Rappresentazione di algoritmi, strutture di controllo

Lezione 2 di Fondamenti di informatica

Docente: Giuseppe Scollo

Università di Catania
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica, I livello, AA 2009-10

Indice

astrazione e raffinamento

la definizione di algoritmo, nelle sue diverse varianti, esige l'**eseguibilità** dei passi, o l'**effettiva computabilità** delle operazioni, da parte dell'esecutore o agente di calcolo

> nella **descrizione** di un algoritmo si fa dunque riferimento, spesso implicito, alle effettive capacità dell'esecutore, o al suo **livello cognitivo**

ad esempio, una descrizione più concisa dell'algoritmo di cottura dell'uovo alla coque già visto può presupporre che l'esecutore sappia riconoscere un uovo fresco, e abbia qualche familiarità con l'uso dei fornelli:

1. accendi una fiamma vivace sotto un pentolino con un uovo fresco completamente immerso in acqua fredda
2. all'ebollizione, modera la fiamma e spegni dopo 100 secondi circa
3. blocca la cottura temperando con acqua fredda

questa descrizione è a un più alto **livello di astrazione** rispetto alla precedente perché, appunto, astrae da alcuni dettagli, inutili al presupposto livello cognitivo dell'esecutore

> per converso, la descrizione più dettagliata è un **raffinamento** di quella ora vista perché ne precisa alcuni passi, sì da renderli eseguibili da un agente meno esperto

livelli di astrazione

astrazione e raffinamento sono concetti fondamentali, inerenti qualsiasi genere di descrizione, non solo quella di algoritmi

si costituisce naturalmente una **gerarchia di livelli di astrazione** in base alla presenza di dettagli, a un livello inferiore, dai quali si astrae al livello superiore

> la struttura di un libro in: indice, capitoli, sezioni, sottosezioni, paragrafi, è un esempio familiare di organizzazione dell'informazione per successivi livelli di astrazione (decrescente)

nell'evoluzione storica dei linguaggi di programmazione è facilmente riconoscibile una tendenza verso livelli di astrazione sempre più alti, cioè meno vincolati a caratteristiche degli agenti di calcolo

> astrazione e raffinamento danno anche luogo a importanti **principi metodologici**, utili al progetto di algoritmi, esposti nella prossima lezione

al momento, questi concetti ci tornano utili per dotarci di una notazione agevole per la **rappresentazione di algoritmi**

primitive di una rappresentazione

i calcolatori non sono gli esclusivi destinatari di descrizioni di algoritmi
spesso accade di voler descrivere una soluzione algoritmica di un problema per comunicarla ad altre persone

il linguaggio naturale sembra conveniente a tal fine, ma si presta ad ambiguità, e inoltre non fornisce supporto immediato ai costrutti più tipici degli algoritmi; d'altra parte, i linguaggi di programmazione pongono vincoli formali spesso eccessivi a fini di comunicazione

le **primitive** di una rappresentazione ne sono i costituenti di base, cioè sono:
le sue componenti elementari, al livello di astrazione della rappresentazione costituite da termini che designano azioni **interpretabili ed eseguibili** dall'esecutore (reale o presupposto) dell'algoritmo

tipici esempi di primitive in descrizioni di algoritmi sono:
lettura/scrittura di un valore in ingresso/uscita
assegnamento di un valore a una variabile
simboli di operazioni aritmetiche e logiche

descrizione di algoritmi in pseudocodice

oltre alle primitive, la rappresentazione di un algoritmo necessita di
regole di composizione

queste si applicano alle primitive, come pure a **espressioni composte** risultanti dall'applicazione delle regole

lo **pseudocodice** è una tecnica informale di descrizione di algoritmi nella quale:
le **primitive** sono espressioni il cui significato si assume noto
le regole di composizione sono espresse da **costrutti** in linguaggio naturale, simili alle strutture di controllo più frequenti nei linguaggi di programmazione, che esaminiamo fra breve: **sequenza, alternativa, iterazione**, etc.

in sostanza, la rappresentazione di un algoritmo in pseudocodice ha le stesse caratteristiche strutturali delle rappresentazioni in linguaggi di programmazione (di alto livello), ma non è soggetta ai rigidi vincoli sintattici di tali linguaggi

ricerca del massimo in un insieme

facciamoci una prima idea della rappresentazione in pseudocodice attraverso un semplice esempio: la ricerca del massimo in un insieme finito di numeri
ipotizziamo che l'insieme sia rappresentato da una sequenza di numeri naturali in ingresso, la fine della quale è segnalata dall'immissione di un numero negativo
conveniamo che **v** ← **E** designi l'assegnamento del valore dell'espressione **E** alla variabile **v**

```
funzione-I/O  massimo
candidato ← -1
leggi nuovo
finché nuovo ≥ 0 ripeti:
   se nuovo > candidato  allora candidato ← nuovo
   leggi nuovo
fine ciclo
se candidato ≥ 0  allora scrivi candidato  altrimenti scrivi "insieme vuoto?"
```

strutture condizionali

quali **primitive** si usano nell'esempio visto?

