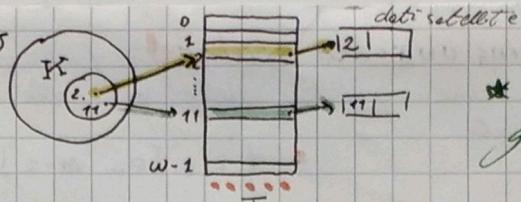


- TABELE HASH AD ARGOSSO/INDIRIZZAMENTO DIRETTO
- U : possibili valori $\rightarrow U = \{0, 1, \dots, w-1\}$
- I : chiavi effettive $\rightarrow |I| = n < |U|$



come sotto
g c'è una

- utilizza un array $T[0 \dots w-1]$
- se K con chiave $x \Rightarrow T[x] = \text{puntatore a } x$
- se K con chiave $x \Rightarrow T[x] = \text{NIL}$

- DIRECT-ADDRESS-SEARCH (array T , chiave x)
return $T[x]$

$$T(n) = \Theta(1)$$

- DIRECT-ADDRESS-INSERT (array T , elem x)
 $T[x.\text{key}] = x$

- DIRECT-ADDRESS-DELETE (array T , elem x)
 $T[x.\text{key}] = \text{NIL}$

! La Tabella è PROPORZIONALE a w e NON al numero di elementi effettivamente memorizzati

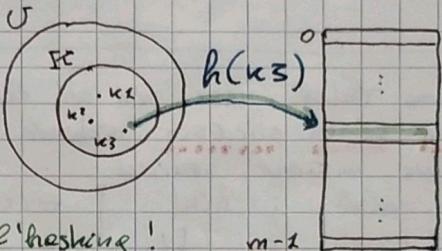
o le chiavi devono essere per forza interi in $[0 \dots w-1]$

- TABELE HASH: NON memorizza un elemento con chiave K nella cella x MA USO una FUNZIONE h che memorizza l'elemento nella cella ottenuta applicando $h(x)$

h : funzione HASH

$h: U \rightarrow \{0, 1, \dots, m-1\}$

m : dimensione della tabella hash, molto più piccolo di $|U|$
l'elemento con chiave x è mappato nella cella $h(x)$



l'hashing!

- può accadere quando $|U| > m$
- accade sicuramente quando $|U| > m$

- PROBLEMA delle collisioni

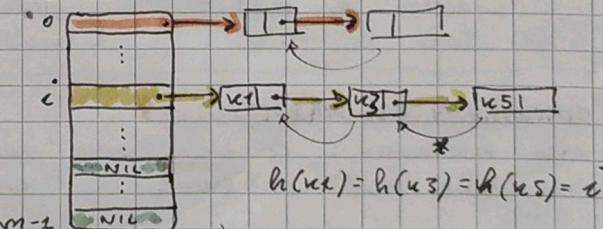
un elemento è mappato in una cella già occupata

GESTIONE COLLISIONI

- concatenamento / liste di collisioni
- indirizzamento aperto

(i) CONCATENAMENTO

- La cella i contiene un puntatore alla testa della lista di tutti gli elementi memorizzati che sono mappati in i , se non ci sono elementi: cioè NIL



$$h(x) = h(x_3) = h(x_5) = i$$

CHAINED-HASH-INSERT (T, x)

inserisce x in testa alla lista $T[h(x.\text{key})]$

$$T(n) = \Theta(1)$$

se il calcolo della fn. hash è costante e non è presente l'elemento da inserire (altrimenti $\Theta(n)$)

CHAINED-HASH-SEARCH (T, K)

ricerca un elemento con chiave K nella lista $T[h(K)]$

tempo proporzionale alla lunghezza della lista contenuta in $h(K)$

CHAINED-HASH-DELETE (T, x)

cancella l'elemento x dalla lista $T[h(x.\text{key})]$

se la lista è doppiamente concatenata!

WORST CASE: quando TUTTE le chiavi sono mappate nella stessa cella, assumendo che sia lunga n , il tempo di ricerca è $\Theta(n)$ più il tempo per calcolare la funzione HASH!!!

il caso medio dipende dal modo in cui la fn. hash distribuisce medianente le chiavi fra le m celle

HASHING UNIFORME SEMPLICE: qualsiasi elemento ha la stessa probabilità di essere mandato in una qualsiasi delle m celle indipendentemente dalle celle in cui sono mandati gli altri elementi.

$$f_i \in [0 \dots m-1] \quad Q[i] = \frac{1}{m}$$

probabilità che una ~~cella~~ chiave qualsiasi finisca nella cella i

chiavi inserite

dimensione tabella

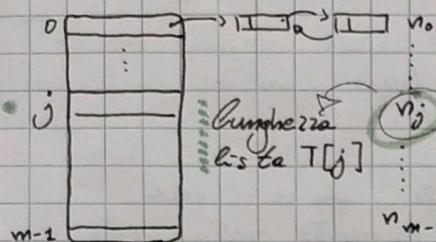
- **FACTOR DI CARICO** di una TABELLA HASH con n chiavi e m celle è

$$\alpha = \frac{n}{m}$$

- $\alpha < 1$ meno chiavi rispetto alla dimensione della tabella

- $\alpha = 1$ tante chiavi quanto le celle

- $\alpha > 1$ si perché le chiavi stanno in strutture esterne!



$$\text{VALORE MEDIO di } v_j \text{ è } \frac{v_0 + v_1 + \dots + v_{m-1}}{m} = \frac{v_j}{m} = \alpha !$$

quindi α è il numero medio di elementi memorizzati in una lista

si assume che la fz. hash sia calcolata in tempo costante

il tempo delle RICERCA di un elemento con chiave k dipende dalla lunghezza $v_{h(k)}$ della lista $T[h(k)]$

in una tabella hash in cui le collisioni sono risolte con il CONCATENAMENTO, una RICERCA SENZA SUCCESSO richiede un tempo $\Theta(1+\alpha)$ nel caso medio e sotto ipotesi di HASHING UNIFORME SEMPLICE

(i) calcolo $j = h(k)$ $\downarrow \Theta(1)$

(ii) accedo a $T[j]$ \downarrow

$$\Rightarrow \Theta(1+\alpha)$$

(iii) scopro la lista $T[j]$ $\downarrow \Theta(\alpha)$ in media, e devo sconfinare tutta lista se è senza successo

se $n = O(m)$ $\Rightarrow \alpha = \frac{n}{m} = \frac{O(m)}{m} = \Theta(1)$ operazioni svolte mediamente in tempo costante

in una tabella hash in cui le collisioni sono risolte con il CONCATENAMENTO, una RICERCA CON SUCCESSO richiede un tempo $\Theta(1+\alpha)$ nel caso medio e sotto ipotesi di HASHING UNIFORME SEMPLICE

