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

Monica Nesi

Università degli Studi dell'Aquila

5 Ottobre 2022

Algoritmi e diagrammi di flusso: contare elementi aventi una data proprietà

Esercizio 5. Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0), *contare* quanti elementi della sequenza hanno o soddisfano una determinata proprietà P.

Esempi: calcolare quanti sono i numeri negativi, oppure quanti sono i numeri pari, oppure i numeri divisibili per un dato numero k, etc.

Input: $x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: numero degli elementi x_i per i quali vale $P(x_i)$

Anche per questo problema è necessario *scorrere tutta la sequenza*.

Inoltre occorre *contare* quegli elementi che soddisfano la proprietà P data.

Contatori ed Accumulatori

Serve una locazione di memoria di tipo intero, detta *contatore*.

Un contatore è un particolare *accumulatore* (i.e. una locazione in cui si memorizzano i risultati parziali della computazione).

Un contatore viene *inizializzato* a 0.

Il suo valore è aggiornato incrementandolo di un'unità solo quando l'elemento in esame soddisfa la data proprietà P.

Il risultato sarà il valore finale del contatore.

Stesse operazioni elementari, oltre alle operazioni aritmetiche su $\mathbb{Z}$ (in particolare, quella di incremento di un'unità).

N.B. Il diagramma di flusso ha una struttura simile a quello del problema del massimo.

Occorrenze di un elemento in una sequenza

Esercizio 6. Dati un intero x ed una sequenza finita (non vuota) di numeri interi (terminata da uno 0), calcolare il numero delle occorrenze di x, ovvero quante volte x compare nella sequenza.

Input: $x \in \mathbb{Z}$, $\quad x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: numero di occorrenze di x nella sequenza

Caso particolare del problema generale precedente, dove $P(n)$ è vera se e solo se $n = x$.

Il diagramma di flusso è un caso particolare del diagramma precedente, in cui `P(n)` è istanziato con `n=x`.

Esempio: dati `x=4` e la sequenza `3 4 -1 5 4 4 -2 0`, il risultato calcolato eseguendo le operazioni nel diagramma di flusso è 3.

Sommatoria degli elementi in una sequenza

Esercizio 7. Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0), calcolare la somma dei numeri nella sequenza.

Input: $x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: $x_1 + x_2 + \ldots + x_n$

Anche per questo problema è necessario scorrere tutta la sequenza.

Ed occorre accumulare il risultato in una locazione di memoria, in cui all'i-esimo passo è memorizzato il valore della somma parziale $x_1 + x_2 + \ldots + x_i$ relativa alla porzione di sequenza già visitata.

L'accumulatore viene inizializzato a 0 (identità per la somma).

Per essere sommati gli elementi non devono soddisfare alcuna proprietà.

Esempio: data la sequenza `3 -1 4 7 5 0`, il risultato è 18.

Produttoria di interi

Variante del problema precedente:
calcolare il prodotto $x_1 * x_2 * \ldots * x_n$ dei numeri nella sequenza.

Basta modificare alcune istruzioni nel diagramma:
l'accumulatore viene inizializzato ad 1 (identità per il prodotto) e viene modificato assegnandogli il valore dell'espressione $s * n$.

Problemi con sequenze di stringhe

Esercizio: data una sequenza finita (non vuota) di stringhe $s_1\ s_2\ \ldots\ s_n\ \epsilon$, calcolare la stringa risultante dalla concatenazione delle stringhe nella sequenza.

L'algoritmo è lo stesso della sommatoria: basta inizializzare l'accumulatore s (che ora deve essere una stringa) con la stringa vuota ϵ (identità per la concatenazione)
e poi ad ogni passo all'accumulatore viene assegnato il risultato della concatenazione $s\,n$ tra il valore di s e la stringa n in considerazione.

Esercizio: dati una sequenza finita (non vuota) di stringhe $s_1\ s_2\ \ldots\ s_n\ \epsilon$ ed un intero $k>0$, contare le stringhe s_i $(i = 1, \ldots, n)$ la cui lunghezza è maggiore di k.

Algoritmi e diagrammi di flusso: problemi di verifica

Esercizio 8. Dati un intero x ed una sequenza finita (non vuota) di numeri interi (terminata da uno 0), dare come risultato *true* se x occorre nella sequenza, altrimenti (ovvero se x non compare nella sequenza) restituire *false*.

