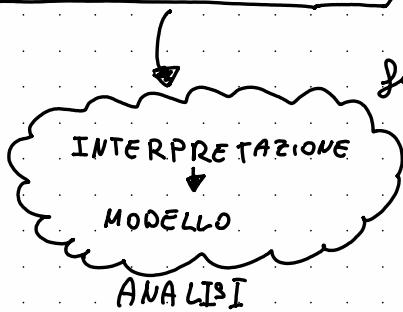


Problema iniziale



formalizzazione
logico-matematica

ALGORITMO

teoria degli
algoritmi
complessità

PROGETTO

CODING

TESTING

linguaggi di
programmazione

REALIZZAZIONE

Struttura di un linguaggio

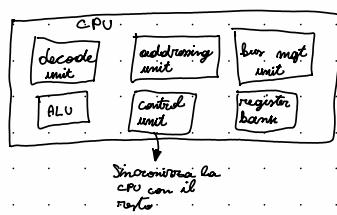
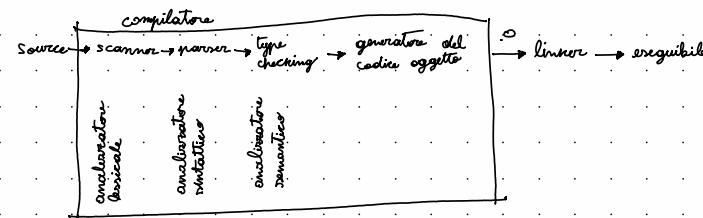
Sintesi:
Lexico: come si scrivono le parole
incangiare

Grammatica: come si compongono le frasi
la tappa grande Teoria

Semantica
significato delle
frasi

la mela mangia pipper

source code
compilatore → genera
interprete → traduce ed
esegue
un'istruzione
per volta



ciclo fetch - execute

recupera una istruzione
della memoria principale

→ decodifica → esegue

Paradigmi

Imperativo

→ Procedurale

→ Orientato agli oggetti

→ Parallello

Dichiarativo

→ Logico
→ Funzionale
→ Database

Procedurale

- estrazione dell'architettura di Von Neumann
- definisce come la memoria cambia dopo ogni azione elementare

Funzionale

- definisce funzioni matematiche sul dominio dei dati e si il codominio

Sintassi e grammatica

Lettura

Insieme delle parole, composto a partire da un insieme di simboli atomici (caratteri) che rappresentano l'alfabeto del linguaggio

Sintassi

fornisce le regole

Graphi

D: coppia $\langle N, V \rangle$

Insieme
di nodi

Insieme di
archi

$$V \leq N^2$$

a cicli

\exists sequenza di archi
non duplicati che parte
da un nodo e torna
allo stesso

connesso

Per ogni coppia di nodi

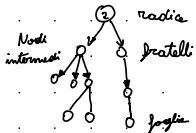
\exists un cammino che lo
unisce

Orientato

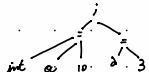
Albero

Graph connesso aciclico

Non ha
cicli



Permettono di definire la sintassi



Sintassi

Definire un linguaggio in maniera appropriata.

$1+1 \checkmark$ $1+-+ \quad ??$ caso di grammatica

Semantica

$1+1$

Cosa significa " $+$ "?

La somma è una convenzione

Grammatica

Alfabeto è un insieme di simboli. Non hanno senso ma
una significato associato

Linguaggio di programmazione (LdP): insieme delle stringhe ammisiibili

Def:

(caratteri)

ALFABETO: insieme finito e non vuoto di simboli

CATENAZIONE: operazione definita sui simboli di
un alfabeto

STRINGA: concatenazione di un insieme finito
e vuoto dei simboli di un alfabeto

LUNGHEZZA DI UNA STRINGA x : numero dei suoi simboli $|x|$

CATENAZIONE DI 2 STRINGHE (x,y) : quando require
i simboli di y a quelli di x

ESPOVIMENTO DI UNA STRINGA x : x^0 vuota \Rightarrow Nessun simbolo

$x^n \in x_1 x_2 \dots x_n$ esponente

$x^3 = x \cdot x^2 = x \cdot x \cdot x = (x \cdot x) \cdot x = (x \cdot x) \cdot (x \cdot x) = x \cdot x \cdot x \cdot x = abab$

$x^1 = x \cdot x^0$

$x^0 = E$
Epsilon

Concatenare un'ante numero di volte
la stessa stringa

PREFISSO DI UNA STRINGA: Si ottiene ricavando

0 o più simboli dalla sua coda

$x = ab$

$y = cd$

$xy = abcd$

$|xy| = |x| + |y| = 2 + 2 = 4$

SUFFISSO DI UNA STRINGA: Si ottiene ricavando 0
o più simboli dalla sua testa

Prefix(x) a, ab, abc, ~~abc~~

Suffix(y) d, cd, bcd

Substring(xy) bc, bcd, b, ab, cd, abc

LINGUAGGIO VUOTO: \emptyset

LINGUAGGIO COSTITUITO DA UNA SOLA STRINGA VUOTA: $\{E\}$

chiarezza di Kleene *

Insieme di tutte le stringhe di lunghezza ≥ 0 che si possono formare concatenando i simboli di un alfabeto.

$LdC \subset$ chiarezza di Kleene

$$A = \{0,1\}$$

$$A^* = \varepsilon, 0, 1, 01, 00, 10, 11, 000$$

Insieme di tutte le stringhe

$$B = \{\}$$

$$B^* = \varepsilon \quad |B^*| = 1$$

Chiarezza positiva $A^+ = A^* \setminus \{\varepsilon\}$

Linguaggio L su alfabeto A $K \subseteq A^*$

Linguaggi di programmazione: notazioni di ASCII *

Programma:

- Non tutte le stringhe sono programmi accettabili
- LdP diversi = insiemi diversi di stringhe
- LdP ha la propria sintassi

Linguaggio informatico definibile in metodi

GENERATIVO: generati da una grammatica

RICONOSCITIVO: stringhe riconosciute da un AUTOMA

ALGEBRICO: stringhe come soluzioni di un sistema di equazioni

ESPRESSIONI ARITMETICHE

consideriamo le espressioni che contengono:

• due operatori binari +, *

• parentesi

• numeri come operandi [0-9*+]

Linguaggio delle espressioni aritmetiche: grammatica

(espr. numeri)

(espr. (espr.)

(espr., espr + espr.)

5+3

(3).

5*3

Espressione: categoria sintattica o non simbolo

Terminal (qualunque stringa nel linguaggio delle espressioni)

simboli non terminali: variabili sintattiche; gruppi di simboli terminali secondo le regole di produzione
dicono le regole per scomporre un simbolo non terminali

simboli terminali: simboli elementari del linguaggio definiti da una grammatica formale caratteri dell'alfabeto

Parte lessicografica per termini grafia

a+2

$$E \rightarrow (E) \rightarrow (E+E) \rightarrow (a+2)$$

Grammatica:

Quadrice

Simboli terminali

Simboli non terminali

Grammatica

A lfabeto

category sintattiche

Categoria
sintattica

for

$$\langle E \rangle, \{a, \dots, z, +, *, (),\}, \{(E, a), (E, z), (E, E+E), (E, E+E)\},$$

$$(E, (E))\}, E$$

Terminali: solo ciò che voglio vedere nel mio
alfabeto

Per definire regole che permettano di dire
frasi corrette/leggibili

$$\left(\{D\}, \{0, \dots, 9\}\right)$$

Non terminali

E: categoria sintattica che rappresenta ??!! ?!

Produzioni: possiamo sostituire E con a, ..., z, E+E, E+E, E

Variabili = Non terminali = set di simboli che
rappresenta il linguaggio

Start symbol:

Production: variabile \rightarrow stringa di variabili e terminali

Eg.

"Soggetto"

↓

je

Grammatiche si classificano in base
all'espressività del linguaggio che generano.

· tipo 0 $(\alpha, \beta), \alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$

dipendenti dal contesto

linguaggi di programmazione:

sottoclassi di grammatiche libere da contesto

con produzioni di forma

$$(\alpha, \beta), \beta \in (N \cup \Sigma)^+$$

Bachus - New form

espressioni ben formate 3 simboli non terminali

$$E ::= E+E \mid E-E \mid E+E \mid (E) \mid I \mid V$$

categoria
intestata indica
simbolo
iniziale

produzioni
separate da |

Tutti i simboli usati per la produzione.

Simboli terminali: $\{I\}$ = simboli terminali e di produzione

$$V ::= \{I\}^*$$

$$I ::= .0|.1|.9$$

Ambiente = funzione

Mappa in T

Declarazioni: hanno effetto sull'ambiente

D: $\Delta L \rightarrow \Delta$

Comando ben formato: derivabile dall'ambiente A.

Espresione ben formata.

Regole di inferenza

$A_1, \dots, A_n \vdash B$ equivale a $A, A_1 \dots A_n \vdash B$.

B

Axioma = Sempre vero.

$\vdash B$ equivale a $\emptyset \Rightarrow B$

Dimostrazione:

$$\frac{\begin{array}{c} D_1, \dots, D_m \\ \hline A_1, \dots, A_n \end{array}}{B}$$

Per dadurre questo.

Espressioni

Benformata: E: $\Delta T, E : \tau$

Se partendo dall'ambiente statico poniamo

associare ad E il tipo del valore che rappresenta.

$$E := \sqrt{|Id|} \cup \text{op } E | E \text{ bop } E | (\tau)$$

op: + - ||

bop: > < != >= <= << >> ||

Val-E := $\Sigma \text{Var } V \{ \text{temporal } \} \cup \{ \text{segnali } \}$

Sono una categoria sintattica

Validità per ottenere un valore

Il tipo di valore ottenuto è uno dei tipi esprimibili del linguaggio.

Se nell'ambiente statico
assegnato a E si ha che:
 $(\Delta(T), z) \models \Delta(G) \vdash L \circ$

$\Delta(T), z \models \Delta(G) \vdash L \circ$
Partendo da E si ha che:
ambiente statico

$$\Delta \vdash \star \tau$$

$\Delta \models \{(n, \text{intLoc}); (g, \text{int})\}$
Ambiente Riconosciuto

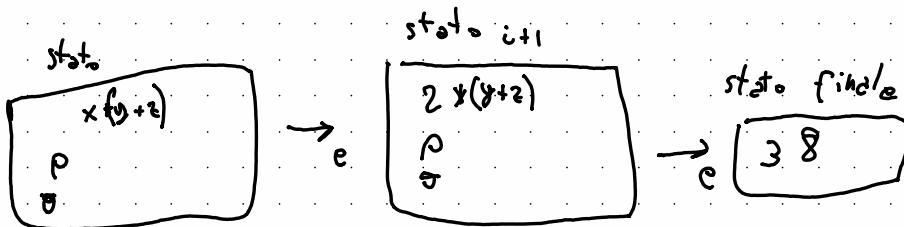
$$(a) \frac{\Delta \vdash \star \tau \quad \Delta \vdash \star \tau \quad \Delta \vdash \star \tau}{\Delta \vdash \star \tau \star \tau}$$
$$(b) \frac{\Delta \vdash \star \tau \quad \Delta \vdash \star \tau}{\Delta \vdash \star \tau \star \tau}$$
$$(c) \frac{\Delta \vdash \star \tau \quad \Delta \vdash \star \tau}{\Delta \vdash \star \tau \star \tau}$$

$$\Delta \vdash \star \tau \star \tau \vdash \star \tau$$

Semantica

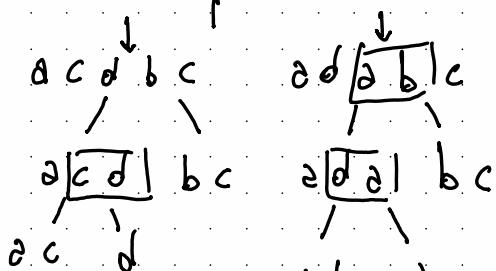
ρ ro ambiente
 σ sigma memoria

$$\rho \{ (x, L_0), (y, 2), (z, L_1) \}$$
$$\sigma \{ (L_0, 2), (L_1, 10) \}$$



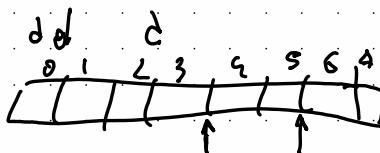
10:25

0	1	2	3	4	5	6	7	8	9
a	c	a	b	c	a	b	a	b	c



step

solo la seconda



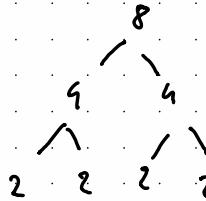
che

$$l = 8$$

$$\frac{a+b}{2} = \frac{11}{2} = 5$$

//impero
if ($i == e$) return false
if ($start - end == 1$) {
//
return $a[start] == a[end];$

