dati

La relazione indice $\leftrightarrow$ dato:

- Ogni casella di un vettore è caratterizzata da un indice (o più indici, nel caso multi-dimensionale) e da un dato
- La casella del vettore può essere quindi utilizzata per mettere in relazione indice e dato

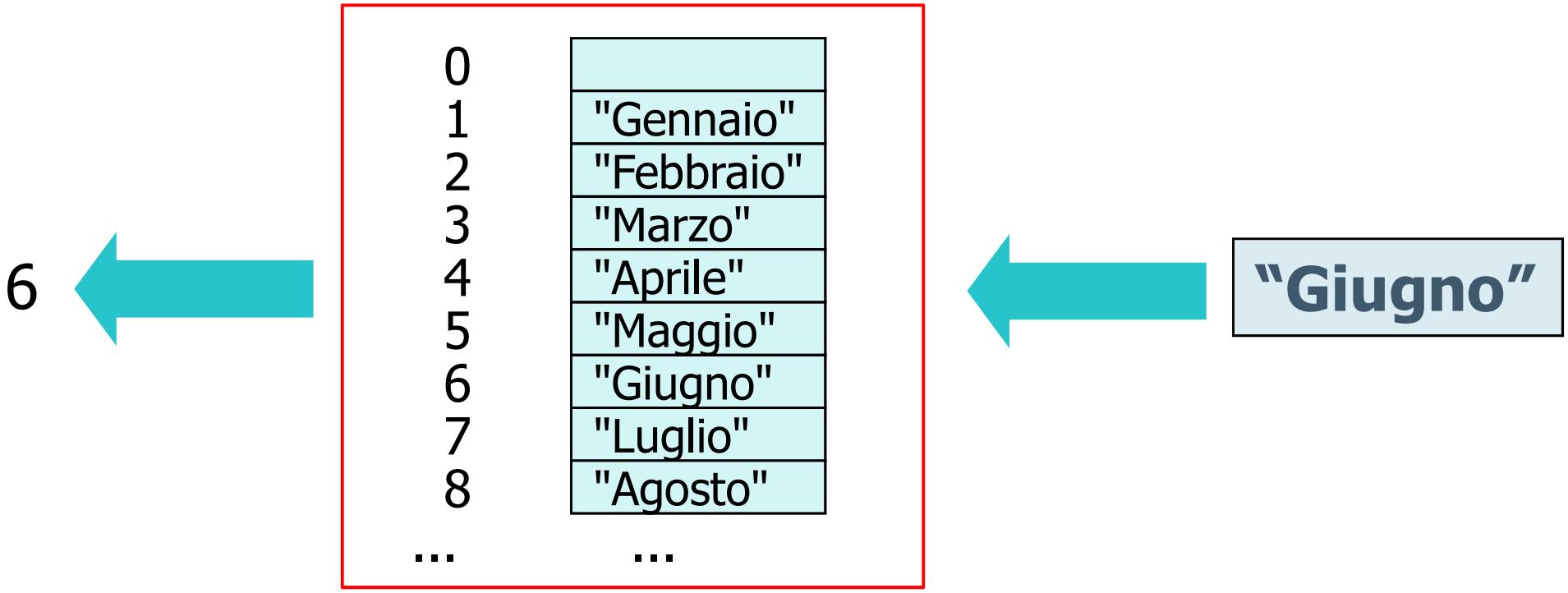
Da indice a dato

A partire dall'indice, calcolare il dato: operazione immediata, grazie al meccanismo di accesso ai vettori



Da dato a indice

A partire dal dato, calcolarne l'indice: operazione non immediata, in quanto occorre cercare il dato



Corrispondenza indice ↔ dato

- Utilizzo:
  - Problemi in cui esistono relazioni/corrispondenze tra numeri interi e dati (numerici o non numerici)
  - L'intero è associato all'indice di una casella, il dato al contenuto
  - Attenzione!
    - L'intero (indice) non può essere troppo grande (non posso allocare memoria illimitatamente)
    - E' possibile che occorra un dato vuoto (nullo) per le caselle non utilizzate

Corrispondenza indice ↔ dato

Esempi:

