Analisi e progettazione di algoritmi

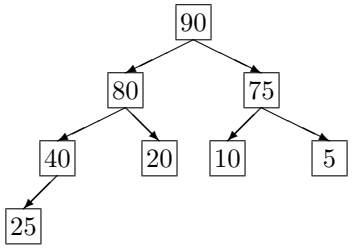
(III anno Laurea Triennale - a.a. 2018/19)

Soluzioni prova scritta 5 giugno 2019

NB: I punteggi sono indicativi.

Esercizio 1 - Ordinamenti

1. Array (heap) come albero:



2. Eseguo *getMax* che preleva 90, lo sostituisce con 25 e poi esegue *moveDown*(25). Scambio 25 con 80, poi con 40. Mi fermo perché foglia.

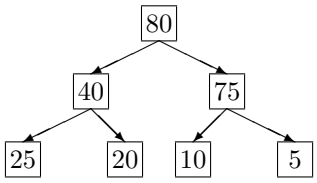
heap:

80	40	75	25	20	10	5

array ordinato:

90

albero:



Eseguo *getMax* che preleva 80, lo sostituisce con 5 e poi esegue *moveDown*(5). Scambio 5 con 75 e poi con 10. Mi fermo perché nodo foglia.

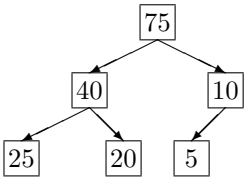
heap:

75	40	10	25	20	5

array ordinato:

80	90

albero:



Eseguo *getMax* che preleva 75, lo sostituisce con 5 e poi esegue *moveDown*(5). Scambio 5 con 40 e poi con 25.

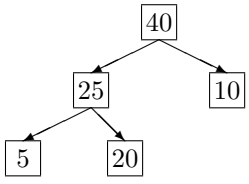
heap:

40	25	10	5	20

array ordinato:

75	80	90

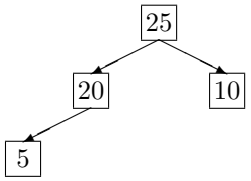
albero:



Eseguo *getMax* che preleva 40, lo sostituisce con 20 e poi esegue *moveDown*(20). Scambio 20 con 25. Mi fermo perché $20 > 5$.

heap: | 25 | 20 | 10 | 5 | array ordinato: | 40 | 75 | 80 | 90 |

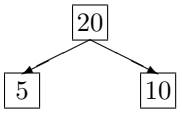
albero:



Eseguo *getMax* che preleva 25, lo sostituisce con 5 e poi esegue *moveDown*(5). Scambio 5 con 20.

heap: | 20 | 5 | 10 | array ordinato: | 25 | 40 | 75 | 80 | 90 |

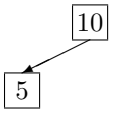
albero:



Eseguo *getMax* che preleva 20, lo sostituisce con 10 e poi esegue *moveDown*(10). Nessuno scambio perché $10 > 5$.

heap: | 10 | 5 | array ordinato: | 20 | 25 | 40 | 75 | 80 | 90 |

albero:



Eseguo *getMax* che preleva 10, lo sostituisce con 5. L'esecuzione di *moveDown*(5) non fa nulla perché la radice è foglia.

heap: | 5 | array ordinato: | 10 | 20 | 25 | 40 | 75 | 80 | 90 |

albero:

Ho finito, l'ultima *getMax* non è necessaria perché l'elemento minimo (5) è già al suo posto (indice zero nell'array) e l'array è ordinato.

Esercizio 2 - Strutture dati

1. Una sequenza che porta al BST migliore è per esempio 40,20,10,30,60,50,70. Non è unica, per esempio anche 40,20,60,10,30,50,70. [disegno]

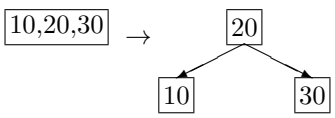
2. La sequenza crescente porta all'albero BST completamente sbilanciato (caso peggiore). [disegno]

3. Inserimento della sequenza crescente 10, 20, 30, 40, 50, 60, 70 in 2-3 albero:

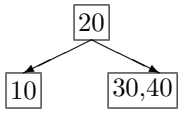
   Inserisco 10 e poi 20 che trovano posto nella radice:

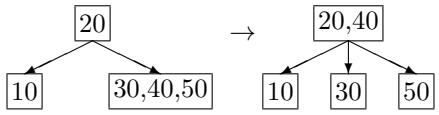
   Inserendo 30, la radice va in overflow. Faccio split, promuovo l'elemento centrale (20) come nuova radice:

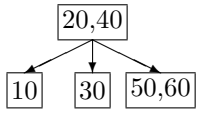
   Inserendo 40, si aggiunge un secondo elemento nella foglia a destra, senza modificare la struttura dell'albero:

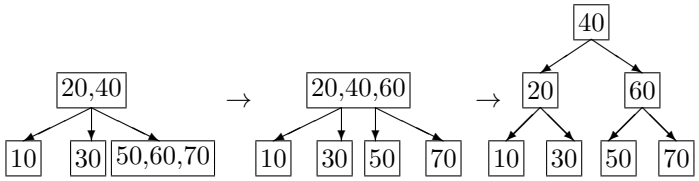
Inserendo 50, la foglia a destra va in overflow. Faccio split, promuovo l'elemento centrale (40) nel padre. Il padre (radice) ha posto, la radice guadagna un elemento ed un figlio:



La chiave 60 viene inserita nella foglia che contiene 50, la struttura dell'albero non cambia:



Inserendo 50, la foglia va in overflow. Faccio split, promuovo 60 nel padre. Il padre (radice) va a sua volta in overflow, faccio split e promuovo 40. Si crea una nuova radice con chiave 40.