// divid:



$$T(n) = 2 T\left(\frac{n}{2}\right)$$

$$a=2$$

$$b=2$$

$$n-1$$

$$\log_2 2 = 1$$

n

$18, 21, 44, 89, 37, 68, 48, 83$

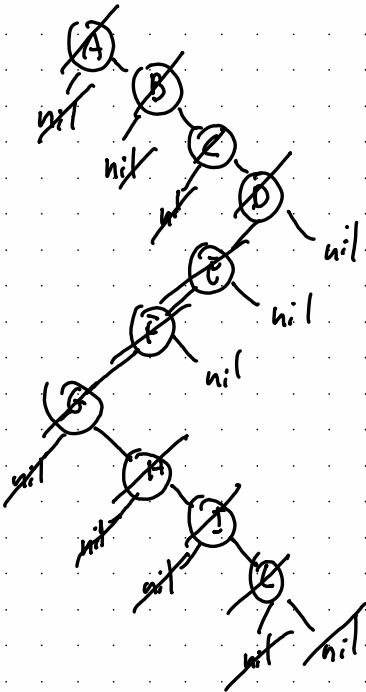
$m = 15$

$$h(k) = k \bmod m$$

$$h_2(k) = (h(k) + i * h(k)) \% m$$

	$h(k)$	$h_2(k)$
18	5	5
21	8	8
44	8	8; 3
89	11	11
37	11	11; 9
68	3	3; 6
48	0	0
83	5	5; 10

18		44		89		68		21		37		83		48
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



$\text{var } a: \text{Int} = 3$

\vdash

$\rho = \emptyset \rightarrow [(a, l_0)]$

$\sigma = \emptyset \rightarrow [(l_0, 3)]$



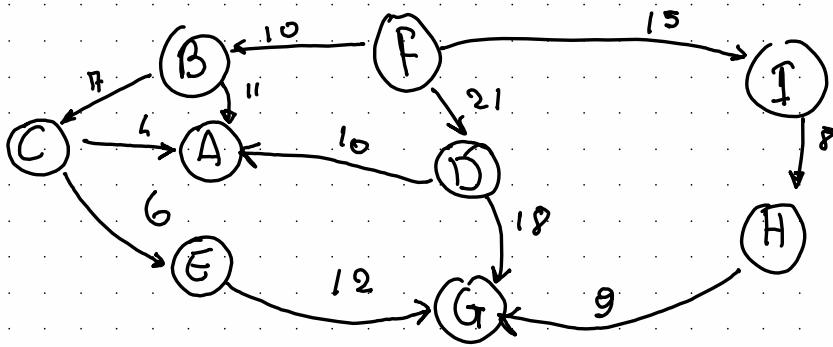
$\text{var } a: \text{Int} = 3$

$\text{var } c: \text{Int} = 2$

\vdash

$\rho = [(a, l_0), (c, l_1)]$

$\sigma = [(l_0, 3), (l_1, 2)]$



Algoritmo cammino minimo F - E

Trovare il cammino minimo e valore indietro

Dijkstra ← Pari positivi

Bellman-Ford ← Pari negativi ma cubico

Abbriamo cammini lunghezza minima

E.Dit. distance

longest distance

Zaino

Da qui riparte la trascrizione
delle lessioni (kinda)

L 12

Semantica statica di L

Studio del significato dei programmi;

Formuli: esiste nei termini delle azioni che il modello deve compiere

stati: proprietà che possono essere dedotte dalla rappresentazione dei programmi

Non coinvolge l'esecuzione

2 tipi

distribuzionale
simile alle funzionali

Possi che la macchina astratta compie

operazionale

Programma considerato come sequenza di istruzioni atomiche che modificano lo stato delle macchine

Se ad un certo punto non riesco a fare un cambio

di stato, c'è un errore nel programma

Stato:
programma + dati
+ elaborare + memoria

ambiente statico funzione:
associa agli identificatori un tipo (σ tipoLoc)

$A: \text{io! Ural} \rightarrow T \cup TLoc$

Nella semantica statica controllo solo che i tipi siano usati coerentemente

Principio di componibilità: definire qualcosa partendo dai suoi componenti elementari (e.g. grammatica attraverso le produzioni)

Axioms Categoria sintattica degli interi

- $\emptyset \vdash_e i : \text{Int} \quad \leftarrow \text{Tipi intero}$



Non mi serve un ambiente

per dire che 5 è un intero

- $\emptyset \vdash_e d : \text{Double}$
- $\emptyset \vdash_e b : \text{Bool}$
- $\emptyset \vdash_e s : \text{String}$

categorie
sintattiche

Regole di inferenza

$$(R1) \frac{\Delta(\text{Id}) = \tau}{\Delta \vdash_e \text{Id} : \tau}$$

Dato
un ambiente,
possiamo dire

che un identificatore è di tipo τ
se nell'ambiente troviamo l'identificatore
con tipo τ o τ_{Loc} . Allora l'espressione
è corretta

partendo dall'ambiente Δ

Se l'espressione è di tipo τ_1 ,
E l'operatore unario prende un
tipo τ_1 e lo trasforma in τ
l'espressione $\text{uop } \bar{c}_1$ è di tipo τ

$$(R2) \frac{\Delta \vdash_e E_1 : \tau_1, \text{uop} : \tau_1 \rightarrow \tau}{\Delta \vdash_e \text{uop } \bar{c}_1 : \tau}$$

$$\Delta[(x, \text{Int})] \quad \boxed{-x;}$$

$$\Delta(x) = \text{Int}$$

$$\Delta \vdash_e x : \text{Int}, - : \text{Int} \rightarrow \text{Int}$$

$$\Delta \vdash_e -x : \text{Int}$$

ESEMPIO

$$(R3) \frac{\Delta \vdash_e E_1 : \tau_1, \Delta \vdash_e E_2 : \tau_2, \text{bop} : \tau_1 \times \tau_2 \rightarrow \tau}{\Delta \vdash_e \tau_1 \text{bop } \tau_2 : \tau}$$

$$(R4) \frac{\Delta \vdash_e E : \tau}{\Delta \vdash_e (\epsilon) : \tau}$$

Comandi

Non hanno tipi in quanto i comandi non rappresentano un valore.

$\emptyset \vdash_c \text{nil}$

$$(R5) \frac{\Delta(\text{Id}) = \tau \text{Loc}, \Delta \vdash_c C : \tau}{\Delta \vdash_c \text{Id} = E}$$

Per assegnare un'espressione ad una variabile deve accadere che il tipo dell'espressione e quello della variabile siano uguali

Se E è ben formato ed ha tipo Bool, C_1 è ben formato e C_2 è ben formato allora il comando è ben formato

$$(R6) \frac{\Delta \vdash_c E : \text{Bool}, \Delta \vdash_c C}{\Delta \vdash_c \text{while}(E)\{C\}}$$



Qui non controlliamo se il while termina

Se la dichiarazione è ben formata e mi restituisce un ambiente Δ' (gli ambienti si aggiungono solo con le dichiarazioni) e se il comando C è ben formato in tutto l'ambiente

Se da Δ posso associare a D l'ambiente statico Δ'

E da Δ esteso con D posso dimostrare che C è ben formato LO ESTENDE!
 $D; C$ è ben formato

$$(R7) \frac{\Delta \vdash_c E : \text{Bool}, \Delta \vdash_c C_1, \Delta \vdash_c C_2}{\text{if}(E)\{C_1\} \text{ else } \{C_2\}}$$

Se l'espressione è ben formata ed ha tipo Bool e se il comando è ben formato, il while è ben formato

Questo Δ' Δ esteso Δ'
contiene solo la nuova dichiarazione,

$$(R8) \frac{\Delta \vdash_d D : \Delta', \Delta[\Delta'] \vdash_c C}{\Delta \vdash_c D; C}$$

$$\Delta[\Delta'](x) = \begin{cases} \Delta'(x) & \text{se } \Delta'(x) \text{ definito} \\ \Delta(x) & \text{altrimenti} \end{cases}$$

Non sovraccrive l'ambiente!

Possibile ridichiarare la variabile

Dichiarazioni

Risultato dopo:

Dichiarazione ben formata $\Delta \vdash D : D'$

Ben formata se a partire dall'ambiente statico Δ , è possibile creare un ambiente statico che contiene i legami per i nomi definiti in D

Assumi:

(A6) $\emptyset \vdash_{\text{mil}} ; \emptyset$

$$(R10) \frac{\Delta \vdash_{\text{e}} E : \tau, T = \tau}{\Delta \vdash_{\text{d}} \text{const } Id : T = E : [(\text{Id}, \tau)]}$$

(R11) $\Delta \vdash_{\text{e}} E : \tau, T = \tau$

$$\Delta \vdash_{\text{d}} \text{var } Id : T = E : [(\text{Id}, \tau \text{ Loc})] \quad \Delta'$$

Quando dichiara una costante, posso dire che è ben formata quando l'espressione è ben formata e di tipo T o di tipo T' .

Sarà discorso per le variabili.

Se da Δ, D_1 genera l'ambiente statico Δ' e da Δ esteso con D_2 genera l'ambiente Δ'' (R12)

$$\frac{\Delta \vdash_{\text{d}} D_1 : \Delta', \Delta[\Delta'] \vdash_{\text{d}} D_2 : \Delta''}{\Delta \vdash_{\text{d}} D_1; D_2 : \Delta''[\Delta'']}$$

Allora da $\Delta, D_1; D_2$ genera l'ambiente statico Δ'' e la dichiarazione è ben formata

$$\Delta' = [(x, \text{IntLoc})]$$

$$\Delta'' = [(y, \text{IntLoc})]$$

$$\emptyset \vdash [\Delta'](x) = \text{IntLoc}$$

$$\emptyset \vdash e : \text{Int}, \text{Int} = \text{IntLoc}$$

$$\emptyset \vdash [\Delta'] \vdash x : \text{Int}, \emptyset \vdash [\Delta'] \vdash y : \text{Int}$$

$$\emptyset \vdash_{\text{d}} \text{var } x : \text{Int} = 3 : \Delta'$$

$$\emptyset \vdash [\Delta'] \vdash_{\text{d}} \text{var } y = x : \Delta''$$

$$\emptyset \vdash_{\text{d}} \text{var } x : \text{Int} = 3; \text{ var } y : \text{Int} = x;$$

L24 | Semantica statica funzioni

$$(FS1) \frac{\Delta \vdash e : \tau}{A \vdash c \text{ return } E}$$

Se da Δ posso associare il tipo τ ad e , il return è ben formato

$$(FS2) \frac{\text{form} : \Delta_0, \Delta[\Delta_0] \vdash c; \text{return } E, \Delta[\Delta'] \vdash e : \tau}{\Delta \vdash \text{Func } \text{Id}(\text{form}) \rightarrow \tau \{c; \text{return } E\} : t(\text{Id}, \overbrace{\text{J}(form)}^T \rightarrow \tau)}$$

Lista tipi formalisi

Tipi di ritorno

form è una lista
[$T : H$]

Nomi nel codice

$$\begin{aligned} T(\text{form}) &= T(\text{var id} : \tau = e; \text{form}') \\ &= \tau, T(\text{form}') \end{aligned}$$

$$T(\text{form}) \left\{ \begin{array}{l} T(\text{nil}) = n : \tau \\ T(\text{const id} : \tau, \text{form}) = \tau, T(\text{form}') \\ T(\text{var id} : \tau, \text{form}') = \tau, T(\text{form}') \end{array} \right.$$

Ogni da parametro è come fosse una variabile

Controllo che il nome non sia già stato usato nella

$$(FS3) \quad n : \emptyset, \frac{\text{form} : \Delta_0, \text{Id} \notin \Delta_0 \text{ lista}}{\text{const id} : \tau, \text{form} : \Delta_0[(\text{Id}, \tau)]}, \quad \text{form} : \Delta_0, \text{Id} \notin \Delta_0 \quad \frac{}{\text{var id} : \tau, \text{form} : \Delta_0[(\text{Id}, \tau)]}$$