- assegnamento: "←"
- operatori di confronto (relativi all'ordinamento dei numeri): "≥", ">"
- dichiarazione di funzione ingresso/uscita: **funzione-I/O** *nome*
- lettura dal canale d'ingresso: **leggi** *variabile*
- scrittura sul canale di uscita: **scrivi** *espressione*

gli operatori di confronto sono di tipo ***booleano***, cioè formano, con i loro argomenti, espressioni la cui valutazione dà un valore di verità: ***vero*** o ***falso***; nell'esempio visto si hanno tre espressioni di tal tipo, costituenti la *condizione* in **strutture di controllo**
strutture condizionali:

- **se** *condizione* **allora** *pseudocodice*
- **se** *condizione* **allora** *pseudocodice* **altrimenti** *pseudocodice*

il costrutto **finché** *condizione* **...** **fine ciclo** è una struttura di controllo **iterativa** :
lo esaminiamo appresso, assieme ad altre strutture di questo tipo

strutture iterative

una struttura di controllo implicita nelle descrizioni di algoritmi è la **sequenza:** in assenza di costrutti che alterino l'ordine di esecuzione, si intende che i passi dell'algoritmo vadano eseguiti nell'ordine in cui sono descritti

come vedremo fra poco, sequenza e iterazione giocano un ruolo fondamentale nella costruzione di algoritmi

stato: la mappa dei valori assegnati alle variabili, in un dato momento dell'esecuzione

ingredienti essenziali di una struttura iterativa (o **ciclo**, ingl. ***loop***) :

inizializzazione : stabilisce uno **stato iniziale** per l'iterazione; lo stato è modificabile nell'esecuzione iterativa del corpo del ciclo

condizione di continuazione : da valutare ad ogni iterazione; quando è falsa, l'esecuzione iterativa termina

corpo del ciclo : costrutto eseguibile iterativamente; di solito **modifica lo stato**, fino a rendere falsa la condizione di continuazione

N.B. talvolta l'inizializzazione è realizzata esternamente alla struttura iterativa propriamente detta, che è composta in sequenza con essa

cicli a condizione anticipata e posticipata

pseudocodice: **finché** *condizione* **ripeti:** *corpo del ciclo* **fine ciclo**

in questa struttura l'inizializzazione è esterna, e precede in sequenza il primo controllo della condizione di continuazione

pseudocodice: **ripeti:** *corpo del ciclo* **finché** *condizione* **fine ciclo**

anche in questa struttura l'inizializzazione è esterna, e precede in sequenza la prima esecuzione del corpo del ciclo

almeno una esecuzione del corpo del ciclo **avviene sempre**, cioè indipendentemente dal valore della condizione di continuazione, poiché quest'ultima viene valutata solo alla fine di ciascuna iterazione

N.B. attenzione alla traduzione inglese ***repeat ... until*** *condition* ***end loop*** : *"**until**"* significa **finché non**, dunque *condition* è una **condizione di terminazione**, ovvero l'opposta della condizione di continuazione

la traduzione che, invece, corrisponde allo pseudocodice in questione è: ***repeat ... while*** *condition* ***end loop***

ricerca di un nome in un elenco

ipotizziamo che il nome da cercare, e il nome di un file contenente l'elenco, siano i due argomenti della funzione desiderata, che produrrà in uscita un numero naturale; questo indicherà la posizione, a partire da 1, (della prima occorrenza) del nome nell'elenco, se il nome è nell'elenco, altrimenti sarà 0

estendiamo le primitive dello pseudocodice con

da *file* **leggi** *variabile*: per la lettura sequenziale di un elemento (nome) da un file

eof *file*: condizione vera sse si è tentato di leggere oltre l'ultimo elemento di *file*

operatori booleani di significato immediato

```
funzione-I/O   trova (nome, file)
posizione ← 0
ripeti:
  da file leggi elemento
  se non eof file allora posizione ← posizione + 1
finché non ( elemento = nome oppure eof file )
fine ciclo
se eof file allora scrivi 0 altrimenti scrivi posizione
```

strutture di controllo fondamentali

il ciclo a condizione anticipata è **la** struttura iterativa fondamentale:

tutte le altre strutture iterative possono essere realizzate mediante (uso opportuno di) sequenza e ciclo a condizione anticipata

non vale il converso

versione ridotta del **Teorema di Böhm-Jacopini:**

sequenza e **ciclo a condizione anticipata** bastano a costruire qualsiasi algoritmo

infatti, ad es., si possono realizzare le strutture condizionali viste prima come segue:

se *condizione* **allora** *pseudocodice*

con un ciclo a condizione anticipata che venga eseguito al più una volta

se *condizione* **allora** *pseudocodice* **altrimenti** *pseudocodice*

con due cicli a condizione anticipata, ciascuno dei quali venga eseguito al più una volta, e aventi opposte condizioni di continuazione (purché l'esecuzione del corpo del primo ciclo non influisca sulla condizione di continuazione del secondo ciclo)

ricerca di un gene in una molecola DNA

problema: ricerca delle occorrenze di una sequenza (contigua) di simboli $S = S_1 \ldots S_k$ in una data sequenza $M = M_1 \ldots M_n$ di simboli dello stesso alfabeto $\{A,T,C,G\}$, con $n > k$

si vuole una funzione che, date le due sequenze come argomenti, produca in uscita la sequenza, eventualmente vuota, delle posizioni iniziali delle occorrenze di S in M

assumiamo di disporre delle seguenti operazioni primitive su sequenze:

lunghezza (X): dà il numero di occorrenze di simboli nella sequenza X

estrai (X, p, k): produce la sottosequenza contigua, di lunghezza k, della sequenza X a partire dalla posizione p (sia $p \geq 1$ e la lunghezza di X sia almeno $p+k-1$)

$X = Y$: vera sse X e Y hanno la stessa lunghezza e $X_i = Y_i$ per ogni posizione i

```
funzione-I/O  trova_gene (S, M)
posizione ← 1
k ← lunghezza(S)
n ← lunghezza(M)
finché posizione + k - 1 ≤ n ripeti:
    se S = estrai(M, posizione, k) allora scrivi posizione
    posizione ← posizione + 1
fine ciclo
```