Problemi, Soluzioni, Programmi

Ivano Salvo
Sapienza Università di Roma
email: salvo@di.uniroma1.it

Anno Accademico 2012-13

Introduzione La presente dispensa si propone di presentare i concetti fondamentali e introduttivi alla programmazione, cercando di presentare i concetti di "problema", "soluzione", "calcolo" ed "esecutore" nella loro forma più generale, indipendentemente da uno specifico linguaggio di programmazione.

Si cercherà dapprima di fornire un'idea il quanto più possibile elementare di cosa significhi "calcolare" o "risolvere un problema", riconducendoci a esperienze comuni sul tema. In un secondo momento, si affrontano problemi numerici elementari e le relative soluzioni, focalizzando l'attenzione sul concetto di "procedura risolutiva di un problema" come sequenza di azioni elementari interpretabili univocamente da un "esecutore", e si cerca di evidenziare quali siano le azioni elementari che un ragionevole esecutore deve sempre poter eseguire.

Nel corso di queste note e di quelle successive, verranno lasciati al lettore numerosi esercizi. Gli esercizi, a volte, hanno dei contrassegni: ♣ significa che l'esercizio ha un contenuto "teorico", oppure è un complemento a quanto viene illustrato nella sezione corrispondente; ★ indica un esercizio che potrebbe risultare impegnativo o che potrebbe nascondere delle finezze; ▶ indica un esercizio che invita alla sperimentazione al calcolatore.

1 Problemi, Classi di Problemi, Soluzioni

La soluzione di problemi vi è nota fin dalla prima infanzia. Ad esempio, vi sarete probabilmente trovati nella spiacevole situazione di desiderare le caramelle e non avere i soldi per acquistarle. In tale situazione, avete dovuto decidere una *strategia* per risolvere tale questione. Vi saranno sicuramente balenate in testa diverse soluzioni, ciascuna con vantaggi e svantaggi, ad esempio:

- chiedo i soldi alla mamma;
- chiedo i soldi a un amico;

- rubo le caramelle;
- picchio un "amico" per ottenere soldi e/o caramelle;
- ...

Con l'età e la scuola, probabilmente vi sono stati posti un gran numero di problemi, prevalentemente di carattere matematico. Un tipico problema era il seguente:

> La mamma va al mercato. Compra 3 pere. Ciascuna pera costa 150 lire. Quanto spende la mamma?

La soluzione in questo caso è semplicemente la risposta "450 lire". Probabilmente, con l'età e l'esperienza, avrete sicuramente isolato un'intera *classe di problemi*, caratterizzata da un identico processo risolutivo, che potremmo chiamare *il problema del mercato*:

> La mamma compra un certo numero n di cose, ciascuna delle quali ha un prezzo p. Quando spende la mamma?

La soluzione consiste nell'eseguire la moltiplicazione $n \times p$. Osservate che, in questo caso, non avete semplicemente risolto un problema, ma indicato una *procedura risolutiva* per un'intera, per quanto semplice, classe di problemi. In generale, sarà proprio questo il nostro obiettivo: *individuare procedure risolutive per classi di problemi.*

Un problema più complicato che vi ha insegnato a risolvere già la maestra elementare è quello di sommare due numeri a più cifre. Cerchiamo di descrivere la procedura necessaria:

1. scrivere i due numeri, allineandoli a destra;
2. cominciando da destra, sommare le due cifre;
3. se il risultato è un numero di una sola cifra, scrivere semplicemente il risultato;
4. se il risultato è un numero di due cifre, scrivere la meno significativa nella riga del risultato e tenere la seconda come riporto;
5. sommare le due cifre immediatamente a sinistra, sommando a questo punto l'eventuale riporto e procedere come nel caso precedente (istruzioni 3 e 4);
6. ripetere queste operazioni, finché non si sono esaurite le cifre di entrambi i numeri da sommare e i riporti.

La maestra ha supposto che voi sapeste interpretare in *modo univoco* questa sequenza di azioni. Più in particolare ha supposto che voi aveste le capacità di:

- allineare due numeri di più cifre a partire dalle cifre meno significative;
- sommare due numeri composti da una sola cifra, distinguendo nel risultato la cifra delle unità (che va trascritta nella riga del risultato) e quella delle decine (che va trascritta nella riga dei riporti);
- ripetere le stesse operazioni, fino al raggiungimento di una certa condizione (fine delle cifre e dei riporti);
- usare il foglio di carta per memorizzare dei dati intermedi del calcolo (ad esempio i riporti) per poi usarli in fasi successive del calcolo.