Se la lista degli ottuali corrisponde a

$$\frac{\Delta \vdash ee : aet, \Delta(\text{Id}) = aet \rightarrow \tau}{\Delta \vdash c \text{ Id}(ee) : \tau}$$

Funzione
Lista ottuali

Lista di tipi

nell'ambiente ho una lista di tipi che restituisce τ allora l'espressione è ben formata

$$\left\{ \begin{array}{l} \Delta \vdash ee \text{ nil} \\ \Delta \vdash e : \tau, \Delta \vdash ee : aet \\ \hline \Delta \vdash ee : \tau, ee : \tau, aet \end{array} \right.$$

$$(RS1) \frac{\Delta[\Delta'_{I_0}] \vdash_D \Delta'}{\Delta \vdash_{rec} D : \Delta'}, I_0 = FI(D) \cap BI(D)$$

$\Delta' = [(\text{fact}, t_n \rightarrow I_n)] = \Delta'_{I_0}$

E stendendo Δ con l'intersezione con l'intersezione fra libere e legate (il nome delle funzioni è legato, l'occorrenza è libera)

Questo si può poi semplificare:

$$(RS1') \frac{\vdash_D D : \Delta \quad (FS2)}{\vdash_D Rec D : \Delta}$$

$$(RS1'') \frac{\vdash_D D : \Delta', \Delta[\Delta'_{I_0}] \vdash_D D}{\Delta \vdash_D Rec D}, I_0 = FI(D) \cap BI(D)$$

$(FS2'')$

$$RS1^* \frac{FS2'}{\vdash_D rec D : \Delta} \quad RS1'' \frac{RS1'}{\emptyset \vdash_D rec D} \quad RS2'' \frac{FS2' \mid FS2''}{\Delta \vdash_D rec D}$$

$$\underline{L25} \quad FI(\text{func}) = FI(c) - FI(\text{for form})$$

Identificatori liberi della funzione

Identificatori liberi
del corpo

Identifieri liber
să formă (nume,
tută legat)

Assunzioni per renderle semplici:

~~Dal punto~~ la prima variabile è il valore nel rettangolo.

Var res: $\tau = E; C$; return res;

1

(FS2) Form: $\Delta_0, \Delta[\Delta_0] \vdash c$ Var res: $\tau = E; C$; return res, $\Delta[\Delta_0][\text{res}, \tau \text{loc}] \vdash_E^T$
 $\Delta \vdash_D \text{func } ID \text{ (form)} \rightarrow \tau \{ \text{var res: } \tau; c; \text{return res} \}: [(ID, \alpha(\text{form}) \rightarrow \tau)]$

Ordinamenti per Compratori: Lower bound:

Almeno n. (dovendo vedere ogni elemento), ma non più basta

gli elementi devono essere confrontabili.

Schematizzazione in abbozzi

(26) Quicksort Pivot

३

15 30 6 8020 11 40

i 1

< pivot | non determinat
 > pivot

Fino ad i i è fatto più
piccolo, del pirof

J è da jore

+ 50 i.e. è tutto più grande

- $i = \text{partenza} - 1; j = i + 1$
 - $A[i] \leq \text{pivot}$
 - $i++$
 $A[i] \leftarrow A[3]$
 $j++$
 $\text{goto } 2$
 - $j++$
 $\text{goto } 2$
 - $A[i+1] \leftarrow \text{pivot}$

L27 | Heap : Albero binario

Albero binario

- al più 2^L nodi al livello L
- Profondità si al più $2^{d+1}-1$ nodi
- n nodi ha profondità almeno $\log_2(n+1)-1$

• Completo

• Quasi - completo : completo tranne per le foglie, che se mancano sono a destra

$$\lfloor \frac{n}{2} \rfloor$$

Heap : Albero binario quasi completo E (con chiavi confrontabili)

ogni nodo interno ha un valore

\geq maggiore = rispetto ai suoi due figli

max & min heap.

Deletion
foglia:
cancellata

interno:
replace con
foglia e
reheap

Insertion
Aggiunge la
foglia e
sostituisce il
padre con la
foglia finché
non è
bilanciato (swap,
reheapification)

operazione a
 $\log(n)$

Implementato tramite array

livello 2^{i-1} nell'array

Genitore di $i = \lfloor \frac{i-1}{2} \rfloor$

Fogli: $i, 2i+1, 2i+2$

Reaport: array

ribaltare non
ordinato
top-to-bottom

Heap(fly)



Bisogna che i
nostri nodi siano heap

Max heapify

Si parte dalle foglie e si procede con i nodi interni

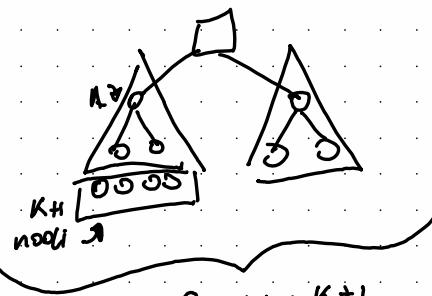
(si parte da $\lfloor \frac{n}{2} \rfloor$,

che è la prima non foglia)

Costo della Max-heapify è

$\Theta(h)$

altezza



$$2K+1 + K+1 = 3K+2 \text{ modi totali}$$

$$K = \frac{n-2}{3}$$

$$Sx = 2K+1$$

$$= \frac{2n-4}{3} + 1$$

percentuale
di modi a
Sx rispetto
a dx

$$\frac{2n-4}{3} + 1 < \frac{2n}{3}$$

caso peggiore

$$T\left(\frac{2n}{3}\right) + \Theta(1)$$

$$\log_{\frac{2}{3}} 1 \text{ vs } \Theta(1)$$

$$\Theta(\log n)$$

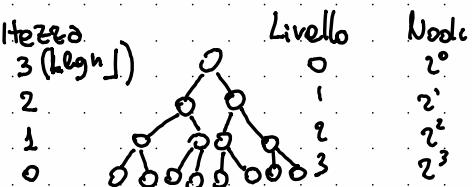
[28]

Build max-heap

$$T(n) = \sum_{i=0}^h n_i h_i$$

costo heapify * numero
modi al livello

Altezza
 $3(\lfloor \log n \rfloor)$



Partendo da un array,
tolgo la radice, e sposto
l'ultima foglia alla radice
e bilancio l'albero

[4 6 3 1 2]



Build heap costo (n)

liste collegate < heap per code di priorità

Accesso diretto

Accesso collegate

Tipi di strutture dati

Max heapify $O(\log n)$

Build max-heap $O(n)$

Heap sort $O(n \log n)$

Max-heap-insert $O(\log n)$

Heap extract max $O(\log n)$

Heap increase key $O(\log n)$

Heap maximum $O(1)$

E29
L30

Numeri pseudo-casuali:

generare ogni numero nel codominio equiprobabilmente

Da un seed si genera in maniera deterministica il numero

L31 Strutture dati

Usati per organizzare dati o valori da manipolare

ADS:

Abstract Data Type

logicamente

fisicamente

Strutture dati

Collegano la struttura fisica con l'organizz.

Operazioni:

- Leggere
- Scrivere / Modificare
- Cancellare / inserire

Strutture:

- Array
- Stack (Vib)
- Code
- Liste collegate
- Alberi e grafici
- ...

Array:

Organizzazione logica: Valori in posizioni logicamente contigue

“ ” Fisica: “ ” indirizzi contigui di memoria

contiguo dipende dalla dimensione del dato

• Accesso diretto:

• facile e veloce accesso e modifica

• cancellazione e inserimento: dipende sulla implementazione

Pila

Logicamente: Valori in posizioni logicamente contigue
LIFO (Push & Pop) (8 top)

Code

Logicamente contigua FIFO

liste collegate

Logicamente elementi non contigui, quindi richiesto un modo per accedere al successivo (ed in corso precedente)

Lettura, inserimento, modifica e cancellazione quasi banali

Possono essere circolari (ad anello)

13h] Sorting lineare

Albero dei confronti avrà $n!$ foglie, quindi ogni

è possibile ordinare senza fare confronti (basandosi sulla proprietà degli attributi degli oggetti da ordinare)

$$[A[i] \in [0 \dots n-1]]$$

E.g. Array di numeri senza ripetizioni \Rightarrow ricrea l'array

Counting sort \Rightarrow conta le ripetizioni in un secondo array (counting sort) (avendo k che è il massimo)

$$\Theta(n) \text{ vs. } \Theta(k)$$

$$\Theta(n+k)$$

Se ne ha questo constraint si della ricorrenza alle comparazioni

ordinamento stabile: preserva l'ordine iniziale fra due elementi con la stessa chiave

Heap & quicksort
NON sono stabili

Radix Sort: Basato sul MSB a Stable sort

Ordina prima le unità, poi poi solire (dalla mano alla più significativa)

Ottimo per intuizione che tutte le cifre precedenti sono ordinate.
se le cifre sono uguali si passa oltre, altrimenti si calcola
l'ordine attraverso un algoritmo stabile in $O(n)$

Semantica dinamica

Cosa fa il programma quando viene eseguito

Funzioni che specificano i passi dinamici (esecuzione dei comandi, valutazione delle espressioni, elaborazione delle dichiarazioni)

Possibilità di individuare alcuni errori logici

L3.7 Questa parte viene gestita dal loader(?)

Identificatore: sequenza di caratteri (stringa); contiene lettere, cifre e - e non inizia con una cifra.

p Ambiente: funzione che associa nomi mnemonici a locazioni

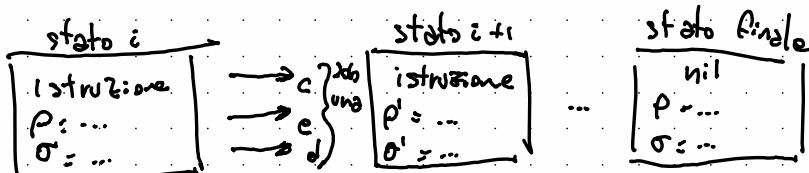
o Memoria: " " " locazioni a valori

2 modi per esprimere:

Loader:

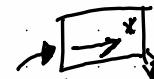
$j : \text{nome} \rightarrow \text{indirizzo}$

Transizioni di stato (includono molteplici regole)



Transizione di stato

Non è formale



Funzioni di interpretazione
semantica

Più transizioni

$$\langle E, P, \sigma \rangle \xrightarrow{e} \langle E', P, \sigma \rangle, \text{Eval}(E, P, \sigma) = v \in \text{Val} \Leftrightarrow \langle E, P, \sigma \rangle \xrightarrow{*} v$$

Sistema di Transizioni: Quadrupla (S, T, F, S_0)

$$S_{\text{ stati.}} = \{\langle \bar{E}, p, \sigma \rangle \cup v \mid v \in \text{Val}_{\neg E}\}$$

$$T \text{ Transizioni} \subseteq S \times S, T = \{(s_1, s_2) | s_1, s_2 \in S \wedge s_1 \rightarrow_c s_2\}$$

F Stati finali $\subseteq S$, $F = \text{Val_E}$

S_0 Stato iniziale $\in S$ (espressione da valutare)

Per le espressioni:

$$\left(\left\{ \langle E, \rho, \sigma \rangle \cup v \mid v \in \text{Val_}E \right\}, \rightarrow_e, \text{Val_}E, (\text{Expr})^* \right)$$

Regole di inferenza → ↑ ↗
inferenza Valore E s'pressione
(dopo) finale de valutare

Le transizioni di stato maschile tuttavia le regole di inferenza per una migliore visibilità sui processi

- Usando le regole di inferenza:

$$(Id_1) \frac{P(Id) = v \vee (P(Id) = L \in Loc \wedge \sigma(L) = v)}{\langle Id, p, \sigma \rangle \rightarrow_e v}$$

Se p associa a Id

un valore o una locazione
me < nella locazione
di L è memorizzato
un valore v

Da $p \in \sigma$ Id viene valutato v in un passo di valutazione

Se in un passo di valutazione (vop1)
o partita da $p \in \sigma$ E si trasforma
in E'

da $p \in \sigma$ vop E si trasforma in vop E'
in un passo di valutazione

$$\frac{}{\langle E, p, \sigma \rangle \rightarrow_e \langle E', p, \sigma \rangle}$$

$$\frac{}{\langle vop\ E, p, \sigma \rangle \rightarrow_e \langle vop\ E', p, \sigma \rangle}$$

$$(vop\ 2) \quad \langle vop\ v, p, \sigma \rangle \rightarrow_{\bar{s}} v' = vop\ v$$

