

ALGORITMO DI ORDINAMENTO QUADRATICO

SELEZIONE SINT

L'ALGORITMO SELEZIONA DI VOLTA IN VOLTA IL NUMERO MINOR: NELL'ESCUZIONE DI PARTECIPARE E LO METTE NELLA SEQUENZA ORDINATA; SI FA PERTO LA SEQUENZA VENDE DIVIDENDA IN 2 PARTI: LA SOTTOSEQUENZA ORDINATA CHE OLTRE LA PRIMA POSIZIONE DELLA M, E LA SOTTOSequenza LS A ORDINARE, CHE COSTITUISCE LA PARTE RESTANTE DELLA M.

6	3	7	2	8	1	x
1	3	7	2	8	6	x
1	2	7	3	8	6	x
1	2	3	7	8	6	x
2	2	3	6	8	7	x
7	2	3	6	7	8	

* SECONDA ETÀ
IL PIÙ PICCOLO TRA LE SEMPRE DA ORDINARE

COMPLESSITÀ (APPROXIMATIVA Dopo) CAPO MILLESIME: $\Theta(n^2)$ CASO CECCHI: $\Theta(n^2)$

CODICE

```
FOR (int i = 0; i < N - 1; ++i) {
    int index = i;
    FOR (int j = i + 1; j < N; ++j) {
        IF (A[j] < A[index])
            index = j;
    }
}
```

PROBLEMA:
ARITM. VRA FUSIONE
AVVILLANTE SCAMBIA

```
INT T TEMP = A[i];
A[i] = A[index];
A[index] = TEMP;
```

IN SEGRETO. SORT

Si assume che la sequenza sia ordinata. Si a partizionata in una sotto sequenza già ordinata, altrimenti composta da un solo elemento, e uno array di array. Alla fine inserisce la sequenza già ordinata contiene k elementi. In ogni inserzione, viene inserito un elemento dalla sotto sequenza non ordinata e inserita così di volta in volta.

6 5 3 1 8 7 2 4 **6** 5 3 1 8 7 2 4 **6 5** 3 1 8 7 2 4 **6 5 3** 1 8 7 2 4

5 6 **3** 1 8 7 2 4 **3 5 6** 1 8 7 2 4 **3 5 6 7** 1 8 7 2 4 **1 3 5 6** 8 7 2 4

1 3 5 6 8 7 2 4 **1 3 5 6 8 7** 2 4 **1 3 5 6 7 8** 2 4 **1 3 5 6 7 8 2** 4

1 2 3 5 6 7 8 4 **1 2 3 5 6 7 18** 4 **1 2 3 4 5 6 7 8** **1 2 3 4 5 6 7 18**

COLUMN.

INT prev, current;

For (int i = 1; i < n; ++i) {

 current = i;

 prev = i - 1;

 while (prev ≥ 0 && A[current] < A[prev]) {

 int temp = A[current];

 A[current] = A[prev];

 A[prev] = temp;

 prev --;

 current --;

}

Complessità: $\Theta(n)$ (caso migliore) $\Theta(n^2)$ (caso peggiore)

Bubble Sort

gli elementi dell'array vengono confrontati a due a due procedendo in verso dritto (no, compare paragrafo Della lista)

per i primi: saranno confrontati il primo e il secondo elemento, poi il secondo e il terzo, poi il secondo e il quarto e così via fino al confronto fra il penultimo e l'ultimo elemento.

inizial:

5 | 3 | 8 | 4 | 6

Step 1

5 | 3 | 8 | 4 | 6

compari 1° < 2° (scambio)

Step 2

3 | 5 | 8 | 4 | 6

compari 2° < 3° (nessun cambio)

Step 3

3 | 5 | 8 | 4 | 6

compari 3° < 4° (scambio)

Step 4

3 | 5 | 4 | 8 | 6

compari 4° < 5° (scambio)

Step 5

3 | 5 | 4 | 6 | 8

preferiti i passi finiti array ordinato

il verde indica che non compare più, per questo si chiama bubble sort: gli elementi più grandi "vengono".

Conclusioni

bubblesort:

```
for (int i = 0; i < N; ++i) {
    cambiati = false;
    for (int j = 0; j < N - i; ++j) {
        if (A[j] > A[j + 1]) {
            int temp = A[j];
            A[j] = A[j + 1];
            A[j + 1] = temp;
            cambiati = true;
        }
    }
    if (!cambiati)
        break;
}
```

$\Theta(n)$ caso minimo

$\Omega(n^2)$ caso pessimo

se non ci sono scambi, array già ordinato

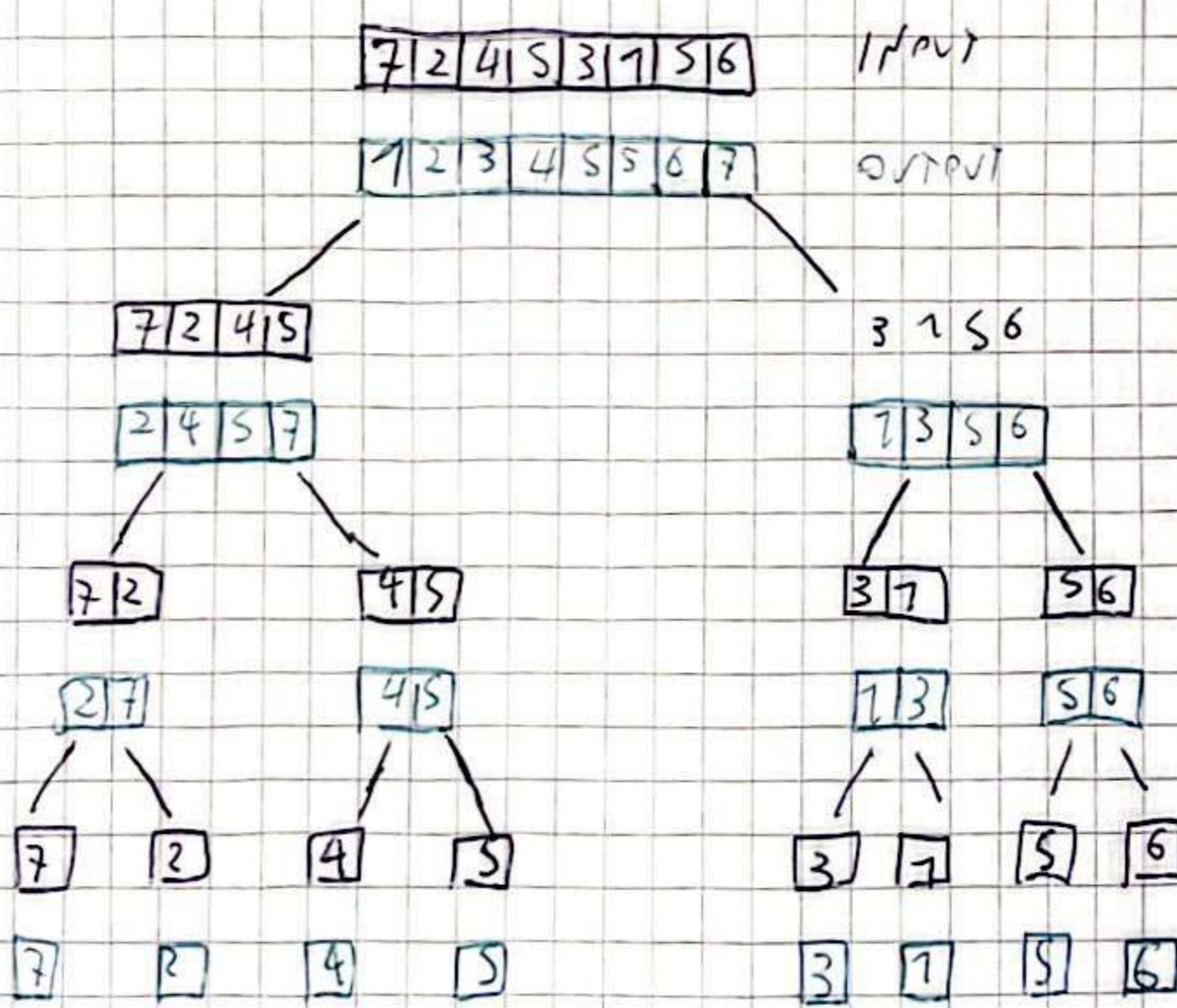
MEGLIE SORT

CORRETTAMENTE, L'ALGORITMO FUSOREA È IL MEGLIO POSSIBILE.

- 1) SE LA SEQUENZA HA ORDINAMENTO INCREMENTALE O DECRESCENTE È CLARIFICATA.
- 2) LA SEQUENZA VERRÀ DIVISA (DIVIDITA) IN DUE METÀ (SE LA SEQUENZA CONTIENE UN NUMERO DI ELEMENTI, VERRÀ DIVISA IN DUE SOLO - SE SONO PARI: DI CUI, LA PRIMA HA UN ELEMENTO IN PIÙ DELLA SECONDA)
- 3) OCCURRA PER QUADRARE SOTTO SEQUENZE VERRÀ ORDINANTE APPLICANDO ALCO ALGORITHM (IMPIEGA)

NICOLA NAMERIC ALGORITHM (IMPIEGA)

- 4) LE DUE SOTTOSERIE DIVIDUTE VERRANNO FUSE (COMBINATE), PER FARLE QUADRARE, SI ESTRAGGONO PRECISAMENTE IL MINIMO DELLE DUE SOTTOSERIE. E LO SI RICHIAMA NELLA SEQUENZA IN USCITA, CHE RIVIRGA' ORDINATAMENTE



IL VERDE APPARTIENE ALL'ALGORITHM NICOLA NAMERIC

CASO MIGLIORE $\Theta(M \log M)$

CASO PEGGIORI $\Theta(M \log M)$

CODE

```
void fondi(vector<int> &v, unsigned int inizio, unsigned int centro, unsigned int fine) {  
    // Dichiario e riempio i due vettori  
    vector<int> vsinistra, vdestra;  
    for(unsigned int i = inizio; i <= centro; ++i) {  
        vsinistra.push_back(v[i]);  
    }  
    for(unsigned int i = centro + 1; i <= fine; ++i) {  
        vdestra.push_back(v[i]);  
    }  
    // dichiaro gli indici necessari  
    unsigned int indicesinistra = 0;  
    unsigned int maxsin = vsinistra.size();  
    unsigned int indicedestra = 0;  
    unsigned int maxdes = vdestra.size();  
    // scorro i due vettori, elemento per elemento, confrontando  
    mano a mano gli elementi tra loro  
    for (unsigned int i = inizio; i <= fine; ++i) {  
        if (indicesinistra < maxsin && indicedestra < maxdes) {  
            if (vsinistra[indicesinistra] < vdestra[indicedestra]) {  
                v[i] = vsinistra[indicesinistra];  
                ++indicesinistra;  
            }  
            else {  
                v[i] = vdestra[indicedestra];  
                ++indicedestra;  
                continue;  
            }  
        }  
        // se uno dei due vettori non confronta elementi, vuol dire  
        // che questi sono i più grandi e vengono aggiunti in coda  
        if (indicesinistra == maxsin && indicedestra < maxdes) {  
            v[i] = vdestra[indicedestra];  
            ++indicedestra;  
            continue;  
        }  
        if (indicedestra == maxdes && indicesinistra < maxsin) {  
            v[i] = vsinistra[indicesinistra];  
            ++indicesinistra;  
            continue;  
        }  
    }  
}  
  
// Funzione 2: Definisce le due metà (gli indici di inizio e di fine),  
// le ordina chiamando fondi e grazie all'ultima chiamata di  
// quest'ultima le mette assieme ordinandole.  
void ms(vector<int> &v, unsigned int inizio, unsigned int fine) {  
    if (inizio < fine) {  
        unsigned int centro = (inizio + fine) / 2;  
        ms(v, inizio, centro);  
        ms(v, centro + 1, fine);  
        fondi(v, inizio, centro, fine);  
    }  
}  
  
// Funzione 1: Gestisce il caso di un vettore vuoto e chiama ms  
// passando inizio e fine del vettore da ordinare.  
void mergeSort(vector<int> &v) {  
    if (v.size() != 0) {  
        ms(v, 0, v.size() - 1);  
    }  
}
```

QUICK SORT

QuickSort è un algoritmo di ordinamento che opera su un array.

Dividendo ricorsivamente l'array in due parti, secondo un elemento detto "pivot", cui elementi minore del pivot sono posti a sinistra del pivot, mentre quelli maggiori sono posti a destra. Quindi, l'algoritmo viene applicato separatamente alle due parti. Questo processo viene ripetuto ricursivamente fino a quando l'array è ordinato.



Caso niente: $\Theta(m \log m)$

Caso Medio: $\Theta(m \log m)$

Caso Peggior: $\Theta(m^2)$

CODE

```
void swap(int a[], int i, int j) {  
    int temp = a[i];  
    a[i] = a[j];  
    a[j] = temp;  
}  
  
int partition(int a[], int inizio, int fine) {  
    int pivot = a[inizio];  
    int i = inizio + 1;  
  
    for (int j = inizio + 1; j <= fine; ++j) {  
        if (a[j] < pivot) {  
            swap(a, i, j);  
            ++i;  
        }  
    }  
    swap(a, inizio, i - 1);  
    return i - 1;  
}  
  
void quickSort(int a[], int inizio, int fine) {  
    if (inizio < fine) {  
        int pivot = partition(a, inizio, fine);  
        quickSort(a, inizio, pivot - 1);  
        quickSort(a, pivot + 1, fine);  
    }  
}
```

COMPLESSITÀ

BIG-O

Siano $f \in \Omega$ due funzioni da numeri naturali ai numeri reali ≥ 0 ($f, \Omega : \mathbb{N} \rightarrow \mathbb{R}^+$)

$f(m) = O(\Omega(m))$ se \exists due costanti $c > 0$ e $m_0 \geq 0$ tali che
 $f(m) \leq c \Omega(m)$ per ogni $m \geq m_0$

(BIG-O IDENTIFICA UNA SOURA STIMA)

THETA

Siano $f \in \Omega$ due funzioni da numeri naturali ai numeri reali ≥ 0 ($f, \Omega : \mathbb{N} \rightarrow \mathbb{R}^+$)

$f(m) = \Theta(\Omega(m))$ se esistono costanti $c_1, c_2 > 0$ e $m_0 \geq 0$ tali che
 $c_1 \Omega(m) \leq f(m) \leq c_2 \Omega(m)$ per ogni $m \geq m_0$

(THETA IDENTIFICA UNA STIMA ESATTA)

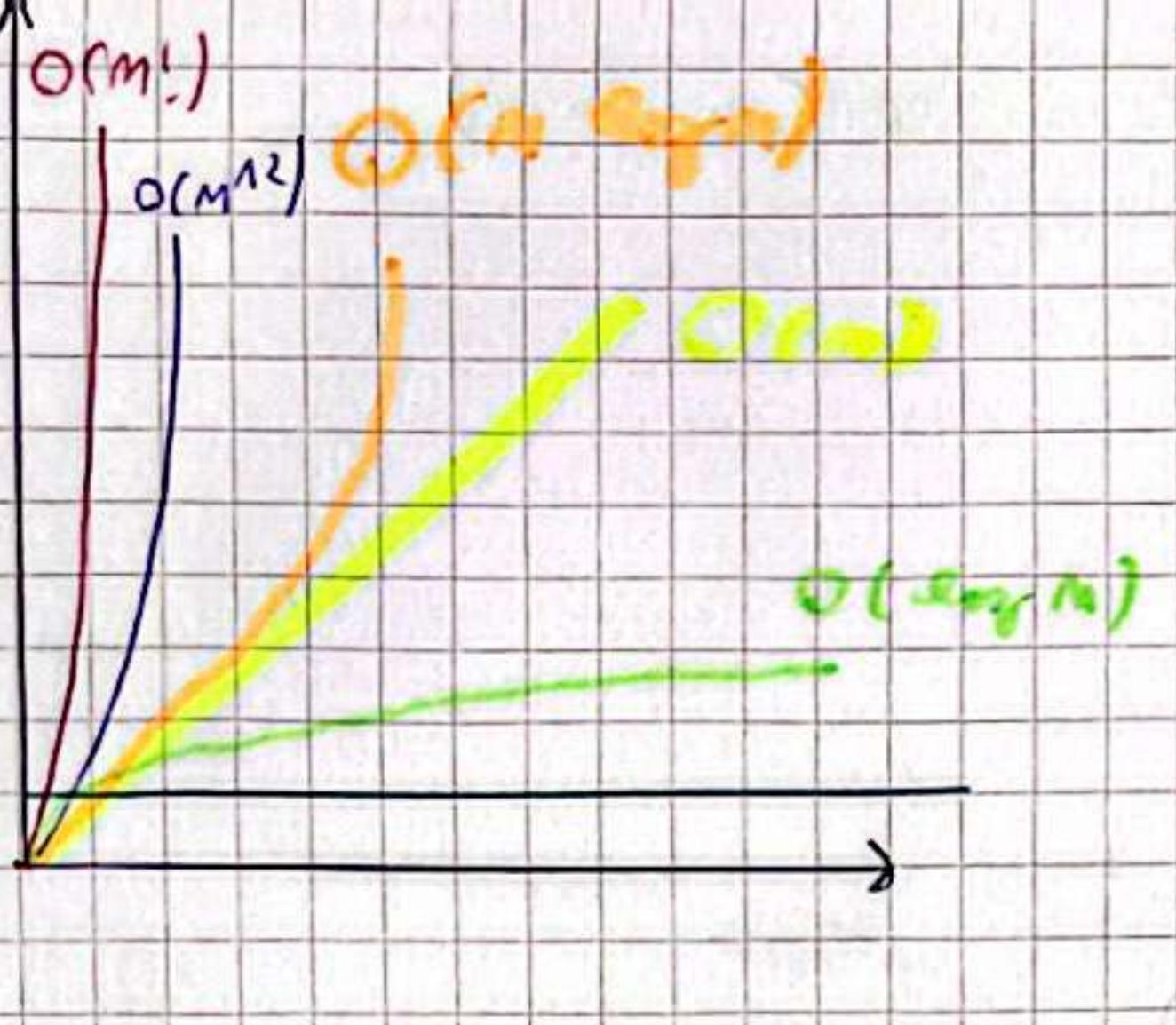
OMEGA

Siano $f \in \Omega$ due funzioni da numeri naturali ai numeri reali ≥ 0 ($f, \Omega : \mathbb{N} \rightarrow \mathbb{R}^+$).

$f(m) = \Omega(\Omega(m))$ se esiste costante $c > 0$ e $m_0 \geq 0$ tali che $f(m) \geq c \Omega(m)$ per ogni $m \geq m_0$

(OMEGA IDENTIFICA UNA STIMA INFERIORE)

SE UNA FUNZIONE E' IN THETA DI UNA ALTRA FUNZIONE ALLORA E' ANCHE IN OMEGA E D



$O(1)$ = Constant

$O(\log n)$ = Logarithmic Time

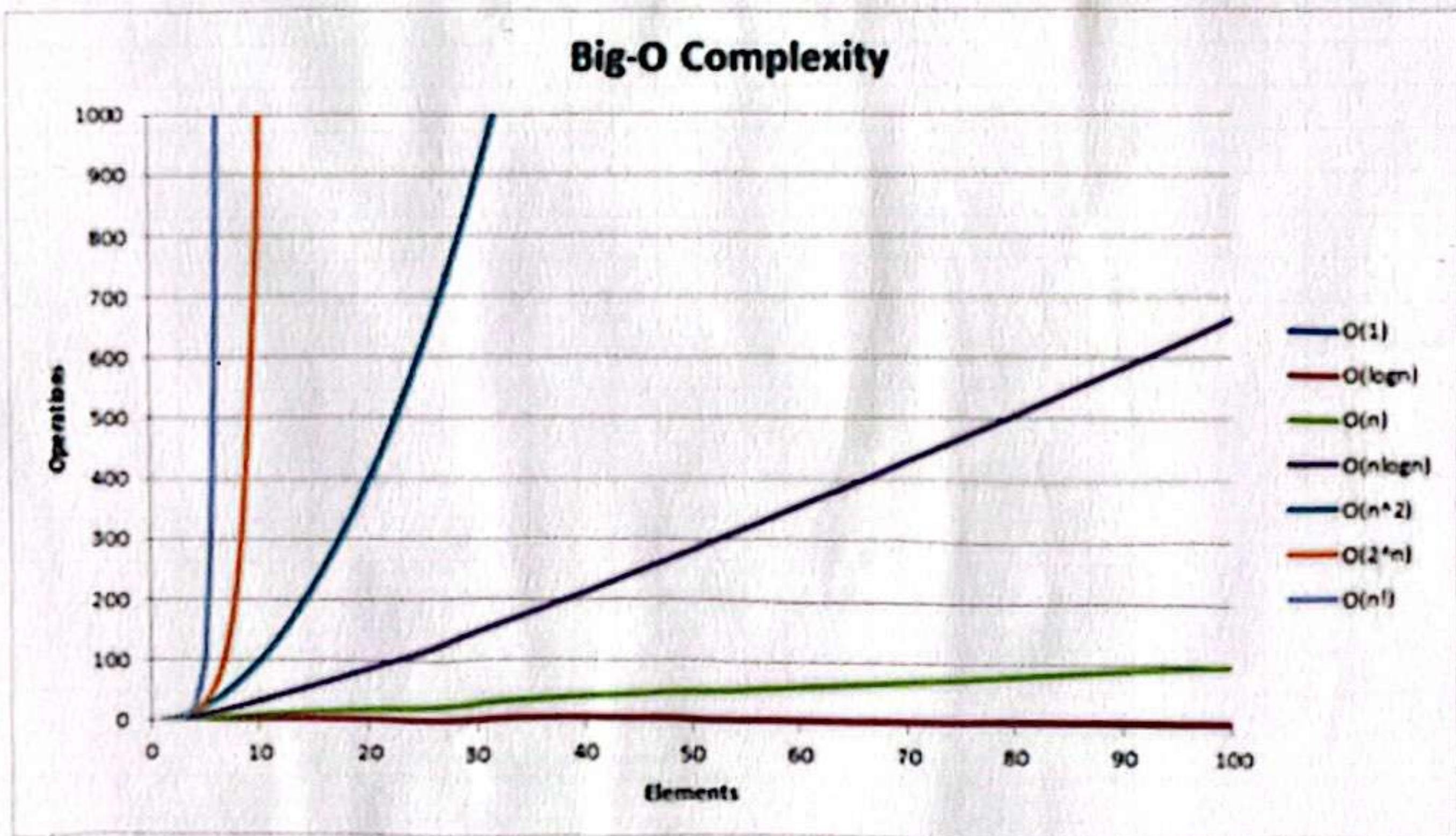
$O(n)$ = Linear Time

$O(n \log n)$ = Quadratic Time

$O(n^2)$ = Quadratic Time

$O(m!)$ = Factorial

Big-O Complexity Chart



[SESION 1] COMPLEXIDAD (AULA WEB)

Ejercicio 1

INT N = 10;

FOR (INT i = 0; i < r; ++i)

COUT << i << endl;

$$\Theta(1)$$

Ejercicio 2

INT N;

CIN >> N;

FOR (INT i = 0; i < m; ++i)

COUT << i << endl;

$$\Theta(m)$$

Ejercicio 3

INT m;

CIN >> m;

IF (m == 42)

COUT << "Buenas DVEs";

ELSE

FOR (INT i = 0; i < m; ++i)

COUT << i << endl;

Caso M(100):	$\Theta(1)$
Caso Peor:	$\Theta(m)$

M == 42

m != 42

Ejercicio 4

INT M;

CIN >> M;

FOR (INT i = 0; i < m; ++i)

FOR (INT j = 0; j < M; ++j)

COUT << (i, j);

$$\Theta(m^2)$$

Ex 4

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Ex 5

int m;

int sum;

for (int i = 0; i < m; ++i) {

 for (int j = i; j < m, ++j) {

 cout << i, j;

$$\Theta(m^2) \Rightarrow \sum_{k=1}^m k = \frac{m \cdot (m+1)}{2} = \frac{1}{2} m^2 \cdot \frac{1}{2} m = m^2$$

m²

(n-1)²

(m-2)²

1

stopia;

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,1)	(1,2)	(1,3)	(1,4)	
(2,2)	(2,3)	(2,4)		
(3,3)	(3,4)			
(4,4)				

Ex 6

void empty (cell)

cell * aux = new cell;

aux -> next = aux;

l1 = aux;

$$\Theta = 7$$

Esercizio 1

void HEAD_INSERT (LIST, VALUE) {

CALL AUX = newcell;

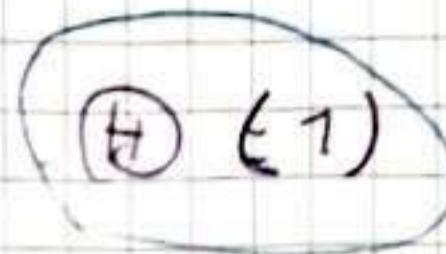
AUX → INFO = NEW → ELEM;

CALL TEMP = LIST → next;

LIST → next = AUX;

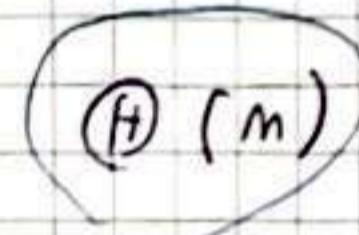
AUX → next = TEMP;

}

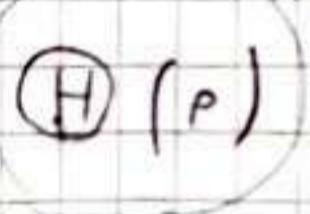


Esercizio 8

void PRINT LIST { ... }



Esercizio 9



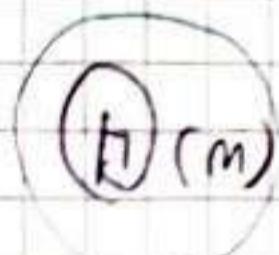
P SAREBBE
LE NUMERO DI DATI DA LEGGERE.