- Problemi numerici (statistiche e conteggi in relazione a dati interi, tabulazione di funzione nel piano cartesiano y=f(x), con ascissa intera o riconducibile a intero)
- Problemi di codifica/decodifica numerica di informazioni (non numeriche)
- Problemi di elaborazione testi (matrice di caratteri in corrispondenza a pagina da visualizzare)
- Problemi di verifica (flag logici in corrispondenza a dati interi)
- Problemi di selezione (ricerca di indice in corrispondenza a una chiave)

Problemi numerici

PROBLEMI NUMERICI CHE RICHIEDONO USO DI VETTORI

Problemi numerici

- Problemi di algebra, geometria, statistica, ecc., simili a quelli risolti mediante dati scalari
- I vettori possono essere utilizzati per:
  - Collezionare e manipolare (insiemi di) numeri
  - Rappresentare dati con struttura lineare (vettori) o tabellare (matrici)
  - Gestire corrispondenze tra numeri (indice $\leftrightarrow$ dato)

Problemi su insiemi di numeri

- Problemi nei quali si gestiscono insiemi (gruppi) di dati numerici, con operazioni di I/O, unione, intersezione, … NON CONTA L'ORDINE
  - Operazioni tipiche degli insiemi
- La soluzione spesso si basa su costrutti iterativi (eventualmente annidati) tali da:
  - Percorrere gli elementi di un insieme
  - Percorrere, per ogni elemento di un insieme, tutti gli elementi di un altro insieme

Intersezione tra insiemi di numeri

Formulazione:

- Acquisire da tastiera un primo gruppo di 10 interi
  - Privo di numeri ripetuti
- Acquisire da tastiera un secondo gruppo di 10 interi
  - Privo di numeri ripetuti
- Calcolare e visualizzare tutti i numeri che sono presenti in entrambi i gruppi

Soluzione:

- Si tratta di un problema di **intersezione tra insiemi**, che richiede di determinare, per ogni elemento del primo insieme, se appartiene anche al secondo

Intersezione tra insiemi di numeri

Algoritmo:

- Input: due iterazioni per acquisire i dati dei due gruppi
- Elaborazioni: doppia iterazione (annidata) per confrontare tutti i dati del primo gruppo con tutti quelli del secondo (o viceversa)
- Output: iterazione sui dati dell'insieme intersezione

Le tre parti possono essere distinte, oppure (parzialmente) integrate: mentre si esegue l'input si calcola l'intersezione e si scrivono i risultati in output

- Memorizzo tutto il primo insieme ma non il secondo

Intersezione tra insiemi di numeri

Struttura dati:

La scelta dipende dall'algoritmo adottato:

- E' necessario almeno un vettore per i dati del primo insieme
- L'utilizzo di un secondo vettore dipende dal fatto di adottare uno schema
  1. Tre fasi separate: input, elaborazioni, output → **serve il secondo vettore**
  2. Elaborazioni ed output fatte durante l'input del secondo gruppo (per ogni numero del secondo gruppo, si determina direttamente se appartiene anche al primo e si fornisce il relativo output) → **non serve il secondo vettore**
- La soluzione proposta utilizza due vettori, con tre fasi separate: il risultato (intersezione) viene sovrapposto al primo vettore

Codice

```
#define NDATI 10
#include <stdio.h>
void leggiVettore (int dati[], int n);
void scriviVettore (int dati[], int n);

int main(void){
  int dati0[NDATI], dati1[NDATI];
  int i, j, ni, trovato;
  /* input */
  leggiVettore(dati0,NDATI);
  leggiVettore(dati1,NDATI);
  /* calcolo intersezione */
  for (i=ni=0; i<NDATI; i++) {
    trovato=0;
    for (j=0; j<NDATI&&(!trovato); j++) {
      if (dati0[i]==dati1[j])
        trovato=1;
    }
    /* se dato appartiene ad intersezione
       riscrivi nella parte iniziale del
       vettore dati0 (già confrontato) */
    if (trovato)
      dati0[ni++]=dati0[i];
  }
  /* output */
  scriviVettore(dati0,ni);
}
```