Rappresentazione dei Dati Osservate che la procedura vista sopra dipende non tanto dalla natura dei numeri naturali, ma da una particolare *rappresentazione* dei numeri: avreste potuto eseguire l'algoritmo anche ignorando il significato delle cifre[1], ma se i numeri vi fossero stati dati in notazione romana, l'algoritmo non sarebbe stato applicabile.

La corretta rappresentazione dei dati è fondamentale affinché sia possibile descrivere procedure risolutive semplici ed efficienti: ed è sostanzialmente alla facilità di eseguire le operazioni aritmetiche che la notazione posizionale per i numeri deve il suo successo: in ogni caso, un *calcolo* può essere visto semplicemente come una *manipolazione di simboli*, che segue delle precise (e univoche) regole formali, indipendenti da cosa i simboli rappresentino.

Ricordate che ad esempio '13' è semplicemente una sequenza di caratteri che rappresenta un numero naturale, il quale potrebbe essere altrimenti rappresentato, ad esempio con una sequenza di 13 simboli 'IIIIIIIIIIIII', oppure con la sequenza di caratteri 'XIII', e più in generale da un insieme di simboli a cui si possa, seguendo una 'semantica' ben definita, associare univocamente il numero 13.

Astrazione Il numero naturale 13, infatti, è sostanzialmente un'*astrazione* che accomuna tutti gli insiemi che contengono 13 oggetti. *Astrarre* significa in informatica (e in matematica, ma più in generale nei processi mentali umani), cogliere gli aspetti essenziali e, dimenticare i dettagli non importanti rispetto al problema in esame. Un banale esempio può ancora essere il problema del mercato: è irrilevante se la mamma compra 3 pere, 3 mele o 3 banane: nella soluzione è rilevante solamente il numero degli oggetti comprati e il loro prezzo. Altre volte *astrarre* significherà generalizzare un ragionamento o la soluzione di un problema, in modo che essa si applichi a una classe più generale di problemi.

L'uso di un elaboratore per risolvere un dato problema, segue quindi le seguenti fasi:

[1] ad esempio vi si sarebbe potuta fornire una tabella esoterica di dimensione 10×10, analoga alla tavola pitagorica per la moltiplicazione, in cui ad esempio all'incrocio della riga 8 e colonna 7 avreste trovato che il risultato è 5 e il riporto è 1.

Codifica dei dati in ingresso nella rappresentazione manipolata dalla procedura risolutiva;

Descrizione di una procedura risolutiva che evidenzia i passi elementari necessari per arrivare alla soluzione: la procedura risolutiva è usualmente dipendente (o parametrica) rispetto a dei dati in ingresso, che identificano specifiche *istanze* del problema;

Decodifica o Interpretazione dei dati in uscita che permetta all'utente di comprendere la soluzione ottenuta: ad esempio, si può interpretare una sequenza di cifre decimali come un numero naturale;

Verifica che la procedura risolutiva effettivamente risolve il problema che ci si era posti, e che i risultati ottenuti siano effettivamente soluzioni del problema.

Linguaggio Un'ultima importante questione riguarda *come* descrivere le procedure risolutive. L'aspetto fondamentale è che l'esecutore interpreti in *modo univoco* la sequenza di azioni elementari: mentre la maestra si affidava al vostro crescente buonsenso nell'interpretare le istruzioni che lei vi dava per eseguire le operazioni aritmetiche, noi studieremo viceversa dei *linguaggi formali* con cui descrivere le procedure risolutive. Tali linguaggi avranno una *precisa* semantica, in modo che l'esecuzione di una istruzione abbia un effetto prevedibile, o se preferite *deterministico*.

Tuttavia, in questo corso, ci limiteremo a descrivere la semantica dei vari costrutti di un linguaggio di programmazione in modo informale, spiegando a parole il loro significato. In un approccio più rigoroso, sarebbe possibile descrivere il significato dei costrutti di un linguaggio in modo *formale*, cioè descrivendo ad esempio i comportamenti di una macchina in seguito all'esecuzione di ciascuna possibile istruzione (*semantica operazionale*).

Algoritmi Useremo usualmente la parola *algoritmo*[2] per indicare le procedure risolutive. Cerchiamo di riassumere quali siano le caratteristiche che si suppone abbiano gli algoritmi:

Descrizione Finita: un algoritmo deve poter essere descritto da un numero finito di azioni elementari;

Illimitatezza dei Dati in Ingresso/Uscita: un algoritmo non fa in genere ipotesi sulla dimensione massima dei dati in ingresso e sulla lunghezza dei risultati (fate attenzione tuttavia la differenza tra *illimitato* e *infinito*!);

[2] l'etimologia di questa parola è legata al nome del matematico persiano *Muhammad ibn Musa 'l-Khwarizmi*, IX secolo, che introdusse in occidente l'idea di rappresentazione posizionale dei numeri. Tuttavia l'idea di *procedura di calcolo* è molto più antica e praticamente comune a tutte le civiltà.