Esercizio 10

$$(H) \in \Theta(q^2) \Rightarrow \frac{1}{2}m^2 + \frac{1}{2}m$$

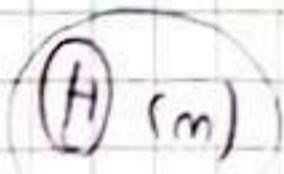
Esercizio 11

FATTORIALE



PER CIECA VIGORE CHIAMARE M VOLTE

Esercizio 12



Esercizio 13

CALL N (1000000) $\in \Theta(1)$

CALL P (1000000) $\in \Theta(m)$

14
[$\text{C}_A = \text{M}_{11, m}$ $(\text{I}) (n)$]
[$\text{C}_B = \text{P}_{k, n}$ $(\text{I}) (m)$]

15
[$(\text{I}) (n)$]
 $(\text{II}) (m)$

16
[$(\text{I}) (n)$]
 $(\text{II}) (m)$

17
[$(\text{I}) (n)$]
 $(\text{II}) (m^2)$

18
In September In autumn In winter Upon Christmas Eve
Autumnal equinox ($\text{II} (n)$)

19
In January In spring In summer Autumnal equinox
($\text{II} (n)$)

20
[$\text{C}_A = \text{M}_{11, m}$: $(\text{II} (1))$ $\text{C}_B = \text{P}_{k, n}$ $(\text{II} (m))$]

21
[$\text{C}_A = \text{P}_{k, m}$: $(\text{II} (1))$ $\text{C}_B = \text{P}_{k, m}$: $(\text{II} (m + m))$]

22
[$\text{C}_A = \text{P}_{k, m}$: $(\text{II} (1))$ $\text{C}_B = \text{P}_{k, m}$: $(\text{II} (m + m))$]

23

CAJO MIGRAZIONE	($\Theta(1)$)	CAJO PEGGIORE (M^2)
-----------------	-----------------	-------------------------

24

CAJO MIGRAZIONE	($\Theta(1)$)	CAJO PEGGIORE ($\Theta(M \log M)$)
-----------------	-----------------	--------------------------------------

Calcolo complessità BINARY SEARCH

IMPIANTO LA MIGRAZIONE BINARIA (BINARY SEARCH) È UN ALGORITMO DI TIPO
UTILIZZATO PER TROVARE UN ELEMENTO ALL'INTERNO DI UN INSERIMENTO ordinato
DI DATI.

CODE

Bool Found = FALSE;

int MAX = N - 1 // N : size array

int MID = 0

int MIN = 0

while (MIN <= MAX && ! Found) {

 MID = (MAX + MIN) / 2;

 if (A[MID] == ITEM) {

 Found = TRUE;

 BREAK;

 } else if (A[MID] > ITEM) {

 MAX = MID - 1;

 } else {

 MIN = MID + 1;

}

if (Found) {

 cout << "trovato";

} else {

 cout << "ELEMENTO non trovato";

}

LA MÉTODA DE ORDENAR ELEMENTOS	LIVELLO NÚMERO JERARQUICO DE CONJUNTO J	Nº ELEMENTO EN EL CONJUNTO DE CONJUNTO J	Nº ELEMENTO EN EL CONJUNTO DE CONJUNTO J	Nº OPERACIONES EN LIVELLO J
	0	m	$m/2^0 = \frac{m}{2^0}$	(H) (1)
[]	1	$m/2^1 = \frac{m}{2^1}$	(H) (1)	
[]	2	$m/2^2 = \frac{m}{2^2}$	(H) (1)	
	⋮	⋮	⋮	⋮
	L	1	$\frac{m}{2^L}$	(H) (1)

$$l = \frac{m}{2^L} \Rightarrow 2^L = m \Rightarrow \log_2(2^L) = \log_2 m \Rightarrow L = \log_2 m$$

(H) ($\log m$)

LA COMPLICACIÓN NEL CASO MISCIONE: (H) (1), QUESTO SI

VERIFICA QUANDO ELEMENTI S'INTRA NEL PUNTO MEDIO DEL ANATI E QUINDI PER E' MEGLIATO EFFETUARE DIVISIONI.

LA COMPLICAZIONE NEL CASO PERCORSI: (H) ($\log m$).

QUESTO SI VERIFICA QUANDO ELEMENTI DELL'INTERVALLO SONO PRESENTI SOLO UNO, TROVA ALL'ESTREMA opposta rispetto al punto MEDIO IN Ogni INTERVALLO. L'ALGORITMO CONTINUA A DIVIDERE L'INTERVALLO DI METÀ A METÀ FINO A QUANDO L'INTERVALLO DIVENTA Vuoto, E' perciò AD Ogni INTERVALLO VERSO DIVISO IN DUE, IL NUMERO di TERMINI NECESSARI E' GRANDEZZA COSTANTE A UNA DIMENSIONE DEL CANTO,

CACCIO A COMPLESSITÀ IN MEGLIO

CACCINO, HAI UNA SEGRETA:

- TROVA LA META'
- CHIAMA M₁ JUCCA PRIMA META' (ARCONDO)
- CHIAMA M₂ IL 2° META' (ARCONDO)
- CHIAMA FONDI JUICE DUE META'

COMPLESSITÀ

VOLTE $\Theta(M \log M)$ SIA NEL CASO MILIONE DI CALCOLI, INVECE NON ESSERADUN
ALCUNO ALTRUI, CREATURA VUOLE CHE CHIAMATI ALCONDO NECESSARIO, E TUTTI
I CONTATORI, ANCHE NEL CASO DI UN UNICO PREZO IN CONSIDERAZIONE SIA CIA'
ABBINATO.

LA COMPLESSITÀ DEL CACCINO DEPENDE DAL CALCOLO DEL COSTO: $m^{\text{LIVELLI}} \cdot \text{COSTO}$

CACCIO PROBLEMA

LIVELLO	NUMERO PROBLEMI	DIMENTICARE PROBLEMA
0	1	m
1	2	$m/2$
2	4	$m/4$
3	8	$m/8$
J	2^J	$m/2^J$

COSTO DI OGNI LIVELLO

AD OGNI LIVELLO J SI HA UNO 2^J SORTO PROBLEMI DI TIPO MELG, Ovvvero
LUNGO $m/2^J$ E DIVISI NELL'UNO IN $\Theta(m/2^J)$. PER CONSEGUENZA IL COSTO

DI OGNI LIVELLO, BISOGNA MOLTIPLICARE IL COSTO MELG PER IL NUMERO

DI VOLTE CHE VIENE CHIAMATO:

$$2^J \cdot m/2^J = m \Rightarrow \Theta(m)$$

NUMERO DI GRANDEZZA:

SE PER UNO C'È UNO ALUVIUMO LIVELLO DELLA MONTAGNA, IL SUO PROBLEMA
AUMENTARE DIMENSIONE 1, E C'È UNO DEL LIVELLO T, DOPO PROBLEMA DELL'
DIMENSIONE $M/2^T$, PER QUALE T SI HA CHE $M/2^T = 1 \Rightarrow T = \log_2 M$

QUINDI IL NUMERO DI GRANDEZZA E': $\log_2 M + 1$ [+1 PER IL PENNINO DI PIANO DA
LIVELLO OGNI]

LE CONCLUSIONI SONO E' DATO DA $(\Theta)(M) \cdot \log_2 M \Rightarrow (\Theta)(M \log M)$

SIA NEL CASO MIGLIORI CAPO DI DUE, PER EFFICIENZA.

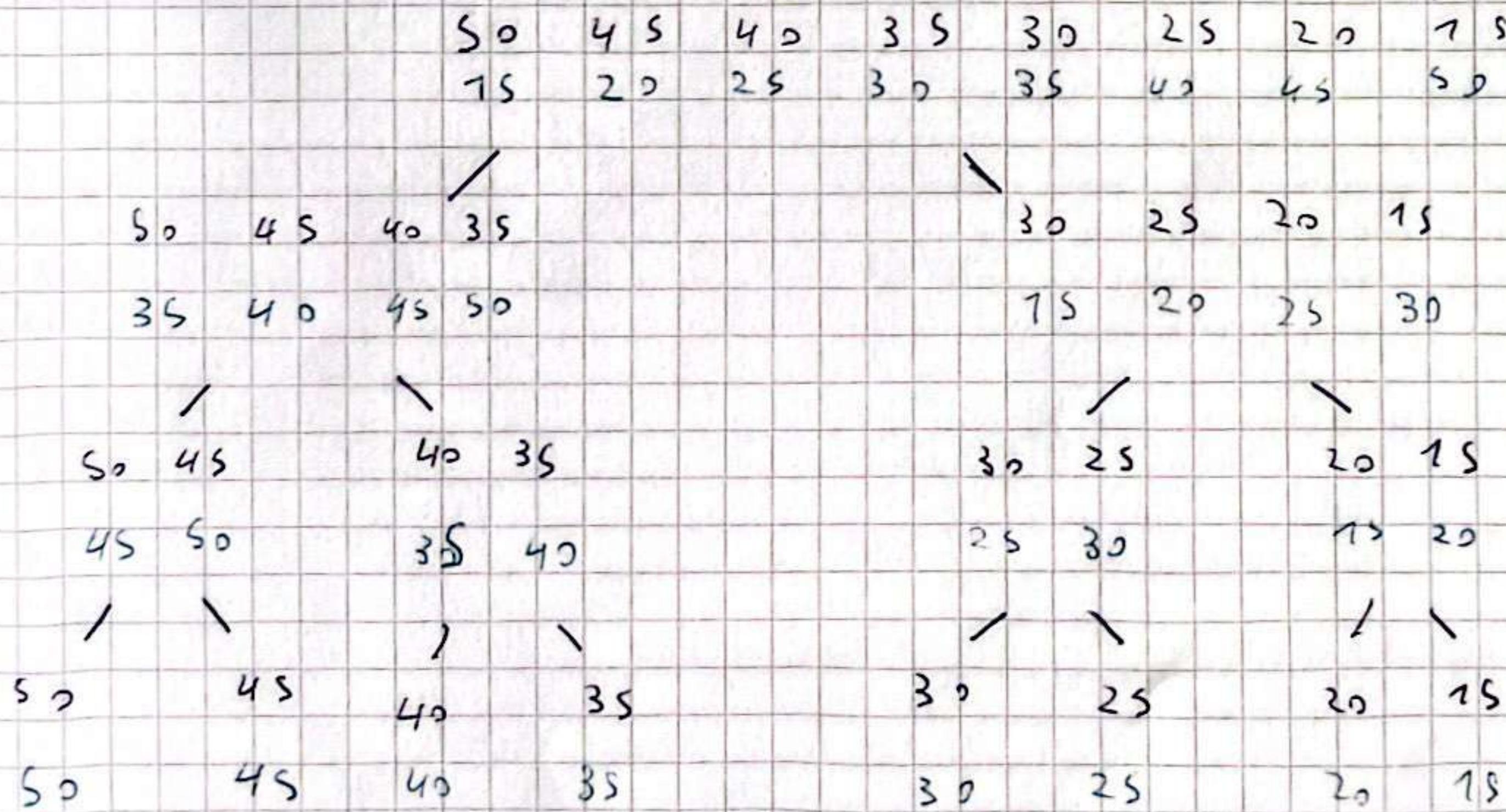
IL "LOG M" DENOTA IL NUMERO DEI GRANDEZZA DELGRANDEZZA DEI PENNINI

MIGLIOR SORT ESEMPIO ESAME

SI SEMPRE UN PENNINO HA UNA CERTA STRUTTURA, CONSIDERARSI IL CASO MIGLIOR, RESENTE O MAGGIOR
DUE PENNINI DI MIGLIORI IN QUESTO CASO?

SE GUERRA, 50 45 40 35 30 25 20 15

SIMULAZIONE CRITICA:



VALORE $(\Theta)(M \log M)$ DI UN CASO MIGLIORI CHE NEL CASO PECCATORI, INFATTI PER
QUESTO VERSO ALCUNO ALGORITMO, EFFETTUANDO TUTTI I PIZZI CHAMATE RECURRENT
NECESSARI, E TUTTI I COMBINATORI, ARRECA NEL CASO IN UNO (PENNINI) PRECISO IS
ORDINE DI GRANDEZZA UNA CIP' ANDAMENTO.

La complessità è della dimensione della calcolo delle fissioni;

m° LIVELLO • COSTO OPERAZIONI DI OTTO LIVELLO

E allora, il costo operazioni si calcola così:

numero problemi • dimensione problema

LIVELLO (τ)	NUERO PROBLEMI (2^τ)	DIMENSIONE PROBLEMA ($M/2^{\tau-1}$)	SEQUENZA
0	1	M	$S_0: 45 \ 40 \ 35 \ 30 \ 25 \ 15$
1	2	$M/2$	$S_1: S_0 \ 45$ $S_2: 30 \ 25 \ 20 \ 15$
2	4	$M/4$	$S_1: S_0 \ 45$ $S_2: 40 \ 35$ $S_3: 30 \ 25$ $S_4: 20 \ 15$
3	8	$M/8$	$S_1: S_0$ $S_2: 45$ $S_3: 40$ $S_4: 35$ $S_5: 30$ $S_6: 25$ $S_7: 20$ $S_8: 15$
4	16	$M/16$	Ora, se avrai già fatto la una lista elementi da cui in poi "si tratta di" nella tabella dei risultati, ove valori sempre minore e maggiore

• COSTO OPERAZIONI DI OTTO LIVELLO:

STESSA SEQUENZA DI PAROLA PRIMA

• NUMERO DI LIVELLI:

STESSA SEQUENZA PAROLA PRIMA

COMPLESSITÀ

DUPLA SONTA

Complessità: L'effettuazione per la divisione si svolge in due sezioni: la part.

Caso migliore: $\Theta(m \log m)$ → E' un caso particolare in quanto si sceglie come pivot il MEDIANO che però non possiamo sapere se sarà PRIMA o DOPPIA. In questo caso, essendo che il pivot è SEMPRE l'elemento mediano, l'array verrà diviso dalla metà in due sottoinsiemi di circa $m/2$. Quindi, come nel caso delle medie sono, ottengo un albero BINARIO BIARRITTO che sta afferma $\log_2(m)$, cioè E' il numero di operazioni della ricorsione necessaria.

CASE MEDIO:

$\Theta(m \log m)$ E' simile alla complessità del Medio-Sont.

- "m" DENOTA IL NUMERO DI PIZZERIE SVOLTE AD Ogni LIVELLO DELLA ALBERGO ALCUNA CONDIZIONE: Al LIVELLO J SI EFFETTUANO 2^J OPERAZIONI PARTIZIONI, CIOE UNA SU UNA PIZZERIA AI PIZZAI UGLIA $m/2^J$. CONSIDERANDO QUESTA CHIAMATE ITA COMPLESSITÀ LINEARE NELL'ORDINE DELLA PIZZERIA: IN PIZZERIA J SI VUOLE DETERMINARE QUINDI $O(m/2^J)$. AD Ogni LIVELLO FAUCIO $2^J \cdot O(m/2^J)$ OPERAZIONI CHE DETERMINANO $O(m)$.

- " $\log m$ " DENOTA IL NUMERO DI LIVELLI DESCRIBENDO DELLA ricorsione.

CASE PESANTE:

$\Theta(m^2)$ Quanto si seleziona il pivot L'ELEMENTO MINIMALE o MAXIMALE DELLA SEZIONE SU CUI PARTIZIONARE viene chiamata O QUANDO LA SEZIONE E' GIÀ ORDINATA. QUESTA E' DATA ALLA PARTIZIONE PORTEGGI A LIVELLO 0 COSTRUI M, A LIVELLO 1, COSTRUI M-1, A LIVELLO 2 COSTRUI M-2 E COSÌ VIA, QUINDI SI HA CHE $M + (M-1) + (M-2) + (M-3) \dots = M \cdot (M+1)/2 = M^2$ QUINDI QUESTA COSTRUZIONE SI SVOLGE A

Quick Sort parziali

Per fare analisi di dimensione M , il tempo di esecuzione del quicksort randomizzato nel caso medio è $\Theta(M \log m)$

Quick Sort DOMANDA ESAME

Quicksort caso peggiore + pivot massimo / minimo

Sequenza: 30 25 20 15 50 45 40 35

Nel caso peggiore durata \propto complessità $\Theta(M^2)$ e si ha

Questo caso risulta essere come pivot l'elemento minore o maggiore.

Dopo separazione in cui partition viene chiamata.

Infatti per un insieme A complessità di tempo $O(n^2)$ dove n è la dimensione dell'insieme.

Ad ogni iterazione, al primo "uno" (valore 0, delimitatore) si

effettua un operazione, al secondo (valore 1) si ha fanno $m-1$

e via avendo fino ad arrivare all'ultimo livello cas essere 1

sia che operazioni, questo calcolo si sviluppa nel sommaioria per

$$1 + 2 + \dots + (m-1) = m + (m-1) + (m-2) + (m-3) \dots = m^2$$

ESEMPIO SU SEQUENZA DATA:

N.B. i valori tra parentesi sono ora le relative posizioni finali

È per venire così a una soluzione più semplice chiamata inversione

1) Secco come pivot o il minore o l' massimo $\rightarrow 15 \in$ sistema

1 minore a sinistra è il massimo a destra

$$\text{SSR} = (15) 30 25 20 50 45 40 35$$

Il pivot è nella posizione finale (in questo caso primo del minore)

Ci sono due su sequenza nestante,

2) Secco come pivot o il minore o il massimo $\rightarrow 50 \in$ sistema

$\text{SEQ} = (15) \quad 30 \quad 25 \quad 20 \quad 45 \quad 40 \quad 35 \quad (50)$

Il pivot è quello caso è molto probabile finale (ultimo elemento)

Cittando Qs su sequenza termine

Quotidiano caso migliore + pivot sempre ultimo elemento

La complessità nel caso peggior è $\Theta(M \log M)$

Questo perche' è un caso pessimo in quanto si scarica

sempre pivot sempre elementi mediante che però non possiamo

sapere se era prima o dopo pivot la sequenza.

Quindi venga quindi dividendo una partizione in due parti scartate

di lunghezza circa $M/2$ ottenendo quindi come per il caso del mergesort.

Albero binario che ha altezza $\log_2(M)$ ossia il numero di operazioni furtose risulta.

Ad ogni livello si eseguiscono \sqrt{M} effettuate su $(2^j) \cdot (M/2^j)$ elementi e ha quindi complessità $\Theta(M \cdot \log_2(M))$.

In complessità prima sarà dunque $\Theta(M \cdot \log_2(M))$

Simulazione Qs con pivot sempre ultimo elemento

Sequenza: 10 20 30 40 50 60 70 80 90 100

Due che in sequenza è ripetuta è che il pivot scelto

è sempre l'ultimo elemento, appunto che di essere più caso peggiore.

Grado m	Pivot	sequenza
1	100	10 20 30 40 50 60 70 80 90 (100)
2	90	10 20 30 40 50 60 70 80 (90) (100)
3	80	10 20 30 40 50 60 70 (80) (90) (100)
.	.	.
10	10	(10) (20) (30) (40) (50) (60) (70) (80) (90) (100)

QUICK SORT COR PIÙ DI PESA MATERIA

SEQUENZA: 12 7 1 2 3 4 9 6 8 10

NUOVO: 12 7 1 2 3 4 9 6 8 10
IN POSIZIONE 9 È IL PUNTO DI SEPARAZIONE

12 7 1 2 3 9 6 8 10

(1 2 3) + (12 7 9 6 8 10)

(1) 2 (3) 4 (7 6 8) 9 (12 10)

1 2 3 4 6 7 8 9 10 12

1 2 3 4 6 7 8 9 10 12

COME SCELGONO IL GUARDO PIVOT?

IDEA SCELTA CASUALE DEL PIVOT:

ASCIATORI CHIAMATA RECUSIVA, IN QUESTO CASO SCELGE PIVOT UNO DEI NUMERI DEL'ANALISI A CASO, CON QUESTO APPROCCIO GLI ALGORITMI SONO PIÙ STABILI, ADDONDO UNO VENDITORE DI QUICK SORT RANDOMIZZATA, NEGLI QUALI ARCAJE JE DUE ESEMPLARI DIVERSI: UNICO STERZO INPUT POSSIBILE VOLTE, IN MODO DIVERSO, I MIGLIORI SONO DUE ESEGUITORI, DUE ESEGUTI CON UNO STERZO,
PER Ogni ANNO DI DIMISSIONE M, IL TEMPO DI ESEGUITORE È
QUICK SORT RANDOMIZZATO PER UNO MEDIO È $O(M \log m)$

Esempio Domanda ESAME - Dati Sopri Aviazione

1) Scrivere la comparsa di un pivote solo nel (1,1), per il caso

comparso: $\begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} m^2 \\ \end{pmatrix}$

Si ricorda nel caso per il quale quando partiamo dalla riga 1 a una "111210" è "fine" se i suoi seguenti simboli come punti elementi marcano o meno. In questo caso, comparsi una "11120" è "fine".

La comparsa si calcola nel seguente modo:

Si sommano le operazioni effettuate ad ogni livello del algoritmo. Si chiamano ricordi. Tali operazioni sono salvati allo stesso livello. Al livello 0 vale m (effettua m operazioni), al livello 1 si fanno $m-1$ operazioni, livello 2 $m-2$ e così via fino ad arrivare all'ultimo livello nel quale viene effettuata una sola operazione.

Questo significa che si svilupperà nella sommatoria per i cui va da 1 a m : $m + (m-1) + (m-2) + (m-3) \dots = m^2 \Rightarrow \oplus (m^2)$

2)

5 6 8 7 9 10 3 4 2 1

In blu il pivot. In rosso gli elementi nella posizione finali:

5 6 8 7 9 10 3 4 2 1

(5 6 3 4 2 1) 7 (8 9 10)

(2 1) 3 (4 5 6) 7 8 9 10

1 2 3 4 5 6 7 8 9 10

CONTI SPORTE AL CASO MILANO: SI QUICHI SONI IN QUANTO AD OGNI CHIAMATA SI PARLIA, IL PIÙ VAI F' IL MESSAGGIO DEGLI ELEMENTI NELLA PONTEGGIO: SE IL GIORNO A TUTTO INIZIO E' FINITO, Ovviamente questo CASO E' TECNICO IN QUANTO FORSE NON' SAREBBERE L'ELEMENTO MESSAGGIO SERVIRÀ A MONITORARE PONTEGGIO GLI ELEMENTI.

COMPRESSSITÀ: $\Theta(\log m)$

TIPO DI DATO (TDD)

TIPO

UN TIPO E' UNO COLLEZIONE DI VALORI

FORMALIZZAZIONE: UN TIPO E' UN INSERIMENTO

DATO

UN DATO E' UN ELEMENTO DI UN TIPO. E' PRODOTTO DA UN TIPO.
AD ESEMPIO, 7 E' UN DATO CHE FA IL TIPO INTERO.

FORMALIZZAZIONE: UN DATO E' UN ELEMENTO PER UN TIPO.

TIPO DI DATO (CORRETTO) TDD

UN TIPO DI DATO E' UN TIPO CON UNA COLLEZIONE DI OPERAZIONI. PER
MAPPAREGLI GLI ELEMENTI (O MEMBERS)

AD ESEMPIO IL TIPO INTERO CON GRADUATORIE. E' UN TIPO DI DATO

FORMALIZZAZIONE: UN TIPO DI DATO (CORRETTO) E' UN ALLEGATO, SOLITAMENTE
ESTENDOCERCA, SU UNA SECONDAVIA

STRUTTURA DATI

UNESCO IL TIPO DI STRUTTURA DATI AFFIGGIBILE A UNA PARTECIPANTE
ORGANIZZATORIO DELLE INFORMAZIONI CHE PERMETTE DI SUPPORTARSI IN MODO
CORRETTO IN RELAZIONE ALLO STATO DI UNO DELL'OGGETTO

IMPLEMENTAZIONE

IMPLEMENTAZIONE DEL TIPO DI DATO = SECURIZZATA MAPPY-SORTED, OVVERO LISTA CERCA

TIPO DI DATO = ACCORDA SU UNA SEQUENZA ESEGUITA

PERMISI DI TIPO DATO APPLICABILI

I TDS SPECIFICANO L'IMPLEMENTAZIONE E IL SIGNIFICATO CHE LI OPERATORI NELL'IMPLEMENTAZIONE SEVERO AUMENTO CONFERISCONO A. RENDIMENTO IN PARTE DELLE ITCR PREVIMENTO DI INFORMAZIONI FONDANTI.

ESEMPIO PER TDS E DEFINIZIONI

DI ES.

ALLEGATA ESEGUENTE INDICA LE DUE FORME LIST, N, BOOL, ELEMENT (Dove ELEMENT CORRISPONDENTI SONO DEDOTTAMENTE SPECIFICATE) E LE OPERAZIONI CON NOTA PRIMA, SONO

- EMPTY(LIST) \rightarrow LIST
- SET $N \times$ ELEMENT \times LIST \rightarrow LIST
- ADD $N \times$ ELEMENT \times LIST \rightarrow LIST
- ADD BACK ELEMENT \times LIST \rightarrow LIST
- ADD FRONT ELEMENT \times LIST \rightarrow LIST
- REMOVE POS $N \times$ LIST \rightarrow LIST
- GET: $N \times$ LIST \rightarrow ELEMENT
- IS EMPTY LIST \rightarrow BOOL
- SIZE LIST \rightarrow N

N, B

$N \times$ LIST SPECIFICA
UN ELEMENTO, ONE LISTA
ACCETTAZIONE
"N" RAPPRESENTA UN ING.
MENTRE "LIST" UNA LISTA.

