Introduzione agli algoritmi
Proff. T. Calamoneri - S. Caminiti- E. Fachini -
15 ottobre 2020

1. Si disegnino tutti i min-heap sulle chiavi {10,20,30,40,50}.
2.

Sol. Un heap binario è un albero quasi completo, cioè completo fino al penultimo livello e con le foglie dell'ultimo livello tutte compattate a sinistra. Alberi di questo tipo possono essere memorizzati in un array, memorizzando la radice nell'entrata 1 e poi a seguire le chiavi ottenute da una visita per livelli da sinistra verso destra e dall'alto verso il basso. Questo permette la "navigazione" nell'albero cioè la risalita da figlio a padre o la discesa da padre in figlio, come nel caso di alberi memorizzati con strutture a puntatori, ma con il vantaggio dell'accesso più veloce fornito dall'uso dell'array. Un min heap è un heap binario in cui la chiave di ogni elemento, esclusa la radice, è maggiore di quella di suo padre.
Quindi i possibili min heap sono:

```
      10
   20    30
40  50
```

```
      10            il simmetrico scambiando le due foglie più a sinistra
   30    20
40  50
```

```
      10
   20    30         ottenuto dal primo scambiando i figli della radice
50  40
```

```
      10     il simmetrico del terzo scambiando le due foglie più a sinistra
   30    20
50  40
```

```
      10
   20    40     40 come figlio destro, 20 deve essere figlio sinistro della radice
30  50
```

```
      10        il simmetrico del quinto scambiando le due foglie più a sinistra
  20      40
50  30
```

```
      10        50 come figlio destro, 20 deve essere figlio sinistro della radice
  20      50
30  40
```

```
      10         simmetrico del settimo scambiando le due foglie più a sinistra
  20      50
40  30
```

Altri non sono ottenibili perchè avendo x come figlio destro della radice necessariamente il più piccolo tra i rimanenti {20,30,40,50} - x va come figlio sinistro della radice.

3. Si imposti la relazione di ricorrenza che definisce il tempo di esecuzione della seguente funzione e la si risolva usando il metodo della sostituzione. Si commentino opportunamente i passaggi del calcolo, si descriva l'albero della ricorsione e come si giunge alla previsione sull'andamento del tempo di calcolo, si imposti l'induzione con chiarezza, sia nello scrivere quanto si vuole dimostrare sia nel formulare l'ipotesi induttiva.

```
Analizzzami(A,i,j)
input: A array di interi ,i e j interi positivi
n = j - i + 1
m = n/4
if n ≤ 1 then return (A[1]);
for k = 1 to 2n/3 do
        A[i] ← A[i] − A[j];
end for
x = Analizzami (A,i,i+m) + Analizzami(A, i + m+1,i+2m)
return (x);
```

Sol. Le istruzioni al di fuori della chiamata sono eseguite in tempo costante, tranne il ciclo for che viene eseguito 2n/3 volte.
Ci sono inoltre due chiamate della funzione su circa un quarto degli elementi quindi la relazione di ricorrenza da risolvere è

$T(n) = 2T(n/4) + \Theta(n)$.

Esplicitando le costanti:

$T(n) = 2T(n/4) + cn$ se $n > 1$

$T(n) = d$ altrimenti

L'albero della ricorsione, che ci consente di semplificare i calcoli per giungere a una previsione dell'andamento di T è binario, visto che ci sono due chiamate, e sotto l'ipotesi semplificatrice che $n = 4^h$, ha altezza h e ogni nodo sul livello i, $0 \le i < h$, è etichettato con $cn/4^i$. Sommando sul livello i si ottiene dunque $2^i\, c(n/4^i) = cn/2^i$. Al livello delle foglie invece ogni nodo corrisponde al costo per n = 1 quindi a d.

In conclusione $T(n) = 2^h d + \sum_{0\ldots h-1} cn/2^i = 2^h d + cn\sum_{0\ldots h-1} 1/2^i \le 2^h d + cn\sum_{0\ldots \infty} 1/2^{2i} = O(n)$, perchè la serie converge a una costante e $2^h d = n^{1/2} d$, e il termine lineare è dominante.