(i) calcolo $j = h(k)$ $\downarrow \Theta(1)$

(ii) accedo a $T[j]$ \downarrow mediamente trovo α $\Rightarrow \Theta(1 + \frac{\alpha}{2}) = \Theta(1+\alpha)$

(iii) scopro la lista $T[j]$ \downarrow metà $\Theta(\frac{\alpha}{2})$

COSTRUZIONE FZ. HASH

- **OSSO FORTUNATO**: le chiavi sono numeri reali: casuali e distribuiti in modo indipendente e uniforme nell'intervallo $0 \leq k < 1$

$\Rightarrow h(k) = \lfloor k \cdot m \rfloor$ soddisfa la condizione di hash uniforme semplice

si assume che le chiavi siano numeri NATURALI \Rightarrow se non sono trasformate chiavi CLRS \rightarrow ASCII $C=67 \ L=76 \ R=82 \ S=83 \Rightarrow CLRS = 67 \cdot 128^3 + 76 \cdot 128^2 + 82 \cdot 128^1 + 83 \cdot 128^0$

I. METODO DELLA DIVISIONE

$$h(k) = k \bmod m \quad \begin{matrix} \text{resto della divisione} \\ \text{tra } k \text{ ed } m \end{matrix}$$

sto prendendo solo un pezzo della divisione

è semplice da realizzare

occorre stare ATTENSI alla SCELTA di m !

- EVITARE le POTENZE di 2; se $m = 2^p \Rightarrow h(k)$ rappresenta i p bit meno significativi di k

- EVITARE le STRINGHE interpretate nella base 2^p : colpa delle PERCORRIBILITÀ

$$h(\text{amor}) = h(\text{roma})$$

- **NUMERO PRIMO** NON troppo vicino a una potenza di 2 o 10

$m = 2000$ e so che posso tollerare 3 collisioni \Rightarrow in media ho l'8% di 3 elementi

$$\approx \frac{m}{3} = 666,66667 \Rightarrow m = 701 \text{ numeri prima vicini e quel valore è lontano dalle potenze di 2 e 10}$$

ii. METODO DELLA MOLTIPLICAZIONE

$$h(k) = \lfloor m \cdot k \rfloor \quad k \in [0, 1)$$

data una chiave $k \in U$ la trasforma in un numero in $[0, 1)$:

1. fisso una costante A t.c. $0 < A < 1$
2. calcolo $k \cdot A$
3. estraggo la parte frazionaria di $(k \cdot A) \bmod 1 \in [0, 1)$

$$\approx 4. h(k) = \lfloor m \cdot ((k \cdot A) \bmod 1) \rfloor$$

m NON è più un valore critico

funzione BENE per tutti i valori di A , specialmente $A = \frac{\sqrt{5} - 1}{2} \approx 0,6180$ (inverso del rapporto aureo)

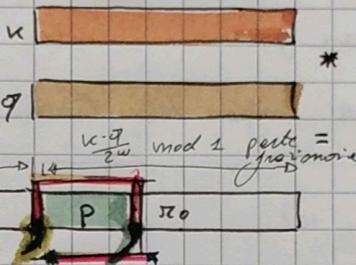
PER SEMPLIFICARE IL CALCOLO DELLA FZ. HASH

- w = lunghezza di una parola di memoria
- K = entra in una singola parola di memoria
- scelgo un intero $0 < q < 2^w$
- scelgo $m = 2^p$
- Scelgo $A = \frac{q}{2^w}$ metto le virgole tra le due parole *

$$1. \text{calcolo } K \cdot A = \frac{K \cdot q}{2^w}$$

$$K \cdot q$$

$$\frac{K \cdot q}{2^w} \text{ parte intera}$$



$$2. h(k) = \lfloor m \cdot (K \cdot A \bmod 1) \rfloor =$$

$$= \lfloor 2^p \cdot (K \cdot A \bmod 1) \rfloor = \text{posiz. } *$$

risultato fz.
bella!

= p bit più significativi della parola meno significativa di $K \cdot q$

iii. HASHING UNIVERSALE: insieme H di funzioni hash opportunamente costruite. il programma sceglie all'inizio una fz. hash $h \in H$ in modo CASUALE (prestazioni molto buone)

(ii) INDIRIZZAMENTO APERTO non ha strutture esterne!

Tutti gli elementi sono memorizzati nella tabella hash.

- ogni cella della tabella contiene un elemento dell'insieme DINAMICO oppure NIL

- per CERCARE un elemento con chiave k :

a. calcolo $h(k)$ ed esamina la cella \rightarrow «ISPEZIONE»

b. se $h(k)$ contiene la chiave $k \Rightarrow$ SUCCESSO

se $h(k)$ contiene NIL \Rightarrow INSUCCESSO

c. se $h(k)$ contiene una chiave che NON è k

\hookrightarrow calcolo un nuovo indirizzo di un'altra cella in base a k e all'ORDINE di ISPEZIONE $(0, 1, \dots, m-1)$

numero d.
ispezioni già fatte
e abv. fallite

d. continuo a scansionare la tabella fino a trovare k (successo) o trovo una cella con NIL (insuccesso) oppure quando ho fatto m ispezioni

FONZIONE HASH: $h: U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$

ordine di ispezione indice celle

$h(k, i)$: posizione della chiave k dopo i ispezioni FALLITE (cella occupata da una chiave diversa!)

per ogni chiave k , la sequenza di ispezioni $(h(k, 0), h(k, 1), \dots, h(k, m-1))$ deve essere una PERMUTAZIONE di $(0, 1, \dots, m-1)$ indici delle tabella hash, in modo che ogni posizione della tabella hash possa essere considerata come una possibile cella in cui vado a inserire una chiave.

IPOTESI: gli elementi sono costituiti solo dalla chiave

• HASH-INSERT (T, k)

• $i = 0$ *

trovato = FALSE

repeat

$j = h(k, i)$

if $T[j] == \text{NIL}$ OR $T[j] == \text{DELETED}$ *

$T[j] = k$

trovato = true

else

$i = i + 1$

vn ispezioni

until trovato OR $i == m$

if trovato

return j

else

ERRORE overflow della tabella hash* // tabella piena

* si inizia SEMPRE dall'ispezione

$i = 0$ per ogni chiave!!!

fallite

* in realtà viene raddoppiata quando è quasi piena con riccalcolo di tutti gli hash

• HASH-SEARCH (T, k)

$i = 0$

trovato = false

repeat

$j = h(k, i)$

if $T[j] == k$

trovato = true doveva esserci la mia chiave ma

è NIL!

else

$i = i + 1$

until trovato OR $T[j] == \text{NIL}$, OR $i == m$

if trovato

return j

else

return NIL

• HASH-DELETE

0	
1	15
2	25
3	118
:	
7	39 X
:	
	73

(a) insert $k = 118$

$h(118, 0) = 2$

COLLUSIONE

$h(118, 1) = 7$

COLLUSIONE

$h(118, 2) = 3$

SUCCESSO

deleted

$\neq \text{NIL}$ Δ
(obv)

(b) delete $k = 39$

(c) search $k = 118$

$h(118, 0) = 2$

$k \neq 118$

$h(118, 1) = 7$

trova NIL e mi dice che 118 non è presente nella Tabella !!