Dopo aver valutato ripetutamente
vop v , e si può concludere la valutazione

Dopo aver valutato
correttamente entrambe
le espressioni, si
raggiunge la
configurazione

$V_1, bop V_2$, che si
può valutare a v

$$(bop\ 1) \quad \frac{}{\langle E_1 bop E_2, p, \sigma \rangle \rightarrow_e \langle E'_1 bop E_2, p, \sigma \rangle}$$

$$\frac{}{\langle E_1 bop E_2, p, \sigma \rangle \rightarrow_e \langle E'_1 bop E_2, p, \sigma \rangle}$$

$$(bop\ 2) \quad \frac{}{\langle E_1 bop E_2, p, \sigma \rangle \rightarrow_e \langle E_1 bop E'_2, p, \sigma \rangle}$$

$$\frac{}{\langle E_1 bop E_2, p, \sigma \rangle \rightarrow_e \langle E_1 bop E'_2, p, \sigma \rangle}$$

$$(bop\ 3) \quad \frac{}{\langle V_1, bop V_2, p, \sigma \rangle \rightarrow_e v = V_1, bop V_2}$$

Queste regole appena mostrate ci permettono di definire un
modello di valutazione delle espressioni

Comandi

$$(id_2) \frac{< E, p, \sigma > \xrightarrow{e}^* v}{< Id = E, p, \sigma > \xrightarrow{c} < Id = v, p, \sigma >}$$

Se, partendo da p e σ l'espressione E (dopo un certo numero di passi) viene valutata ad un valore v

L'assegnamento può essere riscritto come $Id = v$ (senza toccare p e σ dato che la valutazione non li modifica)

L'esecuzione dell'assegnamento (id 3) $< Id = v, p, \sigma > \xrightarrow{e}^{\sigma} [p(Id) = v]$ lascia inalterato l'ambiente modificandolo solo la locazione di memoria $p(Id)$ con v

$$(seq1) \frac{< C_1, p, \sigma > \xrightarrow{c} < C'_1, p, \sigma' >}{< C_1; C_2, p, \sigma > \xrightarrow{c} < C'_1; C_2, p, \sigma' >}$$

Se fa per σ un passo si C_1 lo trasforma in C'_1 con la nuova configurazione della memoria σ'

Allora $C_1; C_2$ può essere riscritto come $C'_1; C_2$ con configurazione della memoria σ'

$$(seq2) \frac{< C_1, p, \sigma > \xrightarrow{c} \sigma'}{< C_1; C_2, p, \sigma > \xrightarrow{c} < C_2, p, \sigma' >}$$

$$(if1) \frac{< E, p, \sigma > \xrightarrow{e}^* \text{true}}{< \text{if}(E)\{C_1\} \text{else}\{C_2\}, p, \sigma > \xrightarrow{c} < C_1, p, \sigma >}$$

$$(if2) \frac{< E, p, \sigma > \xrightarrow{e}^* \text{false}}{< \text{if}(E)\{C_1\} \text{else}\{C_2\}, p, \sigma > \xrightarrow{c} < C_2, p, \sigma >}$$

A seconda di come viene valutata l'espressione (true o false) si eseguono i rispettivi comandi (C_1 o C_2) con la stessa memoria con cui si è iniziata l'esecuzione dell'if

$$(rep_1) \frac{}{\langle E, P, \sigma \rangle \xrightarrow{G}^* \text{true}}$$

$$\langle \text{while}(E)\{C\}, P, \sigma \rangle \xrightarrow{C} \langle C; \text{while}(E)\{C\}, P, \sigma \rangle$$

Se l'espressione viene valutata true, si esegue il

il corpo C del while e poi si torna ad eseguire il while

Se la guardia viene valutata false, non si deve eseguire il corpo C e si può restituire la memoria in cui è eseguito il while

$$(rep_2) \frac{}{\langle E, P, \sigma \rangle \xrightarrow{e}^* \text{false}}$$

$$\langle \text{while}(E)\{C\}, P, \sigma \rangle \xrightarrow{C} \sigma$$

$$(b1) \frac{\langle D, P, \sigma \rangle \xrightarrow{D}^* \langle P', \sigma' \rangle}{\langle D; C, P, \sigma \rangle \xrightarrow{C} \langle C, P[P'] \sigma[\sigma'] \rangle}$$

Allora il comando C si esegue nell'ambiente e nella memoria correnti, estesi con quelli generati da D

Se la dichiarazione genera un nuovo ambiente e memoria

Dichiarazioni

$$(\text{const}) \quad \langle E, P, \sigma \rangle \xrightarrow{*} e \quad V$$

$$\langle \text{const Id: } T = E, P, \sigma \rangle \xrightarrow{D} \langle [(\text{Id}, v)], \sigma \rangle$$

Se da P e σ in un certo numero di passi l'espressione viene valutata ad un valore V .

La dichiarazione di costante associa nell'ambiente generato l'identificatore Id con V , lasciando inalterata la memoria.

Se in un

certo numero (var)

di passi l'espressione viene valutata

ad un valore V ,

la dichiarazione associa all'ambiente generato l'identificatore ad una nuova locazione L (mai usata prima) e aggiorna la memoria scrivendo il valore v alla locazione L

$$\langle E, P, \sigma \rangle \xrightarrow{*} e \quad V$$

$$\langle \text{var Id: } T = E, P, \sigma \rangle \xrightarrow{d} \langle [(\text{Id}, \text{new } L)], [L, v] \rangle$$

$$(\text{dd1}) \quad \langle D_1, P, \sigma \rangle \xrightarrow{D} \langle D'_1, P; \sigma' \rangle$$

$$\langle D_1; D_2, P, \sigma \rangle \xrightarrow{D} \langle D'_1; D_2, P; \sigma' \rangle$$

Se partendo da P e σ la D_1 conduce allo stato $\langle D'_1, P, \sigma \rangle$ allora

la dichiarazione sequenziale conduce a D'_1 con $P; \sigma'$

Continuando la regola

precedente, se da $P; \sigma$ (dd2) $\langle D_2, P[\text{A}_1], \sigma \rangle \xrightarrow{D} \langle D'_2, P[\text{P}_1]; \sigma' \rangle$
 D_2 conduce a D'_2

$$\langle P_1; D_2, P[\text{P}_1], \sigma \rangle \xrightarrow{D} \langle P_1; D'_2, P[\text{P}_1]; \sigma' \rangle$$

La dichiarazione può effettuare
lo stesso passo per D_2

La sintassi non permette
di sovrapporre un ambiente
nel codice, tuttavia
abbiamo questa

$$(\text{dd3}) \quad \langle P_1; P_2, P, \sigma \rangle \xrightarrow{D} \langle P_1[P_2], \sigma \rangle$$

forzatura per permettere al compilatore di funzionare
(l'utente non può)

Scoping statico: Variabili libere vengono legate a tempo di compilazione, costruendo delle chiusure

che assorberà alla

Lambda- astrazione registra i parametri formali di una chiamata

Chiusura: Blocco che lega le variabili libere della funzione usando l'ambiente dinamico (locazioni, non tipi) al momento della dichiarazione

ha un formato D; C

λ -astrazione contiene la chiusura con i formal

f (id) ~~##~~
Free identifiers

La dichiarazione della funzione genera nell'ambiente dinamico un legame tra Id della funzione e la sua λ -astrazione

Scoping dinamico: Variabili libere legate solo al momento di esecuzione \Rightarrow Quando si chiama la funzione, nella chiusura registrare solo il corpo

$$h(1) = 1$$

$$h(N) = 2h(N-1) + 1 \quad N > 1$$

Relazione
di ricorrenza

$$h(N) = 2h(N-1) + 1 = 2[2h(N-2) + 1] + 1$$

$$= 2[2[2h(N-3) + 1] + 1] + 1$$

$$= 2^3 h(N-3) + \underbrace{2^2 + 2 + 1}_{\substack{+4 \\ +2 \\ +1}}$$

Trovo il pattern

$$2^i h(N-i) + \underbrace{2^{i-1}}_{\substack{i-1 \\ -1}} + 3$$

↓
ordine di 2^n

func (α , s_x , e_x) {

$N = \text{array length}$

if ($s_x == e_x$) return $\alpha[s_x]; \boxed{C_1}$

$$h(l) = 1$$

$$c_x = (s_x + e_x) / 2$$

$O(1)$

$$h(n) = ?$$

on 1 = func (α , s_x , c_x)

$$h(N/2)$$

on 2 = func (α , c_x , e_x)

$$h(N/2)$$

return $m_1 + m_2$

C_3

}

$$\begin{aligned} h(N) &= 2h\left(\frac{N}{2}\right) + 1 \\ &= 2\left[2h\left(\frac{N}{4}\right) + 1\right] + 1 \\ &= 2\left[2\left[2h\left(\frac{N}{8}\right) + 1\right] + 1\right] + 1 \\ &= 2 \cdot h\left(\frac{N}{2^i}\right) + (3 \cdot 2^i - 1) \end{aligned}$$

$$\left\lfloor \log_2 N \right\rfloor = i \quad \text{quando}$$

$$2^i + 2^{i-1}$$

$$2^i$$

$$(i = \log N)$$

$$N + N - 1 = O(N) \quad 2^{\log_2 N} \rightarrow N$$

$$T(1) = 1$$

$$T(N) = \frac{N}{2} + 1$$

$$T(N) = \left\lfloor \frac{N}{2} \right\rfloor + 1$$

$$= \left\lfloor \frac{N}{2} \right\rfloor + 2$$

$$= \frac{N}{2} + 3$$

$$\log_2 N + i$$

L15

APPUNTI

funzione da matematica

- Return + (il tipo dell'espressione è in return E') ✓ linguaggi
- Signature (nome + parametri + tipo) ✓ espressioni
- Aumenta la correttezza ✓ (In BNF)
- Definizione (dichiarazione) + uso ≠ definizione / invocazione ✓
- func nome (formali) tipo → T ✓
- name (attuali...) ✓ list di espressioni ✓
- definizioni /
(per invocazione)

compilatore verifica:

- I attuali | = |formali| ✓ ha chiamato "sospese"
- nomi dei formalni distinti ✓ l'esecuzione dal chiamante (formalmente)
- Tipi di attuali e formali nella stessa posizione uguali ✓
- Variabili libere nel corpo della funzione
- 'return' nella funzione a tipo riformato - tipo dichiarato nella firma

$D ::= n; l \mid \text{const} \mid id \mid T \mid E$
 $\text{var} \mid id \mid \text{let} \mid D \mid ; \mid \text{function} \mid \text{td} \mid \text{formal} \mid \text{in} \mid \text{fun}$
 Dichiarazione di corrispondenza ✓
 Nome → Formali
 Tipi → comandi
 → comando

Riuso le sintassi per cose che svolgono lo stesso compito ✓
 sintassi

✓ identificatore
 in posizione di legame (quando abbiamo una definizione) ('var d = 3;') ✓
 occorrenza

Ambiente viene esteso in seguito ad una dichiarazione ✓

↓
 Potranno essere usate nel corpo della funzione ✓

Record di attivazione creato ad ogni chiamata di una funzione

Fine L. 16

$$\begin{aligned}
 T(n) &= ST\left(\frac{4n}{5}\right) + n^9 \sqrt{n} \\
 &= 5 \left[ST\left(\frac{4}{5} \left(\frac{4n}{5}\right)\right) \left(\frac{4n}{5}\right)^9 + n^9 \sqrt{n} \right] \\
 &\quad + 5 \left[ST\left(\frac{4}{5} \left(\frac{4n}{5}\right)\right) + \left(\frac{4n}{5}\right)^9 + \sqrt{\left(\frac{4n}{5}\right)} \right] + n^9 + \sqrt{n} \\
 &= 5 \left[5 \left[ST\left(\frac{4}{5} \left(\frac{4n}{5}\right)\right) + \left(\frac{4n}{5}\right)^9 + \sqrt{\left(\frac{4n}{5}\right)} \right] + \left(\frac{4n}{5}\right)^9 + \sqrt{\frac{4n}{5}} \right] + n^9 + \sqrt{n} \\
 &= 5^i T\left(\frac{4^i n}{5^i}\right) + \left(\frac{4^i n}{5^i}\right)^9 + \sqrt{\frac{4^i n}{5^i}} \quad 9 \log_5 4n ?
 \end{aligned}$$

Mortero Theorem

CHIAMATA DI UNA FUNZIONE

- controllo parma al chiamato
- controllo tipi degli attuali da passare ai formal (non può dare errore perché abbiamo già controllato staticamente)
- TELER + ACCIA
- Assez Valor fondi
- Chiamante (è chi costituisce il controllo)
- Zone di memoria gli lavori

ESECUZIONE - RECORD DI ATTIVAZIONE A fun.