- **Iterazione esterna:** considero un elemento del primo vettore dati0[i]
- **Iterazione interna:** considero uno alla volta gli elementi del secondo vettore dati1[j]
  - Appena trovo un elemento del secondo vettore uguale a dati0[i] **interrompo la ricerca** usando il flag `trovato`

```
int dati0[NDATI], dati1[NDATI];
int i, j, ni, trovato;
/* input */
leggiVettore(dati0,NDATI);
leggiVettore(dati1,NDATI);
/* calcolo intersezione */
```

```
  for (i=ni=0; i<NDATI; i++) {
    trovato=0;
    for (j=0; j<NDATI&&(!trovato); j++) {
      if (dati0[i]==dati1[j])
        trovato=1;
    }
    /* se dato appartiene ad intersezione
       riscrivi nella parte iniziale del
       vettore dati0 (già confrontato) */
    if (trovato)
      dati0[ni++]=dati0[i];
  }
  /* output */
  scriviVettore(dati0,ni);
}
```

Codice

```
void leggiVettore (int dati[], int n) {
  int i;
  printf("Scrivi %d numeri interi (separati da spazio o a-capo):\n", n);
  for (i=0; i<n; i++) {
    scanf("%d", &dati[i]);
  }
}

void scriviVettore (int dati[], int n);
  int i;
  printf("I numeri sono:\n", n);
  for (i=0; i<n; i++) {
    printf("%d ", dati[i]);
  }
  printf("\n");
}
```

Problemi su sequenze di numeri

- I problemi interessano sequenze di dati (**ordinati**) che debbono essere immagazzinati in un vettore prima di venir elaborati, perchè:
  - E' necessario attendere l'ultimo dato prima di poter elaborare i dati
  - Oppure sono necessarie elaborazioni (ripetute) su tutti i dati
- Non si possono trattare sequenze infinite, ma insiemi (ordinati) finiti di dati, organizzati in vettori mono- o multi-dimensionali (matrici)

Normalizzazione di dati

Formulazione: scrivere una funzione C che:

- Acquisisca da file testo una sequenza di dati reali separati da spazi o da a-capo:
  - Il numero N di dati non è noto a priori ma *può essere sovradimensionato* (valore massimo 1000)
- Determini, per l'i-esimo dato d_i ($0 \leq i < N$), le medie dei dati predecedenti (p_i) e dei successivi (s_i)
- Scriva su un secondo file i dati, normalizzati secondo la seguente regola:
  - Ogni dato (d_i) viene sostituito dalla media aritmetica tra d_i, p_i e s_i
    - Cioè la media tra il dato stesso, la media dei precedenti e la media dei successivi
- I nomi dei due file siano ricevuti come parametri

Normalizzazione di dati

Soluzione:

- Occorre calcolare, in modo iterativo, le medie
- Quindi, con una successiva iterazione, le normalizzazioni dei dati

Struttura dati:

- Un vettore per acquisire i dati dal file
- Un vettore per calcolare le medie

NOTA: Si potrebbero evitare i vettori rileggendo più volte il file

- Soluzione suggerita come esercizio, ma sconsigliata: in genere meglio evitare di leggere un file molte volte

Normalizzazione di dati

Algoritmo 1: algoritmo $O(N^2)$

- Input: vettore, riempito mediante lettura iterativa
- Elaborazioni: per ogni dato
  - Calcola la media dei predecessori e dei successori (funzione avg)
  - Fa la loro media in un altro vettore (quello originale deve essere mantenuto per calcolare correttamente le medie)
  - Il predecessore del primo dato e il successore dell'ultimo mancano – ci potrebbero essere due strategie:
    a) estrapolare un valore, ad es. 0 (la strategia usata) oppure un valore uguale al primo/ultimo
    b) non considerare il dato e quindi fare solo la media tra 2 dati
- Output del vettore ricalcolato

Normalizzazione di dati

Algoritmo 1: algoritmo $O(N^2)$

- Input: vettore, riempito med… lettura iterativa
- Elaborazioni: per ogni e…
  - Calcola la media dei predecessor…
  - Fa la loro media in un altro vetto… calcolare correttamente le medi…
  - Il predecessore del primo dato e… essere due strategie:
    a) estrapolare un valore, ad es. primo/ultimo
    b) non considerare il dato e qui…
- Output del vettore rica…

Analisi di complessità: previsione delle risorse (memoria, tempo) richieste dall'algoritmo per la sua esecuzione rispetto alla dimensione N del problema