DELETED

uso una costante DELETED per marcire una cella cancellata (vuota a causa di una cancellazione)

però il tempo di ricerca DIPENDE dalle chiavi presenti e da quelle cancellate, non più del fattore d' carico d \approx degrado delle prestazioni

} usare il concatenamento
in caso di molte cancellazioni!

METODI DI SCANSIONE / ISPEZIONE

ipotesi di **HASHING UNIFORME**: ogni chiave ha la stessa probabilità di avere come sequenza di ispezione una delle $m!$ permutazioni di $\{0, 1, \dots, m-1\}$

- $h(x, 0)$ si distribuisce uniformemente sulle m celle
 - $h(x, 1)$ si distribuisce uniformemente sulle $m-1$ celle
 - $h(x, 2)$ si distribuisce uniformemente sulle $m-2$ celle
 - ...
- ... } circa un hashing uniforme semplice ad ogni iterazione

(i) ISPEZIONE LINEARE

voglio definire $h(x, i)$

- funzione auxiliaria $h': U \rightarrow \{0, 1, \dots, m-1\}$

- il metodo di ispezione lineare per $i=0, \dots, m-1$: $h(x, i) = (h'(x) + i) \bmod m$

$$F.AUX \quad h'(x) = x \bmod 13$$

voglio inserire keys = 69, 4, 31, 53 fondamentale! non posso cambiare!

$$SCANSIONE LINEARE \quad h(x, i) = (x \bmod 13 + i) \bmod 13$$

0	
1	
2	
3	
4	69
5	4
6	31
7	43
8	
9	
10	
11	
12	
13	

quando entra alla fine ripete dall'inizio

$$h(69, 0) = 4 \quad (69 \bmod 13)$$

$$h(4, 0) = 4 \quad \text{COLLUSIONE} "$$

→ scansione lineare

$$h(4, 1) = 4+1=5$$

$$h(31, 0) = 5 \quad \text{COLLUSIONE} "$$

→ scansione lineare

$$h(31, 1) = 6$$

$$h(43, 0) = 5 \quad \text{COLLUSIONE} "$$

→ scansione lineare

$$h(43, 1) = 5+1=6 \quad \text{COLLUSIONE} "$$

$$h(43, 2) = 5+2=6 \quad \text{COLLUSIONE} "$$

$$h(43, 3) = 5+3=7$$

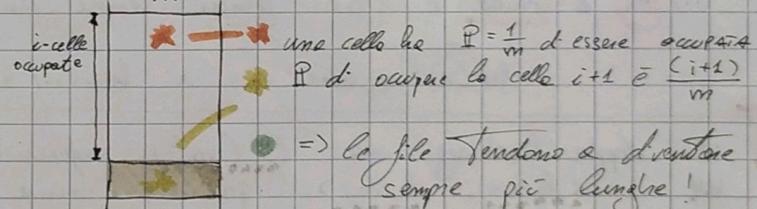
l'ordine di inserimento è

69, 4, 31, 53 fondamentale!

cambiarlo!

molte collisioni sono un problema

ci sono soltanto m sequenze distinte di ispezioni per ogni chiave! appena fisso il punto di partenza sono univocamente determinate



(ii) ISPEZIONE QUADRATICA

$$h(x, i) = (h'(x) + c_1 i + c_2 i^2) \bmod m$$

\rightarrow $\begin{cases} h'(x) \text{ FUNZIONE HASH AUX} \\ c_1, c_2 \neq 0 \text{ e sono costanti aux} \\ i=0, \dots, m-1 \end{cases}$

→ i valori di c_1 e c_2 e m DEVONO essere scelti BENE in modo da generare TUTTI gli indici della tabella: $(h(x, 0), \dots, h(x, m-1))$ sia una permutazione di $\{0, 1, \dots, m-1\}$

Funziona bene con $c_1 = c_2 = \frac{1}{2}$ e $m = 2^p$

anche qui la 1^a posizione $h(x, 0)$ determina l'intera sequenza di ispezioni, quindi ad ogni chiave ci sono soltanto m sequenze di ispezioni distinte "

(iii) ADDENSAMENTO SECONDARIO

se due chiavi DISTINTE x_1 e x_2 hanno lo stesso valore hash per la fz. auxiliare $h'(x_1) = h'(x_2)$ allora hanno la STESSA SEQUENZA di ISPEZIONI. (x_1 e x_2 sono INDISTINGUIBILI!)

(iii) DOPPIO HASHING

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

determina 1^a posizione

ispezioni fatte

\downarrow

$i \cdot h_2(k)$

\rightarrow determina il passo!

$\left\{ \begin{array}{l} h_1 \text{ e } h_2 \text{ f.z. hash aux} \\ i = 0, \dots, m-1 \end{array} \right.$

* rende sempre valori
strettamente minori di m
e quindi h_2 è RELATIVAMENTE
PRIMO di m

0	
1	
2	43
3	
4	69
5	5
6	
7	
8	
9	4
10	
11	
12	

$$h_1(k) = k \bmod 13$$

$$h_2(k) = 1 + (k \bmod 12) *$$

$$h(k, i) = (k \bmod 13 + i(1 + k \bmod 12)) \bmod 13$$

keys $\langle 69, 4, 32, 43 \rangle$

1^a isezione use

$$h(69, 0) = 69 \bmod 13 = 4 \text{ solo } h_2$$

$$h(4, 0) = 4 \bmod 13 = 4 \text{ collisione!}$$

\rightsquigarrow doppio hashing

$$h(4, 1) = h_1(4) + h_2(4) = 4 + 5 = 9$$

$$h(32, 0) = 5$$

$$h(43, 0) = 43 \bmod 13 = 4 \text{ collisione!}$$

\rightsquigarrow doppio hashing

$$h(43, 1) = h_1(43) + h_2(43) = (4 + 1(1 + 43 \bmod 12)) \bmod 13 =$$

$$= 4 + 12 = 15 \bmod 13 = 2$$

SCELTA di h_2

il valore deve essere RELATIVAMENTE PRIMO con la dimensione m della tabella hash ($\text{mcd}(x, m) = 1$!!!)
in modo che possa essere ispezionata l'intera tabella!

