Domande_Problemi esami

Domande frequenti esame

Cosa è un problema di ottimizzazione?

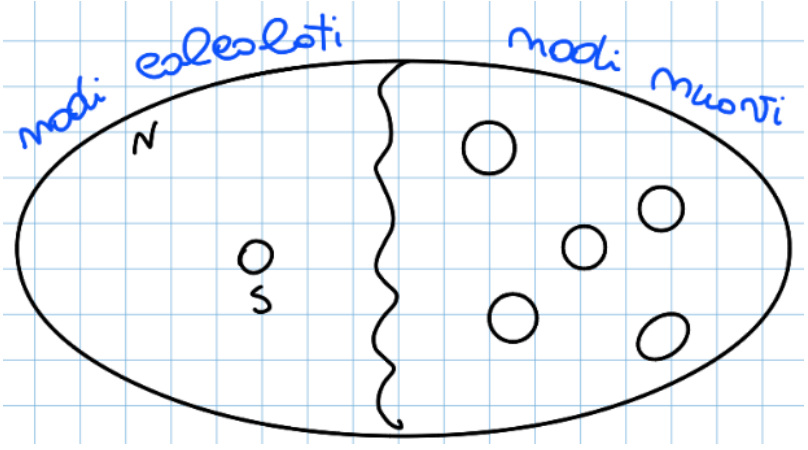
Un problema si dice di ottimizzazione quando esistono varie soluzioni ma solo alcune di queste sono le migliori, per capire qual è la migliore ad ogni soluzione applichiamo la funzione bontà che è definita cosi: $f : S \rightarrow R$, dove S è l'insieme delle soluzioni e R i numeri reali

Ad esempio nel problema dei cammini minimi nei grafi se parto da un nodo A potrei poter scegliere 2 percorsi diversi per arrivare a C: A-B-C o A-D-B-C, normalmente il primo cammino è la soluzione migliore, ma non è sempre così, il grado di bontà assegnato ad ogni soluzione è diverso in base al problema che si affronta

Cosa è un ordinamento topologico di un grafo?

- Un ordinamento topologico di un grafo è un ordinamento lineare dei nodi in modo che ci sia una determinata relazione, se $\exists(u, v) \in E$ tale che $u < v$. Ad esempio: A-F-D-G-C-E-B, questo potrebbe essere un esempio di ordinamento topologico di un grafo
- In un grafo possono esserci più ordinamenti topologici.
- Se il grafo ha un ciclo non posso fare ordinamenti topologici

Come funziona l'insieme dei cammini conosciuti e dei cammini non conosciuti nell'algoritmo di Bellman-Ford?



Questa è la configurazione iniziale prima di iniziare l'algoritmo, man mano che eseguo le relax i nodi si spostano dall'insieme dei nodi nuovi a quello dei calcolati, solo in questa direzione, normalmente l'insieme dei nodi calcolati all'inizio contiene solo la sorgente S, ma nel caso in cui nel grafo ci siano nodi isolati questi si troveranno già qui, solo che noi lo scopriremo alla fine dell'algoritmo (si trovano li perché da soli costituiscono un cammino minimo)

Perché nell'algoritmo di Bellman-Ford nel for principale la i arriva fino a V-1?

```
for i = 0 to V-1 do
for each (u,v) ∈ E do
relax(u,v,w)
```

Perché la lunghezza massima di un cammino minimo è $V-1$, alla fine viene eseguito un ultima volta (ovvero quindi V volte in totale) perché se effettivamente gli archi si rilassano V volte allora c'è un ciclo negativo e l'algoritmo ci avvisa