- **Concetto approfondito con il Prof. Camurati domani**
- Dato N numero massimo di elementi, il programma impiega memoria e tempo proporzionali al quadrato di N

Normalizzazione di dati

Esempio: si supponga di ricevere i seguenti 6 valori:

- 4.0 5.0 3.0 2.0 1.0 7.0

Il risultato sarà:

- dati[0] = (4.0 + 0.0 + (18.0/5)) /3 = 2.53
- dati[1] = (5.0 + 4.0 + (13.0/4)) /3 = 4.08
- dati[2] = (3.0 + (9.0/2) + (10.0/3)) /3 = 3.61
- dati[3] = (2.0 + (12.0/3) + (8.0/2)) /3 = 3.33
- dati[4] = (1.0 + (14.0/4) + 7.0) /3 = 3.83
- dati[5] = (7.0 + (15.0/5) + 0.0) /3 = 3.33

LEGENDA:
- dati d_i
- predecedenti p_i
- successivi (s_i)

Codice

```
// read data from file filename and store
// it in array d, return number or read
// values i
int readFile(char fileName[], float d[],
             int nmax) {
  int i;
  FILE *fp;
  fp = fopen(fileName,"r");
  if (fp==NULL)
    return 0;
  for (i=0; i<nmax; i++) {
    fscanf(fp, "%f", &d[i]);
  }
  return i;
}
```

```
// write data contained in d up to index n
// to file fileName
void writeFile(char fileName[], float d[],
               int n) {
  int i;
  FILE *fp;
  fp = fopen(fileName,"w");
  if (fp==NULL)
    return;
  for (i=0; i<n; i++) {
    fprintf(fp, "%f ", dati[i]);
  }
  fclose(fp);
}
```

Codice

```
#define NMAX 100

/* version 1: O(N^2) */
void normalizeNum(char nfin[],char nfout[]){
  float data[NMAX], dataNew[NMAX];
  int i, j, N;
  N = readFile(nfin,data,NMAX); /* input */
  for (i=0; i<N; i++) {
    float pred = avg(data,0,i-1);
    float succ = avg(data,i+1,N-1);
    dataNew[i] = (pred+data[i]+succ)/3;
  }
  writeFile(nfout,dati,N); /* output */
}
```

```
// returns the average of values from
// index i0 to index i1 (included)
float avg(float d[], int i0, int i1) {
  int i;
  float sum;
  if (i0>i1) {
    return 0.0; // no data: assume 0
  }
  sum = 0.0;
  for (i=i0; i<=i1; i++) {
    sum = sum + d[i];
  }
  return sum/(i1-i0+1);
}
```

Codice

```
#define NMAX 100

/* version 1: C
void normalizeNum(c      nfin[],char nfout[]){
  float data[NMAX], dataNew[NMAX];
  int i, i
  N =                    AX); /* input */
  for
    fl                   ,i-1);
    float succ = avg(data,i+1,N-1);
    dataNew[i] = (pred+data[i]+succ)/3;
  }
  writeFile(nfout,dati,N); /* output */
}
```

Nuovi valori da scrivere nel file

Media degli elementi fino all'i-esimo

```
// returns the average of values from
// index i0 to index i1 (included)
float avg(float d[], int i0, int i1) {
  int i;
  float sum;
  if (i0>i1) {
    return 0.0; // no data: assume 0
  }
  sum = 0.0;
  for (i=i0; i<=i1; i++) {
    sum = sum + d[i];
  }
  return sum/(i1-i0+1);
}
```

Normalizzazione di dati

Algoritmo 2: O(N)

- Input: vettore riempito mediante lettura iterativa
- Elaborazioni: è sufficiente calcolare:
  - La sommatoria di tutti dati (STOT)
  - Per ogni dato, la somma di se stesso e dei predecessori (sum_i)
  - A partire da questa è possibile calcolare le medie di predecessori e successivi:
    - $p_i = sum_{i-1}/i$,
    - $s_i = (STOT-sum_i)/(N-i-1)$
    - I valori normalizzati (sostituendo i dati sul vettore iniziale)
- Output del vettore ricalcolato

Normalizzazione di dati

Algoritmo 2: O(N)

- input: vettore [illegible]mpito mediante lettura iterativa
- elaboraz[illegible]
  - la somma[illegible]
  - per ogni dato, la somma di se stesso e dei predecessori (sum_i). A partire da questa è possibile calcolare:
    $p_i = sum_i/i$,
    $s_i = (STOT-sum_i)/(N-i-1)$
  - i valori normalizzati (sostituendo i dati sul vettore iniziale):
- output del vettore ricalcolato.