assumo $i, i' < m$ e $h(k, i) = h(k, i') \Rightarrow i = i'$ (sotto ipotesi d. sopra)

se si ha

(i) si sceglie m come potenza di 2 e definire h_2 in modo che produce sempre numeri dispari

$$m = 2^k \quad \wedge \quad h_2(k) = 2h'(k) + 1$$

(ii) m è PRIMO e h_2 genera sempre un numero positivo minore di m

$$h_2(k) = k \bmod m \quad \wedge \quad h_2(k) = 1 + (k \bmod m') \text{ con } m' < m$$

il doppio hashing usa $O(m^2)$ sequenze di ispezioni poiché ogni possibile coppia $(h_1(k), h_2(k))$ produce una sequenza DISTINTA di ispezioni
ma offre prestazioni migliori riducendo le collisioni e eliminando il problema dell'addensamento secondario

RISTRUTTURAZIONE

INDIRIZZAMENTO APERTO : raddoppiare la dimensione della tabella hash (deve REINSERIRE TUTTE le chiavi) quando $\alpha > 0.5$
 \hookrightarrow ELIMINA TUTTE le posizioni DELETED

CONCATENAMENTO : quando $d \geq 2$

• COMPRESSENTE HASHING A INDIRIZZAMENTO APERTO

[ved. pag. 3-4]

(ip.1) si assume hashing uniforme

(ip.2) non ci sono cancellazioni

→ analisi in termini del fattore di carico α

$$\alpha = \frac{n}{m}$$

$$0 \leq \alpha \leq 1$$

\rightarrow non lo struttura esterna
 \rightarrow grande n tutte le celle occupate

• [RICERCA] senza successo :

Nell'ipotesi di dato hashing uniforme, data una tabella hash a indirizzamento aperto con un fattore di carico $\alpha = \frac{n}{m} < 1$, il numero atteso (medio) d'ispezioni in una ricerca senza successo è al massimo $\frac{1}{1-\alpha}$

i spessione | probabilità $\xrightarrow{\text{sono celle libere, mi fermo alla 1^a vuota}}$

1^a	1
2^a	$\frac{n}{m} = \alpha \rightarrow$ prob. che la cella della 1^a ispezione fosse occupata
3^a	$\frac{n}{m} \cdot \frac{n-1}{m-1} = \alpha \cdot \frac{n-1}{m-1}$
4^a	$\frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} = \alpha \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2}$

=> VALORE ATTESO di ISPEZIONI

$$= 1 + \alpha + \alpha^2 + \dots + \infty \leq \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha} \quad 1 \leq 1 < 1 !$$

Se α è costante => il tempo medio d'una ricerca senza successo :

Se $\alpha = 0.5$

Se $\alpha = 0.9$

Le prestazioni si degradano al diminuire delle celle libere !

[INSEGNIMENTO]

L'inserimento d'un elemento in una tabella hash a indirizzamento aperto con un fattore d'carico α richiede in media $\frac{1}{1-\alpha}$ ispezioni nell'ipotesi di hashing uniforme.

dim.

un elemento è inserito solo se c'è spazio, quindi $\alpha < 1$. L'inserimento richiede una ricerca senza successo (trovare posizione vuota) seguita dall'inserimento della chiave nella 1^a cella vuota che viene trovata.

ma allora il numero di ispezioni è al massimo $\frac{1}{1-\alpha}$

• [RICERCA] CON SUCCESSO :

Date una tabella hash a indirizzamento aperto con fattore di carico $\alpha < 1$, il numero medio atteso d'ispezioni in una ricerca CON successo è al più $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

Se α è costante => una ricerca con successo è in media $\Theta(1)$

Se $\alpha = 0.5$ => numero medio di ispezioni è $\leq 1,387$

Se $\alpha = 0.9$ => numero medio di ispezioni è $\leq 2,559$

In generale il concatenamento ha prestazioni migliori !

STRUCTURE DATI

• TAB. HASH CON LISTE DI COLLISIONE { medio worst }

• TAB. HASH CON INDIRIZZAMENTO APERTO { medio worst }

• MAX HEAP worst

• ALBERI BINARI DI RICERCA (BST) worst

RICERCA

$\Theta(1+\alpha)$

$O(n)$ unica lista con tutti gli elementi

$O(\frac{1}{1-\alpha})$

$O(n)$ trova tutte le celle occupate

$O(n)$

$O(h)$ ^{h ottenuta all'ultima}

SUCCESSIONE POSIZIONI

$O(m+n)$ chiavi

$O(m+n)$

$O(m)$

$O(m)$

$O(n)$

$O(h)$

COSTRUZIONE

$\Theta(n)$

$\Theta(n)$

$O(\frac{n}{1-\alpha})$

$O(n^2)$

$\Theta(n)$

$\Theta(n \log n)$

MASSIMO

$\Theta(m+n)$

$\Theta(m+n)$

$\Theta(m)$

$\Theta(m)$

$\Theta(1)$

$O(h)$

* PROGRAMMAZIONE DINAMICA * *(tecniche di progettazione di algoritmi)*
 si applica quando un problema si riduce a un insieme di sottoproblemi più piccoli ma i sottoproblemi **NON** sono **indipendenti!** (nel divide - et - impera lo sono) (potrai risolvere più volte lo stesso problema)

RISOLVO ogni SOTTOPROBLEMA UNA SOLA VOLTA, memorizo la soluzione e la uso quando incontro di nuovo il sottoproblema!

* ADATTA A PROBLEMI di OTIMIZZAZIONE: molte possibili soluzioni, ciascuna con un costo e voglio trovare **UNA** soluzione ottima (potrai avere più di una) ↳ costo ritorno o massimo

* COME FARE UN ALGORITMO

- caratterizzazione della **STRUTTURA** di una soluzione ottima
- definizione **RICORSIVA** del valore di una soluzione ottima
- calcolo del **VALORE** di una soluzione ottima ↳ oppure **BOTTOM-UP** o **TOP-DOWN**
- individuazione di **UNA** soluzione ottima sulla base delle informazioni ricavate dal punto (iii)

(i) PROBLEMA del TAGLIO delle ASTE

un'azienda produce delle aste d'acciaio e le vende a pezzi.

