Corso di Laurea in Informatica

Modulo di Laboratorio di Programmazione I (a.a. 2009-10)

Docente: Prof. M. Nesi

Esercizi su Algoritmi e Diagrammi di Flusso

(Versione preliminare)

Dati i seguenti problemi, fornire un algoritmo per ciascuno di essi utilizzando i diagrammi di flusso e mettendo in evidenza le operazioni elementari necessarie. I diagrammi di flusso proposti come soluzione ai vari problemi sono riportati alla fine degli esercizi.

Esercizio 1
Dati due numeri naturali m ed n, calcolare il massimo (risp. minimo) tra m ed n.

Input: $m, n \in \mathbb{N}$
Output: $max(m, n)$

Oltre alle operazioni di input/output per l'acquisizione dei dati e la restituzione del risultato (e.g. lettura in ingresso e scrittura/stampa in uscita), come operazioni elementari di una macchina astratta sono necessarie solo le operazioni di confronto tra numeri naturali ($>$, $\geq$, $<$, $\leq$, $=$, etc.). Nella soluzione data viene utilizzato l'operatore $>$, ma è possibile usare anche uno degli altri operatori (con opportune modifiche all'algoritmo). Nel caso in cui $m = n$, si restituisce uno qualsiasi tra m ed n come valore massimo. L'algoritmo per calcolare il minimo tra m ed n può essere dato in modo simile.

Esercizio 2
Dati tre numeri naturali m, n e p, calcolare il massimo (risp. minimo) dei tre numeri.

Input: $m, n, p \in \mathbb{N}$
Output: $max(m, n, p)$

Le operazioni elementari necessarie sono le stesse dell'esercizio precedente. L'algoritmo proposto fa confronti successivi tra i numeri a due a due. In modo simile può essere dato l'algoritmo per calcolare il minimo di tre numeri.

Esercizio 3
Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0)

$$x_1\ x_2\ \ldots\ x_n\ 0$$

calcolare il massimo (risp. minimo) della sequenza[1].

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

Un possibile modo di procedere consiste nell'esaminare tutti i numeri della sequenza e confrontare ciascun numero con un valore che rappresenta il *massimo corrente*, ovvero il massimo della porzione di sequenza già esaminata. Supponiamo di poter fare uso di due locazioni (o celle) di memoria dove leggere e scrivere valori interi (in un linguaggio di programmazione queste locazioni saranno riferibili tramite due *identificatori di variabili* di tipo intero). Siano le due locazioni riferite simbolicamente con i nomi `max` ed `n`. La locazione `max` funge da *accumulatore*, nel senso che in tale cella viene memorizzato il valore del massimo corrente. Nella locazione `n` viene copiato l'elemento della sequenza considerato ad un certo passo dell'algoritmo per confrontarlo con `max`. Poiché la sequenza in ingresso è non vuota, il suo primo elemento (che esiste ed è diverso da 0) può essere usato per *inizializzare* il contenuto della locazione `max` (notare che l'inizializzazione di `max` con il valore 0 può non essere corretta su $\mathbb{Z}$, per esempio nel caso in cui i numeri della sequenza siano tutti negativi).

L'idea dell'algoritmo è di ripetere i seguenti passi fino a quando non si raggiunge la fine della sequenza: si seleziona un elemento x_i della sequenza (in genere si parte da x_1 fino a quando si raggiunge il primo 0) copiandolo nella locazione `n`, si confronta il valore di `n` con quello di `max` e se `n` risulta (strettamente) maggiore di `max`, il valore di `max` viene aggiornato con il valore di `n`, altrimenti non viene apportata alcuna modifica su `max`.

Le operazioni elementari necessarie sono: lettura/scrittura in memoria, test su 0, selezione del primo elemento di una sequenza e dell'elemento successivo, operazioni di confronto su $\mathbb{Z}$. Nel caso in cui il massimo (risp. minimo) compaia piú di una volta nella sequenza, basta restituire una delle sue occorrenze.