LS = LISTA, CR = CAPI, POS = POSIZIONE, CURSOR = CURSORI, FUSOLE = JULIA, LISTA = SOLO;

- **EMPTY LIST**: CREA LISTA VUOTA
- **SET**: MODIFICA L'ELEMENTO IN POSIZIONE POS
- **ADD**: ACCIUNGE UN ELEMENTO IN POSIZIONE POS
- **ADD BACK**: INSERIMENTO IN Coda
- **ADD FRONT**: INSERIMENTO IN TESTA
- **REMOVE POS**: RIMUOVE ELEMENTO IN POSIZIONE POS
- **GET**: RESTITUISCE L'ELEMENTO IN POSIZIONE POS
- **IS EMPTY**: VERIFICA SE LA LISTA E' VUOTA (VALORE BOOL)
- **SIZE**: RESTITUISCE LA DIMENSIONE

A NAISSI, COMPLESSITA' DELLE OPERAZIONI
SU LISTA

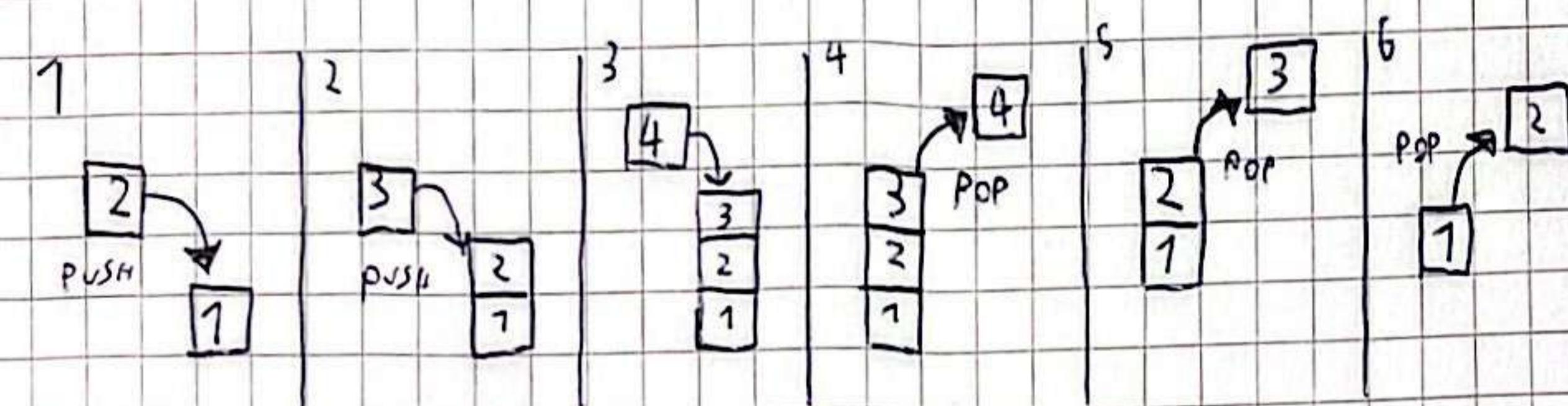
1)

Struttura dati: liste doppiamente collegate, circolari e con sentinella	Caso migliore	Caso peggiore
set: $N \times \text{Elem} \times \text{List} \rightarrow \text{List}$	$\Theta(1)$ primo elemento	$\Theta(n)$ <small>compronta la lista</small>
add: $N \times \text{Elem} \times \text{List} \rightarrow \text{List}$	$\Theta(1)$ se $n=0$	$\Theta(n)$ se $n \geq n$
addBack: $\text{Elem} \times \text{List} \rightarrow \text{List}$	$\Theta(1)$	$\Theta(1)$
addFront: $\text{Elem} \times \text{List} \rightarrow \text{List}$	$\Theta(1)$	$\Theta(1)$
removePos: $N \times \text{List} \rightarrow \text{List}$	$\Theta(1)$ se $n=0$	$\Theta(n)$
get: $N \times \text{List} \rightarrow \text{Elem}$	$\Theta(1)$	$\Theta(n)$
isEmpty: $\text{List} \rightarrow \text{Bool}$	$\Theta(1)$	$\Theta(1)$
size: $\text{List} \rightarrow N$	$\Theta(n)$	$\Theta(n)$
emptyList: $\rightarrow \text{List}$	$\Theta(1)$	$\Theta(1)$

STACK (OPION)

E' una sequenza di elementi di un certo tipo, in cui è possibile accedere a tutti gli elementi soltanto da un estremo. Una pila può essere visualizzata come un caso speciale di lista in cui l'ultimo elemento è anche il primo ad essere inserito e per cui è possibile accedere ad altri tipi avendo.

↳ I elementi citati sono simili in lista



ANALISI COMPISSIMA' D'ESERCIZIO
Stack

1)

Struttura dati: liste doppiamente collegate, circolari e con sentinella

push: $\text{Elem} \times \text{Stack} \rightarrow \text{Stack}$

pop: $\text{Stack} \rightarrow \text{Elem} \times \text{Stack}$

top: $\text{Stack} \rightarrow \text{Elem}$

emptyStack: $\rightarrow \text{Stack}$

isEmpty: $\text{Stack} \rightarrow \text{Bool}$

Caso migliore

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

Caso peggiore

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

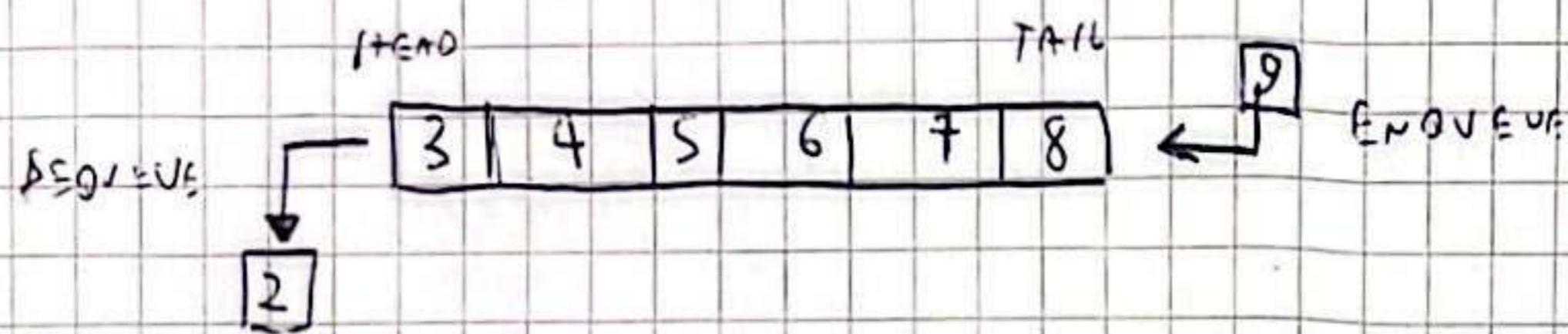
2) INIZIO E TUTTO DALLA TESTA PER AGGIUNTE INIZI E OPERATORI COSTRUTTI

Struttura dati: liste collegate semplicemente	Caso migliore	Caso peggiore
<i>push: Elem x Stack → Stack</i>	$\Theta(1)$	$\Theta(n)$
<i>pop: Stack → Elem x Stack</i>	$\Theta(1)$	$\Theta(n)$
<i>top: Stack → Elem</i>	$\Theta(1)$	$\Theta(n)$
<i>emptyStack: → Stack</i>	$\Theta(1)$	$\Theta(n)$
<i>isEmpty: Stack → Bool</i>	$\Theta(1)$	$\Theta(n)$

3) INIZIO E TUTTO DAL FONDO PER EIMINARE C/ JITTER

Struttura dati: array dinamico + size e maxsize (con ridimensionamento)	Caso migliore	Caso peggiore
<i>push: Elem x Stack → Stack</i>	$\Theta(1)$	$\Theta(n)$ SE DEVO RIDIMENSIONARE
<i>pop: Stack → Elem x Stack</i>	$\Theta(1)$	$\Theta(n)$
<i>top: Stack → Elem</i>	$\Theta(1)$	$\Theta(1)$
<i>emptyStack: → Stack</i>	$\Theta(1)$	$\Theta(1)$
<i>isEmpty: Stack → Bool</i>	$\Theta(1)$	$\Theta(1)$

QUEUE (OVERVIEW)



E' UNA SEQUENZA DI ELEMENTI DI UN CERTO TIPO, IN CUI E' POSSIBILE ACCEDERE AGLI ELEMENTI AD UN ESTREMO E TOLIERE UNA POSIZIONE OPPOSTA. UNA CODA PUO' ESSENTE QUINDI VISITA COME UNA PARTICOLARE LISTA IN CUI E' POSSIBILE ACCEDERE AGLI ELEMENTI AD UN ESTREMO (IL FONDO) E TOLIERE ELEMENTI DALL'ALTRA ESTREMO (TESTA)

COMPLESSITA' OPERAZIONI CODA

1)

Struttura dati: liste doppiamente collegate, circolari e con sentinella

enqueue: *Elem* x *Queue* → *Queue*

dequeue: *Queue* → *Elem* x *Queue*

first: *Queue* → *Elem*

emptyQueue: → *Queue*

isEmpty: *Queue* → *Bool*

Caso migliore

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

Caso peggiore

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

$\Theta(1)$

2) METTI IN FORMA E SCRIVI, PIANA POSSA FAE NOME EFFETTUARE IN FINST

Struttura dati: liste collegate semplicemente	Caso migliore	Caso peggiore
enqueue: $\text{Elem} \times \text{Queue} \rightarrow \text{Queue}$	$\Theta(1)$ METTI IN FORMA FONDO	$\Theta(n)$
dequeue: $\text{Queue} \rightarrow \text{Elem} \times \text{Queue}$	$\Theta(1)$ PIENO UNA TUTTA	$\Theta(1)$
first: $\text{Queue} \rightarrow \text{Elem}$	$\Theta(1)$	$\Theta(1)$
emptyQueue: $\rightarrow \text{Queue}$	$\Theta(1)$	$\Theta(1)$
isEmpty: $\text{Queue} \rightarrow \text{Bool}$	$\Theta(1)$	$\Theta(1)$

3)

Struttura dati: array dinamico + size e maxsize (con ridimensionamento)	Caso migliore	Caso peggiore
enqueue: $\text{Elem} \times \text{Queue} \rightarrow \text{Queue}$	$\Theta(1)$	$\Theta(n)$ SE USO ALTRIMENTI
dequeue: $\text{Queue} \rightarrow \text{Elem} \times \text{Queue}$	$\Theta(n)$ pos == 0 E SOLO SARANNO DA SINISTRA	$\Theta(n)$
first: $\text{Queue} \rightarrow \text{Elem}$	$\Theta(1)$	$\Theta(1)$
emptyQueue: $\rightarrow \text{Queue}$	$\Theta(1)$	$\Theta(1)$
isEmpty: $\text{Queue} \rightarrow \text{Bool}$	$\Theta(1)$	$\Theta(1)$

O NEMO: EFFETTUARE IN ENQUEUE E INERTIALE IN DEQUEUE. O VICEVERSA;

SUPPORE DI METTERE: IN POS. 12: E PERDERSI DA POS 0

IMSIEMI (\subseteq)

ACCIAZIO DI ELEMENTI A VALORI OMOGENEI (PROVERBIO SULLO STUFO)
DISTINTI (SINGOLA OCCORRENZA)

COMPLESSITÀ OPERAZIONI IMSIEMI

1)

Struttura dati: liste doppiamente collegate, circolari e con sentinella	Caso migliore	Caso peggiore
<i>emptySet: → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>insertElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>deleteElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>setUnion: Set x Set → Set</i>	$\Theta(m)$ <small>$\Theta(\max(m, m, m \times m))$</small>	$\Theta(n)$ <small>$\Theta(m, m, m \times m)$</small>
<i>setIntersection: Set x Set → Set</i>	$\Theta(m)$ <small>$\Theta(m \times m)$</small>	$\Theta(n)$ <small>$\Theta(m \times m)$</small>
<i>isEmpty: Set → Bool</i>	$\Theta(1)$	$\Theta(1)$
<i>size: Set → N</i>	$\Theta(n)$	$\Theta(n)$
<i>member: Elem x Set → Bool</i>	$\Theta(1)$ <small>SE E' IL PRIMO</small>	$\Theta(n)$

2)

Struttura dati: liste collegate semplicemente	Caso migliore	Caso peggiore
<i>emptySet: → Set</i>	$\Theta(1)$	$\Theta(1)$
<i>insertElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>deleteElem: Elem x Set → Set</i>	$\Theta(1)$ <small>NON TUTTI SONO VENDIBILI</small>	$\Theta(n)$
<i>setUnion: Set x Set → Set</i>	$\Theta(m)$ <small>$\Theta(\max(m, m, m \times m))$</small>	$\Theta(n)$ <small>$\Theta(m, m, m \times m)$</small>
<i>setIntersection: Set x Set → Set</i>	$\Theta(m)$ <small>$m \times m$</small>	$\Theta(n)$ <small>$m \times m$</small>
<i>isEmpty: Set → Bool</i>	$\Theta(1)$	$\Theta(1)$
<i>size: Set → N</i>	$\Theta(n)$	$\Theta(n)$
<i>member: Elem x Set → Bool</i>	$\Theta(1)$	$\Theta(n)$

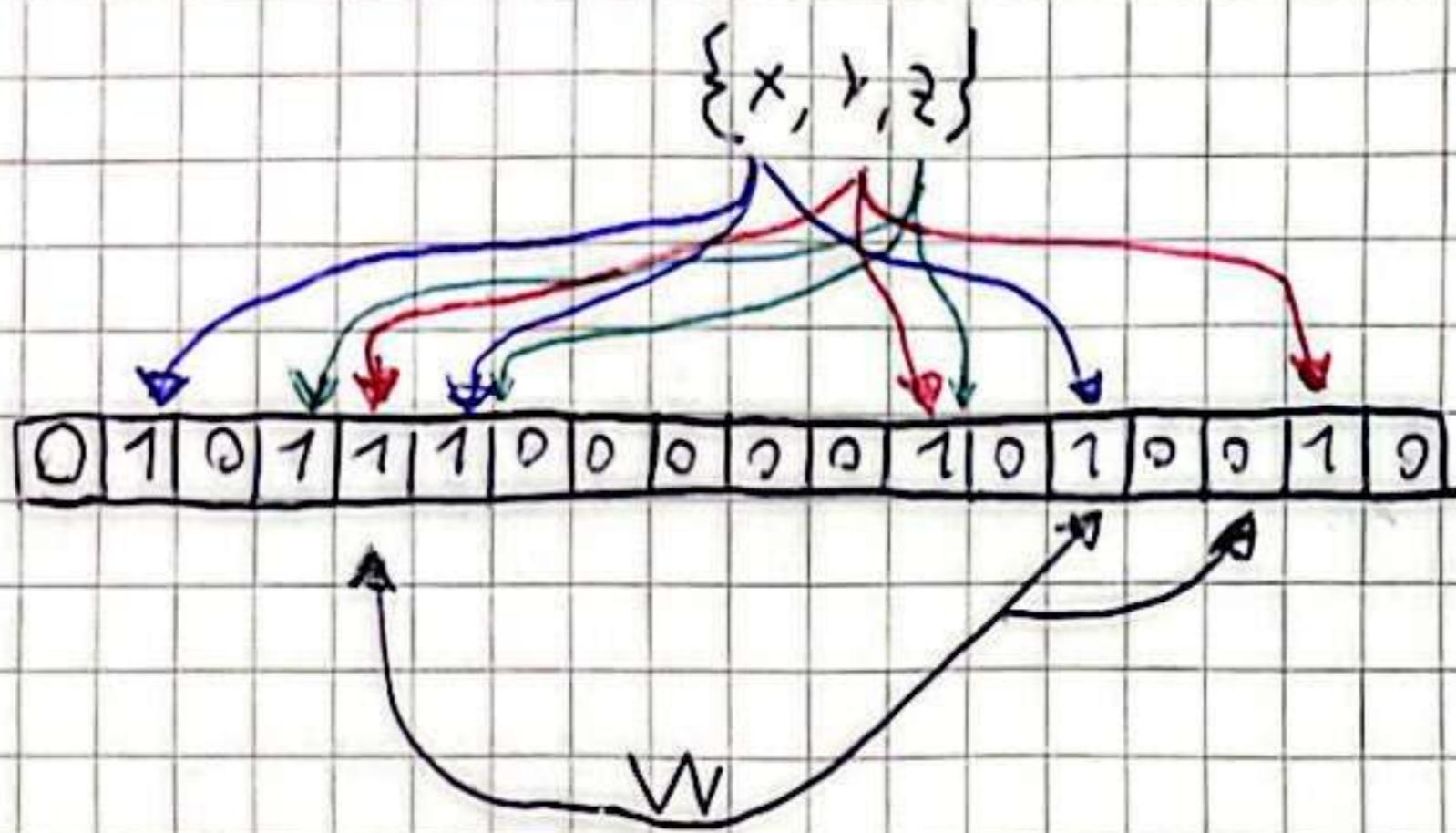
3)

Struttura dati: array dinamico + size e maxsize (con ridimensionamento)	Caso migliore	Caso peggiore
<i>emptySet: → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>insertElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>deleteElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>setUnion: Set x Set → Set</i>	$\Theta(\max(m, m, m \times m))$	$\Theta(\max(m, m, m \times m))$
<i>setIntersection: Set x Set → Set</i>	$\Theta(\min(m, m))$	$\Theta(m \times m)$
<i>isEmpty: Set → Bool</i>	$\Theta(1)$	$\Theta(1)$
<i>size: Set → N</i>	$\Theta(1)$	$\Theta(1)$
<i>member: Elem x Set → Bool</i>	$\Theta(1)$	$\Theta(n)$

4)

Struttura dati: array ordinato con size e maxsize (assumiamo per semplicità di non dover ridimensionare)	Caso migliore	Caso peggiore
<i>emptySet: → Set</i>	$\Theta(1)$	$\Theta(1)$
<i>insertElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>deleteElem: Elem x Set → Set</i>	$\Theta(1)$	$\Theta(n)$
<i>setUnion: Set x Set → Set</i>	$\Theta(\min(m, m))$	$\Theta(m + m)$
<i>setIntersection: Set x Set → Set</i>	$\Theta(1)$	$\Theta(\min(m, m))$
<i>isEmpty: Set → Bool</i>	$\Theta(1)$	$\Theta(1)$
<i>size: Set → N</i>	$\Theta(1)$	$\Theta(1)$
<i>member: Elem x Set → Bool</i>	$\Theta(1)$	$\Theta(\log n)$

BIT VECTOR



LE OPERAZIONI LEGATE ALLA OPERAZIONE BIT A BIT SONO A CONTROLLO, MOLTIPLICANO I COSTANTI, BIT DI UN VALORE BINARIO, DANNO IL RISULTATO IN BASE AI BIT DEI VALORI.

BIT VECTOR E OPERAZIONI &(AND), ~(NOT), |(OR)

- "**&**" (AND): QUESTA OPERAZIONE CONFERMA I BIT CORRESPONDENTI NELLE OPERANDI. SE UN BIT E' 1 SE E' 0 SOLO SE ENTRAMBI I BIT CORRESPONDENTI NELLE DUE OPERANDI SONO 1. AD ESEMPIO, SE HOO 1010 (10 DECIMALE) E 1100 (12) IL RISULTATO DI & SARÀ 1000 (8)

- "**~**" (NOT): QUESTA OPERAZIONE INVERTE TUTTI I BIT IN UN NUMERO, TRASFORMANDO OGNI 0 IN 1 E OGNI 1 IN 0. AD ESEMPIO 1010, UN NUMERO IN DECIMALE, CONSIDERANDO I BIT CORRESPONDENTI NELL'UNICO OPERANDO, SE E' 1 SE ALMENO UNO DEI DUE BIT CORRESPONDENTI NELL'UNICO OPERANDO E' 1, AD ESEMPIO 1010 E

1100 IL RISULTATO DEL'OPERAZIONE "**|**" SARÀ 1110

DIZIONARI

DEFINIZIONE:

- I dizionari si chiamano anche MAPPE o "ARRAY AD ASSOCIAZIONI".
- Senza ad implementare funzioni che non si possono calcolare "al fly" ad esempio, la funzione che associa ad ogn. parola il suo significato (chiave = parola, valore = significato).

TABELLE DI HASH PER IMPLEMENTARE DIZIONARI

Inizieremo, le tabelle di hash nascono per ovviare al problema della spreco di spazio dati TABELLE AD ACCESSO diretto. Le chiavi possono non essere numeri e il loro uscire può anche essere casuale, per accedere alla cella dell'array associata ad una chiave esistono delle funzioni (dette di hash) che trasformano attraverso un algoritmo la chiave in un numero.

Ad esempio, se in chiave è una stringa, una funzione di hash potrebbe effettuare la parte interna del rapporto tra la somma dei valori ASCII delle lettere compresenti la stringa e la size dell'array.

Una buona funzione di hash dovrebbe distribuire gli elementi uniformemente all'interno dell'array. Ad esempio se hai 100 chiavi ed hai a disposizione 10 celle, una buona funzione di hash associa 10 chiavi ad ogni cella.

La funzione di hash deve essere CALCOLABILE in tempo COSTANTE altrimenti si perde il vantaggio di avere questa struttura dati.

Se vogliamo accedere un elemento nella tabella in uno posizionando l'indirizzo occupata si crea una collisione. Per risolvere le collisioni si creano celle (simplici o chainaggio o bucket) in cui vengono memorizzati tutti gli elementi associati alla stessa cella dell'array e all'interno di quest'ultima si inserisce il puntatore alla lista.

Una funzione di hash è perfetta quando non ci sono collisioni.

Una funzione di hash perfetta deve essere INIEZIONE E CALCOLABILE in tempo costante.

SOLUZIONE PARTE 1

TABELLA AD ACCESSO diretto

Sono dizionari basati su una struttura di memoria direzionale che si basa su un array, dove costruisce un up array in cui ogni cella contiene un solo (elemento) e ad ogni elemento è associata una chiave, per accedere ad un determinato elemento è necessario sapere la chiave a cui esso corrisponde.

- FATTORE DI CARICO

Minimizza il grado di sovrapposizioni di una tabella usando la formula di carico di:

$$\alpha = \frac{\text{numero elementi}}{\text{numero celle array}}$$

AL NUMERO DEI CELLE SECONDO, PIÙ' SI AVVICINA A 1 PIÙ' STAMO VERSO IL BORDO SPAZIO.

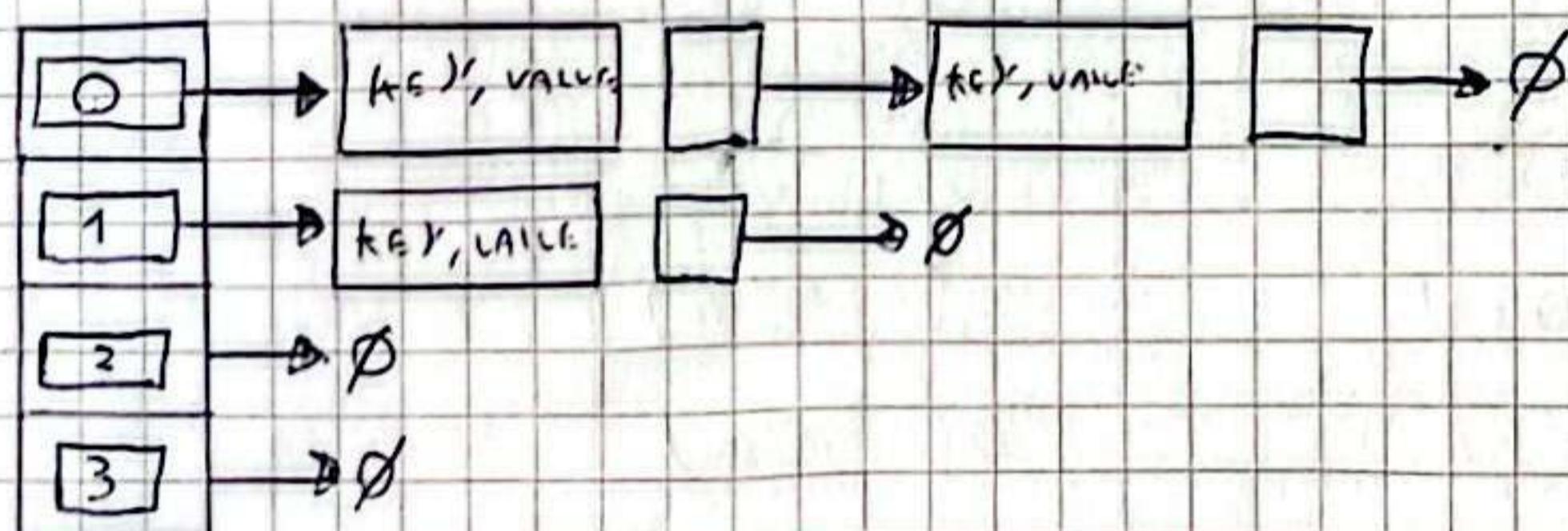
- PRECI: TUTTE LE OPERAZIONI NELL'ORDINE TEMPO $\Theta(1)$ TRATTI LA CREATURA EPPURE UNA LISTA METTENDO IN COSTANTE EQUIVALENTE A "ELEMENTO VUOTO" IN TUTTI LE m CELLE DELLA TABEELA, QUIQUI RICHIESTA $\Theta(m)$

DIRETTA UCCHIA, DEVORO ESSERE NECESSARIAMENTE INTERI IN $[0, m-1]$

LO SPAZIO UTILIZZATO E' PROPORZIONALE A m , MA AL NUMERO M PIÙ' ELEMENTI PUO' ESSERE POSSIBILE UN GRANDE SPACIO DI MEMORIA.

LISTE DI COLLEGAMENTO

INSEGUIMENTO LISTS OF KEY, VALUE PAIRS



Ogni ELEMENTO DOPO CONTENUTO IN LISTA ESSERE ALLA TABEELA (CHIAMATA BUCKET) $V[i]$ PURTA ALLA LISTA VECCHI ELEMENTI TALI CHE $H(k) = i$

INDIRIZZAMENTO APERTO

NEL CASO IN CUI LA POSIZIONE $H(k)$ E' UN INDICES UNA CHIAVE A SIA GIA' OCCUPATA, IL METODO PREVUO' DI POSIZIONARIA IN UN'ALTRA CELLA VUOTA, ANCHE SE QUESTA POSSIBILE SPETTARE DI SINISTRA AD UN'ALTRA CHIAVE, LE OPERAZIONI VERRANO REINVIATE.