- Le aste prodotte hanno una certa lunghezza n
 - Sul mercato i pezzi hanno un **PREZZO** che **DIPENDE** dalla lunghezza
- Trovare il modo di tagliare le aste che **CLASSIFIZZI** il guadagno (assumendo che il costo del taglio sia **irrilevante**)

• n : lunghezza ASTA

• p_i : tabella di prezzi $i = 1, \dots, n \Rightarrow p_i$: prezzo di un pezzo di lunghezza i

• π_n : ricavo massimo per un'asta di lunghezza n (io devo trovarlo!)

TAGLI	i	1	2	3	4	5	6	7	...
PREZZI	p_i	1	5	8	9	10	17	17	...

io ho un'asta
lunga $n=7$ → voi modi!

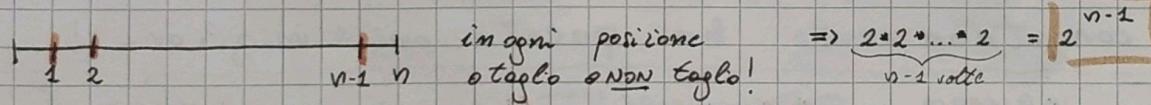
$$\pi_7 = ?$$

$$\pi_7 = 10$$

$$\pi_7 = 18 \quad \left. \begin{array}{l} \text{Ca soluzione} \\ \text{ottima} \\ \text{NON è} \\ \text{unica!!!} \end{array} \right\}$$

se analizzo tutto ottengo
una soluzione ottima
NON è unica!!!

in quanti modi diversi posso tagliare un'asta di lunghezza n ?



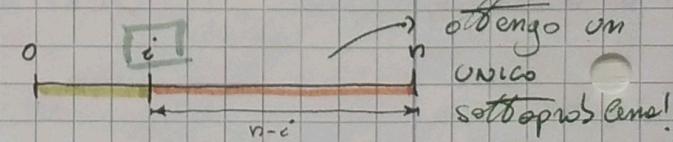
(ii) esprimo il ricavo massimo in modo ricorsivo

$$\begin{aligned} & \pi_0 = 0 \\ & \pi_n = \max \left\{ p_n, \pi_1 + \pi_{n-1}, \pi_2 + \pi_{n-2}, \dots \right\} \\ & \text{non taglio NIENTE} \end{aligned}$$

↳ divido in due pezzi che a loro volta saranno poi divisi

PROPRIETÀ della struttura ottima: La soluzione ottima è assemblabile come combinazione di soluzioni ottime di sottoproblemi

- △ invece di tagliare in 2 parti, ulteriormente divise, posso
- tagliare un pezzo in modo definitivo
 - suddividere la parte che resta



$$\begin{aligned} & \pi_0 = 0 \\ & \pi_n = \max \left\{ p_i + \pi_{n-i} \right\} \text{ dove } i \text{ è la posizione del taglio} \quad (\text{se } i=n \Rightarrow \text{nessun taglio}) \end{aligned}$$

```

    CUTROD (p, n)
    if n=0
        return 0
    else

```

```

        q = -1 /* massimo dei prezzi finire, dato che sono tutti > 0 */
        for i=1 to n
            q = max (q, p[i] + CUTROD (p, n-i))
        return q
    
```

$$T(n) = \begin{cases} ① & \text{se } n=0 \\ ① + \sum_{i=1}^n T(n-i) & \text{se } n>0 \end{cases}$$

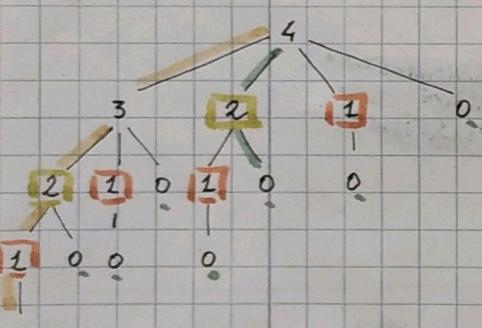
$$T(n) = 1 + \sum_{i=1}^n T(n-i) = 1 + \sum_{\substack{j=0 \\ j=n-i}}^{n-1} T(j) = 2^n \Rightarrow \Theta(2^n)$$

per induzione su n che $T(n) = 2^n$

$$- n=0 \quad T(0) = 1 = 2^0 \quad \text{OK}$$

$$\begin{aligned} - n>0 \quad & \text{assumo vera la proprietà per } n \text{ e dimostro per } n+1 \\ & [\text{per def.}] \quad T(n+1) = 1 + \sum_{j=0}^n T(j) = 1 + \sum_{j=0}^{n-1} T(j) + T(n) = T(n) + T(n) = 2T(n) = \\ & [\text{per ip. induttiva}] \quad = 2 \cdot 2^n = 2^{n+1} \quad \text{QED} \end{aligned}$$

es. ALBERO di ricorsione $n=4$



• ho tante foglie quanti sono i possibili modi di tagliare le ASSE!

$$2^{n-1} = 2^{4-1} = 2^3 = 8$$

• ogni cammino è un possibile modo d'tagliare l'oste!

• uno stesso sottoproblema viene risolto più volte!

★ ci sono $n=4$ sottoproblemi DISTINTI!

↳ mi conviene memorizzare le soluzioni per ogni sottoproblema e riusarle invece di calcolarla di nuovo!

• È UTILE USARE la PROGRAMMAZIONE

se i sottoproblemi distinti sono in numero POLINOMIALE e ognuno si risolve in TEMPO POLINOMIALE quindi memorizzandone le soluzioni ed evitando di ricalcolarla si ottiene un ALGORITMO POLINOMIALE.

« compromesso fra tempo di esecuzione e spazio di memoria »

TECNICHE DI COSTRUZIONE di ALGORITMI

(i) **TOP-DOWN**: ricordo in una tabella (vettore, hash) le soluzioni dei problemi già risolti
 $n \rightarrow \dots \rightarrow 1$
 « memoization »
 ↳ usa la RICORSIONE (più costosa) ma prende le sol. solo le cose interessanti

(ii) **BOTTOM-UP**: ordina i problemi in base alla dimensione, parte dai più piccoli, li risolve e memorizza le soluzioni ottenute
 $1 \rightarrow \dots \rightarrow n$
 ↳ risolve tutto!