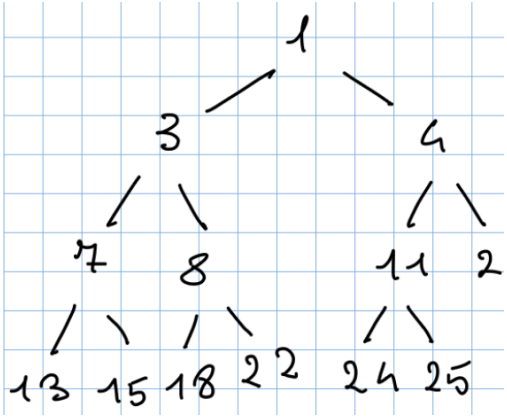
Problemi esami con heap

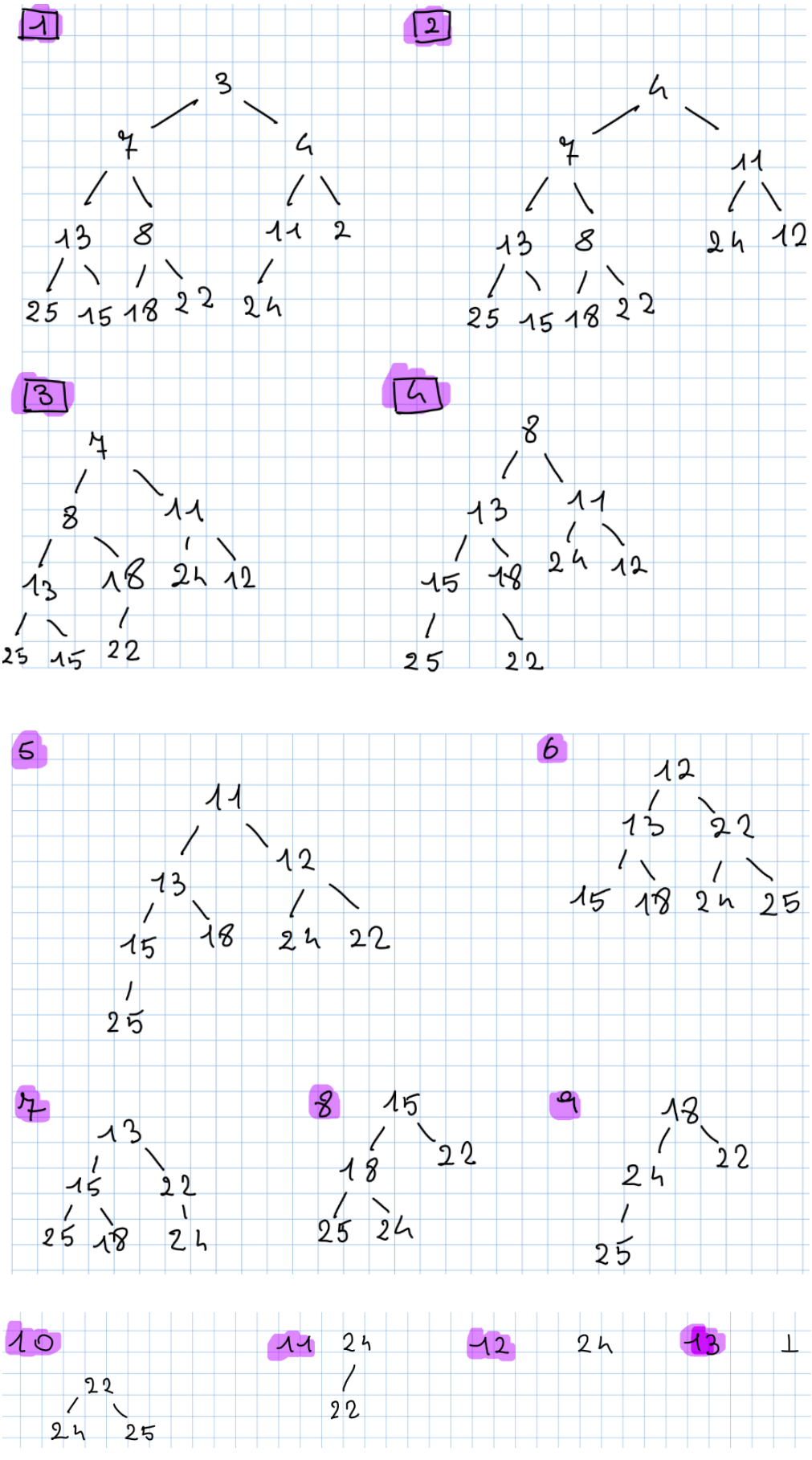
6 Settembre 2024

`A = [1,3,4,7,8,11,12,13,14,15,18,22,24,25]`
`lun(A) = 13`
Questa è la configurazione iniziale di un min-heap

Consegna:
effettuare 13 estrazioni del minimo e mostrare cosa succede al nostro array

Albero iniziale:





((11)questo 22 è un 25 e (12)questo è un 25)

partendo dal min-heap iniziale devo fare 13 estrazioni del minimo che avvengono così:

- Per ogni minimo da estrarre:

1. scambio la root che è il minimo con l'ultimo nodo foglia che sarà il massimo (l'ultimo numero nell' array)
2. Ho estratto il minimo
3. Chiamo heapfy che riordina l'heap trovando il nuovo minimo e facendo scendere il massimo (che era ancora al posto della root) fino in fondo
4. Ricomincio e rieseguo per tutte le estrazioni che devo fare

(nel caso dell'array non è che il minimo lo estraggo e lo tolgo dall'array semplicemente viene messo in fondo e nei passi successivi riduco di 1 la dimensione dell'array)

Possono esserci anche 13 inserimenti, se chiede 13 inserimenti da visualizzare nell'array passo dopo passo, bisogna fare prima l'albero come se fossero 13 inserimenti normali e poi uno alla volta ricopiare sull'array ogni passaggio, facendo un array per passaggio, i figli di un indice i si trovano a $2 * i + 1$ per il left e $2 * i + 2$ per il right

14 Maggio 2025

Consegna:
Scrivere la procedura updateKey(H,i,k)

```
updateKey(H,i,k)
if i > heapsize return
if k > H[i] then increaseKey(H,i,k)
else
H[i] = k
heapfy(H,i)
```