Passaggio per valore (cpia)

- Chiamato (nome)
- Chiamante (per risposta) (nome) (catena dinamica)
- Addr. chiamante (return addr.)
- Cosa (corpo) (risultato della funzione) ← return (addr. result)
- Attuali
- Catena statica (ambiente esterno che la funzione può usare)
- Variabili locali (env?)

record di attivazione

record di
attivazione funzione
chiamante

- Ptr catena dinamica
- " " statica → prossimo record di catena statica
- return address → dove tornare attivazione
- result address → dove salvare
- parametri (formali - attuali)
- variabili locali
- temp variables (compilatore)

catena dinamica

• sequenza di chiamate
• garantisce ordine di esecuzione
catena statica
garantisce referenze dei nomi
nel rispetto della visibilità
(variabili e funzioni)

Record di attivazione

- PUNTATORE CATENA DINAMICA

Record di attivazione funzione chiamante

Poche
due?

- PUNTATORE CATENA STATICA

Record di attivazione precedente (risoluzione nomi non presenti nel blocco corrente, scoping statico)



- INDIRIZZO DI RITORNO

Istruzione da seguire al ritorno della funzione corrente

- INDIRIZZO RISULTATO

Indirizzo dove salvare il risultato attuale della funzione

- PARAMETRI

Associazione formali-attuali

- VARIABILI LOCALI

Ambiente per le variabili locali

- RISULTATI TEMPORANEI

Sposto per variabili temporanee generate dal compilatore

CATENA DINAMICA

Sequenza di chiamate, Garantisce l'ordine di esecuzione LIFO, stack

CATENA STATICA

Scoping statico: Identificatori referenziati nel rispetto della visibilità

Ambiente derivato da dichiarazione di var e const



Variabili locali per

:|

Pila (struttura)

LIFO

LIFO

L.18 Catenza statica: catena, drag

Ci rientrano i parametri formali

= Scoring statico
si riferisce

Divide-et-impera

Paradigma di prog. ~~recursiva~~ ricorsiva

Equazione di ricorrenza (1:02:00)

Equazione ricorsiva che esprime il termine che sto cercando in versioni più piccole di sé stesso

Forma di tipo

$$T(n) = \begin{cases} C & \text{se } n \leq c \\ D(n) + \sum_{i=1}^n T(n_i) + C(n) & \text{se } n > c \end{cases}$$

Divide-et-impera \Downarrow Costo divisione

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{se } n > c \end{cases}$$

Numero dei sottoproblemi dimensione dei sottoproblemi Costo combinazione

+ Esempio Merge Sort

Scriverlo?

Tutto quello a cui posso accedere nell'ambiente in quel momento

Information hiding

L19 | Correttezza

Invariante di ciclo

Initializzazione Vera prima del ciclo

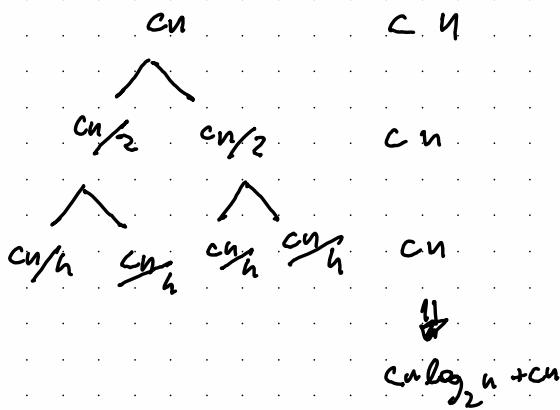
Mantenimento Vera alla fine di un ciclo \rightarrow Vera alla fine della prossima iterazione

Terminazione

Vera alla fine del ciclo

Costo

- Sviluppo
- Albero



$$\log_2 n \\ \text{per divisione}$$

Master Theorem (0:59:00)

Forma $aT(n^b) + f(n)$

$a \geq 1$ $b > 1$ $f(n)$ positiva @ ω

Altrimenti non esce

la soluzione non riduce la complessità

Master theorem

divide - et - impera

$f(n)$	vs	$n^{\log_b a}$	a, b costanti
	↓		$f(n)$ positiva
il più grande			$b > 1$
asintoticamente			$a \geq 1$
è il risultato			

- | | | |
|---|---|----------------|
| 1. $n^{\log_b a} > f(n) \Rightarrow n^{\log_b a}$ | } | A grandi linee |
| 2. $n^{\log_b a} < f(n) \Rightarrow f(n)$ | | |
| 3. $n^{\log_b a} = f(n) \Rightarrow$ whatever | | |

Formalmente:

$$(1) \quad ① \quad f(n) = O(n^{\log_b a - \varepsilon}) . \varepsilon > 0$$

$f(n)$ cresce polinomialmente più lentamente
(d: un fattore n^ε)

$$T(n) = \Theta(n^{\log_b a}) \quad 1:05$$

(2) ②

$$f(n) = \Theta(n^{\log_b a} \log^k n) \quad k \geq 0$$

$f(n)$ e $n^{\log_b a}$ crescono allo stesso modo

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

$$(3) \quad f(n) = \Omega(n^{\log_b a + \varepsilon}) . \varepsilon > 0 \quad f(n) \text{ cresce più velocemente}$$

di un fattore n^ε . E $\frac{f(n)}{f(\frac{n}{b})} \leq c$ $\forall n > n_0$. $T(n) = O(f(n))$

Condizione di continuità

$$T(n) = O(f(n))$$

$f(n)$ deve essere più grande o più piccola polynomialmente

e.g.

$$T(n) = 3T\left(\frac{n}{3}\right) + n$$

$$Q=3$$

$$b=3$$

$$f(n) < n$$

$$n \text{ vs } n^{\log_3 3} \Rightarrow n \text{ vs } n^2$$

$$\overbrace{n^2}$$

$$\begin{aligned} f(n) &= n = O\left(n^{\log_3 3 - \varepsilon}\right) \\ &\quad | \varepsilon=1 \\ &= O(n) \end{aligned}$$

$$\underbrace{f(n) = O\left(n^{\log_3 3 - \varepsilon}\right)}$$

primo caso

$$\boxed{T(n) = \Theta(n^2)}$$

e.g.

$$T(n) = T\left(\frac{2n}{3}\right) + 1 \quad \Theta\left(\frac{n}{b}\right)$$

$$Q=1$$

$$b = \frac{3}{2}$$

$$f(n) = O(1)$$

$$\left. \begin{array}{l} 1 \text{ vs } n^{\log_{\frac{3}{2}} 1} \\ 1 \text{ vs } n^0 = 1 \end{array} \right\} \quad \left. \begin{array}{l} f(n) = \Theta\left(\underbrace{n^{\log_{\frac{3}{2}} 1}}_{k=0} \log^n n\right) \end{array} \right\}$$

$$\boxed{f(n) = \Theta(1)}$$

$$\log(n)$$

Le base
sparisce nel
toto

Eg.

$$T(n) = 3T\left(\frac{n}{4}\right) + n \log n$$

$$\alpha = 3$$

$$b = 4$$

$$f = n \log n$$

$$n \log(n)$$

$$vs \quad n^{\log_4 3}$$

$$n^{\log_4 3}$$

$$\Delta < 1 \quad n^{0.4}$$

$$f(n) = \Omega(n^{\log_4 3 + \varepsilon}) \quad \varepsilon = 0.3$$

$$n \log n = \Omega(n)$$

Secondo controllo

$$a f\left(\frac{n}{b}\right) \leq c f(n) \quad c < 1 \quad n > n_0$$



$$3f\left(\frac{n}{4}\right) \leq \frac{3}{4}(n \log n)$$

$$3\left(\frac{3}{4} \log \frac{n}{4}\right) \leq \frac{3}{4}(n \log n)$$

$$\frac{3}{4} \log \frac{n}{4} \leq \log n$$



sempre maggiore

$$T(n) = \Theta(n \log n)$$

$$\text{eg. } T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

$$a=2$$

$$b=2$$

$$f = n \log n$$

$$n \log n \text{ vs } n^{\log_2 2} = n^2 < n$$

$$\text{caso 3: } \sum_{k=1}^{\infty} (n^{1+\varepsilon})$$

$$f(n) = \sum_{k=1}^{\infty} (n \cdot n^\varepsilon)$$

$$n \log n > n \cdot n^\varepsilon ?$$

$$\log(n) < n^\varepsilon, \varepsilon > 0$$

Non c'è una

differenza polinomiale \Rightarrow Non posso applicare il teorema

$$\text{caso 2: } f(n) = \Theta\left(\underbrace{n^{\log_2 2}}_{n^2} \log^k n\right) \quad k \geq 0 \quad \log_2 2 = 1$$

O: limite superiore

~~S~~: tempo effettivo

Ω : limite inferiore

$$\underbrace{n^2}_{n^2} \underbrace{\log n}_{\log n}$$

$$n \log n$$

L20 (Esercitazione)

L21 prova in itinere

L22 correzione prova (sintassi)

L23

Punti cruciali

per ore:

- Complessità e dimostrazione
- Semantica e sintassi

- Correttezza e discordanza

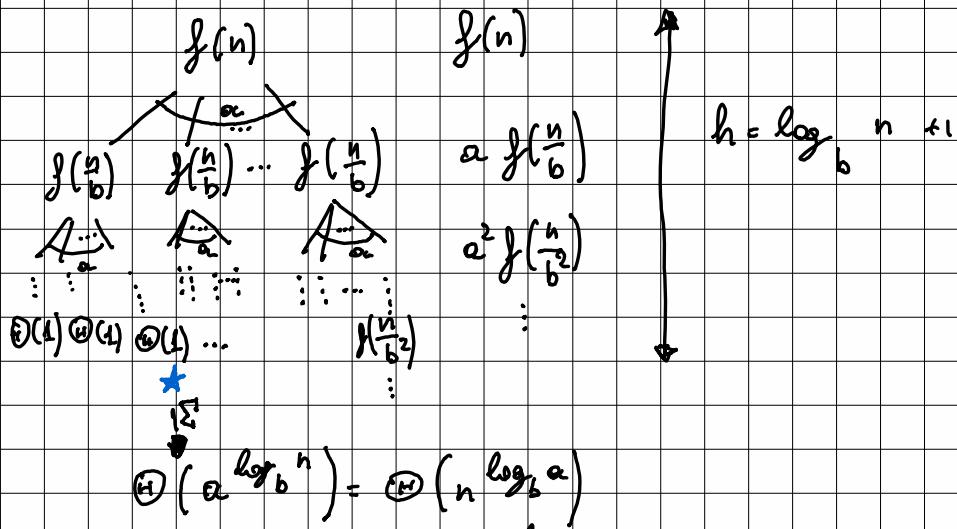
Dimostrazione Master Theorem

- N potenza esatta di b (ipotesi)

$$T(n) = \begin{cases} \Theta(1) & n=1 \\ a T\left(\frac{n}{b}\right) + f(n) & n=b^i \end{cases}$$

Numero di sottoproblemi ↑ Dimensione del sottoproblema ↑ costo unione + divisione
 (semplificato)

Albero di ricorrenza



data che le foglie sono $n^{\log_b a}$
 ed ogni foglia costa 1, il costo è $n^{\log_b a}$

$$T(n) = \underline{\Theta\left(n^{\log_b a}\right)} + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

costo delle foglie il costo effettivo

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)$$

$$\text{①. } f(n) = O(n \log_b a - \varepsilon)$$

$$f\left(\frac{n}{b^j}\right) = O\left(\left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$$

sostituito nella sommatoria



$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon}\right)$$

Sviluppiamo la sommatoria

Da qui si formano le tre strategie (i 3 casi):

$$\text{1. } f(n) = O(n \log_b a - \varepsilon) \quad \varepsilon > 0$$

$$g(n) = O(n \log_b a)$$

$$\text{2. } f(n) = O(n^{\log_b a})$$

$$g(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{3. } a f\left(\frac{n}{b}\right) \leq c f(n) \quad c < 1 \wedge n > n_0$$

$$g(n) = \Theta(f(n))$$

e da qui facciamo uso di una serie di trasformazioni algebriche

$$\begin{aligned}
 & \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon} \\
 &= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{1}{b^j}\right)^{\log_b a - \varepsilon} \\
 &= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{b^\varepsilon}{b^{\log_b a - \varepsilon}}\right)^j \\
 &= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{b^\varepsilon}{b^{\log_b a - \varepsilon}}\right)^j \\
 &= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{b^\varepsilon}{b^{\log_b a - \varepsilon}}\right)^j
 \end{aligned}$$