Illimitatezza della Memoria: l'esecutore ha a disposizione una memoria illimitata per registrare dati intermedi durante l'esecuzione;

Illimitatezza dei Passi Eseguiti: l'esecutore, in generale, eseguirà la sua procedura senza porre limiti al numero di passi eseguiti;

l'Esecutore Opera in Modo Discreto: l'esecutore esegue un passo alla volta, e ciascuna azione modifica lo stato della memoria. L'effetto di ciascun passo è univocamente determinato una volta noto lo stato dell'esecutore (cioè la memoria) che la esegue.

Il lettore non confonda ciò che ha appena letto: un algoritmo verrà descritto presupponendo che non ci siano limiti al numero di passi da eseguire o caselle di memoria da utilizzare, ma considereremo (almeno in questo corso) importante che l'esecuzione di un algoritmo termini dopo un numero finito di passi. Inoltre, ci porremo problemi quali: dato un algoritmo che risolve un problema, ce n'è uno migliore, che risolve lo stesso problema, impiegando meno passi o meno memoria?

1.1 Esercizi e spunti di riflessione

1. Provate a descrivere la procedura per la moltiplicazione di due numeri a più cifre. Cercate di individuare quali siano le capacità necessarie ad eseguirlo. E individuare quali siano i risultati intermedi che è necessario memorizzare.

2. Provate a descrivere la procedura per uscire da un labirinto. Supponete che il labirinto sia costituito di stanze e corridoi e che l'esecutore lo possa percorrere aiutandosi all'occorenza con un pennarello (o dei sassolini!) per marcare stanze e corridoi già percorsi.

3. Prendete un libro di cucina e leggete la descrizione del vostro piatto preferito. Può la ricetta essere considerata un algoritmo? Perché?

4. E uno spartito musicale?

5. Riflettete e provate a spiegare a parole e con esempi, la differenza tra *illimitato* e *infinito*.

6. Considerare il seguente vecchio dilemma: "Un pastore deve attraversare un fiume portando con sè un lupo, una capra e un cavolo. Può utilizzare una barca e può ospitare con sè solo un altro viaggiatore. Sapendo che non può lasciare soli lupo e pecora, e pecora e cavolo, come può il pastore raggiungere il suo scopo?" Riflettere sulla differenza tra *soluzione* e *risultato*.

2 Algoritmi Elementari

Consideriamo in questa sezione alcuni problemi elementari sui numeri naturali e cercheremo di vedere le relative procedure risolutive. Via via distilleremo un *linguaggio* con cui descrivere tali procedure risolutive.

In ogni problema, specificheremo quali siano le capacità dell'esecutore che dovrà eseguire la procedura risolutiva. Faremo sempre l'ipotesi che un esecutore possa:

- eseguire delle azioni in sequenza;
- registrare delle informazioni e recuperarle attraverso il *nome* di una *variabile*;
- modificare la sequenza delle azioni eseguendo un'istruzione di salto;
- prendere delle decisioni sulla base della valutazione di condizioni logiche.

Problema 1: *Sia dato un esecutore capace solamente di testare l'uguaglianza di due numeri e di sommare 1. Scrivere una procedura risolutiva che calcola la somma dei due numeri.*

Si tratta di un problema molto semplice: come si sommano due numeri, sapendo solo sommare 1, o se preferite, sapendo solo contare? Probabilmente la soluzione vi è nota fin dall'infanzia, e probabilmente è il modo con cui vi è stato insegnato a sommare i numeri, contando sulle dita. E lo avete probabilmente dimenticato.

Immaginiamo inizialmente di essere noi stessi gli esecutori dell'algoritmo e di avere a disposizione carta, matita e gomma. L'idea è molto semplice: è sufficiente sommare 1 al primo numero un numero di volte pari al secondo al numero. Potremmo procedere nel seguente modo:

1. scriviamo su un foglio di carta i due numeri con la matita. Appare subito evidente che conviene dare loro un nome, per indicarli con facilità nella descrizione della procedura, diciamo n ed m;
2. cancelliamo con la gomma (la casella contenente) m e riscriviamo al suo posto (il valore) $m + 1$;
3. cancelliamo (la casella contenente) n e riscriviamo al suo posto (il valore) $n-1$;
4. se n è zero, significa che abbiamo sommato 1 a m già n volte, e abbiamo quindi finito, e la somma ora risulta scritta nella casella dedicata ad m. Viceversa torniamo al passo 2 e continuiamo le operazioni.