ASINTOTICAMENTE il costo è lo stesso per entrambi

$p[1..n]$: vettore prezzi
 $p[i]$: prezzo ASSE lungo i
 n : lunghezza testa da tagliare

MEMOIZED-CUTROD (p, n)
 $\pi[0 \dots n]$ /* nuovo vettore dove memorizzare i ricavi massimi */
 for $i=0$ to n
 $\pi[i] = -\infty$ /* $\pi[i]$ = ricavo ottimo per asta di lunghezza i , pren $p_i > 0$ */
 return MEMOIZED-CUTROD-AUX (p, n, π)

MEMOIZED-CUTROD-AUX (p, j, π)
 if $\pi[j] < 0$ /* non ho ancora calcolato il valore */
 if $j=0$ /* posso già metterlo */
 $\pi[j] = 0$ iniziando $\pi[0]=0$ */
 else /* viene fatto una volta per ciascun problema */
 $q = -1$
 for $i=1$ to j
 $q = \max\{q, p[i] + \text{MEMOIZED-CUTROD-AUX}(p, j-i, \pi)\}$
 $\pi[j] = q$ /* memorizza il valore massimo */
 return $\pi[j]$ /* se $\pi[j] > 0$ il problema è già risolto! */

una chiamata ricorsiva per risolvere un problema precedentemente risolto. Termina immediatamente in tempo $\Theta(1)$, si arriva all'else **UNA SOLA VOLTA** per ciascun problema $j=1, \dots, n$

la soluzione di un sottoproblema di dimensione n effettua n iterazioni, in totale oltre

$$\sum_{j=1}^n j = \frac{n(n+1)}{2} = \Theta(n^2)$$

BOTTOMUP-CUTROD (p, n)
 $\pi[0 \dots n]$ /* nuovo vettore */
 $\pi[0] = 0$
 for $j=1$ to n /* j -volte */ /* è già stato calcolato perché
 è minore di $\pi[j]$ */
 for $i=1$ to j /* i -volte */
 $q = \max(q, p[i] + \pi[j-i])$ $\Theta(1)$
 $\pi[j] = q$
 return $\pi[n]$

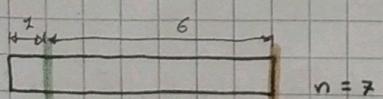
$$T(n) = \sum_{j=1}^n j \cdot \Theta(1) = \Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n^2)$$

no chiamate ricorsive
è "migliore"

voglio anche sapere dove fare il taglio! (come costruire la soluzione (iv))

EXT-BOTTOMUP-CUTROD (p, n)

$\pi[0 \dots n]$
 $s[1 \dots n]$ memorizza le i che mi fa fare il taglio
 $\pi[0] = 0$
 for $j=1$ to n migliore per ora
 $q = -1$
 for $i=1$ to j
 if $q < p[i] + \pi[j-i]$
 $q = p[i] + \pi[j-i]$
 $s[j] = i$
 $\pi[j] = q$
 return $\langle \pi, s \rangle$



i	0	1	2	3	4	5	6	7
$\pi[i]$	0	1	5	8	10	13	17	18
$s[i]$	0	1	2	3	2	2	6	1

PRINT-CUTROD-SOLUTION (p, n)

$\langle \pi, s \rangle = \text{EXT-BOTTOMUP-CUTROD}(p, n)$

while $n > 0$
 print $s[n]$
 $n = n - s[n]$

$$T(n) = \Theta(n^2) + O(n) = \Theta(n^2)$$

ext bottomup
cutrod

PRINT,
quando taglio
tutto i pezzi in 1

$n=7$

$$S[n] = 1 \quad \text{taglio ottimale}$$

$$n = 8 - S[n] = 7 - 1 = 6$$

L'asta viene tagliata in due pezzi 1 e 6
ottenendo sicuramente il valore massimo (18) sia
che fornisce della soluzione

full code
per il
metodo TOP

down!

III MASSIMA SOTTOSEQUENZA COMUNE (LCS)* ▲ va mantenuto l'ordine degli elementi ▲

DNA: A G C T molecole base DNA (alfabeto 4 lettere)
sequenze di DNA

S₁: ACTAACCCTG \Rightarrow LCS: ATAACCA
S₂: ATCACAC

possono essere dei buchi, le sottosequenze non per forza ha carattere consecutivo!

► SOTTOSEQUENZA di $x = x_1 \dots x_m$ è x_{i_1}, \dots, x_{i_n} con $\begin{cases} i_1 \dots i_n \in \{1 \dots m\} & \text{indici} \\ i_1 < i_2 < \dots < i_n & \text{indici crescenti!} \end{cases}$

* Date due sequenze $X = x_1 \dots x_m$ e $Y = y_1 \dots y_n$ trovare una sequenza W tale che
- sia una sottosequenza di X e Y
- sia di lunghezza massima

⚠ NON è unica la LCS \Rightarrow LCS(X, Y): insieme delle LCS di X e Y

$$\begin{aligned} X &= AC \\ Y &= CA \\ \text{LCS}(X, Y) &= \{A, C\} \end{aligned}$$

• BRUTE FORCE ESPONZIALE: genera tutte le sottosequenze di X , verifica se è sottosequenza di Y e memorizza la più lunga
 $X = x_1 \dots x_m \Rightarrow 2^m$ sottosequenze perché ogni sottosequenza può appartenere o meno alla soluzion sequenze

► PREFISSO: date $X = x_1 \dots x_m$, per $k \leq m$, indico con X^k il PREFISSO di lunghezza k di X

$$\begin{aligned} X &= ACG \\ X^0 &= \emptyset \quad /* \text{sottosequenza vuota} */ \\ X^1 &= A \\ X^2 &= AC \\ X^3 &= ACG \end{aligned}$$

$$X^k = x_1 \dots x_k$$

In generale $X = x_1 \dots x_m$ ha $m+1$ PREFISSI quindi riducendo il problema LCS sui prefissi permette di ottenere $O(m \cdot n)$ sottoproblemi dove $X = x_1 \dots x_m$ e $Y = y_1 \dots y_n$

• SOTTOSEQUENZA OTTIMA PER LA LCS

date $X = x_1 \dots x_m$ e $Y = y_1 \dots y_n$ sequenze e $W = w_1 \dots w_k \in \text{LCS}(X, Y)$, allora

(i) se $x_m = y_n \Rightarrow w_k = x_m = y_n$ e $W^{k-1} \in \text{LCS}(X^{m-1}, Y^{n-1})$

(ii) se $x_m \neq y_n$: l'ultimo carattere è in comune!

(a) se $w_k \neq x_m \Rightarrow$ ~~W~~ $W \in \text{LCS}(X^{m-1}, Y)$

(b) se $w_k \neq y_n \Rightarrow W \in \text{LCS}(X, Y^{n-1})$

dimostrazione

(i) se $w_k = x_m = y_n$ allora (per assurdo!) ~~non~~ potrei costruire una sequenza $W' \subsetneq W$ che sarebbe una sottosequenza di X e Y . MA avrebbe lunghezza maggiore di $|W|$ \leadsto assurdo perché $W \in \text{LCS}(X, Y)$

W^{k-1} è sottosequenza di X^{m-1} e Y^{n-1} e voglio dimostrare che $W^{k-1} \in \text{LCS}(X^{m-1}, Y^{n-1})$
se per assurdo non fosse così esisterebbe $W' \in \text{LCS}(X^{m-1}, Y^{n-1})$ e ora che $|W'| > |W^{k-1}| = k-1$

dato che $w_k = x_m = y_n$ ho che $|W'_{w_k}| < |W_{w_k}|$ sono sottosequenze di X e Y
è proprio $|W'| < |W_{w_k}| \leadsto$ assurdo! W NON è più ottimale!