Input: $x \in \mathbb{Z}$, $\quad x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: *true* se x compare nella sequenza, *false* altrimenti

Questo problema *non richiede di contare* quante volte x occorre nella sequenza, ma solo di *verificare* se x vi *compare*.

Ciò significa che, quando si scorre la sequenza, *non appena* si incontra un numero uguale ad x, si restituisce subito il valore booleano *true* senza dover scorrere tutta la sequenza.

Verificare l'occorrenza di un elemento

Si ha il risultato *true* con la sequenza visitata fino in fondo solo quando l'elemento cercato compare proprio come *ultimo* elemento della sequenza.

Invece, per restituire il valore *false* (ovvero affermare che x non compare nella sequenza), è necessario *scorrere tutta la sequenza fino in fondo* senza aver trovato x.

Alle usuali operazioni elementari si aggiungono le operazioni di confronto tra *valori booleani* e le operazioni logiche basilari di *negazione*, *congiunzione* e *disgiunzione* tra espressioni booleane.

Verificare l'occorrenza di un elemento (cont.)

Utilizzo di una locazione di memoria contenente un *valore booleano*, riferita simbolicamente con `trovato`.

Se il valore di `trovato` è *true*, significa che abbiamo trovato un elemento nella sequenza che è uguale ad x.

Se il valore di `trovato` è *false*, significa che x non è stato ancora trovato.

Il valore iniziale di `trovato` è pertanto *false*, in quanto all'inizio non si è ancora trovato niente.

Verificare l'occorrenza di un elemento (cont.)

La sequenza viene scorsa fintantoché valgono *entrambe* le condizioni seguenti:
non si è arrivati alla fine della sequenza ($\texttt{n} \neq 0$)
e (inteso come congiunzione logica, `AND`)
non è stato ancora trovato x (`trovato` vale *false*).

La *congiunzione logica* `p AND q` è vera se e solo se le due espressioni booleane `p` e `q` sono *entrambe* vere, mentre è falsa se *almeno una* tra `p` e `q` è falsa.

Quindi nell'algoritmo si hanno due *condizioni di uscita* dal ciclo:
i) si è arrivati alla fine della sequenza ($\texttt{n} \neq 0$ è falso) *oppure*
ii) è stata trovata la prima occorrenza di x e `trovato` vale *true*.

Eseguire l'algoritmo con input `x=4` e sequenza `3 11 8 4 -7 5 0`, e successivamente con `x=-2` e stessa sequenza.

Verificare l'occorrenza di un elemento con proprietà

Esercizio 9. Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0), restituire *true* se *esiste almeno un* numero negativo nella sequenza, altrimenti restituire *false*.

Input: $x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: *true* se $\exists\, x_i$ negativo, *false* altrimenti

Problema simile al precedente, cambia solo la proprietà da verificare: al posto di `n=x` abbiamo `n<0`.

Testare l'algoritmo con le seguenti sequenze in input:
`3 4 -7 9 -1 3 0` e poi `5 7 1 4 0`.

In generale, tale algoritmo risolve tutti i problemi in cui occorre verificare se in una sequenza di valori ne *esiste almeno uno* che soddisfa una determinata proprietà P.

Verificare tutti gli elementi con proprietà

Esercizio 10. Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0), restituire *true* se *tutti* gli elementi della sequenza soddisfano una data proprietà P, altrimenti restituire *false*.

Input: $x_1\ x_2\ \ldots\ x_n\ 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: *true* se $\forall\, x_i$ si ha $P(x_i)$ vera, *false* altrimenti

Problema "speculare" rispetto ai problemi precedenti (in cui si verificava che *almeno un elemento* avesse una certa proprietà).

Qui occorre verificare che *tutti gli elementi* della sequenza abbiano una data proprietà P.

Per restituire *true* è quindi necessario scorrere tutta la sequenza fino in fondo e verificare che valga $P(x_i)$ per ogni elemento x_i.

Verificare tutti gli elementi con proprietà (cont.)

Invece, per restituire *false* è sufficiente trovare il primo elemento che non soddisfa P senza dover scorrere tutta la sequenza.

Caso particolare: tutti gli elementi tranne l'ultimo soddisfano P, si restituisce *false* e la sequenza è stata visitata fino in fondo.