Serie geometrica!

$$= n \log_b a - \varepsilon \sum_{j=0}^{\log_b n - 1} b^{\varepsilon j} = n \log_b a - \varepsilon \cdot (b^\varepsilon)^{\log_b(n) + 1} - 1$$

Serie geométrica!

$$\sum_{k=0}^{n-1} x^k = \frac{x^n - 1}{x - 1}$$

$$= n \log_b a - \varepsilon \cdot b^\varepsilon \cdot \frac{b^{\varepsilon \log_b n} - 1}{b^\varepsilon - 1}$$

$$= n \log_b a - \varepsilon \cdot b^\varepsilon \cdot \frac{b^\varepsilon \cdot n - 1}{b^\varepsilon - 1}$$

$$= n \log_b a - \varepsilon \cdot \frac{b^{\varepsilon \frac{n}{n-1}} - 1}{b^\varepsilon - 1} \leq n \log_b a - \varepsilon \cdot \frac{b^{\varepsilon} - 1}{b^\varepsilon - 1} =$$

$$= n \log_b a \cdot \frac{1}{y^\varepsilon} \cdot \frac{b^\varepsilon - 1}{b^\varepsilon - 1}$$

$$= n \log_b a \cdot O(\frac{1}{\varepsilon}) \quad b^{\varepsilon \approx \text{const}} = \sim$$

$$= n \log_b a - \varepsilon O(n^\varepsilon)$$

$$= \frac{n \log_b a}{n^\varepsilon} O(n^\varepsilon) = O(n \log_b a)$$

Caso 2

$$f(n) = \Theta\left(n^{\log_b a}\right)$$

$$f\left(\frac{n}{b^j}\right) = \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

$$f(n) = \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}$$

$$= " a^j \cdot n^{\log_b a} \cdot b^{-j \log_b a}$$

$$= " a^j \cdot n^{\log_b a} \cdot a^{-j}$$

$$= " a^j \cdot n^{\log_b a} \cdot \frac{1}{a^j}$$

$$\log_b n - 1$$

$$= \sum_{j=0}^{\log_b n - 1} n^{\log_b a}$$

$$f(n) = \Theta\left(n^{\log_b a}\right)$$

$$\text{allora } g(n) = \Theta\left(n^{\log_b a} \log n\right)$$

$$\Theta\left(n^{\log_b a} \log n\right)$$

$j = ?$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)^{\log_b a}$$

[L24] Ricerca su array di elementi: confrontabili
comparabili

Lineare $O(n)$

dimostrazione che è ottima = dimostrare un lower bound

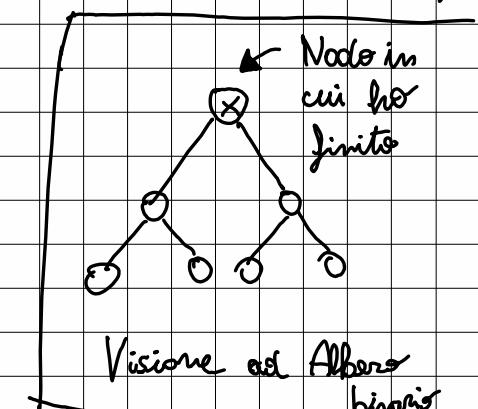
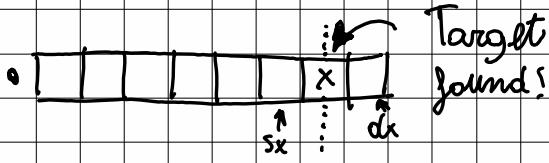
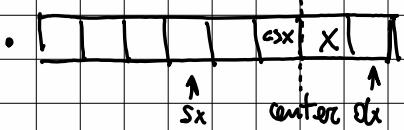
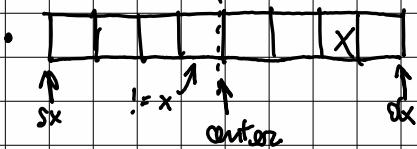


Qualsiasi soluzione S corretta
 non può mettere meno di $O(n)$

(e.g. metterai l'elemento che
 sto cercando in una cella che
 non valuterai)

Array ordinato (collazione ordinata, una collezione
 in cui esiste un
 ordine posizionale
 e.g. $A[i]$ e $A[i+1]$)

$x = \text{target}$ $\text{pos} = -1$ (not found)
 $\exists x = \text{target}$ $\exists x = n$ (len)



Target
 found!

0 1 2 3 4 5 6 7

L 39

Dizionario

Struttura dati, memorizzano un insieme dinamico nel tempo

Operazioni di:

modifica

ordinamento

chiave che individua in modo univoco ~~ogni~~ ogni elemento

u è l'universo delle chiavi

Ricerca $(S, k) = \begin{cases} x & \text{t.c. } x.\text{key} = k \\ \text{NIL} & \text{se } S \text{ non contiene elementi di chiave } k \end{cases}$

Insert (S, x) $S \leftarrow S \cup \{x\}$

Delete (S, x) $S \leftarrow S \setminus \{x\}$

riportamento
da rimuovere

$|S| = \# \text{ di elementi di } S = n \leftarrow$ Complessità in funzione di n

se u è totalmente ordinato, è possibile usare query aggiuntive

Min (S) Max (S) Successore $(S, x) \leftarrow$

elemento di chiave elemento con la più chiave minima massima piccola chiave maggiore di x

Dimensione della Tabella proporzionale con gli elementi nella tabella

$$\dim T = 2 \cdot |S|$$

Usare una chiave per calcolare la posizione di s in una tabella T

$$\dim T = m$$

$$T[0 \dots m-1]$$

Tabella da 0 a m

$$h : U \rightarrow [0, m-1]$$

$$h : U \rightarrow [0, m-1]$$

Ad un valore
nella tabella

funzione
hash mappa
dall'universo
delle chiavi

$$h(k) \in [0, m-1]$$

x viene memorizzato in $T[h(x.\text{key})] = x$

Proprietà della funzione di hashing

- Non può essere iniettiva ($|U| \gg m$)

Può accadere che si verifichino collisioni
(chiavi diverse che risolvono alla stessa posizione)

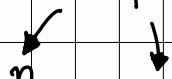
- Deve essere suriettiva (tutto il codominio deve essere coperto)
- Facile da calcolare ($O(1)$)
- Distribuzione uniforme $\forall k \in U. h(k) \in [0, m-1]$
con probabilità $\frac{1}{m}$
- Deterministica ($h(k)$ produce sempre lo stesso valore)

Esempio di funzione

Metodo della divisione

$$\forall k \in \mathbb{U} \quad h(k) = k \bmod n$$

Numeri primi



Distribuisce

bene le chiavi
nelle celle

- ★ $k \bmod n$ deve dipendere da tutte le cifre di k (k deve essere abbastanza piccolo)

Funzione di hashing

Hashing perletto o Gestione delle collisioni

(h iniettiva su S)
difficile da mantenere

Open hash o Concatenamento

- Gestione delle collisioni con concatenamento in HUS (Hashing uniforme semplice) (o liste di trabocco)

T: Array di liste doppie

$$T[j] = \begin{cases} \text{NIL} & \forall x \in S, h(x.\text{key}) \neq j \\ \text{indice di riferimento alla lista doppia} & \text{contenente tutti gli elementi } x \in S, h(x.\text{key}) = j \end{cases}$$

Search

$$T(n) = \begin{cases} O(1) & \text{caso ottimo} \\ O(t) & \text{caso medio} \\ O(n) & \text{caso pessimo} \end{cases}$$

(tutte le chiavi producono lo stesso hash)

Insert (T, x)

inserisce l'elemento x in testa alla lista

$T[h(x.key)]$

$$T(n) = O(1)$$

Delete (T, x)

rimuove l'elemento x dalla lista $T[h(x.key)]$

Dato che $|T| \approx 2|S|$ si cerca di avere liste lunghe 0 o 1

Tabella Elements

$$\text{Fattore di carico } \alpha = \frac{n}{m} = \frac{|S|}{\dim T} \quad \begin{array}{l} \text{Dimensione dizionario} \\ \text{(record)} \end{array}$$

Dimensione tabelle

Si cerca di avere un fattore

$$\alpha = \frac{1}{2} \quad (\text{una cella vuota per ogni cella "piena"})$$

L40

Usando HUS (Hash Uniforme Semplice) e con concatenamento, la ricerca senza successo impiega $\Theta(1 + \alpha)$ al caso medio

$$T_{\text{medio}}(n, m) = \Theta(1 + \frac{n}{m}) = \Theta(1 + \alpha)$$

Dimostrazione:

- Dato la chiave k , accedere alla cella costa $O(1)$, poi è necessario scorrere le liste
- Dato HUS possiamo assumere che gli elementi sono distribuiti uniformemente, quindi ci saranno α elementi inserzioni
- # di accessi alla lista è α

Ricerca con successo

= numero

La ricerca richiede al caso medio $1 + \frac{d}{2} - \frac{d}{2n}$ ispezioni

Dimostrazione

$$T_{\text{medio}}(N, m) = \Theta(1 + d)$$

Calcolata la cella (costante) il # di ispezioni
dipende dal # di elementi che precedono quello
che sto cercando

Dato che ispeziono $\frac{n-i}{m}$ elementi:

Ho n
elementi

Su m celle

$$\# \text{ ispezioni} \underset{\text{al caso medio}}{=} \left(\frac{1}{m} \sum_{i=1}^n \left(1 + \frac{n-i}{m} \right) \right)$$

$$= \frac{1}{m} \sum_{i=1}^n \frac{n-i}{m}$$

$$= \frac{1}{m^2} \cdot n \sum_{i=1}^n (n-i) \quad \sum_{i=1}^n n-i \text{ numeri naturali}$$

$$= 1 + \frac{1}{m^2} \cdot \frac{n(n-1)}{2} = 1 + \frac{n-1}{m \cdot 2}$$

$$= 1 + \left(\frac{n}{m^2} - \frac{1}{m^2} \right) = 1 + \frac{d}{2} - \frac{1}{2 \cdot \frac{n}{d}} = 1 + \frac{d}{2} - \frac{d}{2n}$$

Tabelle Open hash (indirizziamento aperto)

Grazie alla sequenza di inserzione, se una cella è occupata è possibile sceglierne una "successiva"

$$\frac{\text{elementi}}{\text{celle}} \rightarrow \frac{n}{m} = \alpha \leq 1$$

Ci sono più celle che elementi

$$h: U \times [0, m-1] \rightarrow [0, m-1]$$

Universo

delle chiavi

Intero \subseteq

celle nell'array - associabile ai tentativi

Indice nella tabella

h associa quindi una sequenza di inserzione

Ordine con cui si esaminano le celle per la chiave u

$$\langle h(u, 0), h(u, 1), \dots, h(u, m-1) \rangle$$

Dove essere una permutazione dei primi N interi

Inserts (T, u)

$i = \text{tentativi} = 0$

for {

$j = h(u, i)$

$T[j] = \text{nil}?$

$T[j] = K$

ret j

$i + i + 1 > n$?

ret "error overflow"

$$T_{\text{ottimo}} = \Theta(t)$$

$$T_{\text{pessimo}} = \Theta(n) = \Theta(m)$$

$$T_{\text{medio}} = O\left(\frac{1}{1-\alpha}\right) \quad (\alpha < 1)$$

Ricerca (senza cancellazioni)

Search (T, u)

$i = 0$

for :

$j = h(u, i)$

$T[j] == u$?

ret nil

$T[j] == u$?

ret j

$++ i \geq n$?

ret nil

Segue la stessa sequenza per
l'inserimento

Se una cella è vuota, u non è
presente

T ottimo $\Theta(1)$

T pessimo $\Theta(n)$

T medio

operazioni:

$$\left\{ \begin{array}{l} \Theta\left(\frac{1}{1-n}\right) \quad (\alpha < 1) \\ \Theta\left(\frac{1}{\alpha} \ln \frac{1}{1-\alpha}\right) \end{array} \right.$$