La soluzione sopra descritta ha (almeno) un paio di problemi:

- non funziona nel caso in cui n sia 0 all'inizio (verificate);

- il nostro esecutore protesterà, perché la procedura fa riferimento a un'azione che non sa compiere: la sottrazione.

Il primo problema si aggiusta, osservando che è necessario spostare il controllo sul valore di n prima delle operazioni di cancellazione e riscrittura fatte ai passi 2 e 3.

Il secondo problema, viceversa ci impone di scegliere un'altra tecnica per contare: dobbiamo farlo in avanti. L'idea è altrettanto semplice: dobbiamo usare un'altra casella del foglio di carta, che chiameremo i, scriverci all'inizio il numero 0 e sostituire ad ogni passo il valore di i con $i + 1$. In questo caso lasceremo sempre immutato il valore scritto nella casella n e ci assicureremo di aver sommato 1 ad m esattamente n volte, confrontando il valore scritto nella casella i con quello scritto nella casella n. Riassumendo, ecco finalmente la procedura risolutiva al problema che ci eravamo posti:

1. scrivere su un foglio di carta i due numeri da sommare e uno 0. Chiamare rispettivamente con m, n ed i le caselle che contengono i numeri;
2. se il valore contenuto in i è uguale a quello contenuto in n hai finito e nella casella m è contenuto il risultato della somma;
3. sostituire il contenuto della casella m, sommandoci 1;
4. sostituire il contenuto della casella i, sommandoci 1;
5. torna a eseguire il passo 2.

2.1 Esecutori: Capacità Elementari

È opportuno ancora una volta soffermarci con attenzione su questa procedura, per introdurre alcuni concetti base che vanno compresi alla perfezione. Gli ingredienti fondamentali sono:

Memoria: il foglio di carta che ci permette di memorizzare i dati parziali del calcolo;

Variabili: per descrivere in modo conciso ed efficace la procedura, è bene avere la possibilità di indicare con un *nome* le caselle in cui abbiamo memorizzato i dati;

Assegnazione: abbiamo bisogno di modificare il contenuto delle caselle di memoria; azioni come quelle dei passi 3 e 4, "sostituire il contenuto della casella m, sommandoci 1" saranno indicate nel seguito in modo conciso con '$m \leftarrow m+1$'; ma occorre fin da subito fare attenzione a una inerente ambiguità relativa al nome m, tipica di praticamente tutti i linguaggi di programmazione imperativi: il nome m a sinistra del simbolo $\leftarrow$ significa "la *casella di memoria* di nome m", mentre a destra significa piuttosto: "il *valore memorizzato* nella casella di nome m". Correttamente quindi, l'azione '$m \leftarrow m + 1$' va letta:

"prendi il valore contenuto nella casella m, sommaci 1 e scrivi questo nuovo valore dentro la casella m (cancellando il vecchio valore che verrà perso)".

Flusso di Esecuzione: ordinariamente si eseguono le azioni una dietro l'altra, a meno che non ci siano azioni che esplicitamente alterano il flusso dell'esecuzione: queste sono le azioni che esprimono:

Terminazione: informalmente abbiamo scritto: "hai finito". D'ora in poi scriveremo concisamente **exit**.

Salto: in qualche punto abbiamo scritto dei comandi che dicevano al nostro esecutore di non eseguire l'istruzione successiva, ma di tornare ad eseguirne una precedente, in genere allo scopo di ripetere una certa sottosequenza di azioni; d'ora in poi scriveremo: **goto** l, dove l sarà il numero (o etichetta, in inglese *label*) associato a una certa istruzione;

Condizione: in alcuni punti delle procedure risolutive è necessario prendere delle decisioni, sulla base di alcuni condizioni logiche: ad esempio se il contenuto di due caselle è lo stesso si esegue una certa azione, altrimenti se ne esegue un'altra. Scriveremo concisamente **if** C **then** A_1 **else** A_2, dove C è una condizione logica, cioè una espressione che può assumere i valori vero o falso, mentre A_1 e A_2 sono una o più azioni elementari.

Dati di Ingresso/Uscita: i parametri di ingresso di una procedura vengono spesso inseriti dall'utente che usa una procedura risolutiva per calcolare un risultato su una specifica istanza del problema: useremo l'azione "leggi n" per intendere che il nostro esecutore si aspetta di ricevere un dato e quando lo ottiene lo scrive nella casella di memoria chiamata n. Analogamente, useremo l'azione "scrivi n" per indicare che l'esecutore comunica all'utente il contenuto della casella di memoria chiamata n.

