

Principi di Programmazione

Sommario

- Costruzione di algoritmi
- Approccio alla soluzione di problemi complessi
- Sequenza; Selezione; Iterazione; Ricorsione
- Rappresentazione di algoritmi
 - flow chart
 - linguaggio lineare
- Programmazione Strutturata

Algoritmi: Costruzione

- **Esaminare** una specifica realtà o problema
- **Costruirne** un'astrazione
- **Rappresentarla** (più o meno) formalmente
- **Individuare** una sequenza di azioni che, eseguite, risolvano il problema nel mondo dell'astrazione
 - Il processo di analisi e astrazione è difficile da dominare per problemi complessi

Tipi di problemi

- Primitivi
 - Risolubili mediante
 - Un'azione primitiva
 - Una sequenza di azioni primitive
- Complessi
 - Non risolvibili tramite un'azione nota
 - Al solutore
 - All'esecutore
- **Ha senso occuparsi dei problemi complessi!**

Problemi complessi: Soluzione

- Scomposizione del problema in sottoproblemi
 - fino a trovare un insieme di sottoproblemi primitivi che risulti equivalente al problema di partenza
- Individuazione del procedimento che porta alla soluzione
 - Processo di cooperazione tra sottoproblemi dei quali si è in grado di fornire facilmente una soluzione

Problemi complessi: Approccio

- Principio **DIVIDE ET IMPERA**
 - Ridurre la complessità del problema
 - Scomposizione in sottoproblemi, la cui risoluzione è più semplice
 - Individuare le opportune strutture e relazioni
 - Realizzare un algoritmo per ogni sottoproblema
 - Se un sottoproblema continua ad essere troppo complesso, riapplicare ad esso la scomposizione fino ad arrivare a problemi primitivi
 - risolvibili tramite azioni note

Scomposizione di problemi (1)

- Tecnica per raffinamenti successivi
 - Uno dei principi della programmazione strutturata
 - Basata sulla trasformazione di un problema in una gerarchia di problemi di difficoltà decrescente
 - Di un problema si affronta, prima, l'aspetto più generale e si passa, poi, a livelli sempre più dettagliati di descrizione sino ad arrivare agli elementi fondamentali

Scomposizione di problemi (2)

- Conseguenze
 - Aumento del numero di problemi da risolvere
 - Diminuzione della complessità di ciascuno
- Può capitare che, per risolvere il problema complessivo, uno stesso sottoproblema debba essere risolto più volte
 - considerando ogni volta valori diversi dei dati

Scomposizione di Problemi: Requisiti

- Ogni passo della suddivisione deve garantire che
 - la soluzione dei sottoproblemi conduca alla soluzione generale
 - la successione di passi da eseguire abbia senso e sia possibile
 - la suddivisione scelta dia luogo a sottoproblemi “più vicini” agli strumenti disponibili
 - Risolubili direttamente con gli operatori a disposizione

Scomposizione di Problemi: Quando fermarsi?

- Bisogna arrivare al punto in cui
 - Tutti i problemi sono primitivi
 - È fissato l'ordine di esecuzione dei sottoproblemi
 - È previsto il modo in cui un sottoproblema utilizza i risultati prodotti dalla soluzione dei sottoproblemi che lo precedono
 - Livello di cooperazione

Scomposizione di Problemi: Livelli

- I problemi ottenuti dalla scomposizione sono
 - Indipendenti
 - Si può progettare un algoritmo per ciascuno
 - La soluzione può essere delegata a solutori diversi (in parallelo)
 - Cooperanti
 - Ciascuno può usare il risultato della soluzione di altri
- Più complesso è il problema, più livelli saranno necessari in profondità
- Uno stesso problema può essere scomposto in modi diversi

Scomposizione di Problemi: Tecniche

- I linguaggi di programmazione strutturata offrono costrutti per scomposizioni
 - Sequenziale
 - Suddividere un problema in parti disgiunte
 - Selettiva
 - Trattamento differenziato in casi differenti
 - Iterativa
 - Ricorsiva