Sia una *stringa* definita come una sequenza di caratteri. Esempi di stringhe sono `bbccc`, `a1` ed `fg47`. La lunghezza di una stringa s è data dal numero di caratteri in s. Denotiamo la stringa vuota (ovvero la stringa con zero caratteri) con il simbolo ϵ. Le stringhe possono essere concatenate per formare stringhe piú lunghe semplicemente giustapponendo le stringhe una dopo l'altra. Ad esempio, la concatenazione delle stringhe `a1` ed `fg47` dà luogo alla stringa `a1fg47`. L'operazione di concatenazione è associativa, ma non è commutativa. La stringa vuota ϵ è l'elemento neutro o identità per la concatenazione, ovvero valgono le seguenti leggi: per ogni stringa s si ha $s\,\epsilon = \epsilon\,s = s$.

Esercizio 4
Data una sequenza finita (non vuota) di stringhe (terminata dalla stringa vuota ϵ), calcolare la stringa di lunghezza massima (risp. minima) della sequenza[2].

[1] In questi esercizi supponiamo che ogni *sequenza* di elementi sia finita (i.e. costituita da un numero finito di elementi), non vuota (i.e. esiste almeno un elemento nella sequenza diverso da 0 prima di uno 0 di fine sequenza) e terminata da uno 0 (i.e. il numero 0 viene usato come marcatore di fine sequenza ed eventuali numeri dopo uno 0 sono ignorati). Quando passeremo al linguaggio di programmazione, avremo a disposizione altri costrutti per verificare se si è arrivati alla fine di una sequenza, per cui lo 0 tornerà ad essere un numero come tutti gli altri.

[2] Nel caso in cui vi siano nella sequenza in ingresso piú stringhe di lunghezza massima (risp. minima), la specifica del problema può richiedere di restituire una stringa in particolare tra quelle di lunghezza massima

Input: $s_1\ s_2\ \ldots\ s_k\ \epsilon$
Output: la stringa s_j di lunghezza massima

Il problema da risolvere è simile a quello dell'esercizio precedente, la sola differenza è il tipo di elementi considerati. Ciò porta a richiedere operazioni elementari diverse per quanto riguarda, ad esempio, il confronto tra stringhe, in quanto le stringhe sono oggetti in genere piú complessi dei numeri interi, ma l'algoritmo non cambia. Facendo *astrazione* sul tipo degli elementi, il diagramma di flusso che descrive l'algoritmo ha una struttura uguale a quella del diagramma dell'esercizio precedente. Supponiamo ancora di avere due locazioni di memoria atte a contenere valori di tipo stringa, riferite simbolicamente con `max` (contenente la stringa massima corrente) ed `s` (contenente la stringa s_i in esame). Nel diagramma di flusso, oltre a rimpiazzare numeri con stringhe ed `n` con `s`, cambiano solo le operazioni dentro ad alcuni blocchi: si ha $s \neq \epsilon$ per il test di fine sequenza e `lung(s) > lung(max)` per il confronto tra le lunghezze, dove `lung` denota la funzione che calcola la lunghezza di una stringa.

Esercizio 5
Data una sequenza finita (non vuota) di numeri interi (terminata da uno 0), calcolare quanti sono i numeri negativi (oppure, quanti sono i numeri pari, oppure i numeri divisibili per un dato numero k, etc.). In generale, si richiede di *contare* quanti elementi della sequenza hanno o soddisfano una determinata proprietà P.

Input: $x_1\ x_2\ \ldots\ x_n\ 0$
Output: numero degli elementi x_i per i quali vale $P(x_i)$

Come nei problemi precedenti, anche in questo caso occorre scandire tutta la sequenza e contare quegli elementi che soddisfano la proprietà P data. Oltre ad una locazione di memoria `n` che contiene l'elemento della sequenza in esame, utilizziamo una locazione di memoria `c` di tipo intero, detta *contatore*. Un contatore è un particolare accumulatore, tipicamente inizializzato a 0, il cui valore viene aggiornato incrementandolo di una unità solo quando l'elemento in esame soddisfa P. Il risultato sarà il valore finale del contatore `c`. Le operazioni elementari sono ancora quelle di lettura/scrittura (o assegnamento) di valori, selezione del primo elemento della sequenza ed elemento successivo, confronti ed operazioni aritmetiche su $\mathbb{Z}$ (in particolare, quella di incremento di un'unità).