Un possibile algoritmo è simile a quelli precedenti con le stesse operazioni elementari.

La locazione booleana `trovato` è rimpiazzata da una locazione booleana `ok`, tale che:
se `ok` vale *true*, significa che tutti gli elementi già visitati soddisfano la proprietà P;
se `ok` vale *false*, significa che esiste almeno un elemento per il quale P risulta falsa (proprietà non verificata da tutti gli elementi).

La locazione `ok` è inizializzata a *true*.

Verificare almeno k elementi con proprietà

Esercizio 11. Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0) ed un intero $k > 0$, restituire *true* se *esistono almeno k* elementi della sequenza che soddisfano una data proprietà P, altrimenti restituire *false*.

Input: $x_1\ x_2\ \ldots\ x_n\ 0, \quad k > 0$ con $x_i \in \mathbb{Z}$ per ogni $i = 1, \ldots, n$
Output: *true* se per almeno k elementi vale P, *false* altrimenti

Problema "ibrido" o "misto" delle due tipologie considerate prima.

Occorre *contare* gli elementi che soddisfano una proprietà P, ma non necessariamente occorre scorrere tutta la sequenza.

N.B. Non appena sono contati k elementi per cui vale P, si smette di scandire la sequenza e si restituisce *true*.

Si restituisce *false* se si è raggiunta la fine della sequenza senza contare k elementi che soddisfano P.

Verificare almeno k elementi con proprietà: sol.1

Primo algoritmo (vedere diagramma di flusso Es.11):
utilizzo di un contatore c per contare e di una locazione di tipo booleano `trovato` per segnalare che sono stati trovati k elementi che soddisfano P.

Nessuna nuova operazione elementare necessaria.

N.B. Il blocco di azione in cui viene assegnato ad `n` il numero successivo della sequenza può essere eliminato dai *due* punti in cui si trova e "messo a comune" alla fine del blocco condizionale esterno prima di tornare in ciclo.

Ciò significa che ad `n` si assegna sempre il numero successivo, anche nel caso in cui `trovato` sia stato messo a *true*.

Testare l'algoritmo con input: sequenza `3 -4 5 7 -11 5 2 0` e `k=3` considerando la proprietà $P(x) =_{def} x > 0$.

Verificare almeno k elementi con proprietà: sol.2

L'algoritmo può essere modificato per fare a meno della locazione booleana `trovato` (vedere diagramma di flusso Es.11 seconda versione)

Il concetto rappresentato da `trovato` viene rimpiazzato dal confronto `c<k` tra il valore del contatore e l'intero k dato in input.

Dire che `trovato` vale *false* significa che c ha un valore minore di k.

Se `trovato` è diventato *true*, significa che c ha raggiunto il valore k.

Verificare almeno k elementi con proprietà: sol.2 (cont.)

All'uscita dal blocco di iterazione è necessario distinguere per quale condizione l'iterazione è terminata.

Le condizioni di uscita sono due:

i) vale $\texttt{c} \geq \texttt{k}$, quindi abbiamo trovato almeno k elementi che soddisfano P, per cui si restituisce *true*, oppure

ii) $\texttt{c} \geq \texttt{k}$ risulta falso, ne segue che $\texttt{c} < \texttt{k}$ è vero e quindi, affinché la condizione del ciclo sia falsa, si ha $\texttt{n}=0$.

Il caso ii) implica che è stata esaminata tutta la sequenza ed il valore del contatore è rimasto minore di k, quindi si restituisce *false*.

Verificare almeno k elementi con proprietà: sol.3

N.B. Il test finale su c $\geq$ k può essere sostituito da un blocco di output che restituisce il valore di c$\geq$k
(vedere diagramma di flusso Es.11 terza versione)

Osservazione: utilizzare il test n=0 all'uscita del blocco di iterazione, al posto della condizione c$\geq$k, *può non essere corretto* (diagramma di flusso Es.11 versione errata).

Ciò accade se i k elementi che soddisfano P sono ottenuti proprio sull'*ultimo elemento* della sequenza, poiché in questo caso le due condizioni in AND nel blocco iterativo diventano entrambe false alla stessa iterazione.

Esempio: sia data la proprietà $P(x) =_{def} x < 0$ e siano dati in input la sequenza -3 5 -1 7 -11 0 e k=3, verificare il risultato restituito.