Fatte proprie queste notazioni, scriviamo l'algoritmo precedente, come segue:

1: leggi m; leggi n;
2: $i \leftarrow 0$;
3: if $i = n$ then scrivi m; exit;
4: $i \leftarrow i + 1$;
5: $m \leftarrow m + 1$;
6: goto 3;

Probabilmente risulterà strano ai più, ma il semplicissimo linguaggio sopra descritto è sufficiente per esprimere *qualsiasi* procedura risolutiva. La giustificazione rigorosa di questa affermazione fa parte del programma di un corso di Informatica Teorica, ma nelle soluzioni ai problemi successivi (ed esercizi) il lettore dovrebbe

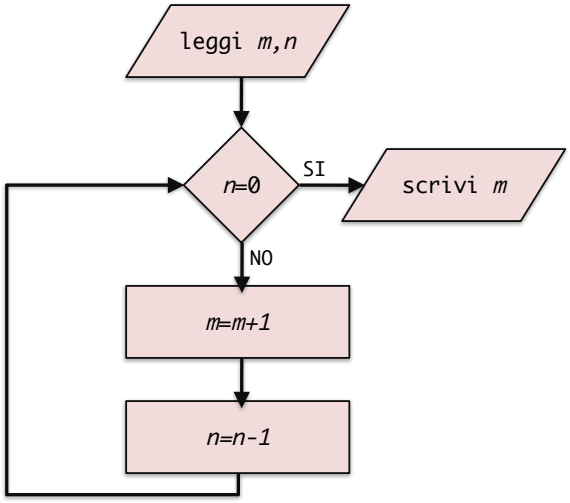


Figura 1: Algoritmo per la somma: diagramma di flusso.

riflettere su come, partendo da abilità così elementari, sia possibile descrivere le procedure risolutive per problemi via via più complessi, magari speculando sul fatto di aver già risolto dei problemi più semplici in precedenza.

Ad esempio, dovendo scrivere una procedura risolutiva per il problema della moltiplicazione tra due numeri, converrà utilizzare il fatto che un esecutore che sa solo sommare 1, in realtà può anche risolvere il problema della somma, e descrivere la procedura che calcola la moltiplicazione in termini della somma, piuttosto che del $+1$, ottenendo una procedura più semplice.

2.2 Digressione: I Diagrammi di Flusso

Una tradizionale metodologia per rappresentare graficamente i programmi, consiste nei cosidetti *diagrammi di flusso* o *flow chart*. Essi sostanzialmente corrispondono al linguaggio che abbiamo appena considerato, ma offrono una rappresentazione grafica che, almeno in linea di principio, dovrebbe facilitare l'individuazione immediata della struttura dell'algoritmo. Di fatto, per algoritmi appena un po' più complessi di quelli visti in questa sezione, il diagramma di flusso diventa subito molto complicato, e quindi in definitiva più difficile da leggere di una rappresentazione testuale del programma. La metodologia dei diagrammi di flusso (o simili) si rivela tuttavia molto efficace quando si rappresentano soluzioni di problemi molto complessi a un livello di dettaglio molto basso (per esempio per rappresentare le relazioni tra varie macro-componenti di un sistema software, o le interazioni dinamiche di componenti software).

Vedremo nel seguito, a titolo di esempio, anche alcune soluzioni dei problemi proposti anche come diagramma di flusso. In Fig. 1, vediamo il diagramma di flusso per l'algoritmo della somma. Senza entrare troppo nei dettagli, nei diagrammi di flusso, diversi tipi di azioni vengono indicati con diversi simboli grafici: i parallelogrammi rappresentano istruzioni di input/output, i rombi (o losanghe) istruzione di decisione e i rettangoli azioni elementari (che usualmente modificano la memoria). Inoltre, per non appesantire troppo le figure con ulteriori simboli grafici per inizio/fine, assumeremo che la procedura risolutiva cominci da un blocco senza archi entranti e finisca nei blocchi senza archi uscenti.