Eliminazione

logica o virtuale (o fotta la ricerca)

"DELETION"

{ Cella libera per inserimento
(Cella occupata per ricerca)

Necessaria
modifica all'algoritmo

$(T[j] == \text{"deleted"}) ? \text{nil}$

Analisi al caso medio

Ipotesi: ① $\alpha = \frac{n}{m} < 1$ ② Non ci sono state cancellazioni ③ Hashing uniforme

Almeno una cella libera

- Contiamo il numero di inserzioni, non il tempo
- Ricerca senza successo

La sequenza di inserzione di ogni chiave è una permutazione delle celle di pari probabilità

di accessi in una ricerca senza successo è $\leq \frac{1}{1-\alpha}$

• Dimostrazione:

$x = \# \text{ accessi alla tabella}$

valore medio (atteso) di $x =$

Somma dei valori che x può assumere · la probabilità di assumere quel valore

Probabilità di fare esattamente gli accessi

$$\sum_{i=1}^{\infty} i \cdot \text{Prob}[x = i]$$

$$\sum_{i=1}^{\infty} \text{Prob}[x \geq i]$$

Probabilità di fare almeno gli accessi

$$i=1 \quad \Pr[x \geq 1] = 1$$

$$i=2 \quad \Pr[x \geq 2] = \alpha \quad (\text{Secondo accesa se il primo è occupato da un'altra chiesa})$$

$$i=3 \quad \Pr[x \geq 3] = \frac{n}{m} \cdot \frac{n-1}{m-1} < \alpha^2$$

prima cella
occupata ↑
II cella
occupata ↑
occupata



Maggiorabilità
con α^{i-1}

$$\begin{aligned} \text{Valore medio } x &= \sum_{i=1}^m \Pr[x \geq i] \\ (\# accesi) &= " \alpha^{i-1} \\ &= \sum_{i=0}^{m-1} \alpha^i \leq \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha} \end{aligned}$$

(Probabilità
di avere le
prime $i-1$ celle
occupate)

$$\text{Inserimento al caso medio} \leq \frac{1}{1-\alpha} \quad (\text{senza successo})$$

Si controllano le n-terre celle della ricerca, trovando la prima vuota

α	occupazione T	# accesi ricerca senza succ.
$1/2$	50%	$\frac{1}{1-\frac{1}{2}} \rightarrow 2$
$1/10$	10 %	$\frac{1}{1-\frac{1}{10}} \rightarrow 11$
$9/10$	90%	$\frac{1}{1-\frac{9}{10}} \rightarrow 10$

Ricerca con successo

$$\# \text{ occorsi al caso medio} \leq \frac{1}{2} \ln \frac{1}{1-d}$$

Dim...

L41

Generazione sequenza di ispezione

1. Scansione lineare

$$h': U \rightarrow [0, 1, \dots, m-1]$$

$$h'(u, i) = (h'(u) + i) \bmod m \quad 0 \leq i \leq m$$

- + Cicliamo la tabella da un punto casuale
- il passo è costante
- Non dipende né da i né dalla chiave

Quando si "colpisce" il cluster, si "appende" al cluster facendolo crescere

Lunghi tratti

di celle occupate

adiacenti

quindi

Produce

agglomerati o "cluster primari"

(male)

- Una collisione genera cluster

secondari \Rightarrow Sequenze di ispezione identiche

"m possibili sequenze vs n! permutazioni"

2. Scansione quadratica

Vuole fare il passo diverso da 1 ogni volta

$$m \text{ numero primo } c_1, c_2 \quad c_2 \neq 0 \quad h(u, i) = (h'(u) + c_1 i + c_2 i^2) \% m$$

punto di partenza casuale e passo che dipende dai

sequenze = n (Il passo è il medesimo per tutte le sequenze)

La sequenza dipende ancora dal punto iniziale e basta

Sono ancora presenti cluster secondari!

3. Doppio hash

Due funzioni hash

$$h_1(u) = u \bmod m \quad h_2(u) = u \bmod n < m$$

$$h(u, i) = (h_1(u) + i h_2(u)) \% m$$

+ Punto di partenza della chiave

+ Passo di scansione: dipende dalla chiave

m numero primo

$h_2(u)$ coprime

} MCD

$(h_2(u), m) = 1$

Permutazione

$h_2(u) < m$

+ Prestazioni simili alla teoria

+ No cluster primari

+ No cluster secondari

- Le permutazioni non sono $n!$ ma n^2

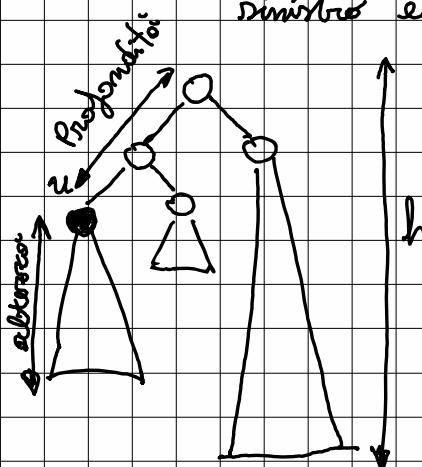
(I possibili distanze $h_2(u)$) \nearrow

Dizionario su albero

Albero binario AB

Struttura definita su un insieme di nodi

- Non contiene alcun nodo (albero vuoto)
- Contiene un nodo detto radice, un sottoalbero sinistro ed uno destro (della radice)



Dimensione: # di nodi

Altezza: Massima distanza di una foglia dalla radice

Altezza di un nodo u: massima distanza del nodo da una foglia

Profondità di un nodo: distanza dalla radice

Rappresentazione in memoria:

padre x.p

Key	
Value	
x.sx	x.dx

foglio sx

foglio dx

Primo il nodo
poi i
sottoalberi

foglio sx
modo
foglio dx

foglio sx
foglio dx
modo

T. root : radice (NIL se vuoto)

Visita Anticipata Simmetrica Posticipata

Alberi binari:

Completo: ogni nodo interno ha due figli

Completamente bilanciato: completo + tutte le foglie sono alla stessa profondità

Bilanciato: di dimensione n ed altezza h se $h = O(\log(n))$

Completo a sinistra/quasi-completo: nodi di profondità $h-1$

formano un albero completamente bilanciato e nodi a profondità h sono tutti a sinistra

1-Bilanciato (bilanciato in altezza): altezza dei sottosalberi
destra e sinistra differiscono al più di un'unità

	Ricerca senza successore	Inserimento	Cancellazione	Build	Spazio
Hash con Concatenamento	$O(n)$ $O(1+\alpha)$	$O(1)$	$\Theta(t)$	$\Theta(m \times (n,m)) \Theta(n)$	$\Theta(\max\{n,m\})$
Hash aperto	$O(n)$ $O(\frac{1}{1-\alpha})$	$O(\frac{1}{1-\alpha})$	$\Theta(1)$	$\Theta(m)$ $\Theta(\frac{n}{1-\alpha})$	$\Theta(m)$
ABR	$O(h)$	$O(h)$	$O(h)$	$O(n \log n)$	$O(n)$
ABR Bilanciato	$O(h)$	$O(h)$	$O(h)$	$O(n \log n)$	$O(n)$
Heap.	$O(h)$ $= O(\log h)$	$O(\log n)$	$O(\log n)$	$\Theta(\frac{n}{h})$ $\Theta(n)$	$O(n)$

Dizionari con alberi binari di ricerca (ABR) (2,4)

Una sintonia simmetrica restituisce la sequenza ordinata delle chiarie

→ Chiavi disposte per
rispettare la
proprietà di ricerca

Costo di una ricerca è $O(h)$

(tracciamo un percorso dalla radice all'albero)

(sendendo exclusivamente
a de o sx)

(In più essere lineare se non ci sono vincoli che lo mantengono bilanciato)

► Tutti i modi minori di
chiave \rightarrow ax e
maggiore a dx

\forall node x in T
 \forall node y nel sottoalbero
 $y \cdot \text{key} < x \cdot \text{key}$

A node z is not sotervalent if
 $z.\text{key} > x.\text{key}$

$$T(n) = O(n)$$

$h = O(\log n)$ caso medio / ottimo

$h = O(n)$ caso pessimo

Successore. (medio y con la più piccola chiave maggiore di x)

- Se il nodo ha un sottoalbero destro: minimo tra i nipoti dx
 - Antenato più vicino ad x, contiene x nel suo sottoalbero sx

$$T(n) = O(h)$$

Non ci serve comparare le chiazze: uriamo la struttura

Insertimento

Nuovi nodi aggiunti come foglie.

$$T(n) = O(\ln)$$

Cancellazione

- Se il nodo è una foglia basta "staccarla"
- Se il nodo ha un solo figlio: collega padre e figlio del nodo
- Due figli: Sostituisce il successore (foglia) con il nodo e poi elimina il nodo

2-3 alberi

L4.6

Binario può essere difficile da tenere bilanciate e si può formare una catena

- Almeno due figli, massimo 3
- Tutti i cammini radice-foglia hanno la stessa lunghezza

$$\log_2 n \leq n \leq \log_3 n$$

Stai valori:

- I dati sono sulle foglie (in ordine crescente)

I nodi intermedi contengono il massimo valore raggiungibile andando a sinistra (e al centro, se ha 3 figli)

Search $\Theta(\log n)$

Insert

- Provare a cercare il valore
- Split, molti nodi com i primi 2 figli, se il genitore ha n figli si fa la chiamata ricorsiva

$$\Theta(\log n)$$

Delete

Se c'è un solo figlio, devo creare un merge con ~~un~~ ^{un} fratello

$\Theta(\log(n))$

Perfoma meglio degli ABR (che sono lineari su h)

Semantica delle funzioni

- Statica: identificatori legati a com tipi corretti
- Dinamica: Simula l'esecuzione (~~ha una memoria~~)

Check di un compilatore nel controllo di una funzione:

1. I formali = l'attuali
2. Nomi dei formalî distinti
3. Tipi di formalî e attuali nella stessa posizione uguali
4. Non ci siano variabili libere e non legabili nel corpo
5. `return` nel corpo della funzione
6. Tipo del return coincide con quello della funzione

Principio di corrispondenza: programmi che hanno
oggetti simili hanno sintassi simile

quinoli

facilità di
apprendimento
e di interpretazione

Semantica statica delle funzioni

Nome della funzione = identificatore in posizione di legame

Id, lista di tipi, tipo di ritorno

occorrenza
di legame

Parametri: formalî in lista di
definizione

Semantica dinamica

1247

$(FD) < \text{func } Id(\text{form}) \rightarrow_d T \{ C; \text{return } E \}, p, o \rightarrow_a \langle (Id,$

$\lambda, \lambda \text{ form.}\{p; C; \text{return } E\}, o \rangle$
astrazione

Scoping statico: mi dico portavo in
pezzo di ambiente che già ho nella chiave

dinamico: mi porto dietro solo il corpo della
funzione

Con la dichiarazione di funzione non ci sono premesse
(quindi non c'è nulla da verificare)

E quindi nell'ambiente creo una coppia Id (nome della
funzione) e λ astrazione (che è composta dai formalii
e la chiusura)

)
composta da

$p';$ corpo della funzione

con scoping dinamico

Variabili libere non legate né dai formalii
né dal corpo

- 1. Assicurarsi che la funzione esiste.
- 2. Valutare gli attuali.
- 3. Applicare regole.

✓ con
scoping
dinamico

$P \models C \rightarrow B \vdash (\text{form})$

Variabili:

libere nel corpo

Formali:

Dichiarazione

Comando

$[(\text{NomeFunzione} . \lambda (\text{form}) . \{ [(\text{variabile}, l_1)] ; C \})]$

chiave (var. libere nel
corpo)

$$(FD2) \quad P(Id) = \lambda \text{ form. } C$$

$$\langle Id(ae), p, \sigma \rangle \xrightarrow{e} \langle \{\text{form. ae}\}, p, \sigma \rangle$$

Se Id esiste
prendo il corpo

e fa la λ astrazione, e lego formalì ed attuali, poi
eseguire il corpo della funzione.

$$\text{form} = ae; \equiv 0;$$

Valutiamo le espressioni degli attuali
e riapplico la funzione
finché E non risolve
ad un valore

$$(FD3) \quad \langle \bar{E}, p, \sigma \rangle \xrightarrow{e} \langle E', p, \sigma \rangle$$

$$\langle E, ae, p, \sigma \rangle \xrightarrow{ae} \langle E', ae, p, \sigma \rangle$$

lista formalì

$$(FD4) \quad \underbrace{\langle ae, p, \sigma \rangle}_{\langle v, ae, p, \sigma \rangle} \xrightarrow{e} \langle ae', p, \sigma \rangle$$

Dopo la regola sopra
per il primo valore,
parm al secondo.