- INSERIMENTO: SE $V[H(k)]$ E' VUOTA, INSERISCI LA COPPIA (EL, k) IN QUESTA POSIZIONE; ALTRIMENTI, A PARTIRE DA $H(k)$, ISPERZI LA CHIAVE DELLA TABEELA E INSERISCI NELLA PRIMA CELLA VUOTA

- RICERCHE: SE, DURANTE LA RICERCA USCE CELLE, NE VIENE TROVATA UNA CON CHIAVE CORRISPONDENTE, RESTITUISCI L'ELEMENTO TROVATO; ALTRIMENTI, SE SI AMMA A UNA CELLA VUOTA O SI È SCARICATO ULTERIORI TABEELA SERIA SUFFICIENTE RESTITUIRE NULL

- CANCELLAZIONE: AFFRANCATE LA CHIAVE CON IL METODO APERTO UGUALE FUNZIONE OCCORRE ADOTTARE UNA STRATEGIA PARTICOLARE PER LA CANCELLAZIONE, OSSIA UTILIZZARE UN VALORE SPECIALE "DELETE ITEM" NEL CAMPO EL DELL'ELEMENTO ONE, CHE VUOLSI RIMUOVERE;

→ U di FSA, mentre la ricerca in ordine parziale

ESEMPIO:

Supponiamo che $m = 8$ e le chiavi siano "a, b, c, d" (associative) e valori che per ogni chiave $H(j) = 3$, $H(b) = 0$, $H(c) = 4$ e $H(d) = 3$.

Usando la strategia di hashage più semplice, il risultato finale è:

$$\text{f.s. } H_i(x) = (H(x) + i) \bmod m$$

COMESI/A:

LISTA di collisioni:

OPERAZIONI	COSTO MIGLIORE	COSTO PEGGIOR
INSERT (ELEMENTO, CHIAVE k)	$\Theta(1)$	$\Theta(1 + M/m)$
DELETE (CHIAVE k)	$\Theta(1)$	$\Theta(1 + M/m)$
SEARCH (CHIAVE k) → ELEMENTO	$\Theta(1)$	$\Theta(1 + M/m)$

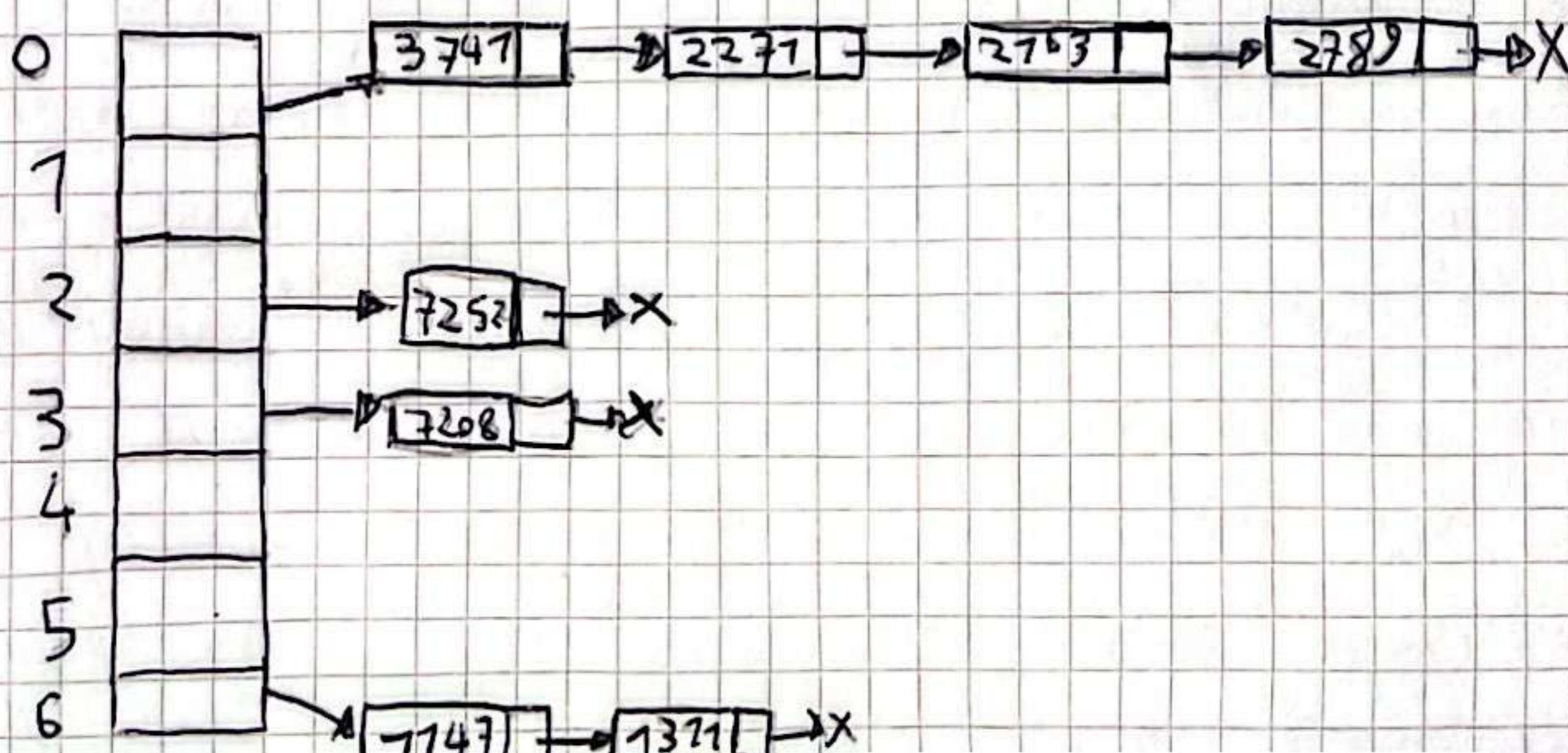
Scenari dominanti. Esempio su hash (avviando)

La funzione di hash $H' = H : (a^2 + b + c + d) \bmod 7$ (può cambiare da esempio a esempio)

3741	2147	2271	7252	2163	7208	7311	2789
"mn"	"lo"	"ln"	"pg"	"le"	"ng"	"ln"	"lv"

∅	6	∅	2	∅	3	6	∅
---	---	---	---	---	---	---	---

DOMANDA 153, 1.



3.2.1

- LA PRIMA PROPRIETÀ È QUELLA DI ESSERE CALCOLATE IN UN TEMPO COSTANTE

- DEVE ESSERE UNIFORME, L'UNIFORMITÀ È UNA PROPRIETÀ PER IL
TIPO DI CARICO VERSO QUESTA PARTE DEL BUCKET. LA FORTUNA HA MOR
DEVE "SOVRAFFARE" ALLE ALTRE PARTI DELLA TABELLA, L'ALTRIMENTE NON VORRE.

3.2.2.

In questo caso la prima proprietà (tempo costante) si soddisfa perché
la nostra funzione di hash è prefabbricata dall'operatore ANDAMENTO SEMPRE:
SOMMA, moltiplicazione e moltiplicazione. Questi operatori sono comunque corrispondenti
operatori a tempo costante perché il tempo di esecuzione con diversi
della dimensione del loro input. Notare in questo caso le carni sotto
SEGUENTI HSI 4 CIFRE DECIMALI che vanno da 0 a 9 quindi anche a
LIVELLO DI CALCOLO non sono numeri elevati

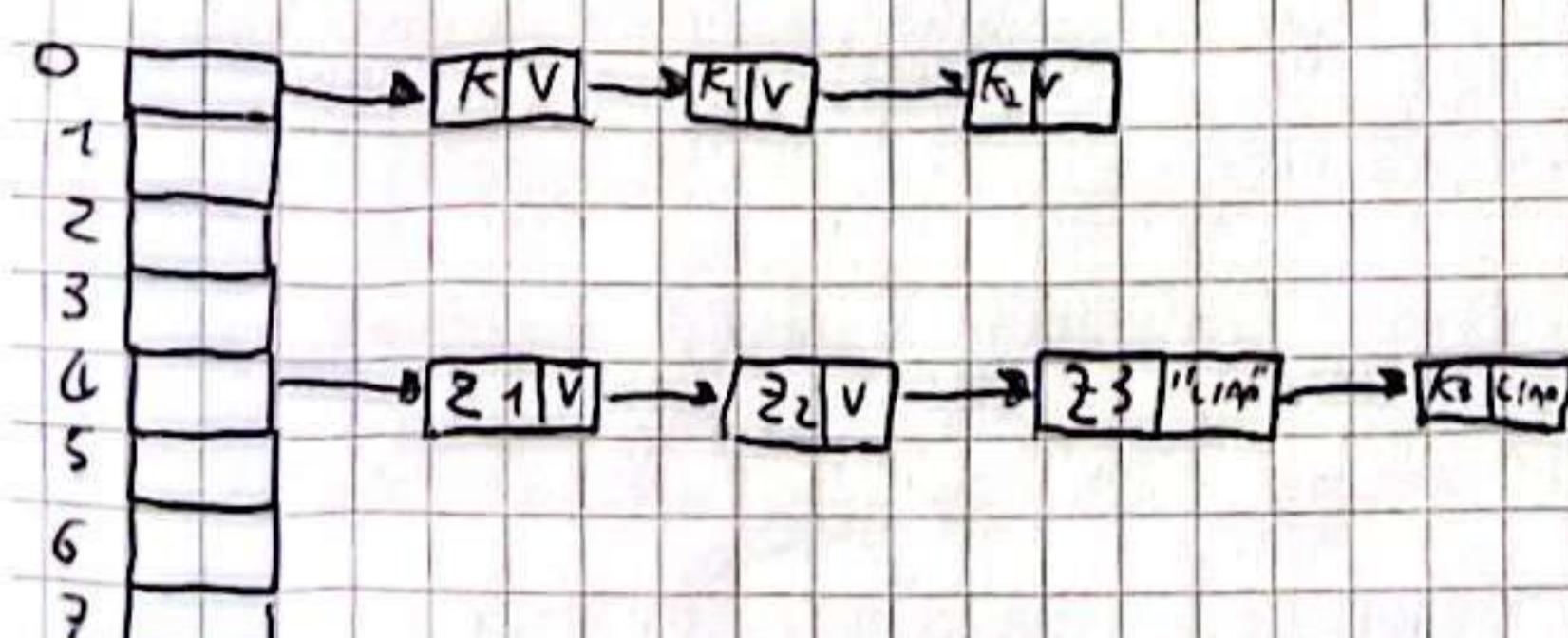
3.2.3.

Per avere rispettata la seconda proprietà (uniformità) pensiamo
che le carni crescano più frequentemente la stessa variazione di punti, come
evidenzieremo dal fatto che altri bucket contengono più carni rispetto
a quelli.

SUMMARY 3.3.1

ESEMPIO CONCRETO:

$$(k_3, v_3) \quad h(k_3) \in [0, m) \subset \mathbb{N}$$



I valori possono essere duplicati, cioè carni no.

3.3.2

THEIA(M) QUANDO TUTTI GLI ELEMENTI SONO INORDINATI STEREO FORTE.

F

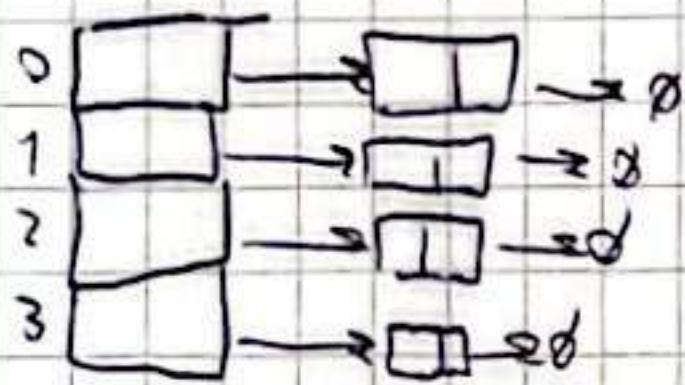


F

L

3.3.3

ABBANDONARE (M/M) NEL CASO DELLA VULCANITA JUMPING



3.3.4

THEIA(i) QUANDO LA MIA FUNZIONE DI STIMA E' INCREMONA

$$H(p) = H(q) \iff p = q$$

ALBERI

Gli alberi sono struttura di visualizzazione quando si trovano per la prima volta. Presentano radici, cenni e rami. A differenza delle liste, ogni elemento può avere alberi come suoi figli elementi.

ALBERI NATURALI E TERMINOLOGIA

Un albero naturale è una coppia $T = (N, A)$ costituita da un insieme N di nodi (o vertici) e un insieme A (sottrazioni, o NRM) di legami. I nodi devono anche.

Il fatto che un albero sia naturale significa che un vertice viene chiamato

Come radice è differente dai altri perché è l'unico senza un padre,

il numero dei figli di un nodo si chiama grado; un nodo ha figli

si chiama foglia è tutti gli altri si chiama nodo interno; gli altri

solo tutti i nodi naturali salvo di padre in padre è il

ALBERGO IN FRONTE.

PROPSITA' DI UN ALBERGO E' IL NUMERO DI ALBERI CHE SONO BISOLTI

AVVENIMENTO: UN ACCIDENTE PANTERINO DANNAGGIOVA,

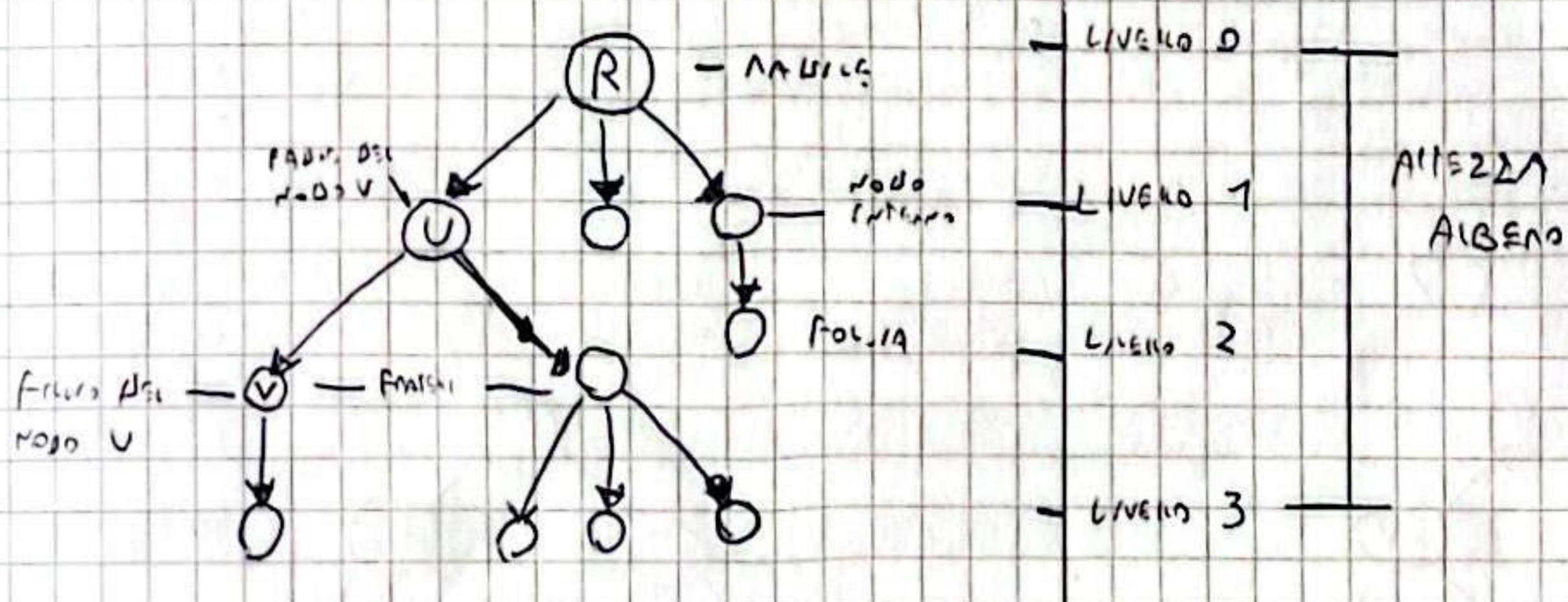
ALLORA HA PROFONDITA' O' SE UN ALBERO HA PROFONDITA' R, ALLORA ESISTE
UN ALBERO PROFONDITA' $R+1$

PER TUTTO LO STESO CERTO CHE VERSO DEGLI ALBERI FINISCE E HANNO LA

ESTRA PROFONDITA'

ALBERA UN ALBERO E' UNA MASSIMA PROFONDITA' A CUI UN ALBERO

OCCURRÀ.



ALBERO SPECIALI

SISTEMA ALBERI TRIPOLARE. DI ALBERI CON VERSO SINGOLAMENTE CHE NON

CONCERNANO LA NARRAZIONE. O CONSIDERA SI NEGLIGEANTI PRESENZA DI

TRIPLICI DI ELEMENTI IN MODO PANTROPONAMENTE EFFETTUATE.

1) PRATICHE, VERSO SINGOLARI ALBERI CON CAUSA DI UN SOLO G

2) PROFONDITA'

UN ALBERO P-ARIO E' UN ALBERO IN CUI I SOLO ALBERI SONO AL PIU'

ALBERI D.

UN ALBERO D-ARIO E' UN ALBERO CON $D=2$ SOLO ALBERI BINARIO

UN ALBERO BINARIO COMPLETO E' UN ALBERO BINARIO IN CUI OGNI

LEVELLO, TRAMITE EVIDENTEMENTE L'ULTIMO, E' COMPLETAMENTE PIENO, E' MUITO

SOLO SOLO IL PIU' A SINISTRA POSSIBILE.

PROPRIETÀ 1: SE UN ALBERO BINARIO COMPLETO È QUASI COMPLETO

PROPRIETÀ 1: UN ALBERO BINARIO COMPLETO DI ALTEZZA H HA $2^{H+1} - 1$ NODI

PROPRIETÀ 2: SIA T UN ALBERO BINARIO QUASI COMPLETO DI ALTEZZA H , ALLORA IL NUMERO DI NODI G DI T È $2^H \leq G \leq 2^{H+1} - 1$

PROPRIETÀ 3: L'ALBERO BINARIO QUASI COMPLETO T CON M

$$\text{NODI } E \quad H = \lceil \log_2 M \rceil$$

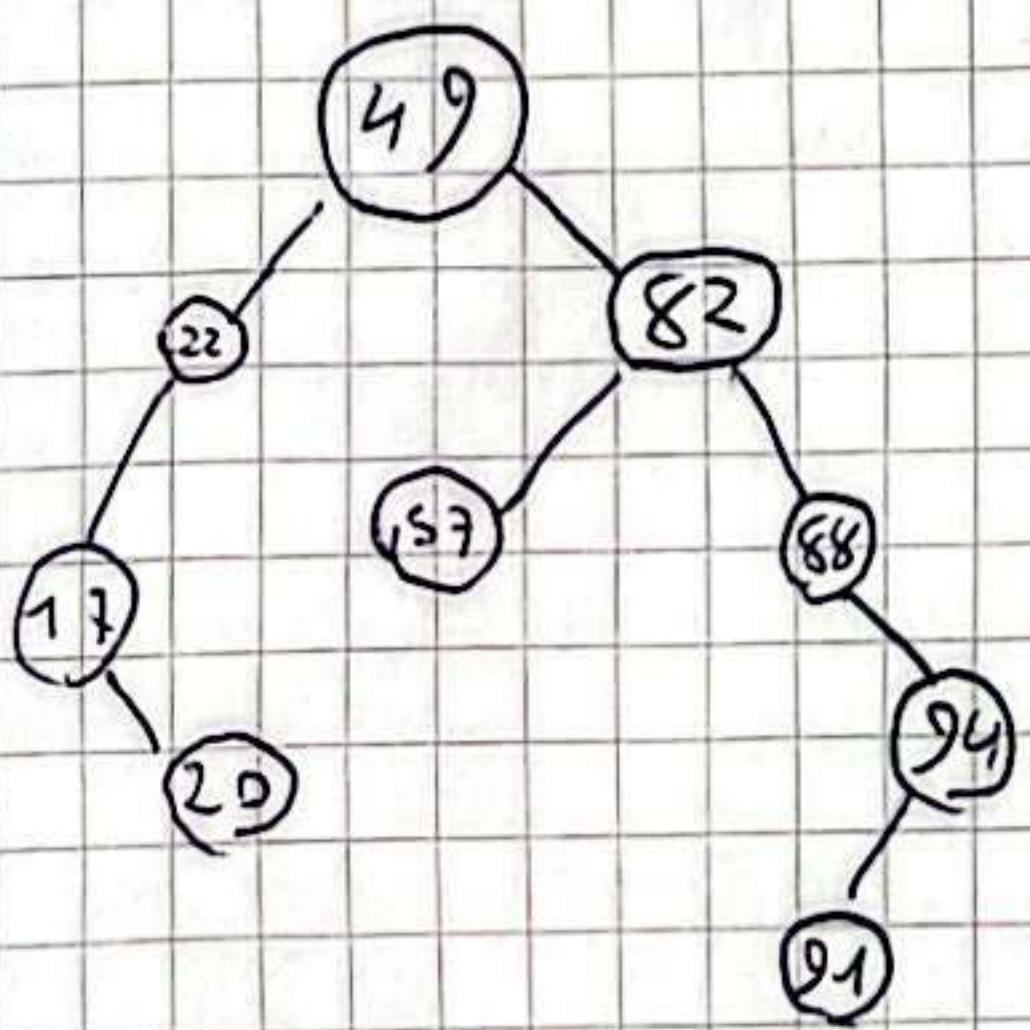
ALBERI BINARI DI ALTEZZA (BST)

Sono Alberi Binari che sono devoluti per fare efficienza complessiva quasi complete.

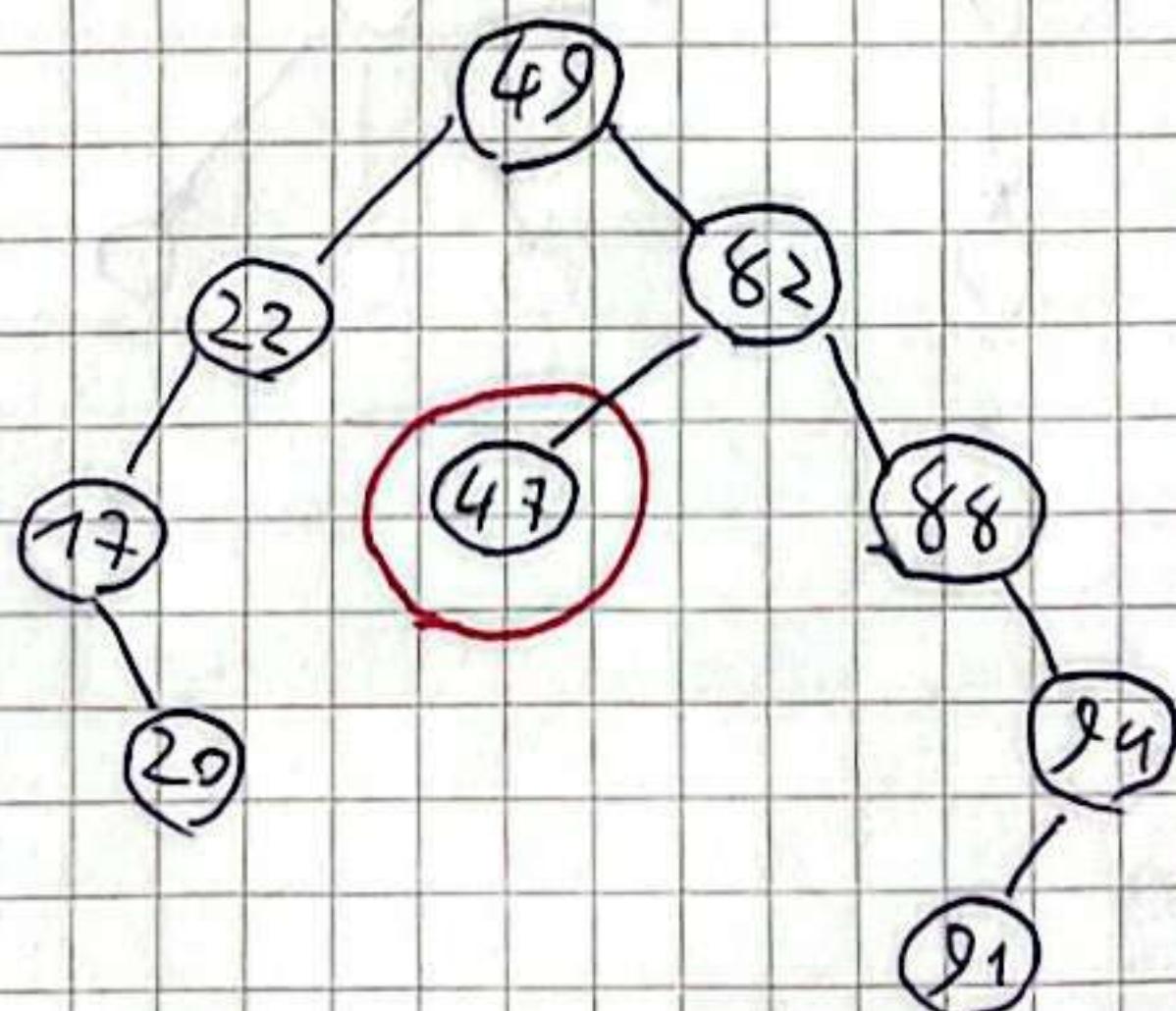
A questo modo è possibile un elemento ed un certo numero preso per dominio totalmente ordinato, quindi ogni catena deve essere continuata con le altre.