Problema 2: *Sia dato un esecutore capace solamente di testare l'uguaglianza di due numeri e di sommare 1. Scrivere una procedura risolutiva che, letti due numeri naturali, risponda 1 se il primo è minore del secondo, 0 se sono uguali, -1 altrimenti.*

Si tratta ancora di un problema molto semplice: un'idea è quella di decrementare parallelamente i due numeri. Il minore sarà quello che per primo raggiunge il valore 0. Purtroppo però l'esecutore ipotizzato non è in grado di sottrarre 1. Tuttavia, la stessa idea funziona contando 'in avanti': si parte da 0 e si aggiunge 1 ad ogni passo: si verifica se il numero sia uguale al primo dei due numeri: se si, bisogna controllare se sia uguale anche al secondo: in tal caso i due numeri sono uguali, altrimenti il primo è minore. Se invece il contatore è diverso dal primo numero, è possibile che sia uguale al secondo: in tal caso è il secondo a essere il minore. Molto più chiaro descrivere concisamente questa idea sotto forma di sequenza di azioni:

1: leggi m; leggi n;
2: $i \leftarrow 0$;
3: if $i = m$ then goto 7;
4: if $i = n$ then scrivi -1; exit;
5: $i \leftarrow i + 1$;
6: goto 3;
7: if $i = n$ then scrivi 0; exit;
8: scrivi 1; exit;

In Fig. 2 vediamo il flow chart di questo algoritmo. Il lettore potrà già rendersi conto che alcune linee cominciano ad incrociarsi. Forse è possibile scrivere un diagramma di flusso più elegante, ma in generale questo problema si fa via via più grave al crescere delle dimensioni delle procedure che si rappresentano con diagrammi di flusso.

2.3 Specifiche

Problema 3: *Sia dato un esecutore capace solamente di testare l'uguaglianza di due numeri e di sommare 1. Scrivere una procedura risolutiva che calcola il*

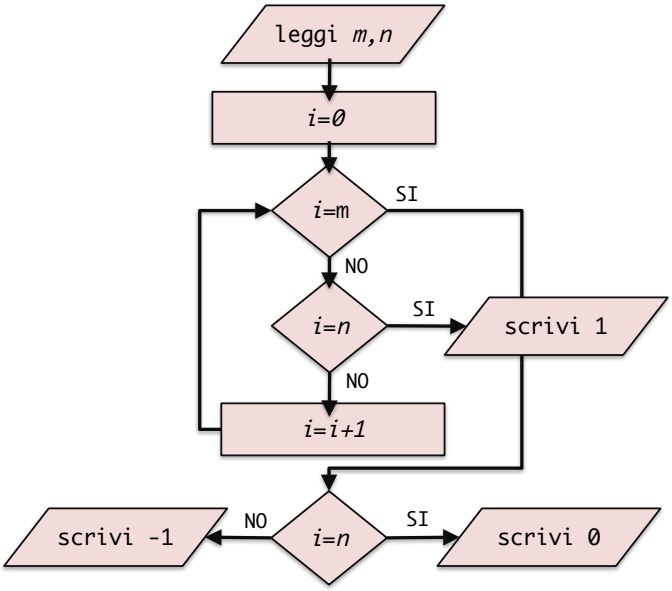


Figura 2: Diagramma di flusso per il confronto di due numeri

predecessore di un numero naturale.
Ricordiamo innanzitutto che il predecessore è definito come segue:

$$\begin{aligned} pred(0) &= 0 \\ pred(n+1) &= n \end{aligned}$$

Osservate che abbiamo definito la funzione su tutti i numeri naturali, definendo il valore della funzione predecessore su 0 e sui numeri naturali che sono successori di un altro naturale.

A dispetto della semplicità, è necessario avere una idea precisa per risolvere questo problema. Com'è possibile togliere 1 a un numero, sapendo solo sommare 1? Nelle procedure precedenti abbiamo più volte contato fino a n, ripetendo alcune operazioni esattamente n volte. Questa volta si tratterebbe di ripetere un'operazione esattamente $n-1$ volte, senza ovviamente saper calcolare il valore $n-1$. Come fare?

Usando due contatori, tenendo uno un passo indietro all'altro, quando il primo avrà raggiunto il valore n, il secondo sarà fermo al valore $n-1$. Per far ciò sarà sufficiente far partire un contatore da 1 e il secondo da 0.

1: leggi n;
2: $j \leftarrow 1$;
3: $i \leftarrow 0$;
4: if $j = n$ then scrivi i; exit;
5: $j \leftarrow j + 1$;

6: $i \leftarrow i + 1$;
7: goto 4;

È opportuno chiedersi cosa accada nel caso patologico in cui n sia zero: si tratta di una precauzione standard: anche quando abbiamo trovato un metodo risolutivo generale, è bene verificare cosa succede nei casi 'limite' (questi dipendono dal problema e in generale farà parte del bagaglio di un buon programmatore individuarli e trattarli correttamente).