Notare che i diagrammi di flusso degli ultimi due esercizi hanno struttura simile.

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: $x \in \mathbb{Z},\quad x_1\ x_2\ \ldots\ x_n\ 0$
Output: numero di occorrenze di x nella sequenza

Questo problema è un caso particolare del problema generale precedente, in cui la proprietà $P(n)$ per un intero n è definita come segue: $P(n)$ è vera se e solo se $n = x$. Il diagramma di

(risp. minima), per esempio quella più a destra o quella piú a sinistra nella sequenza data.

flusso che descrive l'algoritmo che risolve tale problema è un caso particolare del diagramma precedente, in cui `P(n)` è istanziato con `n=x`.
Per 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.

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

Input: $x_1\ x_2\ \ldots\ x_n\ 0$
Output: $x_1 + x_2 + \ldots + x_n$

Anche in questo caso abbiamo bisogno di scorrere tutta la sequenza e di accumulare il risultato in una locazione di memoria, in cui all'i-esima iterazione dell'algoritmo è memorizzato il valore della somma parziale $x_1 + x_2 + \ldots + x_i$ relativa alla porzione di sequenza già visitata. In questo caso non vi è alcuna proprietà che gli elementi della sequenza devono soddisfare, questi vengono semplicemente sommati al valore corrente dell'accumulatore, che viene inizializzato a 0. Per esempio, data la sequenza $3\ -1\ 4\ 7\ 5\ 0$, il risultato è 18.

Nel caso in cui il problema chieda di calcolare il prodotto dei numeri nella sequenza $x_1 * x_2 * \ldots * x_n$, basta modificare alcune istruzioni nel diagramma: l'accumulatore viene inizializzato ad 1 e viene poi modificato assegnandogli il valore dell'espressione $s * n$.

Nel caso in cui in ingresso sia data una sequenza di stringhe e si voglia calcolare la stringa risultante dalla concatenazione delle stringhe nella sequenza, l'algoritmo è ancora lo stesso, basta modificare l'inizializzazione dell'accumulatore s assegnandogli la stringa vuota ϵ e poi ad ogni iterazione all'accumulatore viene assegnato il risultato della concatenazione $s\,n$ tra il valore di s e la stringa n in considerazione.

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: $x \in \mathbb{Z}, \quad x_1\ x_2\ \ldots\ x_n\ 0$
Output: *true* se x compare nella sequenza, *false* altrimenti

Questo è un problema in cui non si richiede di contare quante volte x occorre nella sequenza data, ma solo di verificare se x *compare* nella sequenza. Ciò significa che, quando si scorre la sequenza elemento dopo elemento, non appena si incontra un numero uguale ad x, si restituisce subito il valore booleano *true* senza dover scorrere tutta la sequenza (si ha 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.

Oltre all'usuale locazione `n`, facciamo uso di una locazione di memoria contenente un valore booleano, riferita simbolicamente con `trovato`, tale che se il valore di `trovato` è *true* significa che abbiamo trovato un elemento nella sequenza che è uguale ad x, mentre 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. 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. Ne segue che nell'algoritmo si hanno due condizioni per uscire dal ciclo: si è arrivati alla fine della sequenza ($\texttt{n} \neq 0$ è falso) oppure è stata trovata la prima occorrenza di x e `trovato` vale *true*.
Testare l'algoritmo eseguendo il diagramma di flusso con i seguenti dati in input:
$x = 4$ e la sequenza 3 11 8 4 -7 5 0, e successivamente con la stessa sequenza ed $x = -2$.

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: $x_1\ x_2\ \ldots\ x_n\ 0$
Output: *true* se $\exists\, x_i$ negativo, *false* altrimenti

Questo problema è simile a quello precedente, cambia solo la proprietà da verificare: al posto di `n=x` abbiamo `n<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.
Testare l'algoritmo con i seguenti dati in input:
3 4 -7 9 -1 3 0,
5 7 1 4 0,
7 1 5 -3 0.