Le catene dei nodi elementi nel sottoalbero sinistro di un vertice sono

\leq CARAV(V) mentre gli elementi nel sottoalbero destro sono \geq CARAV(V)

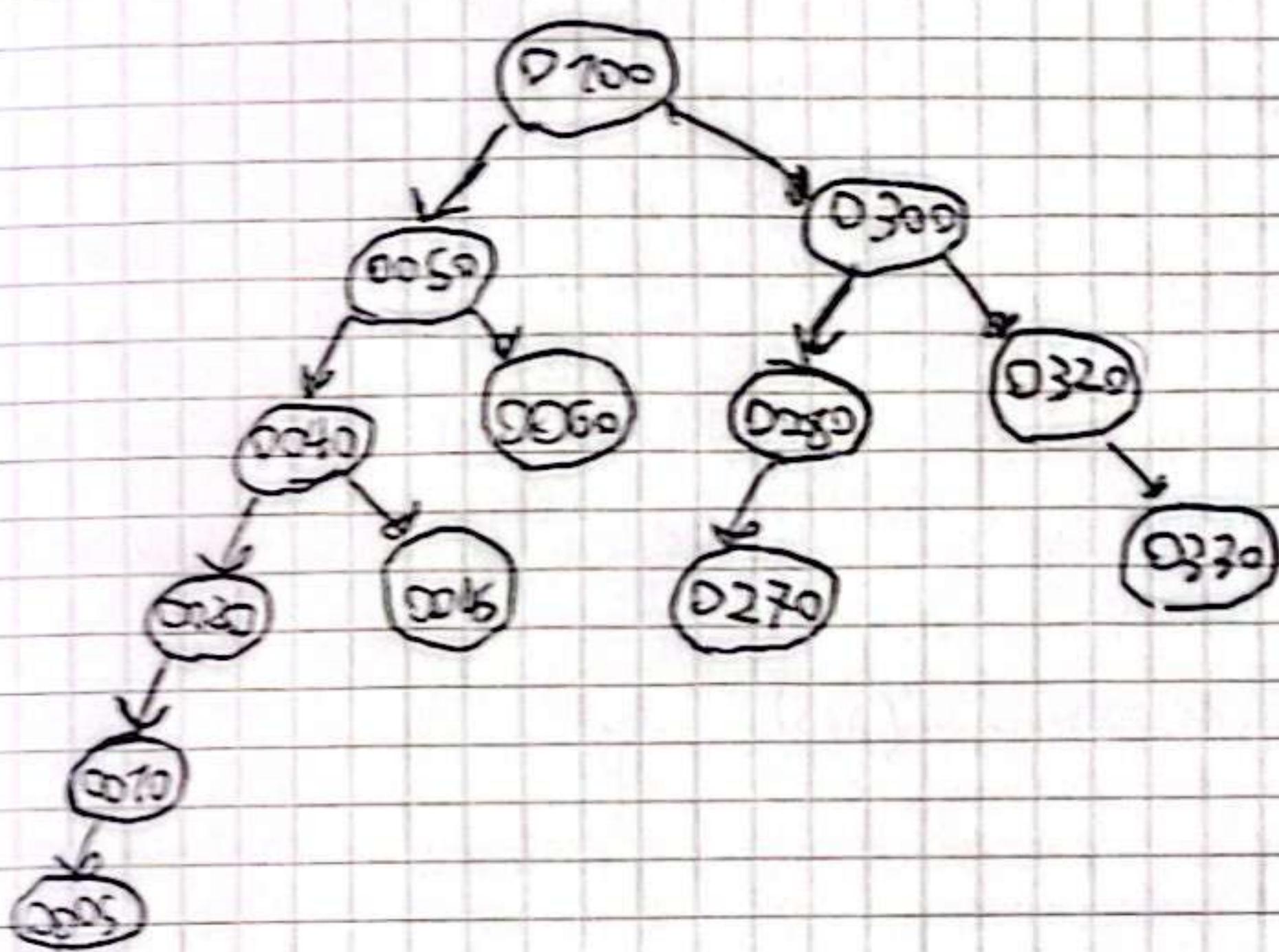


Albero Binario (S)
Altezza



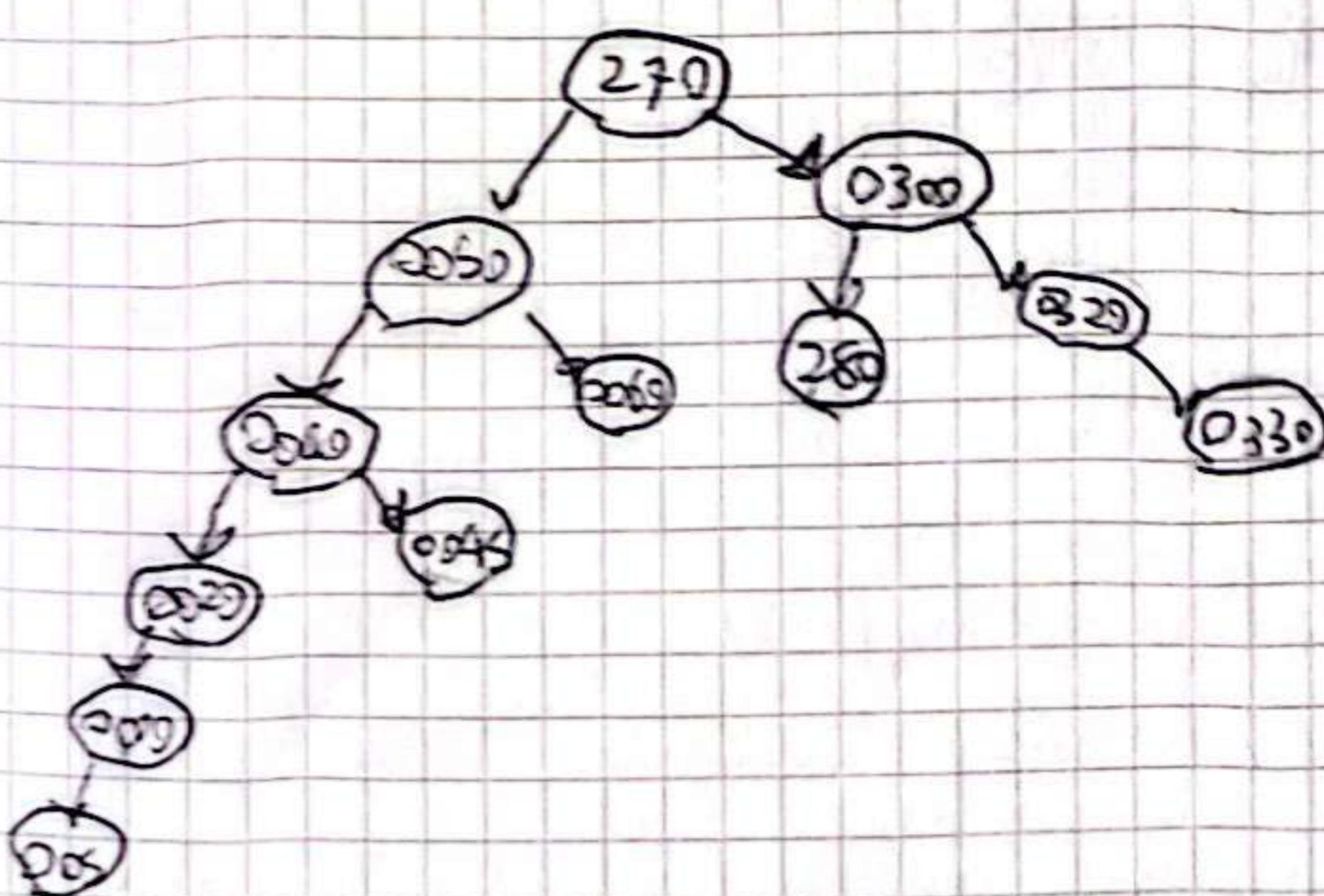
Albero Binario Non Di Altezza

DOMANDA ESAME BST (AVVOLGENTE)



REPRESERVA IL CASO PERMETTENDO L'INSEGNAMENTO E' UN PESA
H, CON H L'ALTEZZA DEGLI ALBERI. PER INSEGNARE SI FA
ETA(H) CHE APPUNTO RAPPRESENTA IL CASO PERMETTENDO

ELIMINARE UN NODO, PER TROVARE SI DOVRAE TROVARE L'ALBERO
CHE VERRA' LA FUSIONE ASSIANDO DETERMINANTI CHE CI
SERVIRANNO A TROVARE IL NODO PIU' PROSSIMO.
TROVIAMO IL PROSPETTO DEI GRANDI. QUINDI IN QUESTO CASO
DETERMINIAMO ELIMINARE 270 E PARTE DA UNA CHIAVE VALORE A
SCHEDE ELIMINARE E' UN DELETE EVENT SOSTITUITO DA 100 COR 270



NEL CASO DI UNA PESANTE PENETRAZIONE AVVIENE FINO IN FONDO

VISITA DI ALBERO (DFI)

Sono algoritmi che consentono di visitare sistematicamente un albero e acciunghi di un albero.

Oli algoritmi di VISITA si distinguono in base al criterio di ordinazione dei nodi,

VISITA GENERICA

VISITA GENERICA VISITA IL NODO R E GUITTI I FIGLI DISORDINATI DI UN ALBERO, SE E' UN'ISTANZA.

ARBITRARIO: TEMPO $\Theta(m)$ PER VISITARE UN ALBERO CON m NODI A PARTIRE DALL'RADICE

VISITA IN PROFONDITA (DFS)

• DA VISITA IN PROFONDITA' INTRINSECAENTE: ricorsiva, si esegue la profondita'

• L'ALGORITMO DI VISITA IN PROFONDITA' PONE UNA RADICE R E PROcede VISITANDO I FIGLI DI QUESTA IN FISSO-PIENO A MIGLIORIA UNA FOGLIA, METTENDO POI AL PRIMO ANTERATO UNTA ALTRA FIGLIA NON VISITATA (SE ESISTE) E RIPETE IL PROCEDIMENTO A PARTIRE DA QUESTA ULTERIORE FIGLIA.

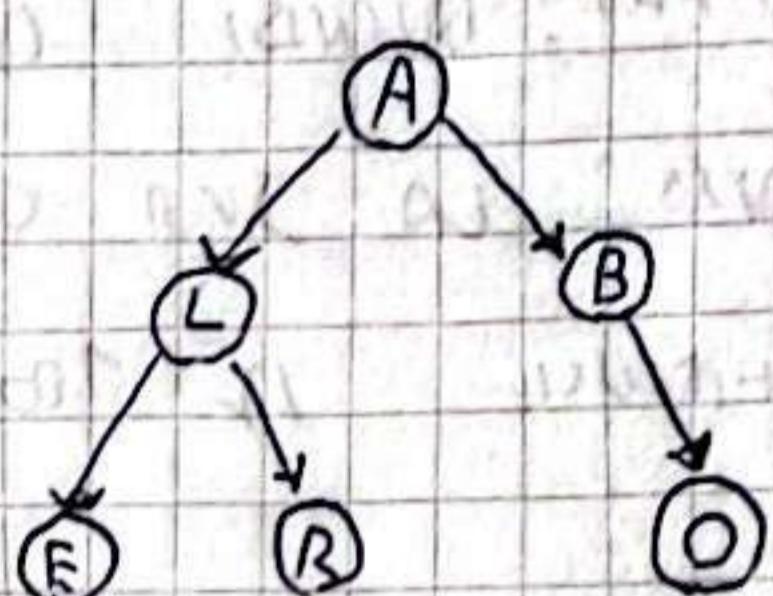
VISITA PREORDINE, SIMMETRICA, POSTORDINE DI ALBERO GENERICO

VISITA IN PREORDINE: RADICE, SOTTOALBERO SX, SOTTOALBERO DX

SIMMETRICA: SOTTOALBERO SX, RADICE, SOTTOALBERO DX

POSTORDINE: SOTTOALBERO SX, SOTTOALBERO DX, RADICE

ESEMPI:



PREORDINE: A L E R B O

SIMMETRICA: E L R A B O

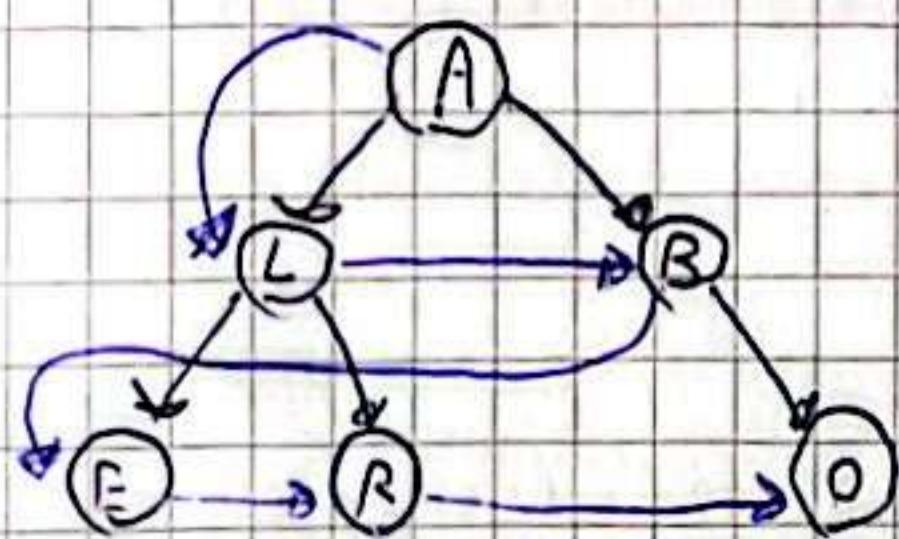
POSTORDINE: E R L O B A

VISITA IN AMPIAZZA (BFS)

UCCONIMO PER USAR E PERMETTE VISITANDO NODI PER LIVELLO

UN NODO SUL LIVELLO I PUO' ESSERE VISITATO SOLO SE tutti i nodi del livello i-1 SONO STATI VISITATI

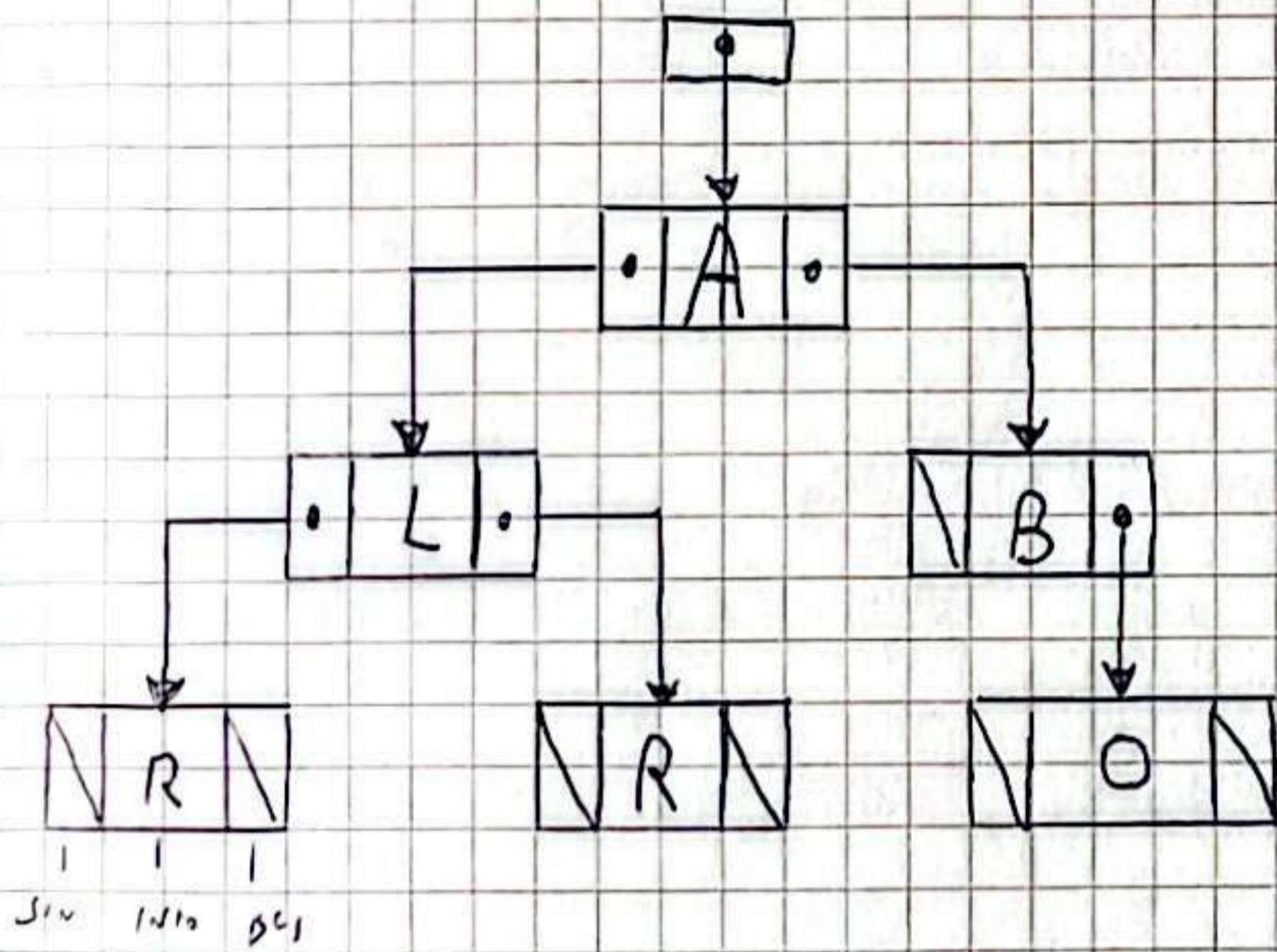
Ejemplo:



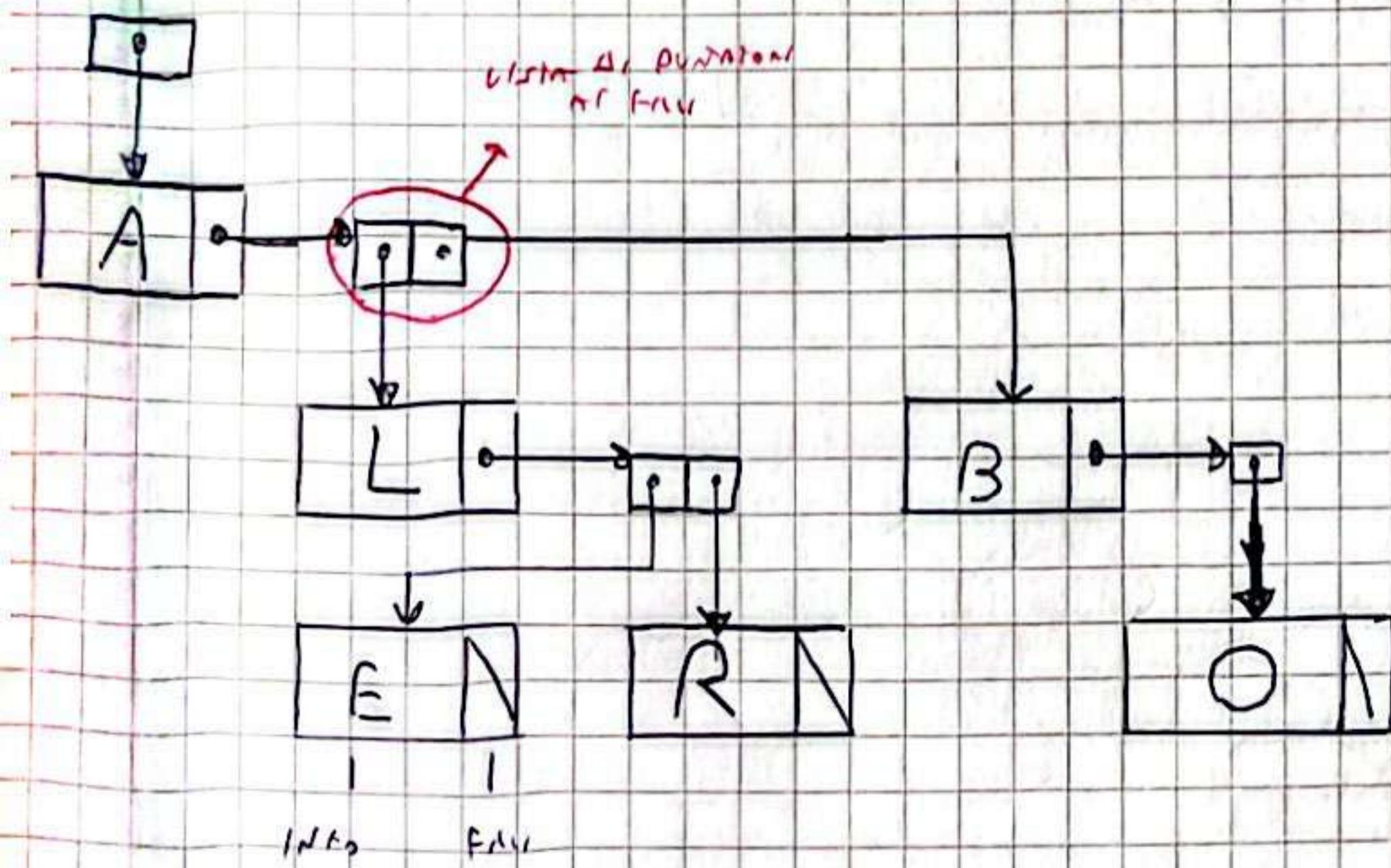
VISITA BFS ALBERTO

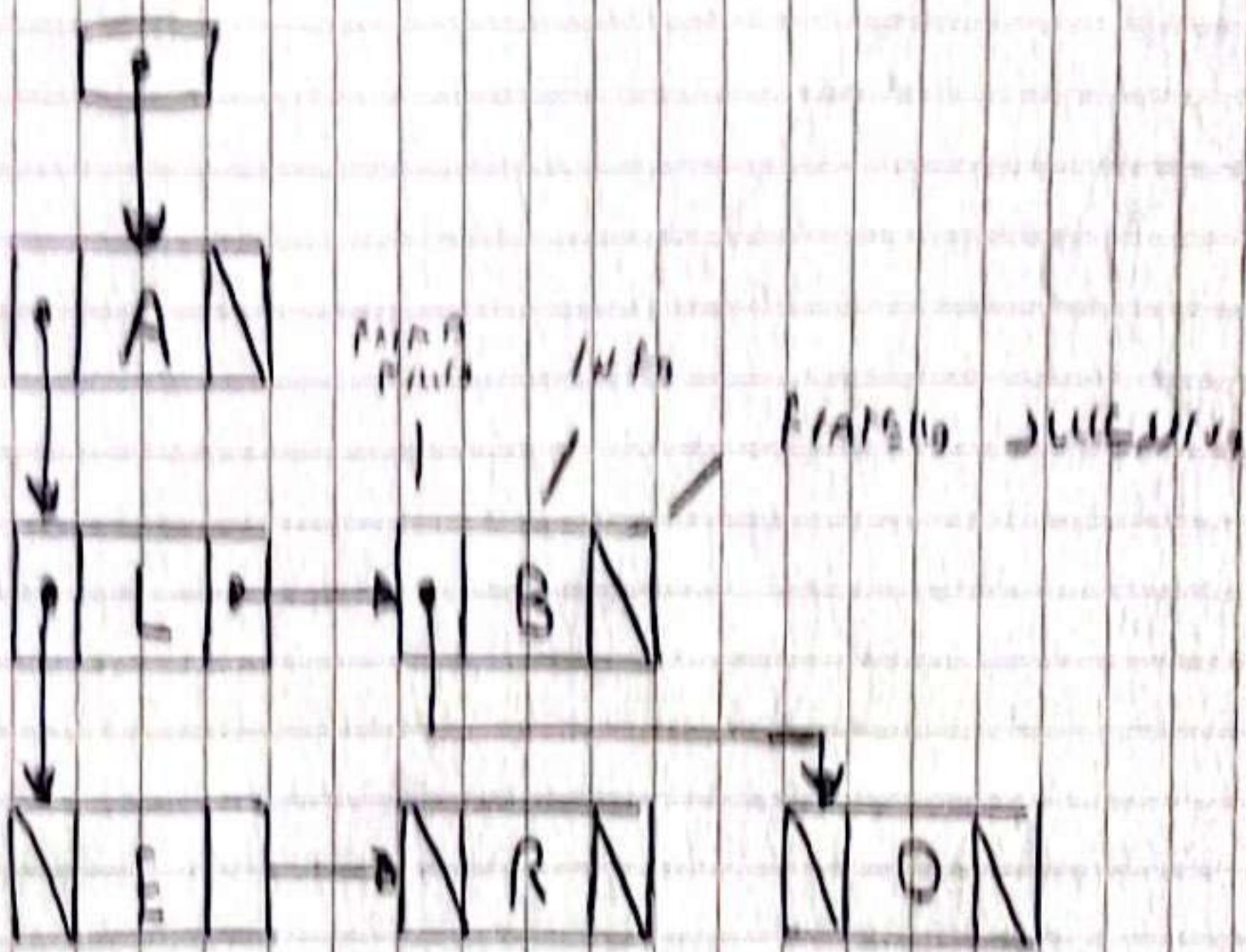
MAPAS DE ALBERTO VISITA DE ALBERTO LENIENTE

IMPLEMENTACIONES CON PERMUTACIONES DE FRECUENCIAS (MAS CON NUMEROS SIMULANDO PIR FREQ)



IMPLEMENTACIONES CON PERMUTACIONES ALFICEL (MAS CON NUMEROS ALGORITMO DE FRECUENCIA)





Graph

Un Grafo $G = (V, E)$ consiste en:

- Un conjunto V de vértices (o nodos)
- = Un conjunto E de aristas de vértice, siendo cada una o simple, o múltiples vértices

$M = \text{numero vértices}$, $m = \text{numero de aristas}$

Bajo un grafo $G = (V, E)$ se entiende un par (x, y) si $x, y \in V$.