- Dato N numero massimo di elementi, il programma impiega memoria e tempo proporzionali a N

Normalizzazione di dati

Esempio:

si supponga di ricevere i seguenti 6 valori:

- 4.0 5.0 3.0 2.0 1.0 7.0

Il vettore somme conterrà:

- 4.0 9.0 12.0 14.0 15.0 22.0

STOT = 22.0

Il risultato sarà:

- dati[0] = (4.0 + 0.0 + (22.0-4.0)/5)/3 = 2.53
- dati[1] = (5.0 + 4.0/1 + (22.0-9.0)/4)/3 = 4.08
- dati[2] = (3.0 + 9.0/2 + (22.0-12.0)/3)/3 = 3.61
- dati[3] = (2.0 + 12.0/3 + (22.0-14.0)/2)/3 = 3.33
- dati[4] = (1.0 + 14.0/4 + (22.0-15.0)/1)/3 = 3.83
- dati[5] = (7.0 + 15.0/5 + 0.0)/3 = 3.33

LEGENDA:
- dati d_i
- predecedenti p_i
- successivi (s_i)

Codice

```
#define NMAX 1000
/* version 2: O(N) */
void normalizeNum(char nfin[],
                  char nfout[]){
  float data[NMAX], sum[NMAX];
  int i, j, N, STOT;
  /* input */
  N = readFile(nfin,dati,NMAX); /*input*/
  /* partial sums */
  sum[0]=data[0];
  for (i=1; i<N; i++)
    sum[i] = sum[i-1]+data[i];
  STOT = sum[N-1];
  /* sum of all numbers */
  ...
```

```
 /* normalize */
  data[0] = (data[0] + (STOT-data[0])/(N-1))/3;
  for (i=1; i<N-1; i++) {
    float pred = sum[i-1]/i;
    float succ = (STOT-sum[i])/(N-i-1);
    data[i] = (pred+data[i]+succ)/3;
  }
  data[N-1] = (data[N-1] + sum[N-2]/(N-1))/3;

  writeFile(nfout,dati,N); /* output */
}
```

Problemi su statistiche per gruppi

- I problemi sono caratterizzati dalla suddivisione dei numeri in classi/gruppi numerabili ed identificabili da un intero
- La raccolta di conteggi o dati statistici può essere effettuata sulle caselle di un vettore
  - A ogni classe o gruppo corrisponde un indice (e una casella del vettore)

Suddivisione in classi

Formulazione: scrivere una funzione C che:

- Riceva come parametro un vettore di interi di valore compreso tra 0 e 100 (il secondo parametro indica la dimensione del vettore)
- Raggruppi gli interi in decine, calcoli e visualizzi i conteggi dei numeri appartenenti a ciascuna decina

Suddivisione in classi

Esempio:

Si supponga di ricevere i seguenti 20 valori:

3	6	9	16	22	23	30	32	40	48
65	78	7	8	10	15	25	90	27	26

I numeri saranno raggruppati come segue:

- (3, 6, 9, 7, 8), (16, 10, 15), (22, 23, 25, 26, 27), (30, 32), (40, 48), (-), (65), (78), (-), (90), (-)

I conteggi visualizzati saranno:

- 5, 3, 5, 2, 2, 0, 1, 1, 0, 1, 0

Suddivisione in classi

- Soluzione:
  - Si potrebbe fare una doppia iterazione (per ogni decina, interare su tutti i numeri e contare quelli appartenenti alla decina): il costo sarebbe `O(decine*numeri)`
  - Più efficiente (`O(numeri)`): utilizzare un vettore di contatori, sfruttando la corrispondenza indice-dato:
    - Per ogni dato in ingresso si calcola l'indice corrispondente alla decina di appartenenza, e si aggiorna il relativo contatore
- Struttura dati:
  - Un vettore di interi come parametro alla funzione
  - Un vettore di contatori (con indici da 0 a 10)