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: $x_1\ x_2\ \ldots\ x_n\ 0$
Output: *true* se $\forall\, x_i$ si ha $P(x_i)$ vera, *false* altrimenti

Questo problema è "speculare" rispetto ai due problemi precedenti, in cui occorreva verificare che almeno un elemento avesse una certa proprietà. In questo caso, infatti, occorre verificare che *tutti* gli elementi della sequenza abbiamo una data proprietà, per cui per restituire *true* è necessario scorrere tutta la sequenza fino in fondo e verificare che valga $P(x_i)$ per ogni elemento x_i. Invece, per restituire *false* è sufficiente trovare il primo elemento che non soddisfa P senza dover scorrere tutta la sequenza (un caso particolare si verifica quando tutti gli elementi tranne l'ultimo soddisfano P, in tal caso si restituisce *false* e la sequenza è stata visitata fino in fondo).

Un possibile algoritmo è simile a quelli precedenti, in cui 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, mentre se `ok` vale *false* significa che esiste almeno un elemento per il quale P risulta falsa, e quindi la proprietà non è verificata da tutti gli elementi nella sequenza. La locazione `ok` è inizializzata a *true*.

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: $x_1\ x_2\ \ldots\ x_n\ 0, \quad k > 0$
Output: *true* se per almeno k elementi vale P, *false* altrimenti

Questo problema può essere visto come un "ibrido" delle due tipologie di problemi considerate negli esercizi precedenti. Infatti, in questo caso occorre contare gli elementi che soddisfano una proprietà P, ma non necessariamente occorre scorrere tutta la sequenza. Non appena sono contati k elementi per cui vale P, si smette di scandire la sequenza e si restituisce *true*, mentre si restituisce *false* se si è raggiunta la fine della sequenza senza contare k elementi che soddisfano P. Per contare si usa un contatore `c` e per segnalare che sono stati trovati k elementi che soddisfano P possiamo usare ancora una locazione di tipo booleano `trovato`. Un primo diagramma di flusso per risolvere questo problema è dato in **11(a)**. Notare che il blocco di istruzione 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 nella sequenza, anche nel caso in cui `trovato` sia stato messo a *true* e quindi non sia necessario continuare a scorrere la sequenza. Ad esempio, testare l'algoritmo con i seguenti dati in input: la sequenza $3\ -4\ 5\ 7\ -11\ 5\ 2\ 0$, $k = 3$ e $P(x)\ =_{def}\ x$ positivo.

L'algoritmo, e quindi anche il diagramma di flusso, può essere ulteriormente modificato adottando una soluzione che utilizzi solo il contatore `c` e la locazione `n` *senza* la locazione booleana `trovato` (vedi **11(b)**). Il concetto rappresentato da `trovato` viene rimpiazzato da un confronto tra il valore del contatore e l'intero k dato in input. In altre parole, dire che `trovato` vale *false* significa che `c` ha un valore minore di k, mentre se `trovato` è diventato *true* significa che `c` ha raggiunto il valore k. All'uscita dal blocco di iterazione è necessario distinguere per quale condizione l'iterazione è terminata. I casi sono due: i) vale `c`$\geq$`k`, quindi abbiamo trovato almeno k elementi che soddisfano P, per cui si restituisce *true*, oppure ii) `c`$\geq$`k` risulta falso, ne segue che `c<k` è vero e quindi, affinché la condizione del ciclo sia falsa, si ha `n=0`. Ciò implica che è stata esaminata tutta la sequenza ed il valore del contatore è rimasto minore di k, quindi si restituisce *false*.

Si osservi che utilizzare il test `n=0` all'uscita del blocco di iterazione, al posto della condizione `c`$\geq$`k`, può non essere corretto. 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. Ad esempio, dati la sequenza $-3\ 5\ -1\ 7\ -11\ 0$, $k = 3$ e $P(x)\ =_{def}\ ''x$ negativo$''$, verificare il risultato restituito dal diagramma di flusso **11(b)** in cui il blocco condizionale finale sia stato rimpiazzato come in **11(c)**.