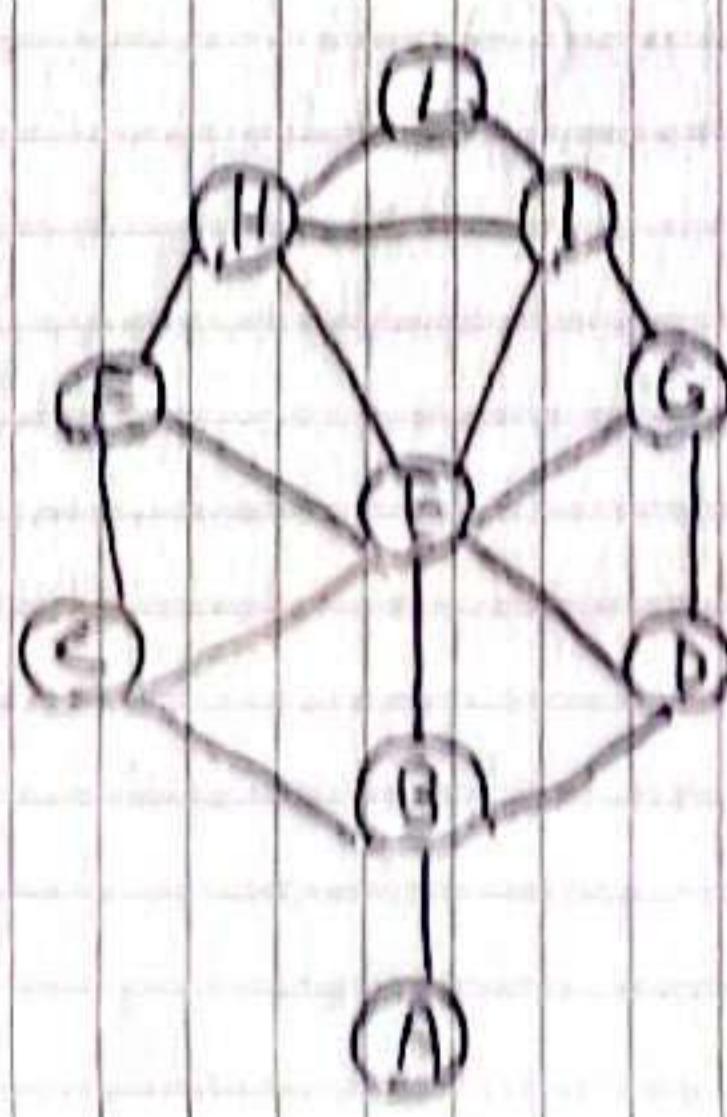
$x \in V$ se dice que x es vecino de y o viceversa.

Se $G \in V$ se dice que G es completo si para todos $x, y \in V$ se cumple que $x \sim y$.

Si $V = \{x, y\}$ se dice que V es completo si $x \sim y$.

Si G es un grafo completo, se le dice completo.

Si G es un grafo no completo, se le dice incompleto.



IL GRAFO D'UN VERTICE, CONSIDERATO CON IL PESO AL NUMERO DI ANCI
DISEGNATI IN ES.

PER ESEMPIO NEL DISEGNO DEL PRIMO VERTICE GRAFO 4: $\delta(i) = 4$

SIA G UN GRAFO ORDINATO:

- IL GRAFO IN INGRESSO DI UN VERTICE V È PARI AL NUMERO DI ANCI, CHE
SARÀ IN V È SI INDICA CON $\delta_{in}(v)$

- IL GRAFO IN USCITA DI UN VERTICE V È PARI AL NUMERO DI ANCI CHE
ESCE DA V È SI INDICA CON $\delta_{out}(v)$

- $\delta(v) = \delta_{in}(v) + \delta_{out}(v)$ È IL CAMPO DEL VERTICE V

- UN CAMMINO IN UN GRAFO È UNA SEQUENZA DI VERTICI $[v_0, v_1, \dots, v_m]$ CHE
DA UN VERTICE A UN ALTRO È POSSIBILE CONSEGUENTI REGOLE SEQUENZA (v_i, v_{i+1}) È CORRETTA
PER OGNI ANGOLO E PER TUTTI OGNI ANGOLI (v_i, v_{i+1}) SONO ESISTENTI TRA LORO.

- UN CAMMINO È UNO JGMNIO SE TUTTI I VERTICI $[v_0, v_1, \dots, v_m]$ SONO
DISTINTI: AL PIÙ DEL PRIMO E DELL'ULTIMO SONO LIMITI.

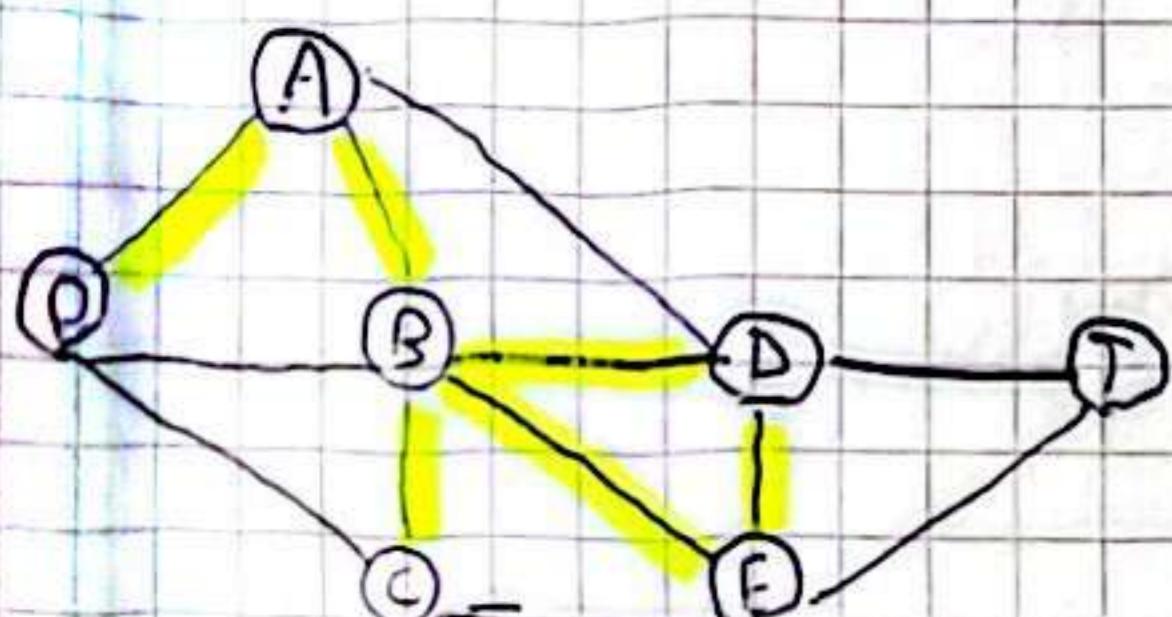
- UN CAMMINO SEMPRE IN CUI IL PRIMO È CHIUSO VERTICE SONO CONSECUTIVI
E SONO CHIUSI.

GRAFI ESTRAZIONI

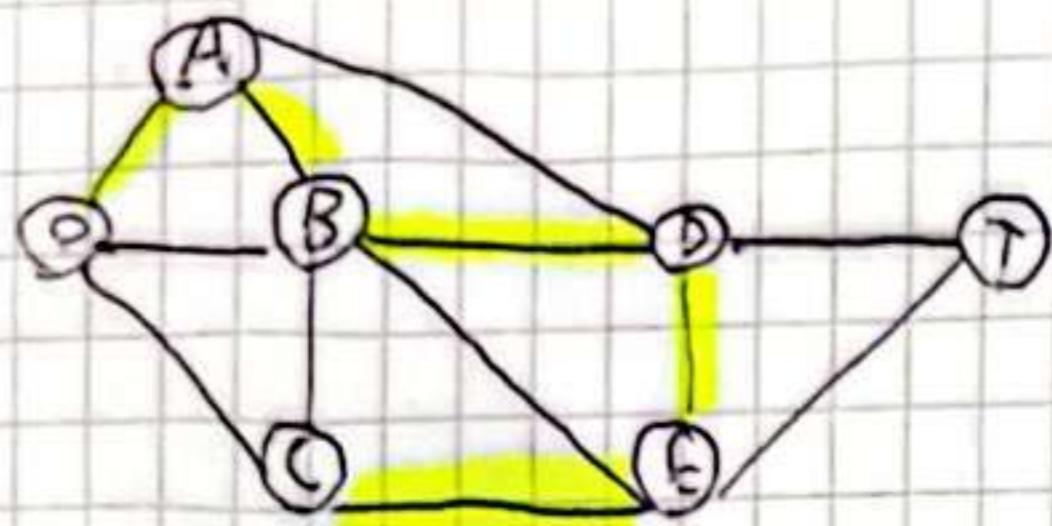
I GRAFI ESTRAZIONI SONO GRAFI IN CUI OGNI VERTICE SONO ASSOCIAZI.

ESTRAZIONE MENO ACCI ANCHE POSSONO ESSERE ASSOCIAZI DEI VERTICI
NUMERICI DENITI PESI.

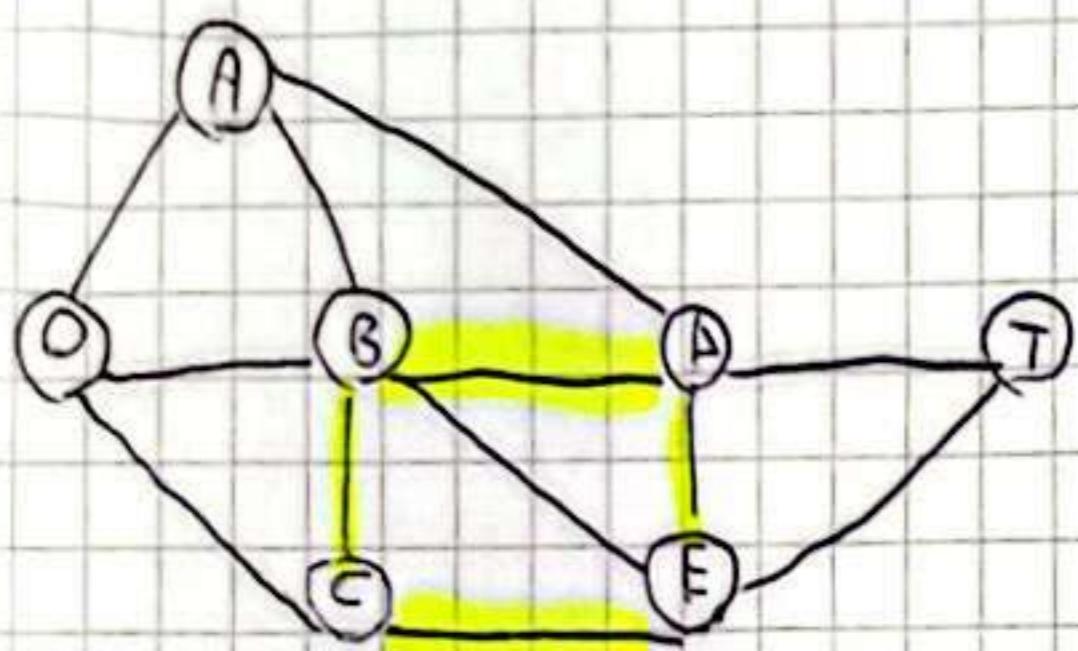
GRAFO NON ORDINATO CAMMINI E CICLI



(O, A, B, D, E, B, C) È UN CAMMINO



(D, A, B, D, E, C) è un cammino senza anelli.



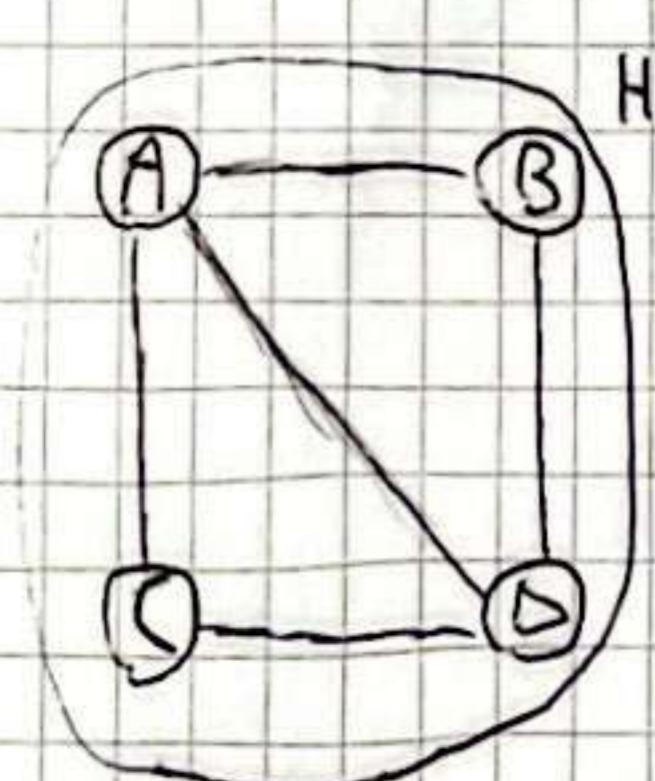
(B, D, E, C, B) è un cammino semplice.

Anello vuoto.

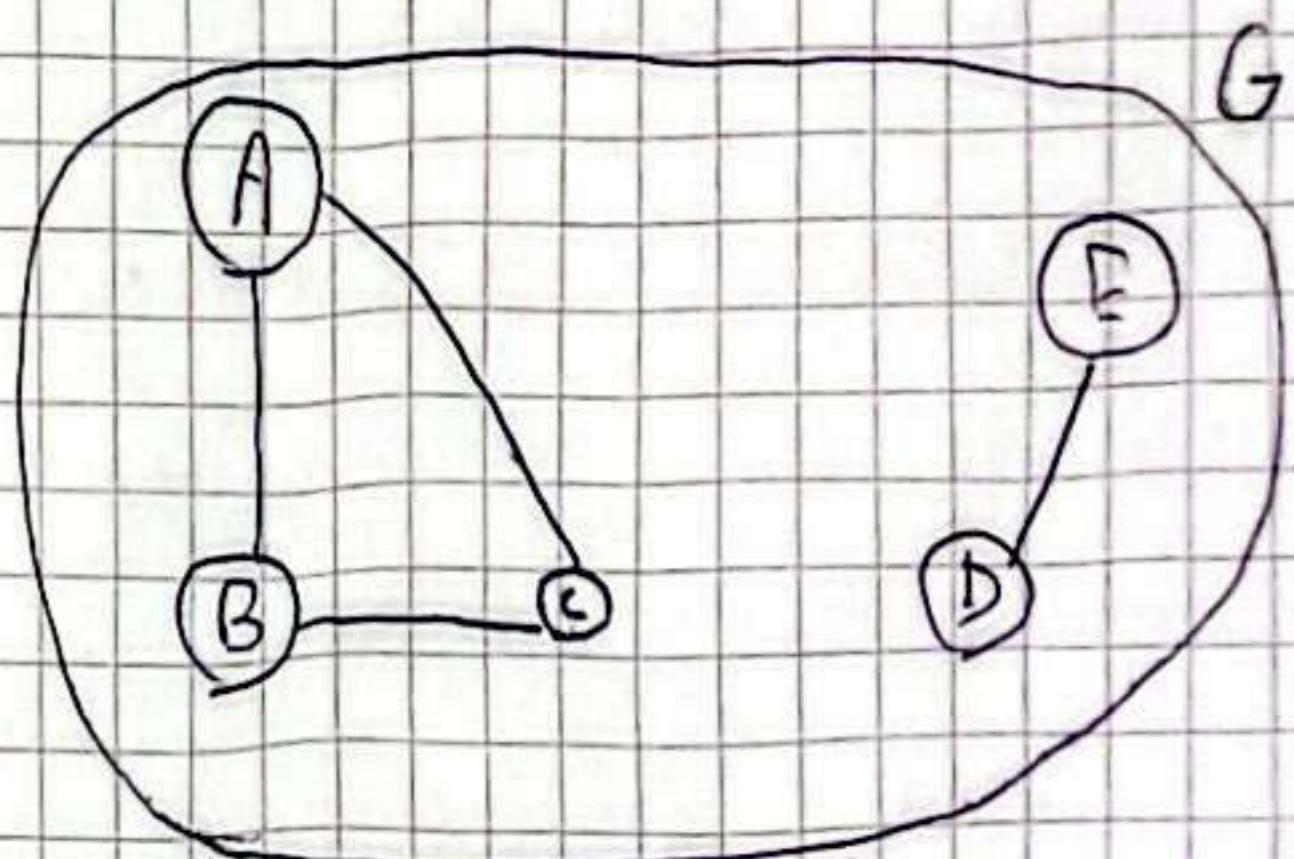
GRAFI DI CONNESSIONE

Un grafo non orientato $G = (V, E)$ è connesso se e solo se per

ogni coppia (v, w) di vertici di G esiste un cammino che li unisce.



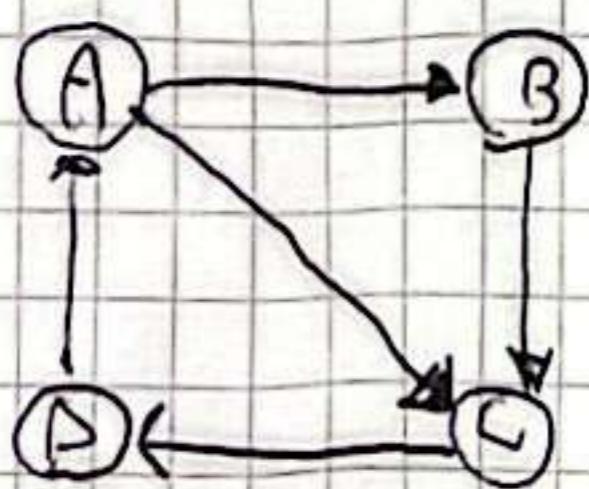
grafo connesso



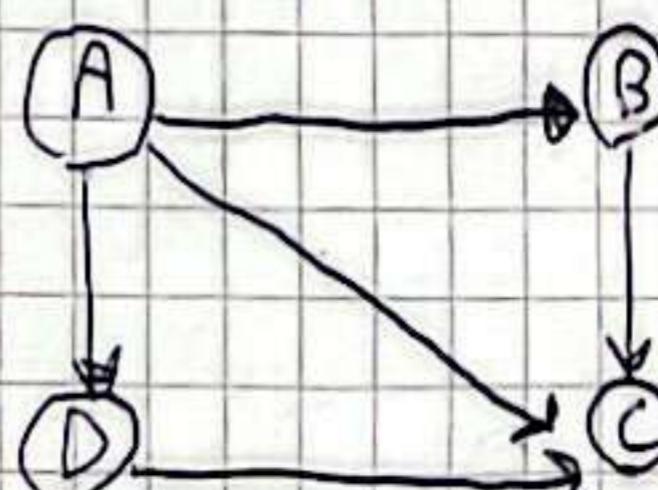
grafo non connesso

GRAFI ORIENTATI CONNESSIONE

Un grafo orientato G è **fortemente connesso** se esiste una strada da ogni vertice v di G ad ogni vertice v' di G .



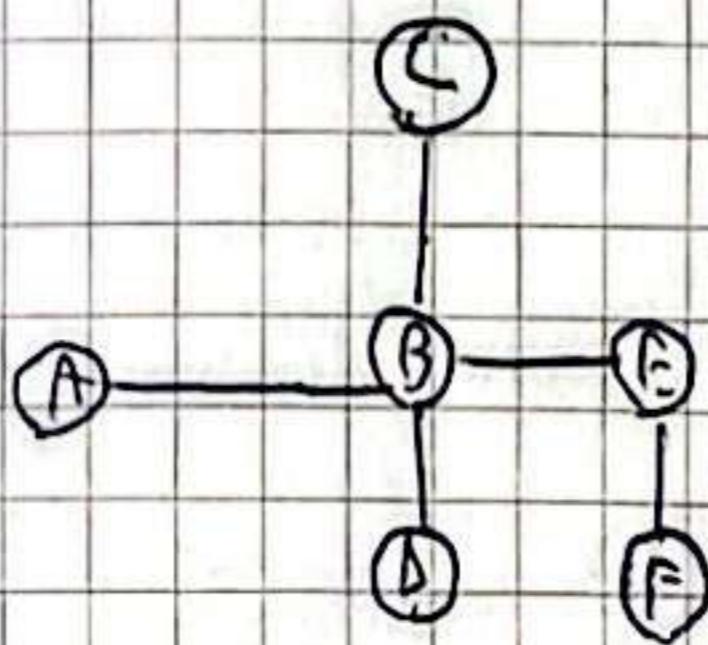
grafo forte
connesso (orientato)



grafo orientato non
fortemente connesso
(nesun cammino da B,C,D a E)

Un Albero, chiamato anche **grafo non orientato**, consiste di un circuito.

A differenza di un albero numerato, un albero chiamato non è possibile.



Proprietà:

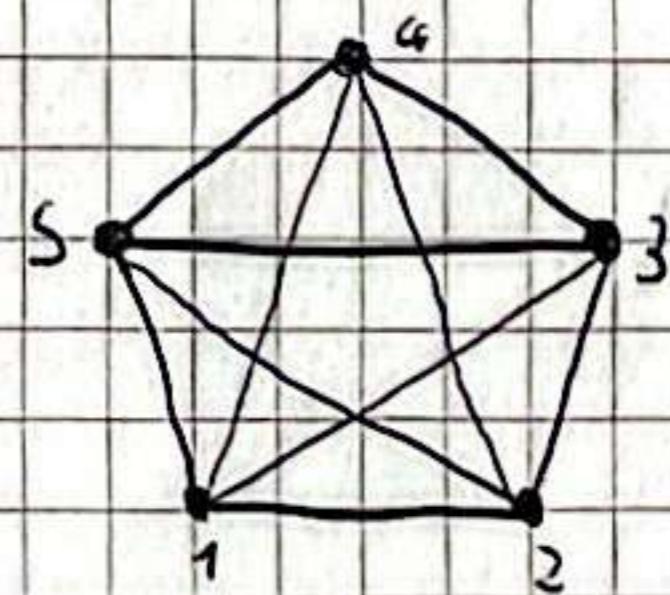
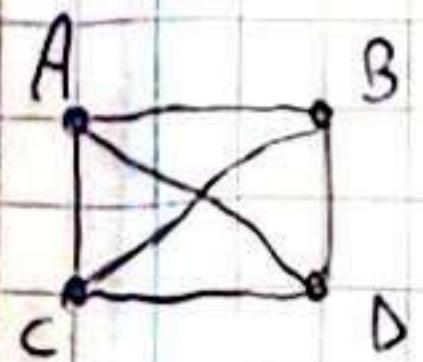
- Se muoviamo un arco da un albero chiamato, il grafo risultante è sempre连通的 (connesso).
- Se in un numero di vertici di un albero chiamato n , allora ha esattamente $n-1$ archi.

GRAFO COMPLETO NON ORIENTATO

Un grafo non orientato G , in cui ogni vertice è connesso da tutti gli altri.

Allora occorre avere almeno $m-1$ vertici, è detto **grafo completo**.

Un grafo non orientato completo con m vertici ha esattamente: $m(m-1)/2$ archi.

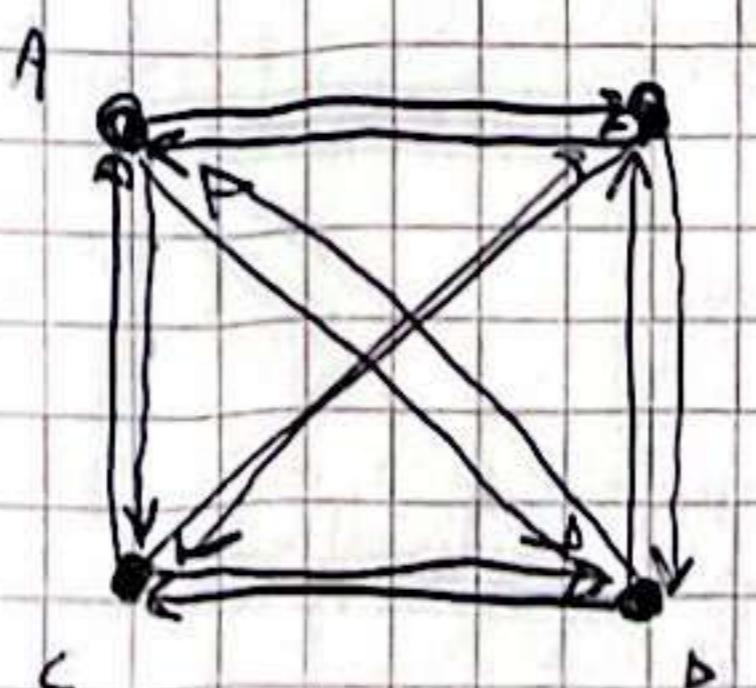


GRAFO COMPLETO ORIENTATO

Un grafo orientato G , in cui ogni vertice è connesso da tutti gli altri.

Ovvero se i restanti $m-1$ vertici è detto completo.

Un grafo completo orientato ha esattamente $m(m-1)$ archi.

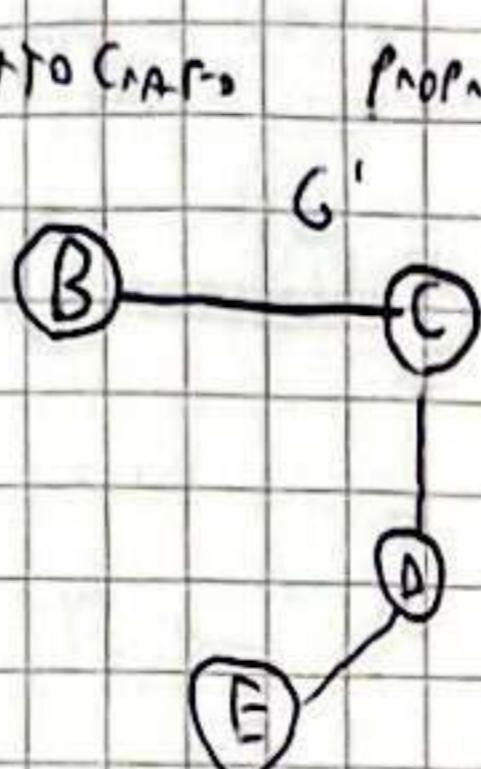
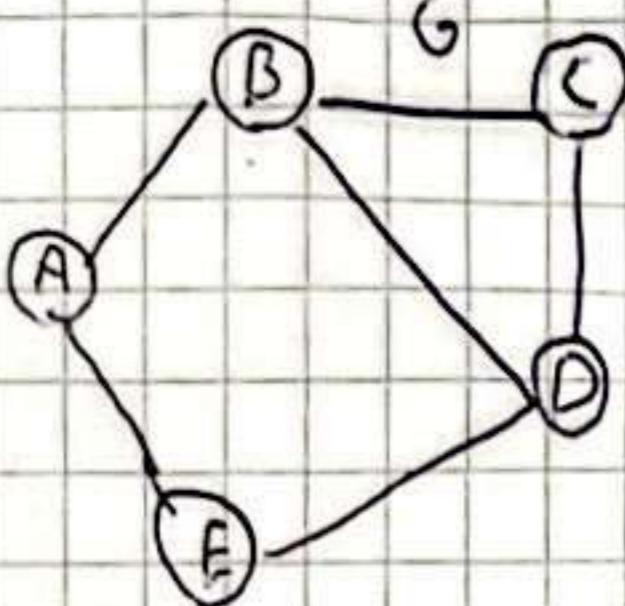


DEFINIZIONE
Un sottografo di un grafo $G = (V, E)$ è un grafo $G' = (V', E')$ tale che $V' \subseteq V$ e $E' \subseteq E$

Un sottografo G' è detto un sottografo proprio di G se $G' \subsetneq G$.

