

Gli errori

- Il processo di risoluzione di un problema non è meccanico: parte delle scelte necessarie è affidata ad esperienza e creatività di analista, progettista e programmatore
- quindi è soggetto ad errori di varia natura, che possono accadere nelle varie fasi
- Più alta è la fase in cui accade l'errore, più gravi sono le conseguenze che comporta, poiché la mole di lavoro che dovrebbe essere rifatto cresce
- È necessario quindi controllare passo passo i risultati intermedi
- così facendo, le 4 fasi non saranno più lineari ma a volte verranno parzialmente ripetute

Gli errori

- **Analisi:**
 - identificazione inappropriata del problema
 - dovuta ad un'analisi incompleta, superficiale o errata
 - spesso si manifesta alla fine del lavoro o dopo un periodo d'uso
- **Progettazione:**
 - algoritmo che non rispetta le specifiche dell'analisi
 - spesso il problema si manifesta solo in alcune istanze di problema, rendendone difficile l'individuazione
- **Programmazione:**
 - dai più ai meno gravi:
 - il programma non implementa l'algoritmo progettato
 - uso scorretto del linguaggio di programmazione (errori statici e dinamici)
 - errori sintattici nella scrittura del programma
 - gli errori sintattici vengono scoperti dal compilatore
- **Esecuzione:**
 - il calcolatore opera diversamente da come ci si aspetta, a causa di malfunzionamenti

Evitare gli errori

- durante le fasi di analisi e progettazione, la correttezza dipende da analista e progettista, che possono seguire determinate metodologie al fine di ridurre la probabilità di errore
- durante la fase di programmazione, possiamo avere tre supporti:
 - il linguaggio di programmazione ha caratteristiche (formali) tali per cui è meno semplice usarlo scorrettamente, ed i cui programmi, essendo chiari, sono più facilmente verificabili staticamente
 - gli strumenti di programmazione segnalano alcuni degli errori statici
 - l'esecuzione del programma evidenzia alcuni errori dinamici

Il linguaggio di programmazione...

- è fondamentale nella costruzione di programmi (ovviamente)
- le sue caratteristiche influiscono anche sulla fase di progettazione, in certi casi
- un buon linguaggio di programmazione deve:
 - essere un aiuto in fase di progettazione
 - costituire una guida ai processi mentali che permettono di passare dall'idea iniziale dell'algoritmo al programma vero e proprio
 - permettere la scrittura di programmi
 - chiari
 - facilmente verificabili
 - facilmente modificabili

Il linguaggio dei diagrammi di flusso

- è un linguaggio con poche regole di composizione, di conseguenza lascia molta libertà di scelta
- per programmi non banali, il diagramma potrà quindi essere estremamente intricato
- E quindi sarà difficile:
 - garantire la correttezza del programma (cioè garantire che risolva il problema per cui è stato realizzato)
 - identificare gli errori
 - risalire dal programma al problema (utile se un altro programmatore lo osserva, o allo stesso programmatore dopo un po' di tempo...)
 - modificare il programma se sopraggiungono nuove esigenze o per risolvere un problema simile

I difetti del linguaggio DF

- poche regole di composizione:
 - un blocco di inizio, almeno uno di fine, un solo arco uscente
- questo garantisce una grande libertà:
 - (~) positiva dal punto di vista della creatività
 - di fatto significa assenza di strategie e criteri di scelta
 - quindi il programmatore può programmare sia bene che male
 - e quindi costruirà programmi buoni e cattivi
- il programma dovrebbe essere un prodotto industriale/artigianale, non artistico...
- cerchiamo un linguaggio più vincolante, ma che risolva gli stessi problemi di DF

Diagrammi di flusso strutturati

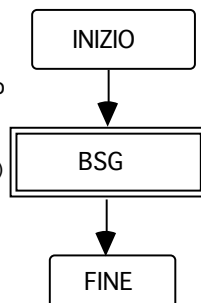
- DFS: linguaggio di programmazione molto simile a DF
- con regole di composizione più vincolanti
- che permettono di costruire un sottoinsieme dell'insieme dei diagrammi costruibili con DF
- ma che bastano per risolvere gli stessi problemi
- nuove regole di composizione:
 - definizione *ricorsiva* di diagramma di flusso strutturato
 - concetto di *blocco strutturato generico*
 - in generale, un solo arco entrante per blocco
 - un solo blocco di fine