Nel nostro caso, l'indice j, inizializzato a 1, continuerà a crescere, non raggiungendo mai il valore di n, che è 0. L'esecuzione non termina mai: in generale è sempre bene porsi il problema della terminazione dei programmi. Possiamo viceversa convincerci facilmente, eseguendo mentalmente, o meglio, con carta e penna, che la procedura funziona quando n assume i valori 1, 2, 3 ...

Osserviamo che questo programma potrebbe essere considerato corretto se assumessimo che non ha senso fare il predecessore di 0. In generale, la correttezza di un programma non è assoluta, ma relativa a una aspettativa sui risultati che il programma deve calcolare: tale aspettativa è chiamata *specifica*, e di solito esprime in un linguaggio logico-matematico la descrizione di:

- *precondizioni*, ossia le assunzioni che ci aspettiamo sui dati di ingresso;
- *postcondizioni*, ossia una relazione tra dati di ingresso e risultati finali.

Quindi, il programma appena visto, implementa correttamente la funzione matematica $pred'$ specificata da:

$$pred'(n) \quad = \quad n - 1, \quad n > 0$$

che non specifica il valore della funzione sul valore 0. Il programmatore ha il diritto di formulare una sorta di contratto, in cui richiede (come precondizione) che il numero di ingresso sia un numero strettamente positivo. Viceversa, per soddisfare la specifica precedente è necessario trattare a parte il caso $n = 0$, anteponendo un opportuno controllo:

1: leggi n;
2: if $n = 0$ then scrivi 0; exit;
3: $j \leftarrow 1$;
4: $i \leftarrow 0$;
5: if $j = n$ then scrivi i; exit;
6: $j \leftarrow j + 1$;
7: $i \leftarrow i + 1$;
8: goto 5;

Ancora una volta cerchiamo di trarre un insegnamento generale: cercate di fare un'analisi dei casi rilevanti nella soluzione dei problemi.

2.4 Invarianti

Un altro aspetto interessante è che ad ogni esecuzione del ciclo, è soddisfatta la relazione matematica:

$$i = j - 1$$

questo perché è vera grazie alle inizializzazioni dei valori di i e j, e rimane vera perché entrambe le variabili ad ogni esecuzione del ciclo vengono incrementate di 1. Proprietà di questo tipo (valide durante tutta l'esecuzione di un ciclo) si dicono proprietà *invarianti*: esse formalizzano l'idea che dietro alla ripetizione di una sequenza di istruzioni ci sia una uniformità nei valori che le variabili (o se preferite, lo stato della computazione) assumono durante l'esecuzione di un ciclo. Osserviamo anche che la proprietà invariante $i = j - 1$, nel momento in cui si esce dal ciclo (cioè quando $j = n$), implica che $i = n - 1$, che è proprio la proprietà matematica che desideriamo dal risultato della procedura risolutiva.

Concludiamo l'analisi di questo problema, osservando ancora due fatti:

- benché ogni istruzione sia etichettata con un numero, sono rilevanti solo i numeri delle istruzioni che sono il punto di arrivo di un salto; potremmo pensare di usare un linguaggio ulteriormente semplificato in cui, solo occasionalmente le istruzioni hanno una *etichetta* o *label*.
- è possibile scrivere l'algoritmo precedente, facendo uso di una sola variabile contatore.

Vediamo, alla luce di queste due osservazioni, una nuova procedura risolutiva:

leggi n;
if $n = 0$ then scrivi 0; exit;
$i \leftarrow 0$;
l : if $i + 1 = n$ then scrivi i; exit;
$i \leftarrow i + 1$;
goto l;

Osservate che nel test di uguaglianza $i + 1 = n$, il valore della casella di memoria chiamata i **non** viene modificata: le modifiche (per ora), avvengono solo per effetto di azioni di assegnazione.

Problema 4: *Sia dato un esecutore capace solamente di testare l'uguaglianza di due numeri e di fare somme. Scrivere una procedura risolutiva che calcola il prodotto di due numeri naturali*

Come molti di voi sanno, possiamo calcolare un prodotto come segue:

$$m \times n = \underbrace{m + m + \ldots + m}_{n \text{ volte}}$$

Si tratta ancora una volta di iterare un'operazione un numero prefissato di volte. Osservate tuttavia, che iterare l'azione: $m \leftarrow m + m$; sarebbe scorretto, perché modificherebbe il contenuto della variabile da sommare. È necessaria una casella di memoria ausiliaria, *prod* in cui accumulare le somme parziali: tale casella viene inizializzata al valore 0 (elemento neutro della somma) per non influenzare il calcolo della sommatoria.