Considerare per G

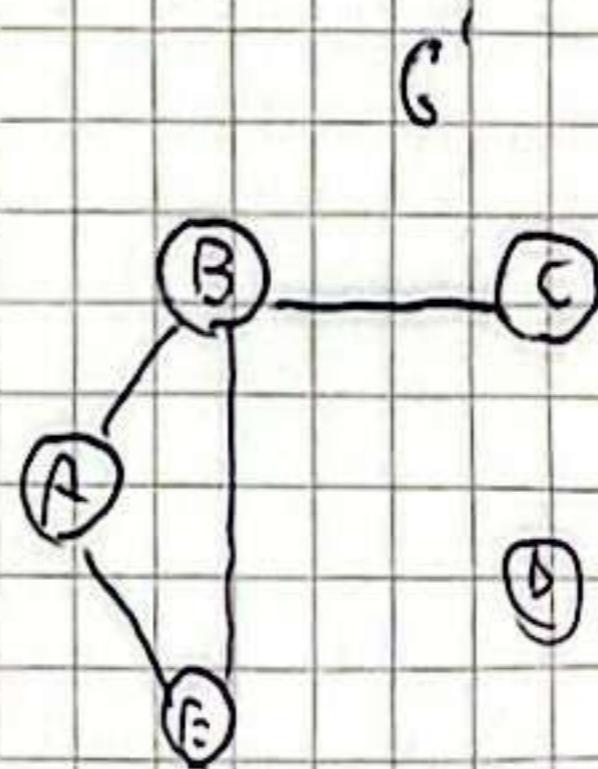
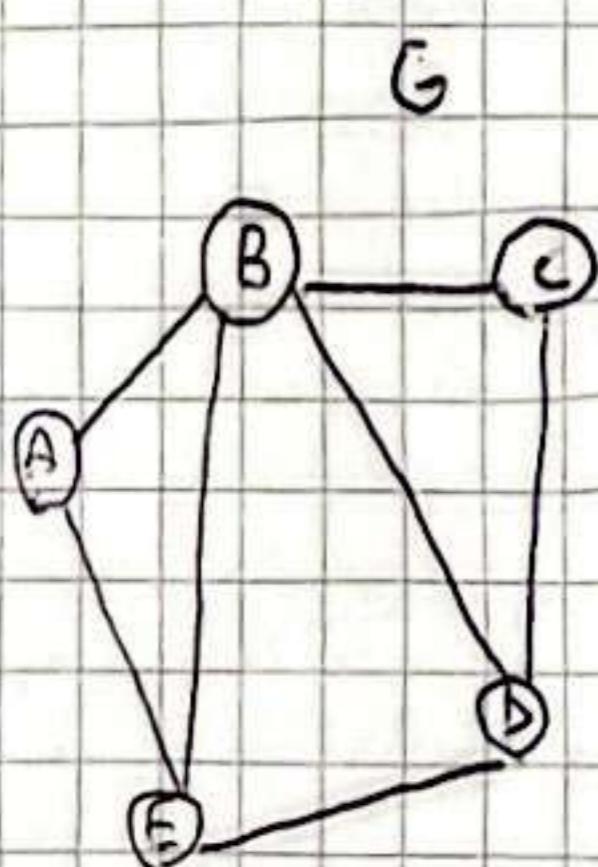
ESEMPIO: Un grafo G è un sottografo proprio di G se G



Sottografo di ricoprimento
(SPANNING GRAPH)

Un sottografo $G' = (V', E')$ di un grafo $G = (V, E)$ tale che $V' = V$ è detto sottografo di ricoprimento di G .

ESEMPIO: G' è un sottografo di ricoprimento di G



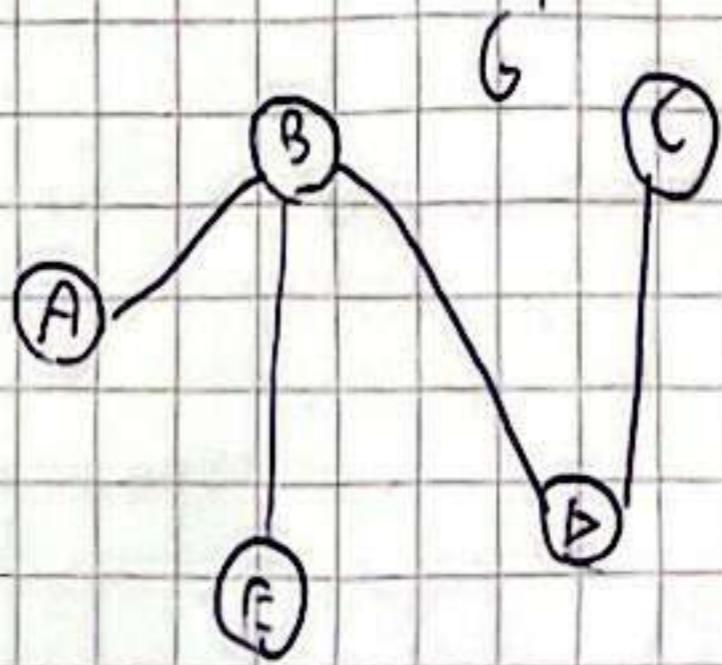
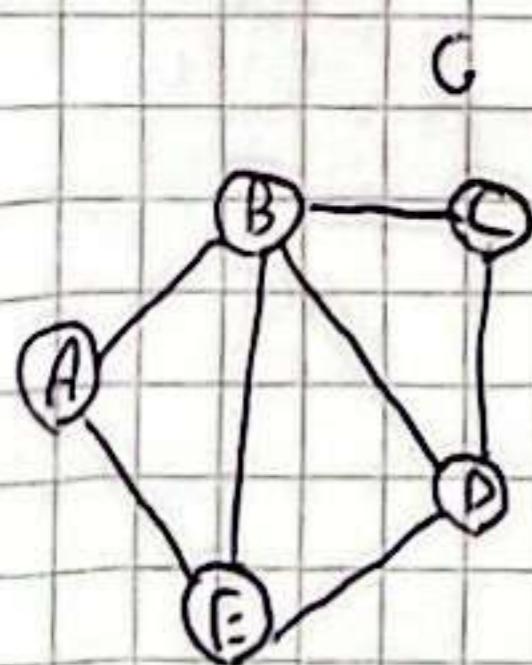
Albero di ricoprimento
(SPANNING TREE)

Un sottografo di ricoprimento di un grafo $G = (V, E)$ tale che G

è un albero chiamato G' detto albero di ricoprimento di G .

ESEMPIO:

G' un albero di ricoprimento di G



G = (V, E) è un grafo G' = (V', E') nuovo
 $\subseteq E$
 un sottografo proprio di G è G' now

un sottografo proprio G' di G

ALGORITMO
 GRAFO

GRAFO G = (V, E) TALE CHE V' = V E' DEUTTO

ALGORITMO SUL GRAFO G

ALGORITMO
 TESI

(V, E)' DI UN GRAFO G = (V, E) TALE CHE G'

G' È UN ALGORITMO SUL G

DEL GRAFO G

OPERAZIONI SUL GRAFO PER OPERAZIONI
 E' UNA ELIMINAZIONE

ADDVERTEX(D, G)

ADDEDGE(a, c, w, G)

REMOVEEDGE(B, C, G)

REMOVEVERTEX(D, G)

STRUTTURE DATI PER
 GRAFI PIÙ FATTURE

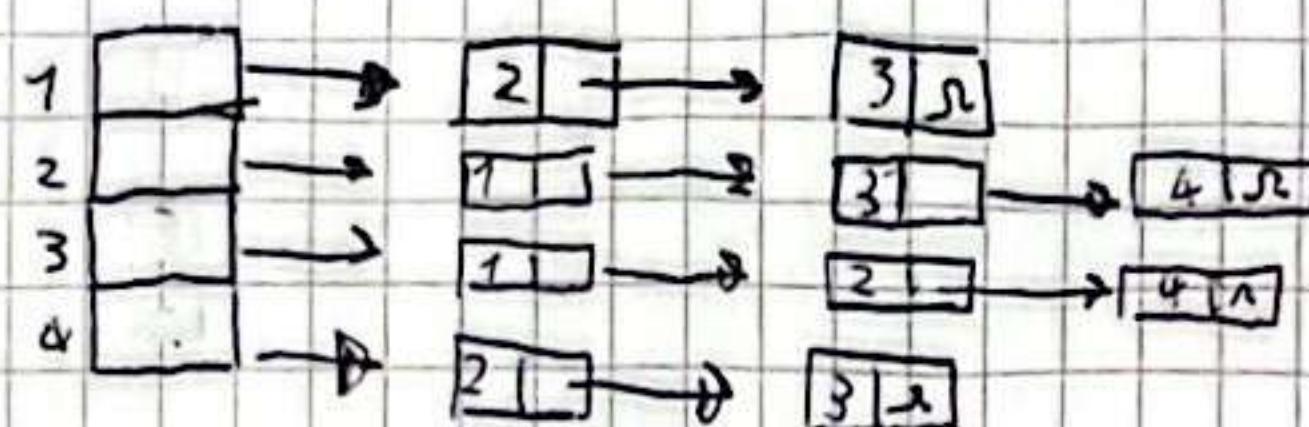
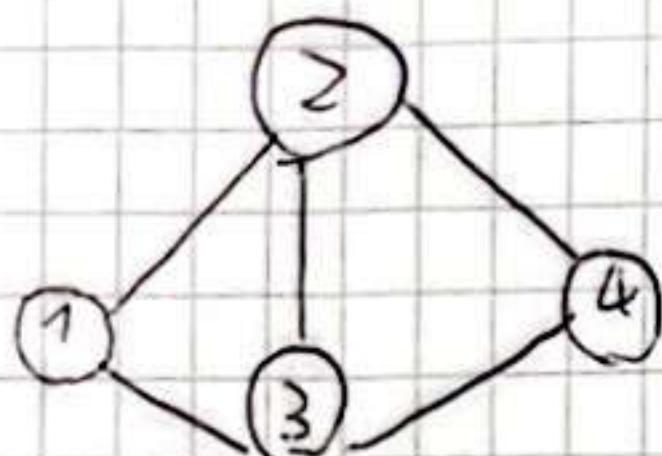
- LISTA DI ADJACENZA

- VERTICI MEMORIZZATI IN UN ARRAY
- VERTICI MEMORIZZATI IN UNA LISTA

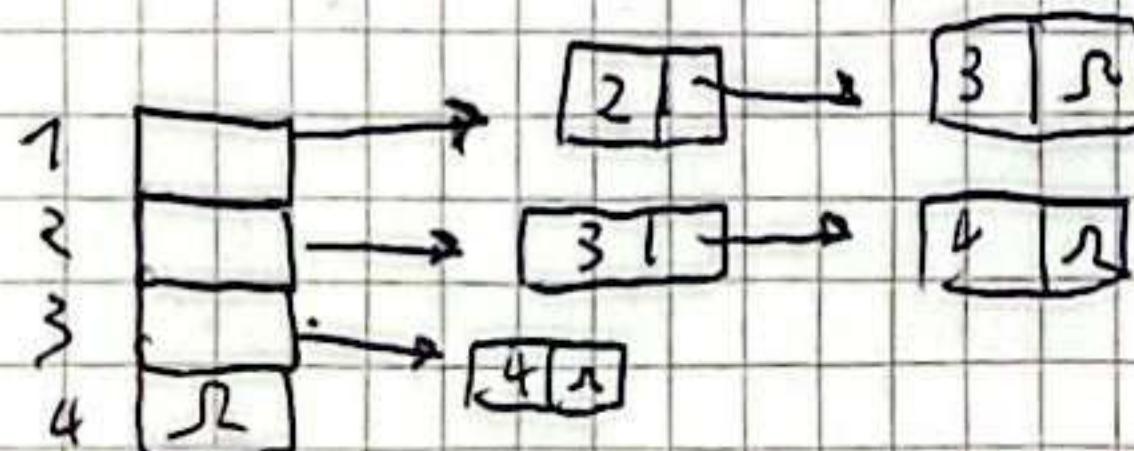
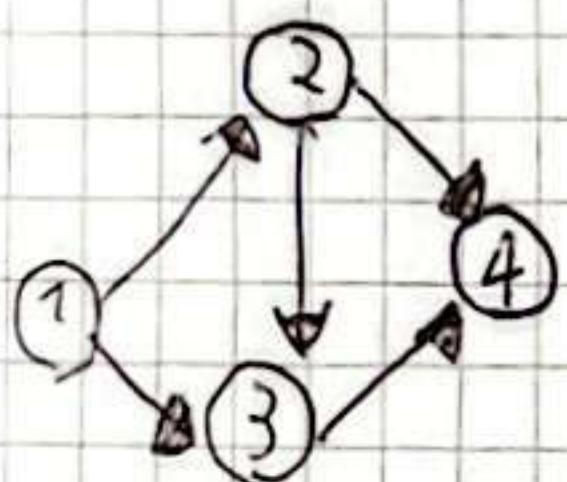
- MATELLA DI ADJACENZA

LISTA DI ADIACENZA

OGLIUMORE: SENZIALE



IN UN GRAFO SONO ORGANIZZATI UN ANCO E' RAPPRESENTATO DUE VOLTE



IN UN GRAFO DIRETTO, UN ANCO E' RAPPRESENTATO UNA VOLTA

E - Ogni Anco nel Grafo G e' definito da

- 1) Un elemento nelle liste di adiacenza, se G e' un grafo orientato
- 2) Due elementi (corrispondenti ai suoi due vertici estremi) nelle liste di adiacenza se G e' non orientato.

- Se m il numero dei vertici ed n il numero di archi

- Le liste di adiacenza costituiscono un totale di:

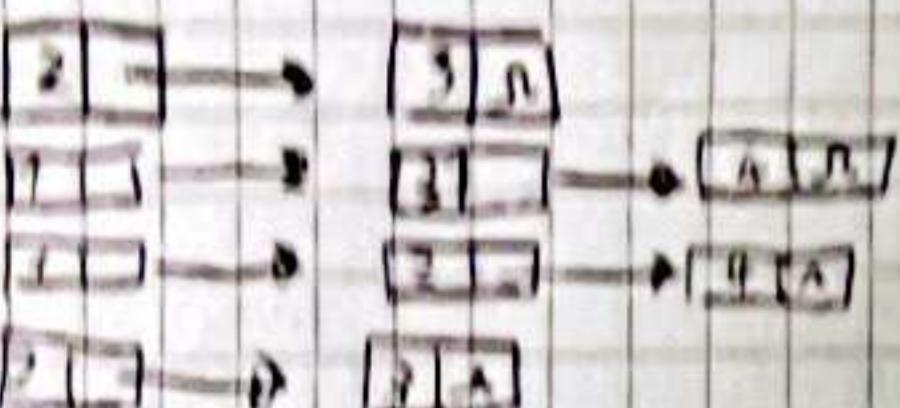
- m elementi se G e' orientato
- $2m$ elementi se G non e' orientato

- Se ogni lista di adiacenza e' implementata come una lista singola:

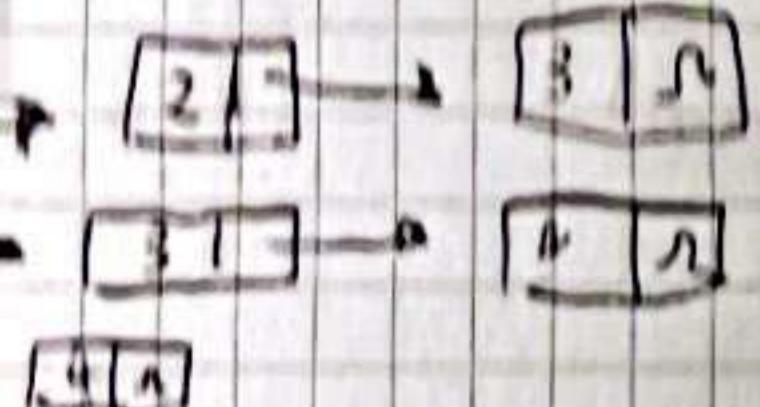
E - Collezione, struttura dati associativa:

- $m + m$ "blocki" per un grafo orientato
- $m + 2m$ "blocki" per un grafo non orientato

digitata
funzione



• Anche l'implementazione DUE VETTORI



• E' rappresentato, una lista, vettore

DEFINIZIONE

• ADJACENZA, SE G E' UN GRAFO ADJACENZA

Per ogni due vertici esiste una lista di

• IL NUMERO DI ARCHI.

• UN VETTORE DI

• ELEMENTI

NON E' obbligatorio.

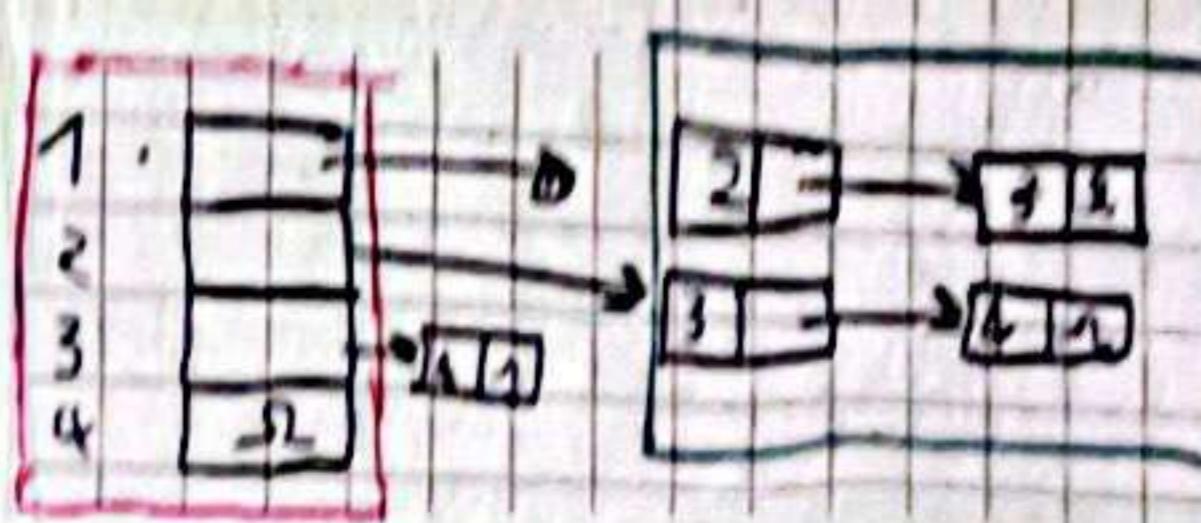
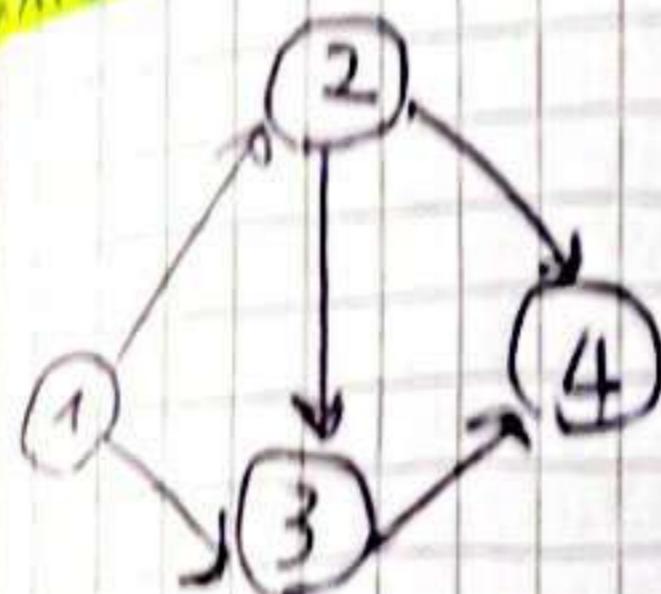
• IMPLEMENTAZIONE CON UN GRAFO SIMMETRICO

SE:

• GRAFO SIMMETRICO

• LISTA NON SIMMETRICA

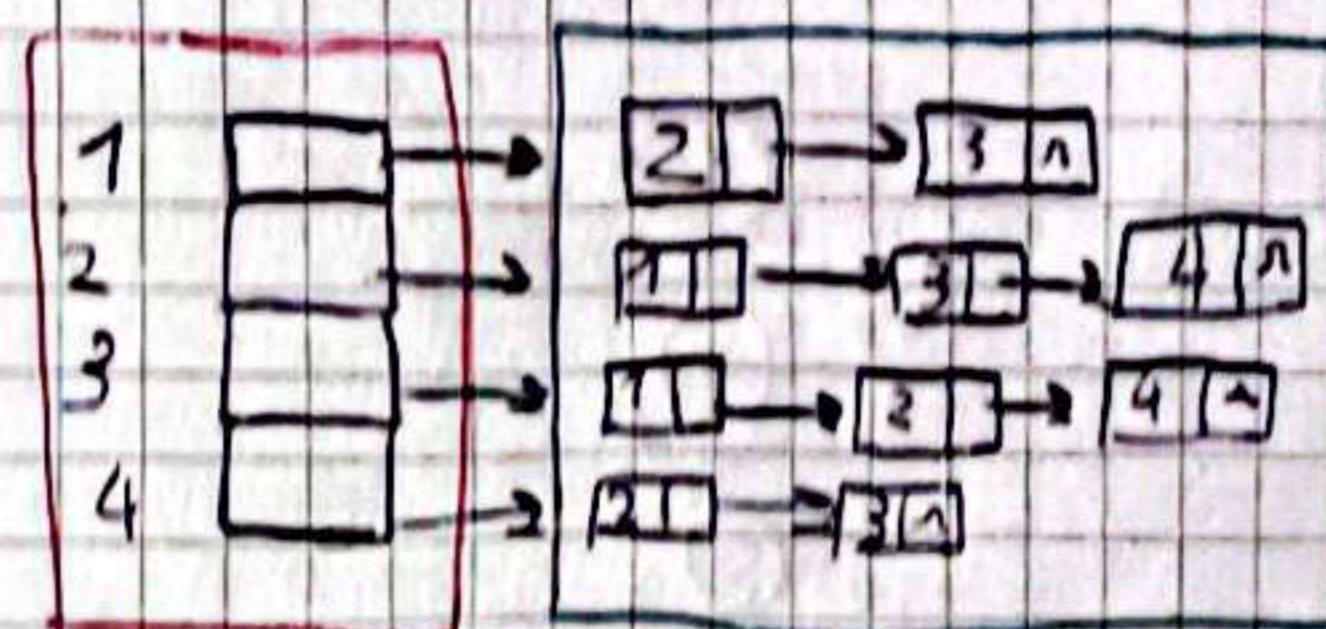
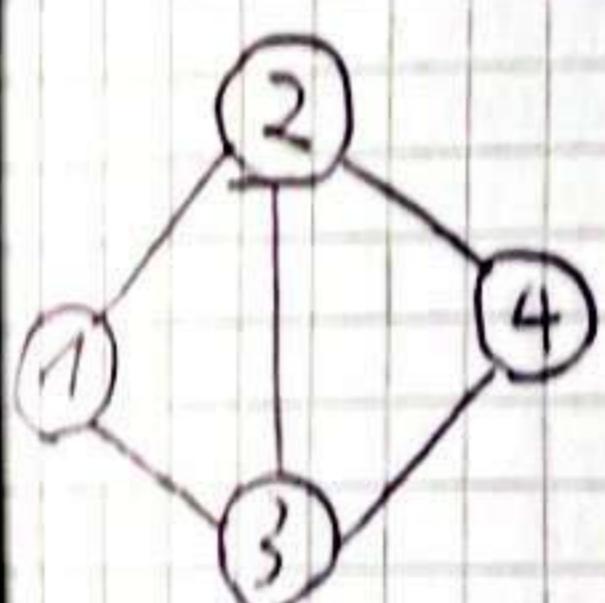
• ADJACENZA



M ELEMENTI PER RAPPRESENTAZIONE / MOD

M ELEMENTI NELLA LISTA DI ADJACENZA

• ADJACENZA



LISTA DI ADJACENZA COM

• VERTICI MEMORIZZATI IN PARTE

SIA $G = (V, E)$ UN GRAFO COM M VERTICI ED M ARCHI

• CONSIDERANDO I MIGLIORI IMPLEMENTAZIONI AD AMM per i VERTICI

- UN AMM CONTIENE UN ELEMENTO PER OGNI VERTICE v DI G

- Ogni ELEMENTO DEL VERTICE E' UNA LISTA DI VERTICI, OVRRA ALTRI VERTICI

DEI VERTICI DI G ADJACENTI AD v .

- LA LISTA DI ADJACENZA DI v CONTIENE $E(v)$ ELEMENTI.