Da DF a DFS

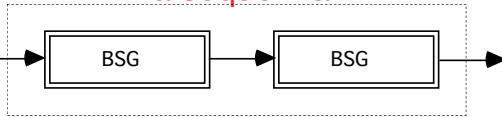
- individuazione di alcune configurazioni tipiche di blocchi che appaiono frequentemente nei diagrammi:
 - sequenza
 - selezione condizionale
 - ciclo
- costruzione delle regole che identificano tutti e soli i diagrammi che usano queste configurazioni
- verifica che il nuovo linguaggio permette di risolvere gli stessi problemi

Diagramma di flusso strutturato

- è uno schema a blocchi definito come da figura
- BSG significa **blocco strutturato generico**, e corrisponde ad uno di quattro possibili costrutti:
 - sequenza
 - condizionale doppia (semplice)
 - ciclo a condizione iniziale
 - blocco funzionale semplice
- **regola di sostituzione**
- ogni costrutto ha un arco entrante ed uno uscente

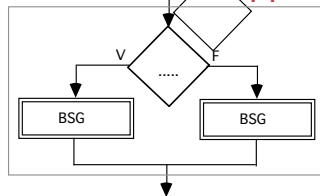


La sequenza



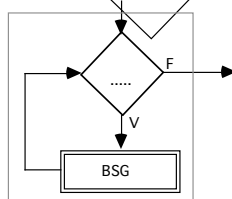
- è una **struttura di controllo**
- identifica l'esecuzione sequenziale di due BSG, che a loro volta potrebbero essere sostituiti da altre sequenze (o altri BSG...) per ottenerne di più lunghe
- rispetto alla sequenza DF che già conosciamo, non si prevede che altri archi puntino ai due BSG

Condizionale doppia



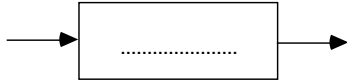
- è una **struttura di controllo**
- definisce qualcosa di più preciso del solo blocco condizionale: scelta tra due azioni in base ad una condizione logica
- se uno dei due BSG è vuoto, si ottengono i due casi particolari detti **condizionale semplice**

Ciclo a condizione iniziale



- è una **struttura di controllo**
- il BSG viene ripetuto finché la condizione nel blocco decisionale è vera
- si noti che il BSG è eseguibile solo all'interno del ciclo (non ci si può arrivare tramite altri archi)

Blocco funzionale semplice



- corrisponde al blocco funzionale del linguaggio DF, con un solo arco entrante
- contiene un'istruzione elementare

La regola di sostituzione

- un diagramma di flusso strutturato si ottiene, a partire dallo schema iniziale, sostituendo per un numero arbitrario di volte un blocco strutturato generico con
 - una struttura di controllo
 - oppure con un blocco funzionale semplice
- si noti che sostituendo un BSG con una struttura di controllo si ottiene almeno un nuovo BSG da sostituire ulteriormente;
- mentre sostituendo un BSG con un blocco funzionale semplice, il processo si ferma

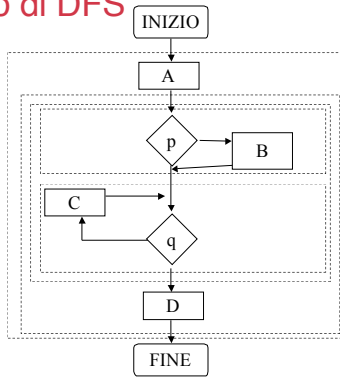
Costruzione di DFS

- il metodo comunemente utilizzato per costruire DFS è quello di applicare ordinatamente la definizione
- cioè, partendo dallo schema iniziale, si sostituisce passo passo ad ogni BSG una struttura di controllo o un blocco funzionale semplice, finché tutti i BSG non siano stati eliminati
- come effetto collaterale, procedendo in questo modo il programma viene specificato dall'astratto al concreto, specificando sempre più dettagliatamente le operazioni
- è il metodo di lavoro noto come **top-down**