Se dalla lista riusco ad arrivare alla lista ae',
lo "risolve" (l'elemento della lista), che mi farà
aggiungere ad una lista di valori

Lego i formalì agli
attuali avendo i

formalì come valori

lista di formalì = lista di valori

$$(FD5) \quad \langle ae, p, \sigma \rangle \xrightarrow{e} \langle ae', p, \sigma \rangle$$

$$\langle \text{form} = ae, p, \sigma \rangle \xrightarrow{a} \langle \text{form} = ae', p, \sigma \rangle$$

$$(FD6) \quad \text{avr} + \text{form} : p_0, \sigma_0$$

$$\langle \text{form} = av, p, \sigma \rangle \xrightarrow{s} \langle p_0, \sigma_0 \rangle$$

Ho una lista di valori
seguita da una lista di
formalì. Quindi la cosa
deve produrre un nuovo
ambiente e memoria

- nil + nil: \$, \$\xrightarrow{}\$ Da niente non genera un nuovo ambiente

$\text{av} \vdash \text{Form} : \rho, \sigma$

$v, \text{av} \vdash \text{const Id} : \tau, \text{Form} : \rho [(\text{Id}, v)], \sigma$

Una costante

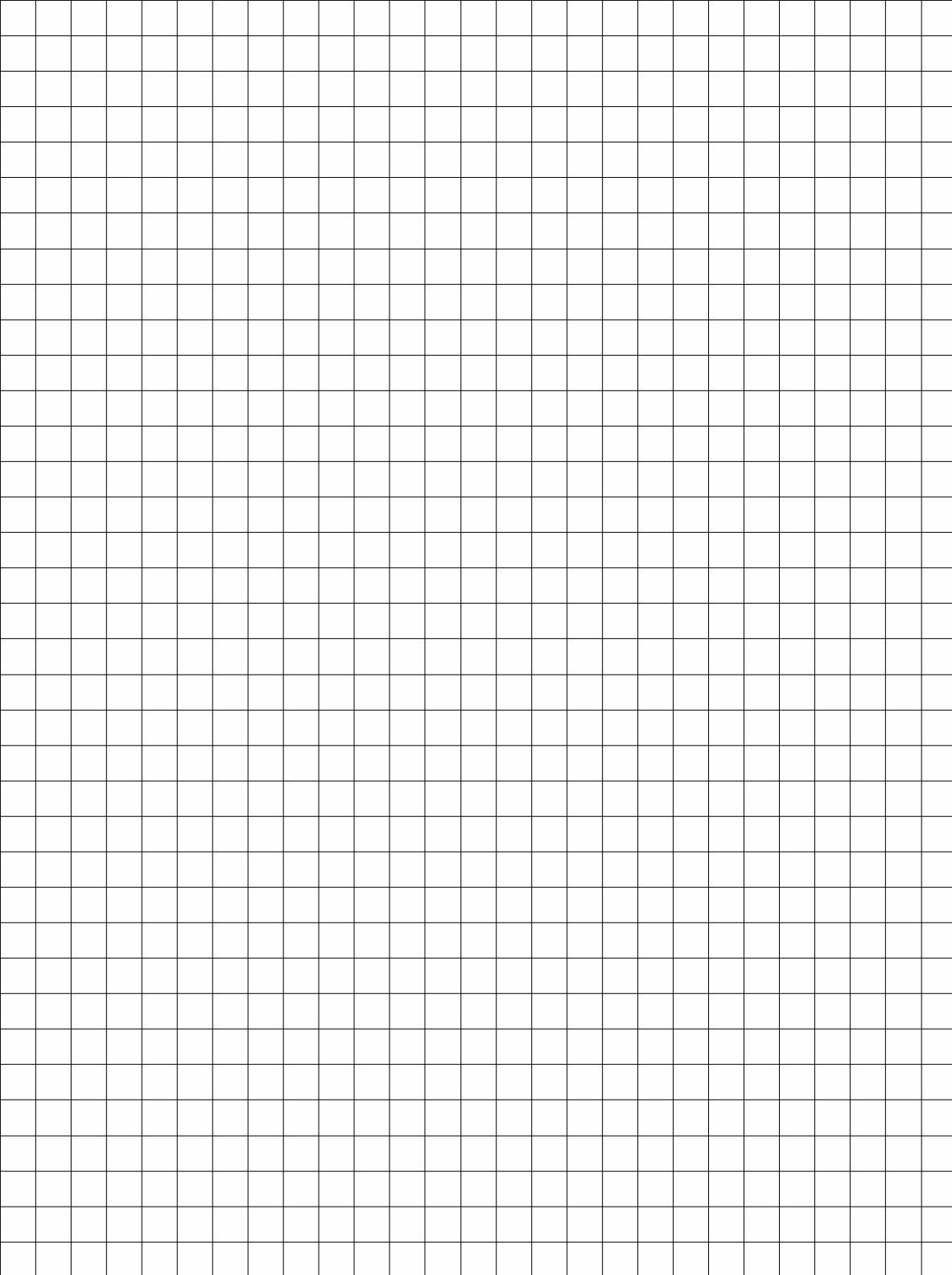
Si genera solo un
nuovo ambiente,
che usiamo per
estendere quello vecchio

$\text{av} \vdash \text{Form} : \rho, \sigma$

$v, \text{av} \vdash \text{var Id} : \tau, \text{Form} : \rho [(\text{Id}, \ell_{(\text{new})})], \sigma [(\ell_{(\text{new})}, v)]$

(FO f) $\langle \text{return } \bar{e}, \rho, \sigma \rangle \rightarrow_c \langle \bar{e}, \rho, \sigma \rangle$

L 56



Programmazione dinamica

Tecnica algoritmica

Per operazioni su dati non disgiunti (ottimizza ogni sottoproblema una volta)

Ricchezza: Top-down

dinamica: bottom-up

Problema di ottimizzazione

Necessari per l'approccio:

- Sottostruttura ottima: la soluzione ottima può derivare dai sottoproblemi con soluzioni ottime
- Sovrapposizione dei sottoproblemi
sottoproblemi diversi deve essere polinomiale nella dimensione dell'input

Fasi della struttura degli algoritmi

1 Definizione dei sottoproblemi e dimensionamento della tabella delle soluzioni

2 Riempiti la tabella con le soluzioni elementari e soluzione dei sottoproblemi

3 Combinazione dei risultati dei sottoproblemi per ottenere la soluzione ottima del problema

4 Restituzione del risultato del problema iniziale

LCS Longest Common Subsequence

Sottosequenza di caratteri fra due stringhe

\emptyset	R	I	S	T	O	R	O		No match
\emptyset	O	O	O	O	O	O	O		Match
\emptyset	O	L	I	I	I	I	I		Match
I	O	I	2	2	2	2	2		
S	O	1	2	3	3	3	3		
O	O	1	2	3	3	h	h		
T	R	L	2	3	h	h	h		
T	O	1	2	3	h	h	h		
O	O	1	2	3	h	9	S	S	

Priorità:

- ↑ up
- ← left
- ↖ ride

R I S O O

Edit distance

Minimizza la distanza (in termini di operazioni per trasformare una stringa in input)

\emptyset	L	A	B	B	R	O			
\emptyset	O	I	2	3	h	S	6		
A	1	1	2	2	3	h	5		No match
L	2	1	2	2	3	h	5		
B	3	2	3	2	2	3	4		
E	h	3	3	3	3	h	4		
R	S	L	h	4	4	4	3	4	
O	6	5	6	5	5	4	3		

L A B B R O

A L B E R O

• • • (3)

Mismatch

Zaino

Ottimo globale attraverso sottostruzione ottima

Strategia greedy - euristica : Ad ogni passo mi fa una scelta localmente ottima per trovare l'ottimo globale

Tecnica che non garantisce la strategia ottima

Non è dinamica

Non considera i sottoproblemi

P.D.

kg trasportabili:

val peso	0	1	2	3	4	5	6	7	8
10 5	0	0	0	0	0	0	0	0	0
6 4	1	0	0	0	0	10	10	10	10
5 4	2	0	0	0	6	10	10	10	10
3	3	0	0	0	6	10	10	10	11

5+6

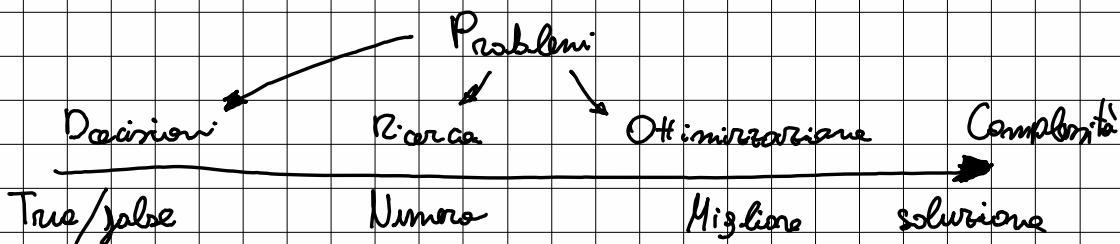
(2+3)

Poco polinomiale: costo polinomiale nel numero ma esp. nel numero di cifre (si hanno ~~wz~~ colonne)

Problemi computazionali

L59

Si basano su un modello computazionale (macchina di Turing)
ed algoritmi e strutture dati usati



Almeno tanto difficile quanto un problema decisionale

Time($f(n)$) e Space($f(n)$)
sono gli insiemi dei problemi decisionali risolvibili in tempo e spazio $f(n)$

Limite inferiore polinomiale
nessuno l'ha provato (non NP/NP-hard)

Classe P: Algoritmo polinomiale ($S \circ T$). Date $c, n_0 > 0$
(Numero di passi elementari sono al più n^c se $n > n_0$
(input di dimensione n)

Classe dei problemi risolvibili in tempo polinomiale

$$P = \bigcup_{c \geq 0} \text{Time}(n^c)$$
$$P\text{Space} = \bigcup_{c \geq 0} \text{Space}(n^c)$$

$P \subseteq P\text{Space}$ (Un algoritmo deve accadere alle altre

$$P\text{Space} \subseteq \text{Exp Time}$$

Graphs

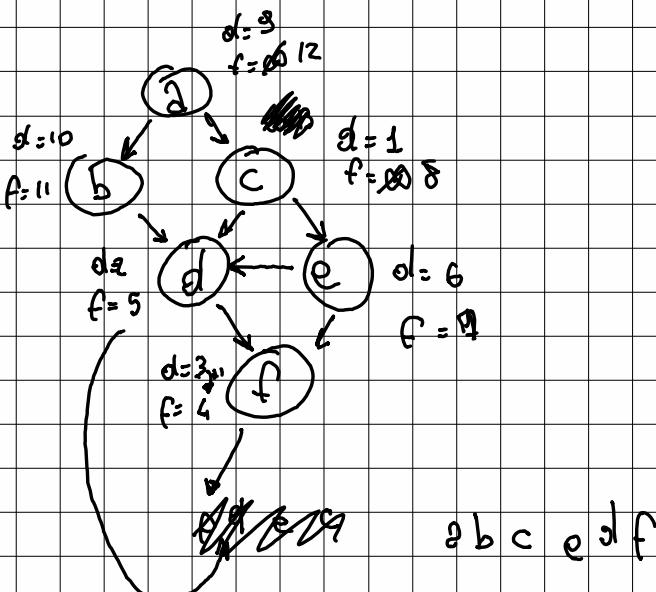
BFS (Breadth, prima i figli)
Non richiede i colori

DFS, Notazione archi

- Tree Albero
- Back Indietro
- Forward Avanti
- Cross Attraversamento

Topological sort (DAG) DFS

Ordinazione secondo topologia (ordina le dipendenze)



$$P \cap \text{Exptime} = \emptyset$$

Letterali: x, y, z

Epressione congiuntiva: espressione come congiunzione di clausole

Quantificato: Presieduta da quantificatori esistenziali ed universali che legano tutte le variabili

Certificato: una soluzione per un problema

NP classe di problemi che ammettono certificati verificabili in tempo polinomiale

Non deterministico - Polinomiale

$$P \subseteq NP$$

$$NP \subseteq P \text{ space}$$