Esercizio 3 - Grafi La seguente tabella mostra per ogni iterazione: nella prima colonna il nodo che viene estratto; nelle successive i nodi per i quali viene modificata `dist` e come; nell'ultima gli archi dell'albero ricoprente (in grassetto quelli definitivi).

estratto	A	B	C	D	E	F	G	*albero*
A	–	7	∞	5	∞	∞	∞	$(A,B),(A,D)$
D	–	7	∞	–	15	6	∞	(A,B), **(A,D)**, $(D,E),(D,F)$
F	–	7	∞	–	8	–	11	(A,B), **(A,D)**, **(D,F)**, $(F,E),(F,G)$
B	–	–	8	–	7	–	11	**(A,B)**, **(A,D)**, **(D,F)**, $(B,C),(B,E),(B,G)$
E	–	–	5	–	–	–	9	**(A,B)**, **(A,D)**, **(D,F)**, **(B,E)**, $(E,C),(E,G)$
C	–	–	–	–	–	–	9	**(A,B)**, **(A,D)**, **(D,F)**, **(B,E)**, **(E,C)**, (E,G)
G	–	–	–	–	–	–	–	**(A,B)**, **(A,D)**, **(D,F)**, **(B,E)**, **(E,C)**, **(E,G)**

Esercizio 4 - Design e analisi di algoritmi

1. Il problema si può risolvere con una variante della ricerca binaria.

```
get_first(x,a)// a[0..n-1]
        return get_first(x,a,0,n-1)

get_first(x,a,inf,sup)
  if (inf <= sup)
    mid = (inf + sup)/2
    if (x=a[mid]) && (mid = inf || a[mid-1]<a[mid]) return mid
    if (x ≥ a[mid]) return get_first(x,a,inf,mid-1)
    else return get_first(x,a,mid+1,sup)
  return n//a.length
```

2. Per induzione forte. L'algoritmo è banalmente corretto nel caso `inf > sup` (0 elementi). Nel caso $n > 0$ elementi, si considera l'indice centrale dell'array `mid`. Perché questo sia l'indice della prima occorrenza di `x`, occorre ovviamente che l'elemento corrispondente sia `x` e l'elemento precedente sia strettamente minore (oppure non vi sia elemento precedente). Se l'elemento centrale è uguale a `x` ma non è la prima occorrenza, oppure è maggiore, basta cercare nella prima metà, mentre se l'elemento centrale è minore di `x`, basta cercare nella seconda metà. Dato che queste ricerche sono effettuate su porzioni di array di dimensione strettamente minore si ha la correttezza per ipotesi induttiva.

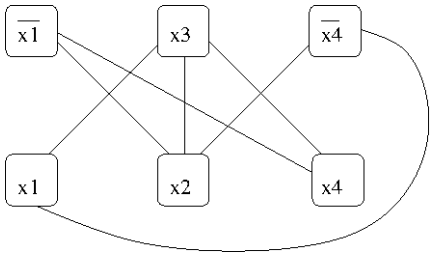
3. La complessità è quella della ricerca binaria, quindi $O(\log n)$.

4.

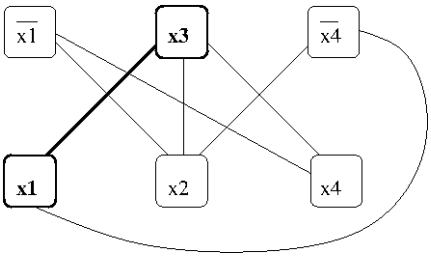
```
get_first(x,a,inf,sup)
  if (inf <= sup)
    mid = (inf + sup)/2
    if (x=a[mid]) && (mid = inf || a[mid-1]<a[mid]) return mid
    if (x ≤ a[mid]) return get_first(x,a,mid+1,sup)
    else return get_first(x,a,inf,mid-1)
    else return true
  return n//a.length
```

Esercizio 5 - NP-completezza (Punti 7)

1. La corrispondente istanza del problema *CLIQUE*, ottenuta attraverso la riduzione vista a lezione, è data dalla coppia $(G, 2)$ dove G è il grafo seguente:



2. Un'assegnazione di valori di verità che rende vera ϕ è, per esempio, $x_1 = T$, $x_2 = F$, $x_3 = T$, $x_4 = F$. La corrispondente clique è evidenziata sotto.



3. Una riduzione da *SAT* a *CLIQUE* si ottiene come composizione della riduzione da *3SAT* a *CLIQUE* con l'altra riduzione da *SAT* a *3SAT* vista a lezione.