Scomposizione Sequenziale

- La soluzione si ottiene tramite una sequenza di passi
 - I passi sono eseguiti **uno alla volta**
 - Ogni passo è eseguito **una sola volta**
 - Nessuno è ripetuto o omesso
 - L'ordine in cui i passi vanno eseguiti è lo stesso in cui sono scritti
 - L'ultimo passo equivale alla terminazione del procedimento

Scomposizione Sequenziale: Esempio (1)

- Problema: Preparare una tazza di tè
- Soluzione: un algoritmo per la preparazione della tazza di tè
 - Passo1: Bollire l'acqua
 - Passo2: Mettere il tè nella tazza
 - Passo3: Versare l'acqua nella tazza
 - Passo4: Lasciare in infusione
- Non sono problemi primitivi
 - Ciascuno va ulteriormente scomposto

Scomposizione Sequenziale: Esempio (2)

- Bollire l'acqua:
 - Riempire d'acqua il bollitore
 - Accendere il gas
 - Aspettare che l'acqua bolla
 - Spegnerne il gas
- Mettere il tè nella tazza:
 - Aprire la scatola del tè
 - Prendere un sacchetto-filtro
 - Chiudere la scatola del tè
 - Mettere il sacchetto nella tazza
- Versare l'acqua nella tazza:
 - Versare l'acqua bollente nella tazza finché non è piena
- Lasciare in infusione:
 - Aspettare 3 minuti
 - Estrarre il sacchetto-filtro

Scomposizione Selettiva

- Soluzione ottenuta tramite scelte multiple
 - Strutture di selezione all'interno di altre strutture di selezione
- La scomposizione fa ricorso a strutture multiple
 - Nidificazione (o annidamento) di strutture

Scomposizione Selettiva: Esempio (1)

- Problema: trovare il maggiore fra tre numeri **a**, **b**, **c** (diversi tra di loro)
 - Supponiamo che verificare se un numero è maggiore di un altro sia un'azione primitiva

Scomposizione Selettiva: Esempio (2)

- Algoritmo:
 - se** $a > b$
 - allora se** $a > c$
 - allora** a è la soluzione
 - altrimenti** c è la soluzione
 - altrimenti se** $b > c$
 - allora** b è la soluzione
 - altrimenti** c è la soluzione

Scomposizione Iterativa

- Successione di problemi tutti dello stesso tipo
 - Ripetere un **numero finito di volte** un passo di soluzione in modo da ottenere, alla fine, la soluzione completa

Scomposizione Iterativa: Requisiti

- Individuare una catena di sottoproblemi che:
 - sono uguali, oppure
 - differiscono solo per i dati su cui agiscono
- ed inoltre i dati su cui agiscono
 - sono uguali, oppure
 - sono in una qualche relazione d'ordine
 - ciascun problema differisce dal precedente perché opera sul dato successivo

Scomposizione Iterativa: Esempio (1)

- Trovare il più grande fra $n > 3$ numeri tutti diversi fra loro
 - Supponiamo che i dati siano in relazione d'ordine
 - Si può parlare di 1° dato, 2° dato, ... , n-esimo dato

Scomposizione Iterativa: Esempio (2)

Lo stesso passo (trovare il più grande) è ripetuto $n-1$ volte, su diverse coppie di dati

- Trova il più grande tra i primi 2 numeri
 - Trova il più grande fra il risultato del problema precedente e il 3° numero
 - ...
 - Trova il più grande fra il risultato del problema precedente e l'ultimo numero (n-esimo dato)
- Più concisamente:
 - Trova il più grande fra i primi 2 numeri
 - **Finché** ci sono numeri da esaminare esegui
 - Esamina il primo numero non ancora considerato
 - Trova il più grande tra questo e il più grande precedentemente trovato

