Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

11 Ottobre 2022

Algoritmi e diagrammi di flusso: strutture dati bidimensionali

Finora abbiamo considerato problemi di ricerca e verifica di proprietà all'interno di una sequenza *lineare* o *monodimensionale* di elementi.

In molti ambiti si ha a che fare con *dati strutturati in piú dimensioni*, e.g. matrici, tabelle, etc.

Quindi è necessario definire e manipolare strutture dati piú complesse, applicando su di esse ricerche e verifiche simili a quelle già viste per il caso monodimensionale.

In questo corso tratteremo solo *strutture dati bidimensionali*, ma il ragionamento può essere esteso in modo simile al caso generale *multidimensionale*.

Sequenze di sequenze: operazioni elementari

Supponiamo che, oltre alle operazioni elementari necessarie per operare su sequenze lineari, la macchina astratta fornisca le operazioni elementari per manipolare *sequenze di sequenze*, e.g.

- selezione della prima sequenza,
- selezione della sequenza successiva a quella in esame,
- test di fine sequenza di sequenze, etc.

senza dare molti dettagli al riguardo.

Non appena passeremo al linguaggio di programmazione in cui codificare i nostri algoritmi, avremo a disposizione costrutti precisi per implementare le operazioni elementari necessarie.

Sequenze di sequenze: assunzioni

Data una sequenza S di sequenze di elementi $s_1\ s_2\ \ldots\ s_n$, abbiamo le seguenti assunzioni:

- S è finita, i.e. costituita da un numero finito di sequenze lineari s_i;
- ogni sequenza lineare s_i (riferita anche come *riga* per la similarità con le matrici) in S è finita, non vuota e terminata da un opportuno marcatore di fine sequenza;
- esiste un marcatore di fine sequenza di sequenze, denotato $\{\}$;
- S è non vuota, i.e. esiste *almeno una sequenza* in S prima del marcatore $\{\}$.

Nel seguito estendiamo alle sequenze di sequenze i problemi e gli algoritmi considerati per le sequenze lineari.

Contare le occorrenze di un elemento

Esercizio I. Dati un intero x ed una sequenza di sequenze di interi S, calcolare il numero delle occorrenze di x in S.

Input: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$ con $s_i = x_1\ x_2\ \ldots\ x_{n_i}\ 0$ ed $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n_i$
Output: numero di occorrenze di x in S

Come nel caso di una sequenza lineare, occorre *scorrere tutta la sequenza* S e registrare il numero di occorrenze di x in un *contatore*.

Un algoritmo consiste nel ripetere le seguenti operazioni fino a quando non viene raggiunto il marcatore di fine sequenza $\{\}$:

- selezionare una riga (a partire dalla prima riga di S),
- scorrerla tutta alla ricerca di elementi uguali ad x,
- passare alla riga successiva.

Contare le occorrenze di un elemento (cont.)

Il diagramma di flusso risultante contiene due blocchi di iterazione, uno *annidato* dentro l'altro.

Il ciclo esterno itera le operazioni sulle righe di S, mentre il ciclo interno itera la ricerca sugli elementi della sequenza s_i selezionata (vedere diagramma Es. I, dove seq denota la sequenza s_i) .

In modo simile possono essere dati gli algoritmi ed i relativi diagrammi di flusso per i problemi in cui sia richiesto di calcolare:

- quanti elementi di una sequenza di sequenze S soddisfano una data proprietà,
- la somma o il prodotto o la concatenazione degli elementi di S,
- l'elemento massimo o minimo in S, etc.

Verificare l'occorrenza di un elemento

Esercizio II. Dati un intero x ed una sequenza di sequenze di interi S, restituire *true* se x occorre in S, altrimenti restituire *false*.

Input: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$ con $s_i = x_1\ x_2\ \ldots\ x_{n_i}\ 0$ ed $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n_i$
Output: *true* se x occorre in S, *false* altrimenti

Il diagramma dato per lo stesso problema nel caso di una sequenza lineare (Es. 8) viene esteso con i blocchi necessari per la scansione di una struttura bidimensionale (vedere diagramma Es. II).

N.B. *Non appena* viene trovata la prima occorrenza di x in una qualche riga di S, si termina restituendo *true* (il confronto `trovato = false` compare nella condizione booleana di *entrambi* i cicli).

Si restituisce *false* se sono state esaminate tutte le sequenze in S senza aver trovato x.

Verificare l'occorrenza di un elemento in ogni riga

Esercizio III. Dati un intero x ed una sequenza di sequenze di interi S, restituire *true* se x occorre in *ogni* sequenza s_i di S, altrimenti restituire *false*.