All'esame vanno scritte anche le funzioni heapfy e increaseKey, in generale se faccio uso di altre funzioni le devo scrivere

17 Febbraio 2025

Scrivere una procedura UpdateKey(H, i, k) che aggiorni, in un Heap Binario Massimo, la chiave in posizione i con il nuovo valore k, e ripristini le proprietà dell'heap massimo. L'algoritmo deve gestire correttamente entrambi i casi in cui $k > H[i]$ e $k < H[i]$

```
updateKey(H,i,k)
if i > heapsize return
if k > H[i] then increaseKey(H,i,k)
else if k < H[i] then H[i] = k max-heapfy(H,i)`
```

```
INCREASE-KEY(H,i,k)
H[i] = k
while (i > 1 and H[parent(i)] < H[i]) do
```

```
swap(H,i,parent(i))
i = parent(i)
```

quando ho un max heap in questo caso per gestire entrambi i casi devo chiamare increaseKey quando $k > H[i]$ e quando invece è minore chiamo max-heapfy, con il min heap funziona al contrario devo chiamare decreaseKey

16 Giugno 2025

Consegna:
scrivere la procedura heapMerge(H1,H2,n) (n è il numero di chiavi)
La procedura deve avere complessità $O(n)$

```
Heap-merge(H1,H2,n)
H = newArray(2n)
for i = 0 to n do
H[i] = H1[i]
H[i+n] = H2[i]
buildMax-heap(H,2n)
```

1 Settembre 2025

Si fornisca un funzione ricorsiva, IsHeap(A, i), che preso in input un Max-Heap A (rappresentato come un array) e un indice i > 0, verifica se il sotto albero radicato in nel nodo i di A sia effettivamente un Heap, ovvero rispetti la proprietà dell'ordinamento parziale. La procedura restituisce True/False.

```
IsHeap(A, i)
// Calcolo indici dei figli
l = 2*i + 1
r = 2*i + 2
// CASO BASE IMPLICITO:
// Se i nodi l o r escono dall'array, quel ramo è valido (è NULL).
// I controlli "if l < heapSize" e "if r < heapSize" gestiscono le foglie.
// 1. CONTROLLO SOTTOALBERO SINISTRO
if l < heapSize then
// Violazione locale: il figlio è maggiore del padre
if A[l] > A[i] then return False
// Violazione ricorsiva nel sottoalbero sinistro
if IsHeap(A, l) == False then return False
// 2. CONTROLLO SOTTOALBERO DESTRO
if r < heapSize then
// Violazione locale: il figlio è maggiore del padre
if A[r] > A[i] then return False
```

```
// Violazione ricorsiva nel sottoalbero destro
if IsHeap(A, r) == False then return False
// Se non ho trovato violazioni, è un Heap valido
return True
```

versione senza commenti:

```
IsHeap(A, i)
l = 2*i + 1
r = 2*i + 2
if l < heapSize
if A[l] > A[i] or IsHeap(A, l) == False
return False
if r < heapSize
if A[r] > A[i] or IsHeap(A, r) == False
return False
return True
```

27 Ottobre 2025

Un sistema registra in tempo reale i punteggi dei giocatori di un videogioco online. Si vuole man- tenere in ogni momento la Top-Ten dei giocatori con i punteggi più alti, aggiornandola in tempo efficiente man mano che arrivano nuovi risultati. Per ciascun giocatore si considera solo il suo punteggio migliore. Progettare un algoritmo e una struttura dati che permettano di aggiornare la classifica dopo ciascun nuovo punteggio. Descrivere l'idea di base e stimare la complessità dell'operazione di aggiornamento della struttura.

Useremo un min heap che conterrà i 10 punteggi più alti, quindi la root sarà il punteggio più alto in assoluto

```
topTen(H,i,p)
if p > H[0] then
H[0] = p
min-heapfy(H,0)
```

1 Luglio 2025

In un albero rosso-nero, l'altezza nera di un nodo x, denotata bh(x), è definita come il numero di nodi neri presenti lungo un qualunque cammino da x a una foglia NIL, escludendo il nodo stesso, ma includendo il nodo NIL finale (che si assume sempre nero). Per definizione degli alberi rosso-neri, tutti i cammini da un nodo verso le foglie devono avere la stessa altezza nera. Scrivere lo pseudo-codice di una procedura ricorsiva che, dato un nodo x, restituisca la sua altezza nera

Assumo che bh sia sempre almeno uguale ad 1, anche nel caso che x sia la root e anche l'unico nodo ha comunque 2 figli null che sono neri quindi bh = 1, assumo anche che l'albero sia valido

```
BH(x,bh)
    if (x = NULL) then
        return bh
    if (left(x) = NULL) then
        return bh
    if (left(x).colore = black) then
        bh++
    return BH(left(x),bh)
```

qui controllo ogni volta solo i figli sinistri fino ad arrivare alla foglia, scorrendoli come fosse una lista concatenata perché mi basta arrivare semplicemente alla fine e contare quante volte incontro un nodo nero