Oggetto Ricorsivo

- Definito in termini di se stesso
 - Esempi
 - Numeri naturali:
 - 1 è un numero naturale
 - Il successore di un numero naturale è un numero naturale
 - Strutture ad albero:
 - • è un albero (albero vuoto)
 - Se $t1$ e $t2$ sono alberi, allora anche la struttura
 -



Definizione Ricorsiva

- Ottenuta in termini di versioni più semplici della stessa cosa che definisce
- 2 livelli:
 - Passo
 - Operazione che riconduce la definizione ad un dato livello a quella di livello inferiore
 - Base (o livello assiomatico)
 - Definizione di partenza
 - Necessario per ricostruire i livelli successivi
 - Consente di definire problemi di ordine superiore

Ricorsione: Esempio



Scomposizione Ricorsiva

- Un problema di ordine n è definito in termini del medesimo problema di ordine inferiore
 - Entità definita in termini più semplici della stessa entità
 - Nella definizione deve apparire il livello assiomatico (o di ricostruzione)
 - Problema risolubile tramite un'azione primitiva

Scomposizione Ricorsiva: Requisiti

- Almeno uno dei sottoproblemi è formalmente uguale al problema di partenza, ma di ordine inferiore
- Al momento della scomposizione è noto l'ordine del problema
 - Scomposizione mediante regola di copiatura
 - Sostituire ad ogni occorrenza del problema la scomposizione ricorsiva
 - fino ad avere problemi primitivi (raggiungere il livello assiomatico)

Scomposizione Ricorsiva: Esempio (1)

- Invertire una sequenza di lettere
 - se la sequenza contiene una sola lettera allora
 - Scrivila (il problema è risolto)
 - altrimenti:
 - Rimuovi la prima lettera dalla sequenza
 - Inverti la sequenza rimanente
 - Appendi in coda la lettera rimossa

Scomposizione Ricorsiva Esempio (2)

- Invertire “roma”
 - “roma” (4 lettere): rimuovi ‘r’, inverti “oma”
 - “oma” (3 lettere): rimuovi ‘o’, inverti “ma”
 - “ma” (2 lettere): rimuovi ‘m’, inverti “a”
 - » “a” (1 lettera): è già invertita
 - Appendi ‘m’: “am”
 - Appendi ‘o’: “amo”
 - Appendi ‘r’: “amor”
- Risultato: “amor”

Scomposizione Ricorsiva: Caratteristiche

- Un problema esprimibile ricorsivamente si può risolvere iterativamente
- Una funzione esprimibile ricorsivamente è computabile
 - Esiste un algoritmo che la calcola e termina sempre
- Ogni funzione computabile per mezzo di un programma è ricorsiva

Iterazione e Ricorsione: Esempio (1)

- Calcolo del prodotto di due interi n e m
 - Definizione iterativa:
 - $n * m = m + m + \dots + m$ (n volte)
 - Definizione ricorsiva:
$$n * m = \begin{cases} 0 & \text{se } n = 0 \\ (n - 1) * m + m & \text{se } n > 0 \end{cases}$$

Iterazione e Ricorsione: Esempio (2)

- Soluzione iterativa
 - inizialmente sia il risultato uguale a 0
 - ripeti per ogni intero da 1 a n
 - somma m al risultato per ottenere un nuovo risultato
 - il prodotto è il risultato finale
- Soluzione ricorsiva
 - se n è uguale a 0
 - allora il prodotto è 0
 - altrimenti
 - calcola il prodotto di n - 1 per m
 - somma m al prodotto

Scomposizione di Problemi: Conclusioni

- Può esserci più di una scomposizione di un problema in sottoproblemi
- Cause di difficoltà nella scomposizione
 - Comprensione intuitiva della complessità di un problema
 - Scelta tra diverse possibili scomposizioni
 - Necessità di una formulazione “adeguata” per ciascun sottoproblema