Input: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$ con $s_i = x_1\ x_2\ \ldots\ x_{n_i}\ 0$ ed $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n_i$
Output: *true* se x occorre in s_i *per ogni* $i = 1, \ldots, n$,
false altrimenti

Nell'esercizio precedente la proprietà da verificare può essere vista come una proprietà *globale* rispetto ad S (è sufficiente verificare che x compaia in *una qualche* sequenza di S).

Qui invece si richiede che questa proprietà sia verificata *localmente per ogni sequenza* di S.

Verificare l'occorrenza di un elemento in ogni riga (cont.)

Si consideri il diagramma di flusso per l'Es. III.

Affinché sia restituito *true* occorre esaminare *tutte* le sequenze di S e verificare che ogni s_i soddisfi la proprietà in questione, ovvero avere *almeno* un'occorrenza di x (locazione booleana `ok`).

N.B. *Non appena* viene trovata tale occorrenza all'interno di una sequenza, si può smettere di scorrere la sequenza in esame e passare a considerare quella successiva.

L'algoritmo invece termina restituendo *false non appena* si trova una sequenza s_i che non ha alcuna occorrenza di x al suo interno, in quanto la condizione "per ogni sequenza" non può essere soddisfatta.

Verificare l'occorrenza di un elemento in ogni riga (cont.)

L'algoritmo proposto utilizza due locazioni di tipo booleano:
- `ok` che, se vale *true*, significa che tutte le sequenze già esaminate soddisfano la proprietà richiesta;
- `trovato` che, se vale *true*, indica che è stata trovata un'occorrenza di x nella sequenza in esame.

N.B. Il valore di `trovato` deve essere rimesso a *false* prima della scansione di ogni sequenza s_i fatta nel blocco di iterazione interno.

All'uscita da tale blocco occorre distinguere per quale motivo il ciclo interno è terminato:
- se `trovato` vale *true*, s_i soddisfa la proprietà e quindi si può passare a considerare la sequenza successiva,
- altrimenti si ha che s_i è stata visitata fino alla fine senza trovare alcuna x, per cui `ok` viene messo a *false* in modo da uscire subito dal blocco di iterazione esterno e terminare l'intero algoritmo.

Verificare l'occorrenza di un elemento in ogni riga (cont.)

Esempio: verificare il risultato restituito dall'esecuzione dell'algoritmo dato con $x = 3$ e la sequenza di sequenze S

```
1 4 3 3 0
5 -3 3 0
3 -1 5 3 7 0
{}
```

e successivamente con la stessa sequenza S ed $x = 5$.

Verificare almeno k elementi in ogni riga

Esercizio IV. Dati una sequenza di sequenze di interi S ed un intero $k > 0$, restituire *true* se in *ogni* sequenza s_i di S *esistono almeno* k elementi che soddisfano una data proprietà P, altrimenti restituire *false*.

Input: $S = s_1\ s_2\ \ldots\ s_n\ \{\}$, $k > 0$ con $s_i = x_1\ x_2\ \ldots\ x_{n_i}\ 0$ ed $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n_i$
Output: *true* se *ogni* s_i $(i = 1, \ldots, n)$ ha *almeno* k elementi che soddisfano P, *false* altrimenti

Anche in questo caso occorre verificare *localmente* una proprietà, ovvero ogni sequenza s_i di S deve soddisfare la condizione che s_i abbia almeno k elementi per i quali sia vera una data proprietà P.

Verificare almeno k elementi in ogni riga (cont.)

Si consideri il diagramma di flusso per l'Es. IV.

Ad esempio, dati $P(n) =_{def} pari(n)$, $k = 2$ e la sequenza di sequenze S

```
3 4 -2 6 0
7 -10 8 0
4 5 21 -8 10 0
{}
```

l'algoritmo restituisce *true*, poiché ogni riga di S ha almeno 2 numeri pari.

Esercizi

Dare un algoritmo ed il relativo diagramma di flusso per i seguenti problemi.

1. Dati due numeri $m, n \in \mathbb{N}$ tali che $m < n$, stampare tutti i numeri pari compresi tra m ed n, ovvero tutti quei numeri x tali che x è pari e $m \leq x \leq n$.
2. Dato $n \in \mathbb{N}$, calcolare il fattoriale di n, definito come segue:

   $0! = 1$
   $n! = n \times (n-1) \times \ldots \times 2 \times 1$ per $n \geq 1$.
3. Dato $n \in \mathbb{N}$, calcolare l'n-esimo numero di Fibonacci, definito come segue:

   $fib(0) = 1$
   $fib(1) = 1$
   $fib(n) = fib(n-1) + fib(n-2)$ per $n \geq 2$.