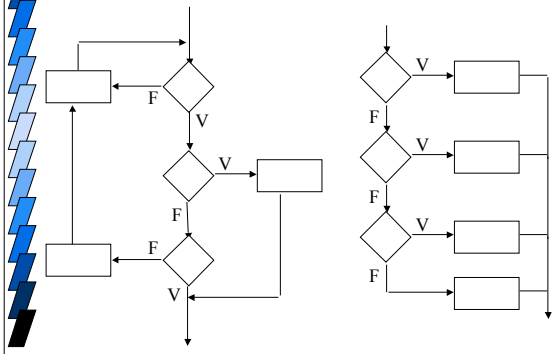
Verifica di DFS

- sempre in base alla definizione, è facile verificare se un qualsiasi diagramma di flusso sia strutturato o meno
- basta verificare che ogni diagramma di flusso parziale sia un blocco strutturato generico
- la scomposizione facilita lo studio del diagramma, in quanto ogni BSG ha un solo arco entrante ed un solo uscente, e quindi è semplice studiarne separatamente il comportamento

Un esempio di DFS



Controesempi (parziali)



Confronto tra DF e DFS

- due problemi:
 - tutte le funzioni computabili con DF sono computabili anche con DFS? Cioé: tutti i problemi risolubili lo sono anche con DFS?
 - tutti gli algoritmi esprimibili con DF sono esprimibili anche con DFS? Cioé: posso risolvere un problema risolubile con uno qualunque degli algoritmi risolvendi anche esprimendolo in DFS?
- usiamo due concetti:
 - funzione computata da un programma (con ovvio significato)
 - sequenza di computazione generata da un diagramma di flusso: sequenza dei blocchi funzionali e decisionali generata dall'esecuzione del diagramma su dei dati iniziali. Fornisce il criterio di uguaglianza tra algoritmi

Equivalenza dei diagrammi

- **Equivalenza debole**
Due diagrammi di flusso X e Y si dicono *debolmente equivalenti* se e solo se, per ogni possibile dato d'ingresso, le funzioni da essi computate assumono lo stesso valore o sono indefinite
- **Equivalenza forte**
Due diagrammi di flusso X e Y si dicono *fortemente equivalenti* se e solo se, per ogni possibile dato d'ingresso, le sequenze di computazione generate sono uguali
- l'equivalenza forte implica l'equivalenza debole

Confronto tra DF e DFS

- i due problemi:
 - tutte le funzioni computabili con DF sono computabili anche con DFS?
 - tutti gli algoritmi esprimibili con DF sono esprimibili anche con DFS?
- diventano:
 - per ogni diagramma di flusso generico, esiste un diagramma di flusso strutturato debolmente equivalente?
 - per ogni diagramma di flusso generico, esiste un diagramma di flusso strutturato fortemente equivalente?

Il teorema di Böhm-Jacopini

- Per ogni diagramma di flusso esiste un diagramma di flusso strutturato debolmente equivalente
- ciò significa che con i DFS possiamo computare tutte le funzioni computabili (primo dei due problemi)
- secondo problema:
non per tutti i DF esiste un DFS fortemente equivalente.
- Ciò significa che posso computare tutte le funzioni, ma non per tutte posso scegliere l'algoritmo con la libertà che avevo con i DF

Perché?

- i costrutti condizionale doppia e ciclo a condizione iniziale non catturano tutte le varianti rispettivamente di selezione e ripetizione; alcuni diagrammi infatti presentano delle composizioni di questi tipo che non sono riscrivibili in modo strutturato
- la soluzione è introdurre delle regole leggermente più ampie che rendano più espressivo il linguaggio,
- in modo non solo di risolvere qualsiasi problema, ma di utilizzare anche uno qualsiasi degli algoritmi possibili

I diagrammi di flusso ben formati

- DFBF: nuovo linguaggio dei diagrammi di flusso, simile a DFS, in cui si hanno tre costrutti:
 - sequenza, uguale a quella di DFS
 - ciclo generalizzato, con un arco di ingresso e più archi di uscita governati da regole precise
 - blocco funzionale semplice, come in DFS
- per DFBF, esiste il:
- Teorema di Peterson-Kasami-Tokura:
Per ogni diagramma di flusso esiste un diagramma di flusso ben formato fortemente equivalente
- DFBF rispetta quindi i principi che avevano spinto verso DFS, ma senza limiti per gli algoritmi



Diagrammi di flusso e linguaggi di programmazione

- Un linguaggio di programmazione è di solito almeno debolmente equivalente ai diagrammi di flusso (DF)
- normalmente mette a disposizione strutture di controllo un po' più varie di quelle disponibili con i diagrammi di flusso strutturati
- un'altra caratteristica dei linguaggi di programmazione è una gestione più raffinata dei tipi di dati, necessaria per gli stessi motivi per cui avevamo introdotto i DFS: chiarezza, verificabilità, modificabilità