Rappresentazione degli algoritmi

- Negli esempi, gli algoritmi sono rappresentati in un linguaggio simile a quello naturale
- Vantaggi
 - Intuitività
 - Facilità di scrittura
- Svantaggi
 - Ambiguità
 - Ridondanza
 - Scarso rigore

Diagrammi di Flusso

- Linguaggio grafico tipicamente utilizzato per trasmettere ad un esecutore umano la descrizione di un algoritmo o processo
 - Si parte dal punto iniziale
 - Si seguono i percorsi indicati, intraprendendo le azioni che via via si incontrano
 - In caso di percorsi alternativi, se ne sceglie uno a seconda della condizione specificata
 - fino al raggiungimento del punto finale

Diagrammi di Flusso Elementi Costitutivi

- Operazioni

- Calcolo



- Ingresso/Uscita



- Decisione



- Controllo

- Inizio/Fine



- Flusso



- Connessione



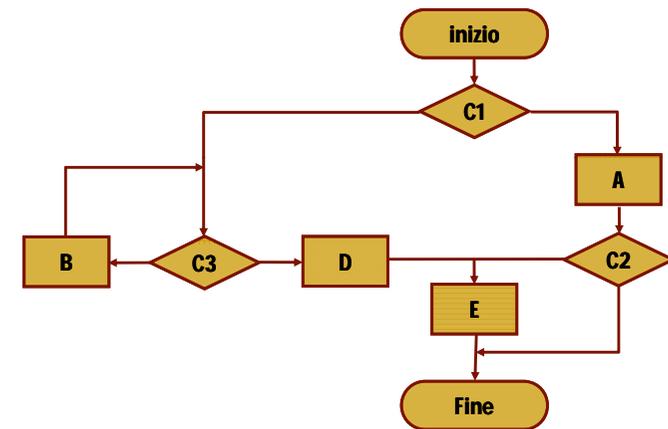
Diagrammi di Flusso Regole di Costruzione

- Un solo blocco iniziale e un solo blocco finale
 - Ogni blocco è raggiungibile da quello iniziale
 - Il blocco finale è raggiungibile da ogni blocco
- I blocchi sono in numero finito
 - Ogni blocco di azione (calcolo o ingresso/uscita) ha una freccia entrante ed una uscente
 - Ogni blocco di decisione ha una freccia entrante e due uscenti
- Ogni freccia parte da un blocco e termina in un blocco o su un'altra freccia

Principi di Programmazione

37

Diagrammi di Flusso Esempio



Principi di Programmazione

38

Diagrammi di Flusso: Vantaggi

- Grafici
 - Adatti agli esseri umani
 - Adatti a rappresentare processi sequenziali
 - Immediatamente visualizzabili
- Rispondono all'esigenza di divisione del lavoro
- Documento base per l'analisi organica
- Non ambigui
- Traducibili in vari linguaggi di programmazione

Principi di Programmazione

39

Diagrammi di Flusso: Svantaggi

- Spesso non entrano in una pagina
 - Difficili da seguire e modificare
- Non naturalmente strutturati
 - Spesso le modifiche portano a de-strutturazione
- Lontani dai linguaggi dei calcolatori
 - Possono rivelarsi errati in fase di programmazione

Principi di Programmazione

40

Programmazione Strutturata

- Uso di schemi fondamentali
 - Uso di soli diagrammi strutturati
 - Configurazioni standard di blocchi elementari, comuni a molti processi della vita quotidiana
- Sviluppo per raffinamenti successivi
 - Ogni schema fondamentale ha un solo punto di ingresso ed un solo punto di uscita
 - Sostituibile ad un blocco di azione
 - Nella sostituzione, si possono omettere i blocchi di inizio e fine dello schema che si sta inserendo

Programmazione Strutturata: Schemi fondamentali (1)