Strutture dati bidimensionali
Finora abbiamo considerato problemi di ricerca e verifica di proprietà all'interno di una sequenza *lineare* o *monodimensionale* di elementi. Sia in ambito scientifico che nella realtà quotidiana possiamo avere a che fare con dati strutturati in piú dimensioni, e.g. matrici, tabelle, etc. Si pone quindi la questione di 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 il caso di strutture dati bidimensionali, ma il ragionamento può essere esteso in modo simile al caso generale multidimensionale.

Come già fatto per le sequenze lineari di elementi, supponiamo che la macchina astratta fornisca le operazioni elementari per manipolare sequenze di sequenze (e.g. selezione della prima sequenza e della 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. Infine, data una sequenza S di sequenze di elementi $s_1\ s_2\ \ldots\ s_n$, abbiamo le seguenti assunzioni:
i) S è finita, i.e. costituita da un numero finito di sequenze lineari s_i;
ii) 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;
iii) esiste un marcatore di fine sequenza di sequenze, denotato {};
iv) S è non vuota, i.e. esiste almeno una sequenza in S prima del marcatore {}.

Negli esercizi seguenti vediamo come i problemi, considerati prima per le sequenze lineari, possono essere risolti nel caso in cui si abbiano sequenze di sequenze.

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

Input: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$
Output: numero di occorrenze di x in S

In maniera analoga a quanto fatto nel caso di una sequenza lineare, per risolvere questo problema occorre scorrere tutta la sequenza S e registrare il numero delle occorrenze di x in un contatore. Un modo per farlo 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, e poi passare alla riga successiva. 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.

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 prodotto o concatenazione degli elementi di S, l'elemento massimo o minimo in S, etc.

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: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$
Output: *true* se x occorre in S, *false* altrimenti

Il diagramma dato per lo stesso tipo di problema nel caso di una sequenza lineare (Esercizio 8) deve essere esteso con i blocchi necessari per la scansione di una struttura bidimensionale tenendo conto che, non appena troviamo la prima occorrenza di x in una qualche riga di S, si termina restituendo *true*, mentre *false* è restituito se sono state esaminate tutte le sequenze in S senza aver trovato x.

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: $x \in \mathbb{Z}$, $S = s_1\ s_2\ \ldots\ s_n\ \{\}$
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, in quanto è sufficiente verificare che x compaia in una qualche sequenza di S. Ora invece si richiede che questa proprietà sia verificata *localmente per ogni sequenza* di S. 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. Inoltre, *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.

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

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

$$S = \begin{array}{l} 1\ 4\ 3\ 3\ , \\ 5\ -3\ 3\ , \\ 3\ -1\ 5\ 3\ 7\ , \\ \{\} \end{array}$$

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

Esercizio IV
Dati una sequenza di sequenze di interi S ed un intero $k > 0$, restituire *true* se in ogni se-

quenza s_i di S esistono almeno k elementi che soddisfano una data proprietà P, altrimenti restituire *false*.

Input: $S = s_1\ s_2\ \ldots\ s_n\ \{\}, \quad k > 0$
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 proprietà che s_i abbia almeno k elementi per i quali sia vera una data proprietà P. Ad esempio, dati $P(n) =_{def}$ "n pari", $k = 2$ e la sequenza di sequenze

$$S = \begin{array}{l} 3\ 4\ -2\ 6\ , \\ 7\ -10\ 8\ , \\ 4\ 5\ 21\ -8\ 10\ , \\ \{\} \end{array}$$

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

Esercizio
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. Si ricorda la definizione del fattoriale: $0! = 1$ ed $n! = n \times (n-1) \times \ldots \times 2 \times 1$ per $n \geq 1$.

3. Dato $n \in \mathbb{N}$, calcolare l'n-esimo numero di Fibonacci. Si ricorda la definizione dei numeri di Fibonacci: $fib(0) = 1$, $fib(1) = 1$, $fib(n) = fib(n-1) + fib(n-2)$ per $n \geq 2$.