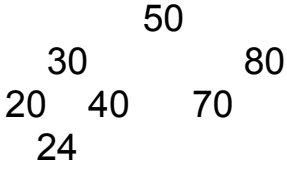
Ora verifichiamo la previsione per induzione.
Faremo vedere che esistono due costanti positive k e n' tali che $T(n) \le kn$, per ogni $n \ge n'$.
Sia vero per ipotesi induttiva per ogni $m < n$, consideriamo T(n):
$T(n) = 2T(n/4) + c\, n \le$
$2k\,(n/4) + c\, n =$
$k\, n/2 + c\, n.$
Vediamo se troviamo due valori per k e n' tali che $k\, n/2 + c\, n \le k\, n$
Sviluppando:
$k\, n/2 + c\, n \le k\, n \Leftrightarrow k\, n + 2c\, n \le 2k\, n \Leftrightarrow 2c\, n \le k\, n$ quest'ultima disuguaglianza è vera per $k = 2c$ e per ogni $n \ge 1$

Considerando il caso base abbiamo che $T(1) = d$ e $T(4) = 2d+4c$ e $T(4) \le 4k$ è vero prendendo $k = d+2c$.
Questo valore va bene in tutti i casi, prendendo $n_0 = 4$.

3. Si progetti un algoritmo il più efficiente possibile che, dato in input un albero binario di ricerca *T* e un intero *k*, verifichi se nell'albero c'è un elemento la cui chiave sommata a quella minima dà il valore *k*. Si assuma che tutte le chiavi in *T* siano interi positivi distinti e che l'albero è memorizzato con struttura a puntatori con quattro campi: il campo chiave, il puntatore al nodo padre, i puntatori ai figli sinistro e destro. Si descriva a parole l'idea algoritmica, si produca lo pseudocodice e si analizzi il tempo di esecuzione asintotica. Lo pseudocodice delle funzioni studiate durante il corso, eventualmente utilizzate come moduli per costruire l'algoritmo, deve essere esplicitato.

Sol. Basta cercare nell'albero il valore x = k - min. Il minimo in un ABR è il nodo sul cammino più a sinistra radice-foglia che non ha figlio sinistro. Quindi si determina in O(h), ove h è l'altezza dell'albero. Poiché le chiavi sono positive allora se k minore del minimo e quindi x < 0 si può immediatamente dare in output falso. Altrimenti si deve eseguire una ricerca di x nell'albero. Poiché anche quest'operazione ha un tempo di esecuzione in O(h) complessivamente il tempo è O(h).

Es.



Il minimo è 20.
Se k = 10 10 - 20 < 0 non c'è alcun numero che sommato a 20 può dare 10.
Se k = 50 50 - 20 = 30 che viene trovato con la ricerca alla prima chiamata sul figlio sinistro, visto che 30 è minore di 50.
Se k = 80 k-20 = 60 la ricerca dà 0 perchè nel sottoalbero destro 60 non c'è.
se k =100 allora k-20 = 80 e la ricerca termina con successo nel sottoalbero destro.

Pseudocodice:
Useremo la funzione minimum che dà in output il puntatore al nodo di chiave minima e la funzione di ricerca di una chiave in un ABR.

```
Minimum(x)

precond:x ≠ nil e x è un ABR

postcond:restituisce il puntatore al nodo di chiave minima in x

  while x.left ≠ nil do
        x = x.left
  return x

Tree-Search(x, k)

Input: un puntatore x e una chiave k

prec: x è un ABR

output: il puntatore (riferimento) al nodo di chiave k, se presente, nil altrimenti

   if x == NIL or k == x.key then return x
   if k < x.key then  return Tree-Search(x.left, k)
              else return Tree-Search(x.right, k)
```

La nostra funzione sarà dunque:

```
SommaMin(T,k)
 if T == NIL then  return 0
 p = Minimum(T)
 x = k - p.key
 if x < 0  then  return 0
 if Tree-Search(T,x) == NIL then return 0 else return 1.
```