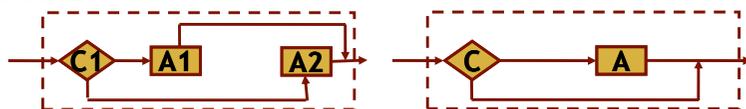
- Sequenza
 - Concatenazione di azioni
- Selezione
 - Scelta di azioni alternative
 - Dipendenza da una condizione
- Iterazione
 - Ripetizione di una certa azione
 - Dati potenzialmente diversi
 - Dipendenza da una condizione

Programmazione Strutturata: Schemi fondamentali (2)

- Sequenza



- Selezione



- Iterazione



Diagrammi Strutturati (1)

- Base: Dato un blocco di azione A,



è strutturato.

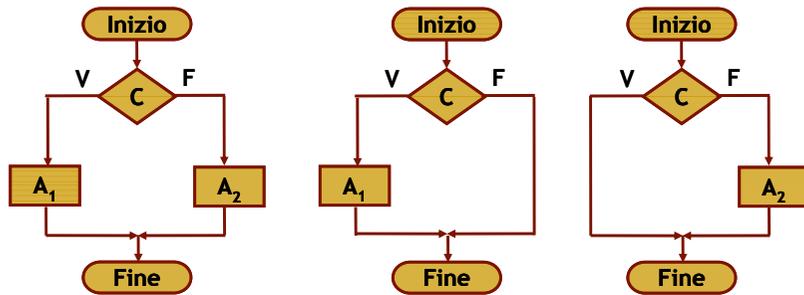
- Sequenza: Se A1, ..., An sono strutturati,



è strutturato

Diagrammi Strutturati (2)

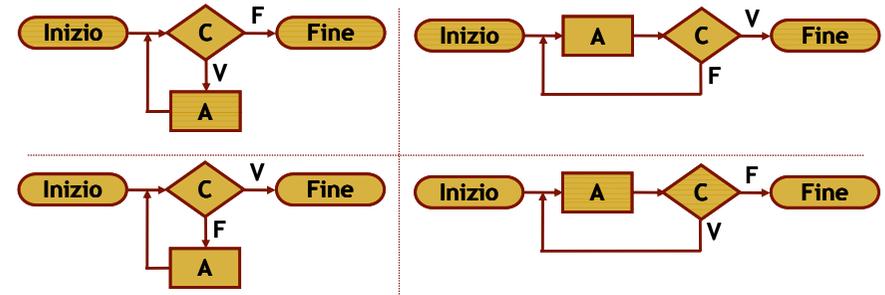
- Selezione: Se A1 e A2 sono strutturati,



sono strutturati

Diagrammi Strutturati (3)

- Iterazione: Se A è strutturato allora...

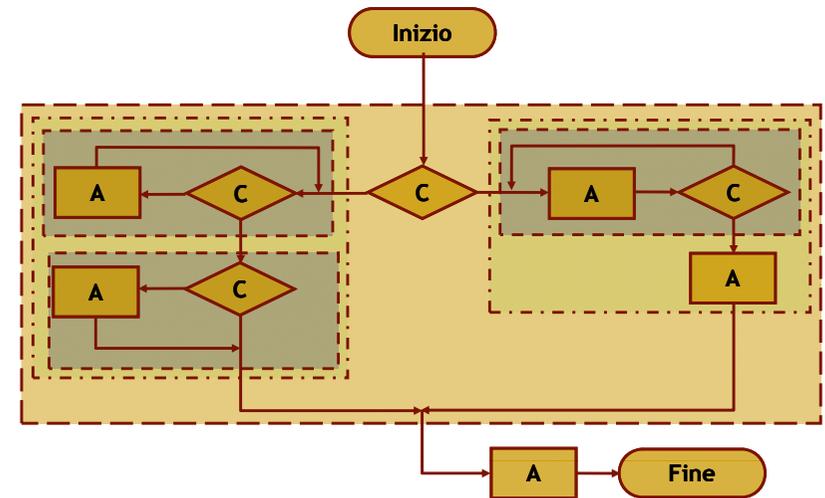


sono diagrammi strutturati

Diagrammi Strutturati (4)