(ii) [a] se $x_m \neq y_n$ e $w_k \neq x_m$ allora W è sottosequenza di X^{m-1} e Y (per ipotesi)

da voglio dimostrare che $W \in \text{LCS}(X^{m-1}, Y)$

per assurdo suppongo che ci sia una sottosequenza comune di X^{m-1} e Y più lunga di W , chiamate W' $\Rightarrow |W'| > |W|$

essendo W' una sottosequenza comune di X^{m-1} e Y lo è anche di X e Y ma questo contraddice il fatto che W è d. lunghezza massima $\in \text{LCS}(X, Y)$

SOLUZIONE RICORSIVA
dati $X = x_1 \dots x_m$ e $Y = y_1 \dots y_n$
indico con $c[i, j]$ = la lunghezza della $\text{LCS}(x^i, y^j)$ con $0 \leq i \leq m$ e $0 \leq j \leq n$

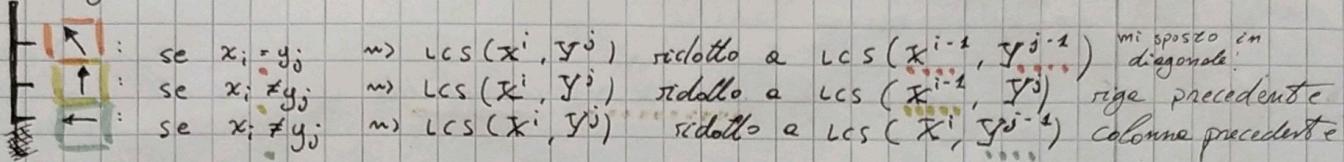
$$c[i, j] = \begin{cases} 0 & \text{se } i=0 \text{ OR } j=0 \text{ una stringa è vuota!} \\ c[i-1, j-1] + 1 & \text{se } x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & \text{se } x_i \neq y_j \end{cases}$$

sto usando il teorema!

(m · n)

BOTTORUP

- $c[i, j]$ = lunghezza della $\text{LCS}(x^i, y^j)$
- $b[i, j]$ = informazioni utili per recuperare la soluzione



BOTTORUP - LCS(X, Y)

$b[1 \dots m, 1 \dots n]$

$c[0 \dots m+1, 0 \dots n+1]$

$m = X.\text{Length}$

$n = Y.\text{Length}$

for $i=0$ to m /* riempio la 1^a riga con zero */
 $c[i, 0] = 0$

for $j=0$ to n /* riempio la 1^a colonna con zero */
 $c[0, j] = 0$

for $i=1$ to m

 for $j=1$ to n

 if $x_i == y_j$

$c[i, j] = c[i-1, j-1] + 1$

$b[i, j] = \nwarrow$

 else

 if $c[i-1, j] \geq c[i, j-1]$ l'uguale da precedente all'riga *

$c[i, j] = c[i-1, j]$

$b[i, j] = \uparrow$

 else

$c[i, j] = c[i, j-1]$

$b[i, j] = \leftarrow$

return $\langle b, c \rangle$

calcola tutto i sottoproblemi!

$$T(m, n) = \Theta(m) + \Theta(n) + \Theta(m \cdot n) = \Theta(m \cdot n)$$

X	y	0	1	2	3	4	5
i	ϵ	A	C	D	A	B	
0	ϵ	0	0	0	0	0	0
1	A	0	1 \nwarrow	1 \leftarrow	1 \leftarrow	1 \nwarrow	1 \leftarrow
2	B	0	1 \uparrow	1 \uparrow	1 \uparrow	1 \uparrow	2 \nwarrow
3	C	0	1 \nwarrow	1 \uparrow	1 \uparrow	2 \nwarrow	2 \uparrow
4	A	0	1 \nwarrow	2 \nwarrow	2 \leftarrow	2 \uparrow	2 \uparrow

$c[m, n]$ è il valore ottimo!

$X = A B A C A$
 $Y = A C D A B$

$i=1$

$j=1$ $A == A$ ✓ $\rightarrow \nwarrow + 1 \Rightarrow 0 + 1$

$j=2$ $A \neq C$ max tra riga e colonna precedente $\Rightarrow 1$

$j=3$ $A \neq D$

$j=4$ $A == A$ +1

$j=5$ $A \neq B$

PRINT-LCS = ACA

stampa al contrario!

* PRINT-LCS(X, Y)

$\langle b, c \rangle = \text{BOTTOMUP-LCS}(X, Y)$] $\Theta(nm)$

PRINT-LCS-REC($X, b, X.\text{length}, Y.\text{length}$)] $\Theta(n+m)$

$\Theta(nm)$

* PRINT-LCS-REC(X, b, i, j)

$\rightarrow O(i+j)$ perché ad ogni

if $i > 0$ and $j > 0$
if $b[i, j] == R$

diamette decremente uno tra i e j

PRINT-LCS-REC($X, b, i-1, j-1$)

print x_i

else

if $b[i, j] == \uparrow$

PRINT-LCS-REC($X, b, i-1, j$)

else

PRINT-LCS-REC($X, b, i, j-1$)

$O(i+j)$

* OTIMIZZAZIONI

rispetto all'uso della memoria!

(i) La tabella b ~~non~~ è fondamentale, l'info è derivabile da c in modo "locato"
guardando le tre celle ~~adiacenti~~ di c !