Suddivisione in classi

Algoritmo:

- Azzeramento dei contatori, per ogni decina
  - Le decine sono numerate da 0 a 10, ho 11 decine in totale
- Iterazione sui dati interi. Per ognuno:
  - Si calcola la decina di appartenenza con una divisione intera
    `d = dati[i]/10;`
  - Si accede al vettore dei conteggi (utilizzando la decina come indice) incrementando il contenuto
- Iterazione sui contatori per visualizzare i conteggi

Codice

```
void contaPerDecine (int dati[], int n){
  int i, d, conta[11];
  for (i=0; i<=10; i++)
    conta[i]=0;
  for (i=0; i<n; i++) {
    d = dati[i]/10;
    conta[d] = conta[d]+1;
  }
  for (i=0; i<=10; i++)
    printf ("%d dati in decina %d \n",
            conta[i], i+1);
}
```

Codice

```
void contaPerDecine (int dati[], int n){
  int i, d, conta[11];
  for (i=0; i<=10; i++)
    conta[i]=0:
  for (i=0; i<n; i++) {
    d = dati[i]/10;
    conta[d] = conta[d]+1;
  }
  for (i=0; i<=10; i++)
    printf ("%d dati in decina %d \n",
            conta[i], i+1);
}
```

Conosco l'**indirizzo** del (primo elemento di) dati, ma non so **quanti elementi contiene** → devo ricevere come parametro il numero di elementi n per fare iterazioni sugli elementi del vettore

Problemi di codifica

PROBLEMI DI CODIFICA DI NUMERI E TESTI

Problemi di codifica di numeri

Nei problemi di codifica di numeri, i vettori possono essere utilizzati per:

- immagazzinare le cifre in una data codifica
- manipolare i numeri, lavorando a livello di codifica in cifre.

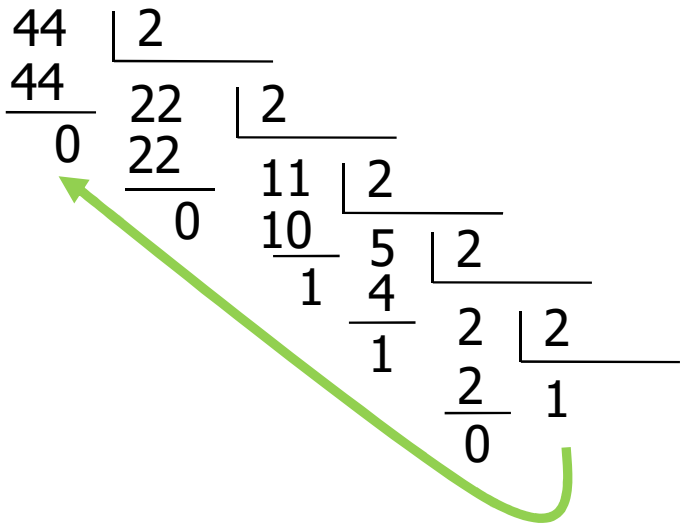
Codifica binaria di un intero

Formulazione: realizzare una funzione C che, ricevuto come parametro un intero ($0 <= n <= 2^{32} -1$), ne determini la codifica binaria e visualizzi i bit

Soluzione:

- Costruzione iterativa dei bit, a partire dai meno significativi
  - Divisioni successive per 2, i resti delle divisioni sono i bit della codifica
  - Classico algoritmo di codifica visto a Informatica

Codifica binaria di un intero



$44_{10} = 101100_2$

Vettore con codificazione su 32 bit:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0

Codifica binaria di un intero

Struttura dati:

- Un parametro formale (intero): n
- Un vettore (bit) per i bit della codifica

Algoritmo:

- Iterazione per generare, in codifica binaria, i bit, a partire dai meno significativi (al massimo 32 bit)
- Iterazione per visualizzare i bit (dal più significativo)

Codice

```
void binarioVettore (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Codice

```
void binarioVettore (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Calcola bit meno significativo

Codice

```
void binarioVettore (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Continua finchè n non è 0
Se inizialmente n=0 calcola 1 bit