1: leggi m; leggi n;
2: $prod \leftarrow 0$;
3: $i \leftarrow 0$;
4: if $i = n$ then scrivi p; exit;
5: $prod \leftarrow prod + m$;
6: $i \leftarrow i + 1$;
7: goto 4;

Anche in questo caso è interessante notare che il programma termina sotto la precondizione $n \geq 0$ e che durante il ciclo nelle linee 4–7 vale la relazione $prod = m \cdot i$, che, come prima, quando $i = n$ implica la postcondizione desiderata ($prod = m \cdot n$).

Problema 5: *Sia dato un esecutore capace solamente di testare la relazione di minore tra due numeri e di fare somme e differenze. Scrivere una procedura risolutiva che calcola quoziente e resto della divisione tra due numeri naturali*

In quest'ultimo problema consideriamo un esecutore molto più abile in grado di eseguire anche somme e differenze e testare la relazione di minore. Il lettore attento avrà comunque già intuito come la differenza si possa fare iterando l'operazione di predecessore (problema 3), mentre il calcolo del minore è in buona sostanza stato descritto nella soluzione del problema 2.

Ricordiamo, che il problema della divisione intera tra due numeri naturali n ed m consiste nel trovare due numeri interi q (quoziente) ed r (resto) per qui valga la seguente relazione:

$$m = qn + r \quad r < n$$

In questo caso itereremo la sottrazione di n da m, finché il resto non sia minore di n. La variabile q memorizzerà il numero di volte in cui eseguo la sottrazione. Osservate che, inizializzando il valore di q a 0 e di r al valore di m, la relazione

$$m = qn + r$$

sarà soddisfatta banalmente all'ingresso del ciclo. Mantenendo questa relazione (a ogni iterazione sommando 1 a q e sottraendo n da r) e smettendo di fare le sottrazioni non appena r diventa minore di n otterremo la soddisfazione della specifica desiderata:

1: leggi m; leggi n;
2: $q \leftarrow 0$;
3: $r \leftarrow m$;
4: if $r < n$ then scrivi q, r; exit;
5: $r \leftarrow r - n$;
6: $q \leftarrow q + 1$;
7: goto 4;

2.5 Esercizi e Spunti di Riflessione

1. Considerando un esecutore capace solo di sommare e sottrarre 1 e testare se il contenuto di una variabile è uguale o diverso da 0, scrivere un programma per testare l'uguaglianza di due numeri.

2. Dettagliare la procedura risolutiva per la moltiplicazione, supponendo che il nostro esecutore non sia in grado di fare le somme, ma solo sommare 1.

3. Considerando un esecutore capace solo di sommare 1 e testare l'uguaglianza, scrivere una procedura risolutiva che scriva 1 se un numero letto in ingresso è dispari, e 0 altrimenti.

4. Scrivere una procedura risolutiva per calcolare l'esponenziale come iterazione di prodotti.

5. Scrivere una procedura risolutiva che calcola la somma dei primi n numeri.

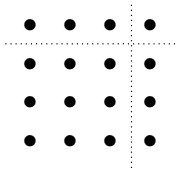
6. Scrivere una procedura che somma una sequenza di numeri positivi introdotta dall'esterno. Supporre che la sequenza termini quando viene introdotto uno 0.

7. Scrivere una procedura risolutiva che calcola il quadrato di un numero n come somma dei primi n numeri dispari. Infatti:

   $$1^2 = 1 = 1 \qquad 2^2 = 4 = 1 + 3 \qquad 3^2 = 9 = 1 + 3 + 5$$

   e così via.

   Questo problema ha un interessante interpretazione geometrica:

   (Per passare dal quadrato di lato n a quello di lato $n + 1$ bisogna aggiungere due file di n pallini, più un ultimo pallino, cioè $2n + 1$ pallini).

   Per chi preferisce l'algebra, banalmente $(n + 1)^2 = n^2 + 2n + 1$.

8. Scrivere una procedura che letti tre numeri, li stampi ordinati. Disegnate il diagramma di flusso corrispondente.

9. Scrivere una procedura risolutiva che calcola la radice quadrata intera di n, cioè trovi un numero r, tale che $r^2 \leq n < (r+1)^2$.

10. Scrivere una procedura risolutiva che scrive 1 se n è primo, e 0 altrimenti.

11. Scrivere una procedura risolutiva che scrive il più grande numero primo che divide n.

12. Scrivere una procedura risolutiva che calcola il logaritmo base 2 intero di n, cioè trovi un numero r, tale che $2^r \leq n < 2^{r+1}$.