$c[i, j]$ dipende da $\begin{cases} c[i-1, j-1] \\ c[i-1, j] \\ c[i, j-1] \end{cases}$

⚠ L'ordine dei test
è fondamentale

(prova con $X=AC$ e $Y=BA$)

* PRINT-LCS-REC-OPT(X, c, i, j)

if $i > 0$ and $j > 0$

if $c[i, j] == c[i-1, j]$

PRINT-LCS-REC-OPT($X, c, i-1, j$)

else

if $c[i, j] == c[i, j-1]$

PRINT-LCS-REC-OPT($X, c, i, j-1$)

else /* if $c[i, j] == c[i-1, j-1] + 1$ */

PRINT-LCS-REC-OPT($X, c, i-1, j-1$)

print x_i

• Lo spazio in memoria
è $m * n$

(ii) Se mi interessa solo la lunghezza della LCS (senza ricostruire la soluzione) si può evitare di mantenere TUTTA la tabella $c[i, j]$ calcolando la riga $i+1$ tramite i

\rightsquigarrow spazio ridotto a $O(n)$

• TOP-DOWN-LCS (X, Y)

$m = X.length$

$n = Y.length$

$c[0..m, 0..n] = -1 \quad \Theta(m \cdot n)$

return TOP-DOWN-LCS-AUX (X, Y, c, m, n) $\Theta(m \cdot n)$

$\Theta(m \cdot n)$

• TOP-DOWN-LCS-AUX (X, Y, c, i, j) *

if $c[i, j] = -1$ /* NON ho ancora risolto il sottoproblema di dimensione $[i, j]$! */

if $i == 0$ OR $j == 0$
 $c[i, j] = 0$

else

if $x_i == y_j$ /* non so ancora se esiste il valore, quindi vedo in ricorsione */
• $c[i, j] = \text{TOP-DOWN-LCS-AUX} (X, Y, c, i-1, j-1) + 1$

else

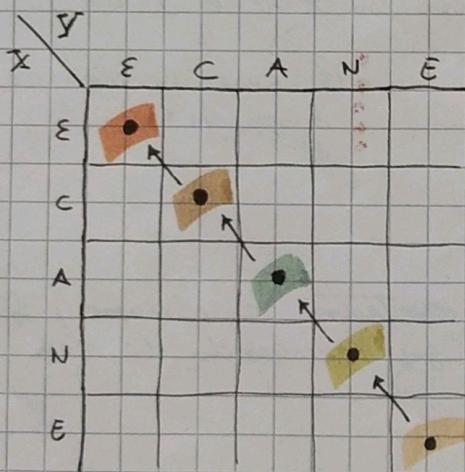
• $c[i, j] = \max (\text{TOP-DOWN-LCS-AUX} (X, Y, c, i-1, j),$
 $\text{TOP-DOWN-LCS-AUX} (X, Y, c, i, j-1))$

return $c[:i, :j]$ /* se voglio la soluzione rendo c e uso le punti corsori */

* il numero complessivo d. chiamate è AL PIÙ $m \cdot n$, tanti quanti sono i possibili sottoproblemi Bz.

queste coste $\Theta(m \cdot n)$

MA se considero il numero di sottoproblemi (chiamate) e NON è init, vedo e fare solo le chiamate NECESSARIE!



se le stringhe sono uguali ho $\Theta(m)$ chiamate invece di $\Theta(m \cdot n) = \Theta(m^2)$.

asintoticamente è la stessa complessità

anche se meglio risolvere l'LCS con il metodo TOP-DOWN

$A = [1 \dots n]$ non ordinato

MASSETTA lunghezza d'una sottosequenza di A composta da elementi in ordine crescente

• $L[j] =$ lunghezza massima di una sottosequenza crescente di $A[1 \dots j]$ avente $A[i]$ come ultimo elemento.

La più lunga sottosequenza crescente di $A[1 \dots j]$ che termina in $A[i]$ avrà un penultimo elemento $A[u]$ per $u < i$
solo che non so quale sia questo u
quindi prendo il massimo valore $L[u]$ dove $A[u] < A[i]$ e sommo 1 perché aggiungo l'elemento $A[i]$

$$L[j] = \begin{cases} 1 & \text{se } j=1 \\ 1 + \max_{\substack{1 \leq u < j \\ A[u] < A[i]}} L[u] & \text{altrimenti} \end{cases}$$

se $j=1$ la sottosequenza composta da un elemento è crescente e ha lunghezza 1

LIS (A)

$L[1 \dots n]$

$L[1] = 1$

int max = $L[1]$

for $i=2$ to n

int maxnow=0

for $u=1$ to $j-1$

if $A[u] < A[j]$ and $L[u] > maxnow$

$maxnow = L[u]$

$L[j] = 1 + maxnow$

if $L[j] > max$

$max = L[j]$

return max

tempo $O(n^2)$
con i prefissi

longest common palindromic subsequence!

$X = x_1 \dots x_n$ la trovo come la LCS tra X e il suo rovescio (X al contrario)

sia $Y = y_1 \dots y_n = x_n \dots x_1$

• $L[i, j] =$ lunghezza massima di una sottosequenza palindroma da $x_i \dots x_j$

$$L[i, j] = \begin{cases} 0 & \text{se } i=j \text{ or } j=0 \\ L[i-1, j-1]+2 & \text{se } x_i = y_j \\ \max(L[i-1, j], L[i, j-1]) & \text{altrimenti} \end{cases}$$

(diamo $n \times n$)

$O(n \cdot n)$ diametralmente

TOP DOWN

A B B A caso $\mathcal{O}(n) = O(n^2)$



fatidico

perderà funzione?

Le LCS tra X e reverse(X) è lunga al più come la LCS X e reverse(X)
se forse più lunga anche la LCS sarebbe più lunga!
Le LCS tra X e reverse(X) è lunga almeno come la LCS X e reverse(X)

LONGEST CORROU PALINDROMIC SUBSTRINGS (caratteri consecutivi!)

$X = x_1 \dots x_n$

$L[i, j] \equiv$ Lunghezza massima di una sottostringa palindroma $x_i \dots x_j$

$$L[i, j] = \begin{cases} 1 & \text{se } i=j \\ 2 + j - i - 1 & \text{se } x_i \neq x_j \wedge x_i = x_j \wedge L[i+1, j-1] = j-i-1 \\ \max\{L[i, j-1], L[i+1, j]\} & \text{altrimenti} \end{cases}$$

LCPSUBS (X)

~~LCPSUBS-MAIN~~

$n = X.\text{Length}$

$L[1 \dots n, 1 \dots n] = -1$

return LCPSUBS-AUX ($X, L, 1, 1, n$)

LCPSUBS-AUX (X, L, i, j)

if $L[i, j] == -1$

 if $j > i$ /* stringa vuota */
 return 0

 else if $i == j$ AND $x_i == x_j$ AND LCPSUBS-AUX ($X, L, i+1, j-1$) == $j-i-1$
 ~~max~~
 $L[i, j] = 2 + j - i - 1$

 else

$L[i, j] = \max(LCPSUBS-AUX(X, L, i+1, j), LCPSUBS-AUX(X, L, i, j-1))$

return $L[i, j]$

approccio TOP DOWN

complessità $T(n) = O(n^2)$

il numero di chiamate ricorsive è al più n^2

se l'intera stringa fosse palindroma ~~non~~ si avrebbero $O(n)$ chiamate ricorsive, dove n rappresenta la lunghezza delle stringe!

	A	B	B	A
A				4
B				3
B				2
A	1			