- NELL'IMPLEMENTAZIONE CON ADJACENZA GLI ELEMENTI SONO I VERTICI DELLA LISTA

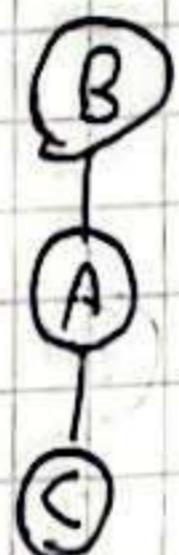
MA SOLO VERTICI ADJACENTI

VISITA A FJ:

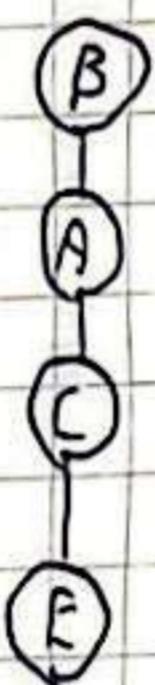
a) Partiamo da S, visitiamo A da B si ottengono:



b) Visito C partendo da A si ottengono:



c) Visito E partendo da C si ottengono:



d) Visito G partendo da E si ottengono:



e) Visito F partendo da G si ottengono:



AVANTAGGIO

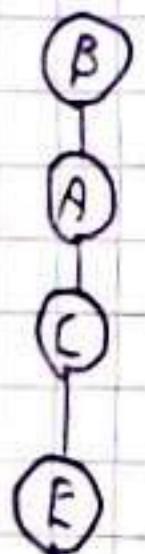
VISITIAMO A DA B E OTTERIAMO:



DA A E OTTERIAMO:



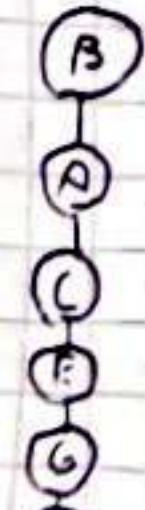
DA C E OTTERIAMO:



DA E E OTTERIAMO:



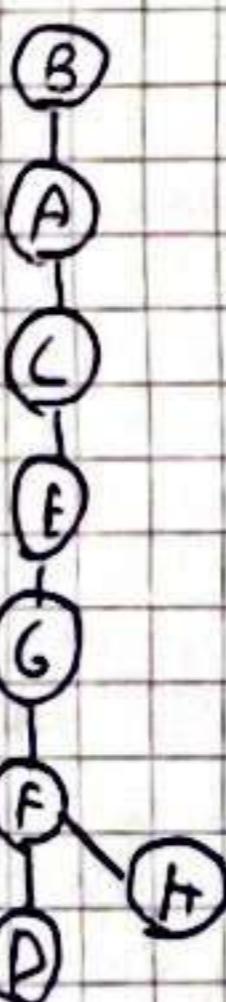
DA G E OTTERIAMO:



F) DA
SCELTA DI VISITARE D. OTTERIAMO:

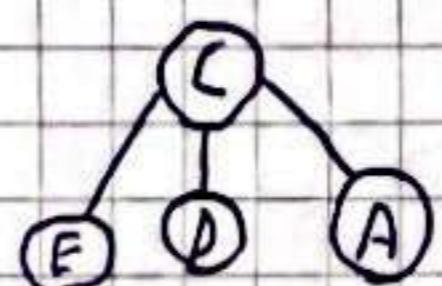


6) PIANO DA D NEL CASO PIÙ VISITARE NEGLI ULTIMI NODI IN QUANTO C ED E SONO GRA' MOLTI, QUINDI TUTTI AD F E DA F POSS. VISITARE ULTIMO ROSSO, OVVERO H:

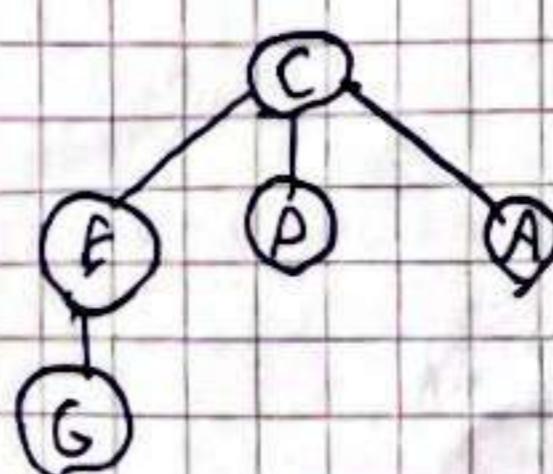


VISITA BFS:

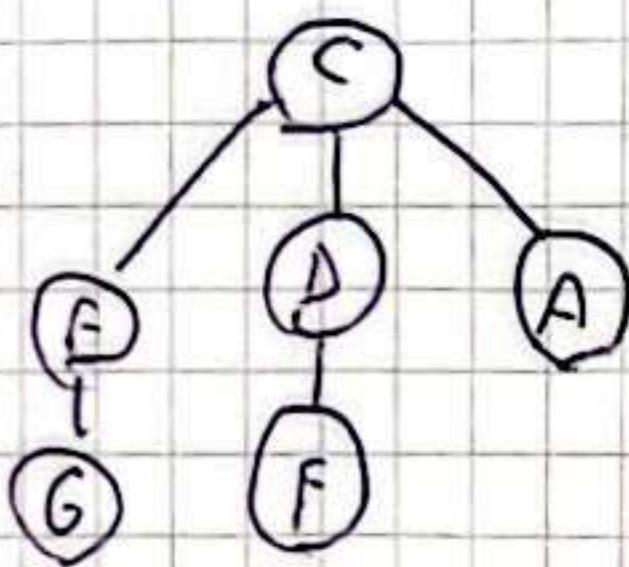
7) DA C POSS. VISITARE E, D ED A QUINDI:



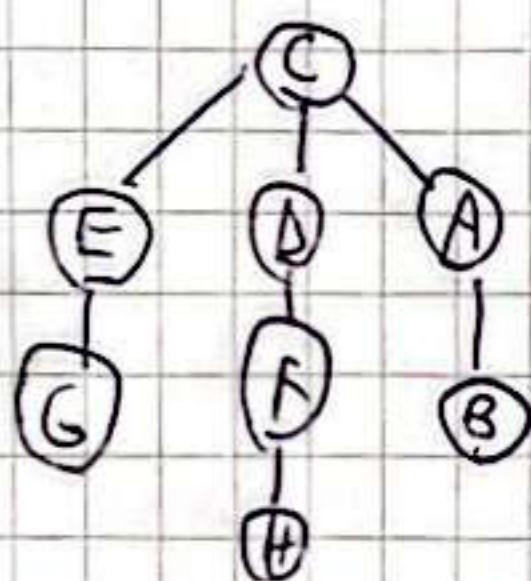
8) DA E POSS. VISITARE SOLO G QUINDI:



C) DA G NON POSSO VISITARE NODI IN QUANTO SONO
MANATI, ALLORNO A D. DA D POSSO VISITARE F QUINDI:



D) DA F VUOTO H E INFINE DA A VUOTO B

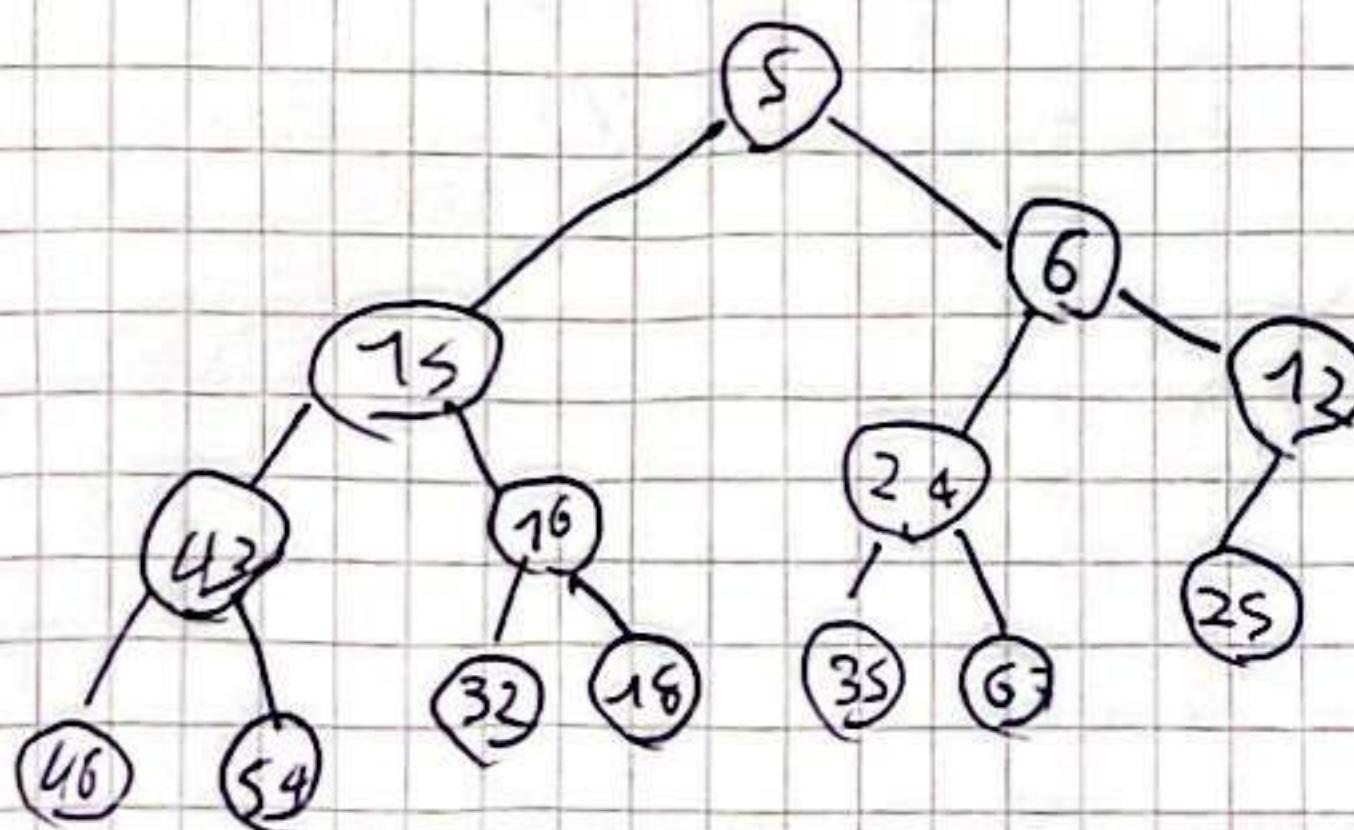


DOMANDA: E' RAMA HEAP BINARIO

ESERCIZIO 7;

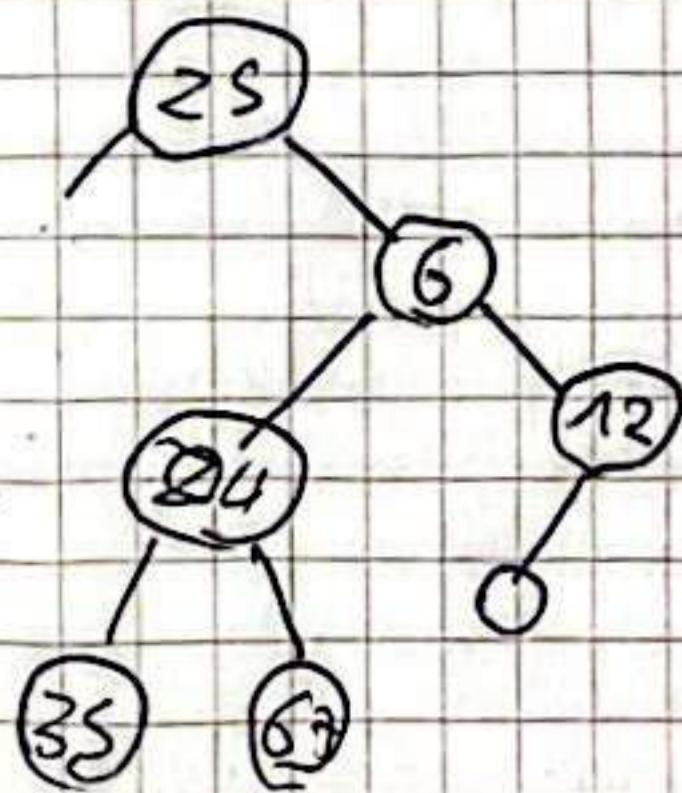
1) NUOVO NODO VIENE RICONTA COME FOGLIA NEL PRIMO POSTO

2) POSSIBILE OLTRE COME FIGLIO SINISTRO DI 25, MA C, PERCHE'
LA PROFONDITA' BFTC SISTEMA I NODI, QUINDI IL NUOVO NODO VIENE
FATTO "VIRGULATO" FINO ALLA RADECE. ($25 \rightarrow 12 \rightarrow 6$)



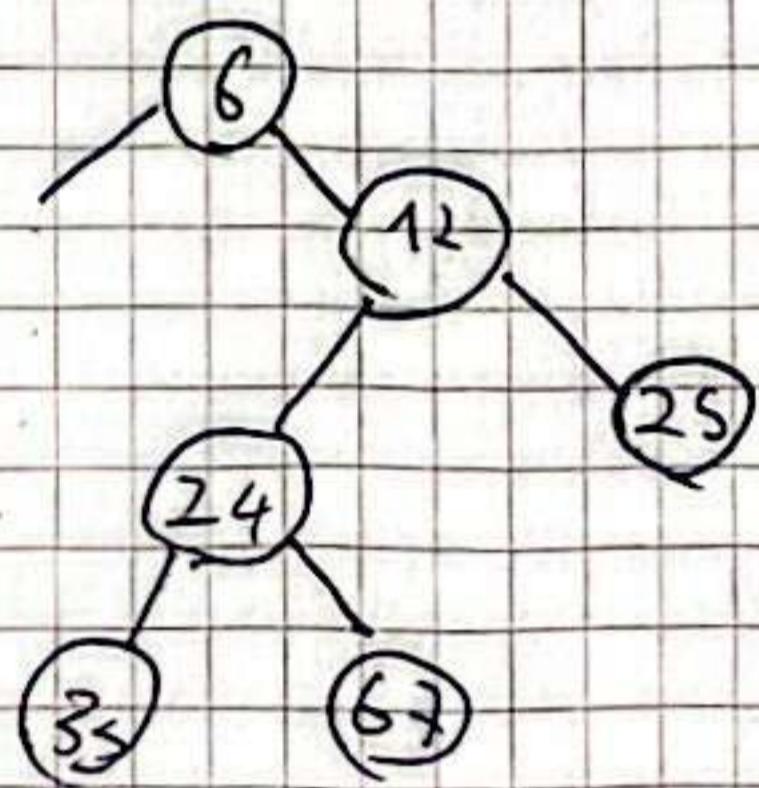
DATO CAE DOBBIAMO MAINTENERE L'HEAP PROPRIETÀ NEL CASO PERICOLO
PER L'INSEGNAMENTO CHE VALE $\Theta(\log m)$

110 NODO 5 E JWAPPO CON L'ULTIMO (2S)



120 CA PROBLEMI MUOVI BAJSO (J, CONVERGIR 2S CON 1S E JWAPPO CON IL MINORE (6))

CONTINUO A SORTEARE IL NODO 2S CONVERGIRI CON I SUOI FIGLI
FINE, L'ALBERO E' DI NUOVO UN ALBERO DI ARRICCHIMENTO PER IL PROBLEMA.



A PIANO SINISTRA RIMANE UCILE.

IMPLEMENTAZIONE: LISTA DI GRAFI

TAD: GRAFO NON ORIENTATO CON VERTICI, ETICHETTATI E ANCII PESATI

STRUCT

ABBIAMO 2 TIPI DI STRUCT:

- 1) "MEZZO ARCO", HA IL PUNTIATORE DEL VERTICE DI DESTINAZIONE E PESO DELL'ARCO, TIENE INOLTRE IL PUNTIATORE AL PROSSIMO MEZZO ARCO NELLA LISTA DI ADIACENZA
- 2) VERTICE, ABBIAMO: ETICHETTA DEL VERTICE, PUNTIATORE ALLA LISTA DEI "MEZZI ANCII" ADIACENTI, PUNTIATORE AL PROSSIMO VERTICE E BOOLEAN PER VERIFICARE SE IL MONDO E' STATO VISITATO.

1)

STRUCT HALFEDGEVERTEX {

LABEL LABEL; // ETICHETTA VERTICE DESTINAZIONE

VERTEXNODE* VERPAT; // PUNTIATORE AL VERTICE

WEIGHT WEIGHT; // PESO DELL'ARCO

HALFEDGEVERTEX* NEXT; // MEZZO ARCO SUCCESSIVO

};

2)

STRUCT GRAPH: VERTEXNODE {

LABEL LABEL; // ETICHETTA VERTICE

HALFEDGEVERTEX* ADJLIST; // PUNTIATORE LISTA MEZZI ANCII ADIACENTI

VERTEXNODE* NEXT; // PROSSIMO VERTICE

BOOL IS VISITED; // BOOLEANO PER VERIFICARE SE IL MONDO E' VISITATO

};

ADD VERTEX

IL PRIMO CONTROLLO A FAR E' QUELLO DI VERIFICARE SE IL GRAFO E' Vuoto, NEL CASO SI GRAFO VUOTO AGGIUNGERE UN NUOVO VERTICE E' FA' CREATO UN NUOVO NODO CHIAMATO: VERTEXNODE* AUX = NEW VERTEXNODE;

PER TUTTI GLI ARCI CHE DOBBIANO FINIRE IN UN VERTICE, VERIFICA SE IL MONO

PRESENTE, SE NON FAISCE SETTORE GRADO, INSERISCE IN CODA.

AHO EDGES

LA FUNZIONE SI OCCUPA DI INSERIRE UN ARCO TRA DUE VERTICI.
CONFIGURA UN PUNTATORE SG A DUE NODI ESISTENTI, PER FARLO CREARE DUE PUNTI

E GLI ASSEGNA A NULLPTR.

PER CONTROLLARE SE G E SG SONO VERSO IL MATESTO \rightarrow LABEL == FROM ACCORDA
NODO1 = CUR, METTERE CUR CON NODO2.

SE UNO DEI DUE (P ENTRANTI) SONO ARCI DA NULLPTR RETURN FALSE, SENZA
DUE CONTROLLARE SE ESSE SONO GI' UN ARCO;

HALFEDGEVERTEX * ARCO = NODO1 \rightarrow ADJUST;

WHILE (ARCO != NULLPTR) {

IF (ARCO \rightarrow LABEL == NODO2 \rightarrow LABEL) {

} RETURN FALSE;

ARCO = ARCO \rightarrow NEXT;

5

SE NON ESISTE NODI POSSO PROCEDERE CON LA RICORSIONE:

HALFVERTEXNODE * ARCO1 = NEW HALFVERTEXNODE;

ARCO1 \rightarrow LABEL = NODO2 \rightarrow LABEL

ARCO1 \rightarrow VERTICE = NODO2;

ARCO1 \rightarrow WEIGHT = W;

ARCO1 \rightarrow NEXT = NODO1 \rightarrow ADJUST;

NODO1 \rightarrow ADJUST = ARCO1;

STESSO PROCEDIMENTO CON ARCO2.

NUM EDGES

QUESTA FUNZIONE HA IL COMITO DI RITORNARE IL NUMERO DI ARCI DEL GRAFO.

PER FARLO ABBIAMO BISOGNO DI DUE CICLI, UNO PER CUI != NULL
E L'ALTRO PER L'ARCO, ASSEGNIAMO HALFEDGEVERTEX ARCO = CUR \rightarrow ADJUST
ALL'INIZIO DEL PRIMO CICLO, ALLA FINE DELLA FUNZIONE ESSESSE COUNT / 2 IN

CLEAR

WHILE ($G \neq \text{NULLPTR}$) {

VERTEXNODE \times TEMP = G;

G = G \rightarrow NEXT;

HALFEDGE VERTEX \times ADJ = TEMP \rightarrow ADJLIST;

WHILE (ADJ != NULLPTR) {

HALFEDGE VERTEX \times TEMPADJ = ADJ;

ADJ = ADJ \rightarrow NEXT;

DELETE TEMPADJ;

}

DELETE TEMP;

{

G = EMPTYGRAPH;

LISERA LA LISTA DI ADJLIST
DEL VERTICE CORRENTE.

PRINT

VERTEXNODE \times CUN = G;

WHILE (CUN != NULLPTR) {

COUT << CUN \rightarrow LABEL << " : ";

HALFEDGE VERTEX \times ADJ = CUN \rightarrow ADJLIST;

BOOL FIRST = TRUE

SENZA PUNTI VEDI FINE SE SI TROVA BSI = TRUE

WHILE (ADJ != NULLPTR) {

IF (! FIRST) {

COUT << ", ";

{

COUT << " " << ADJ \rightarrow LABEL << "(" << ADJ \rightarrow WEIGHT << ")"

FIRST = FALSE;

ADJ = ADJ \rightarrow NEXT;

E

{

COUT << ENDL;

CUN = CUN \rightarrow NEXT;

{

SWAP

FUNZIONE AUSILIARE, MI SERVE PER SOSTituIRE DUE ELEMENTI:

```
VOID SWAP(PRIORITYQUEUE::PriorityQueue &PQ, INT pos1, pos2) {
```

```
PRIORITYQUEUE::ELEMENT aux = PQ.DATA[pos1];
```

```
PQ.DATA[pos1] = PQ.DATA[pos2];
```

```
PQ.DATA[pos2] = aux;
```

}

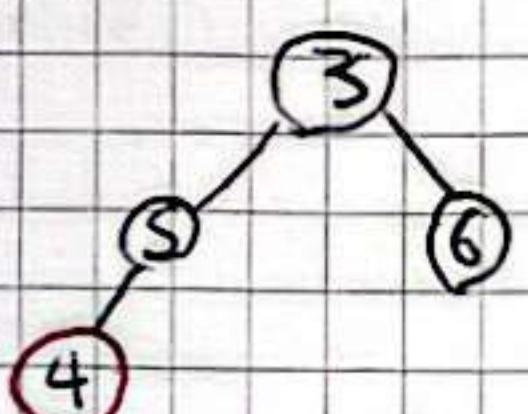
MOVE UP

FUNZIONE AUSILIARE, MI SERVE PER RIPARZIUMARE L'ORDINE DELLE HEAP, VISTO CHE

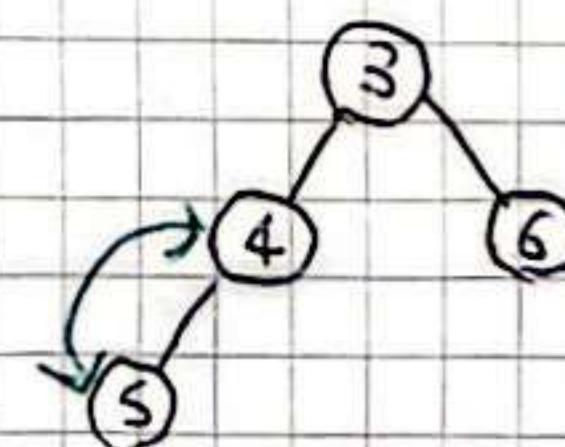
NELL'INSETTOPOPOVA APPARISCE L'ORDINE.

ESEMPIO:

```
INSET(4);
```



MOVE UP



IL 4 E' PIU' PICCOLO DI 5,

DNA L'HEAP E' SISTEMATO

CHIAMIAMO MOVE UP

CODE:

```
VOID MOVE_UP(PRIORITYQUEUE::PriorityQueue &PQ, INT INDEX) {
```

```
WHILE(INDEX > 0) { METTO > 0 PER EVITARE IL CASO COPPIA VUOTA
```

```
INT PARENT_INDEX = (INDEX - 1) / 2; // AVVIA DALL'INDEX SECONDO PESO TANNO  
// VEDRE AL CERCHIO.
```

```
IF(PQ.DATA[INDEX].TIMESTAMP >= PQ.DATA[PARENT_INDEX].TIMESTAMP) {
```

BREAK;

```
SWAP(PQ, INDEX, PARENT_INDEX);
```

INDEX = PARENT_INDEX

SE LA DIMENSIONE E' PLENA, PARTE RETORNANTE FA UNA (SIEGEL = MAXSIZE), DELL'ALBERO

INDICANTE IN CODA E' RISERVAMENTE NUOVE UP;

PQ.DATA[PQ.SIZE] = ELEMENT

MOVEUP(PQ, PQ.SIZE);

++PQ.SIZE;

RETURN TRUE;

FINDMIN

IL SUO COMITTO E' QUELLO DI RESTITUIRE L'ELEMENTO MINIMO (O UNO, L'ADICE) E RESTITUIRE.

IF (!EMPTY(PQ)) {

} RETURN FALSE;

RES = PQ.DATA[0];

RETURN TRUE;

MOVEDOWN

FUNZIONE AUXILIARE UTILIZZATA PER LA DELETION, IL SUO COMITTO E' DI ESSERE
RIPARATRICE L'ORDINE DOPO L'ELIMINAZIONE.

PARSING, DEPRECATION:

1) CALCOLO DEGLI INDICI DEI FIGLI

INT LEFTCHILD = 2 * INDEX + 1

INT RIGHTCHILD = 2 * INDEX + 2

QUESTI CALCOLI DETERMINANO GLI INDICI DEI FIGLI SINISTRA E DESTRA

2) INIZIA A UTILIZZARE DEL PIU' PICCOLO

INT SMALLEST = INDEX;

ASSUME CHE L'ELEMENTO CORRENTE (ACCORDO ALL'INDEX) SIA IL PIU' PICCOLO.

3) CONFRONTO CON IL FIGLIO SINISTRA

IF (LEFTCHILD < PQ.SIZE AND PQ.DATA[LEFTCHILD].TIMESTAMP < PQ.DATA[SMALLEST].TIMES

DELL'ELEMENTO CONNENTE, ACCORDANDO ALL'INDICE SMALLEST PER PENSARE L'INDEX DELL'ELIMINATO.

L'ELEMENTO CONNENTE SOSTITUISCE QUELLO ALL'INDICE INDEX (PASSATO COME ARGOMENTO DELLA FUNZIONE).

4) CONVERSIONE CON FIGLI DECENDI

IF (RIGHT CHILD < PQ.SIZE AND PQ.DATA[NODE INDEX], TIMEStamp < PQ.DATA[SMALLEST], TIMEStamp)

3 SMALLEST : RIGHT CHILD

SE E' FIGLIO DESTRO ED E' TRUE (NODE INDEX < PQ.SIZE) E' PIU' PICCOLO

DELL'ELEMENTO CONNENTE ACCORDANDO SMALLEST,

DETERMINE

PQ.DATA[0] = PQ.DATA[PQ.SIZE - 1];

PQ.SIZE --;

MOVEDOWN(PQ, 0)

RETURN TRUE;

5) SCAMBIO E' CONDIZIONE

IF (SMALLEST != INDEX) {

SWAP(PQ, INDEX, SMALLEST);

MOVEDOWN(PQ, SMALLEST)

}

SE SMALLEST NON E' PIU' GRANDE A INDEX SIGNIFICA CHE UNO

DEI FIGLI E' PIU' PICCOLO DELL'ELEMENTO CONNENTE.

SCAMBIA L'ELEMENTO CONNENTE CON L'ELEMENTO PIU' PICCOLO E OTTIME

RICORDIVAMENTE: MOVEDOWN SULL'INDICE DEL PIU' PICCOLO PER

CONTINUARE IL PROCESSO.