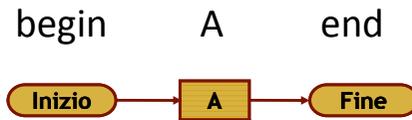
- Nessun altro diagramma è strutturato
- Note:
 - La definizione è ricorsiva

Diagrammi Strutturati Esempio



Linguaggio Lineare Sequenza

- Costrutto base:

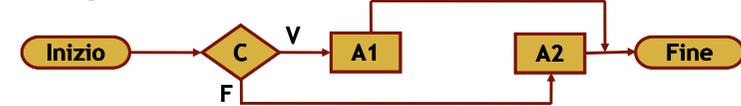


- Inoltre, ogni blocco di azione si può sostituire con uno dei seguenti costrutti



Linguaggio Lineare Selezione

- begin if C then A1 else A2 end



- begin if C then A end



- begin if not C then A end



Linguaggio Lineare Iterazione (while...do)

- begin while C do A end



- begin while not C do A end

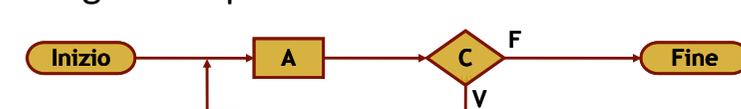


Linguaggio Lineare Iterazione (repeat...until)

- begin repeat A until C end

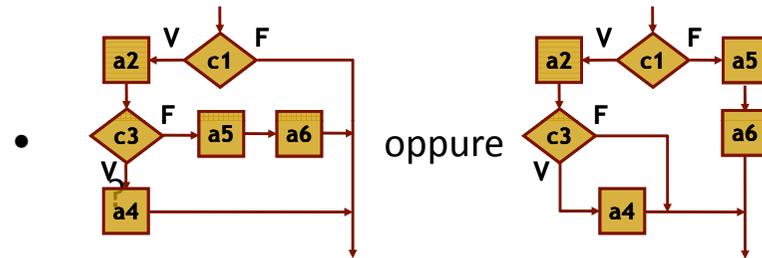


- begin repeat A until not C end



Linguaggio Lineare Ambiguità

- if c1 then a2; if c3 then a4 else a5; a6



- Uso dell'indentazione
 - Aiuta ma non risolve

Risoluzione delle Ambiguità Convenzioni aggiuntive

- Ogni descrizione di un sottoprocesso che sia composizione in sequenza di descrizioni di azioni elementari o sottoprocessi deve essere racchiuso tra le parole begin – end
 - Vale, in particolare, per le descrizione di un sottoprocesso che segue le parola chiave
 - then
 - else
 - while
 - quando non è un'azione basica

Risoluzione delle Ambiguità Convenzioni aggiuntive

- In alternativa:
 - aggiungere i seguenti terminatori di istruzione
 - Selezione: endif
 - Iterazione di tipo while: endwhile
 - Non necessario per l'iterazione di tipo repeat
 - È già presente la clausola until come delimitatore

Esempio Algoritmo Euclideo per il MCD

- Leggi la coppia di numeri
- Fintantoché i numeri sono diversi ripeti
 - Se il primo numero è minore del secondo allora
 - Scambiali
 - Sottrai il secondo dal primo
 - Rimpiazza i due numeri col sottraendo e con la differenza, rispettivamente
- Il risultato è il valore dei due numeri della coppia (uguali)

Esempio

Algoritmo Euclideo per il MCD

begin

leggi a, b

while (a <> b) do

 if (a > b) then

$a \leftarrow a - b$

 else

$a \leftarrow b - a$

MCD \leftarrow a

end

