

23-12

Domanda 33 Si consideri una tabella hash di dimensione $m = 8$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \bmod m$ e $h_2(k) = 1 + k \bmod (m - 2)$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 14, 22, 10, 16, 8.

HASHING =

$$\text{Doppio hash} = h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$$

$$\textcircled{14} \quad h(14, 0) = (14 \bmod 8) = 6$$

$$\textcircled{22} \quad h(22, 0) = (22 \bmod 8) = 6$$

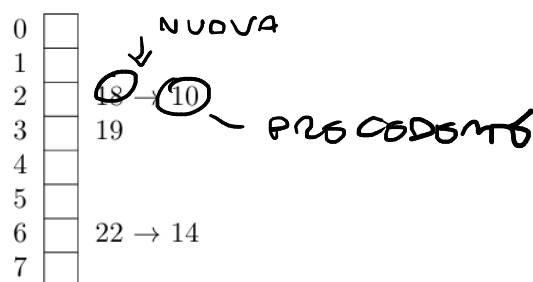
$$h(22, 1) = (22 \bmod 8) + 1 * (1 + (22 \bmod 6)) \bmod 8$$

$$= [6 + (1 + 3)] \bmod 8 = 10 \bmod 8 = 2$$

Domanda 34 Si consideri una tabella hash di dimensione $m = 8$, gestita mediante chaining (liste di trabocco) con funzione di hash $h(k) = k \bmod m$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 14, 10, 22, 18, 19.

$$h(k) = 14 \bmod 8 = 6$$

Soluzione: Si ottiene



Esercizio 1 (9 punti) Scrivere una funzione $\text{Anc}(T, k1, k2)$ che dato un albero binario di ricerca T nel quale tutte le chiavi sono distinte, e due chiavi $k1, k2$ presenti in T , verifica se il nodo contenente $k1$ è antenato del nodo contenente $k2$. Valutare la complessità della funzione.

$\text{ANC}(T, k1, k2)$

- 1 $X = T.\text{root}$
- 2 $\text{WHILE } (X.\text{KEY} < k1 \text{ AND } X.\text{KEY} < k2)$
- 3 $\text{IF } (k2 < X.\text{KEY})$
- 4 $X = X.\text{LEFT}$
- 5 ELSE
- 6 $X = X.\text{RIGHT}$
- 6 $\text{IF } (X.\text{KEY} = k1) \text{ RETURN TRUE}$
- 7 RETURN FALSE

Domanda A (8 punti)

Si consideri la seguente funzione ricorsiva con argomento un intero $n \geq 0$

$$\text{val}(n) = \begin{cases} \text{return } 0 & \text{if } n = 0 \\ \text{return } \text{val}(n-1) + n + 1 & \text{else} \end{cases}$$

$$T(n) \rightarrow [T(n) = T(n-1) + c]$$

Dimostrare induttivamente che la funzione calcola il valore $(n+3)n/2$. Inoltre, determinare la ricorrenza che esprime la complessità della funzione e mostrare che la soluzione è $\Theta(n)$. Motivare le risposte.

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

$$T(n) = \Omega(n) \Rightarrow [T(n) \geq \Omega(n)] \text{ ①}$$

$$T(n) = O(n) \Rightarrow T(n) \leq C(n)$$

$$\text{① } \begin{aligned} T(n) &= T(n-1) + c \\ \Omega(n) &\geq \Omega(n-1) + c \end{aligned} \Rightarrow \begin{aligned} d_n &\geq d_{n-1} + c \\ d &\geq c, \forall n \in \mathbb{N} \end{aligned}$$

$$\text{val}(n-1) = \frac{(n+3)n}{2}$$

Soluzione: Per quanto riguarda la funzione calcolata, procediamo per induzione su n . Se $n = 0$, la funzione ritorna 0 che è in effetti $\frac{(0+3)0}{2} = 0/2 = 0$. Se $n > 0$, allora per ipotesi induttiva, $\text{val}(n-1) = \frac{(n+2)(n-1)}{2}$. Quindi $\text{val}(n) = \text{val}(n-1) + n + 1 = \frac{(n+2)(n-1)}{2} + n + 1 = \frac{(n^2 + n - 2 + 2n + 2)}{2} = \frac{(n^2 + 3n)}{2} = \frac{(n+3)n}{2}$, come desiderato.

osservazione

$$\begin{array}{c} \text{val}(n-1) \\ \text{volta} \end{array} \rightarrow \frac{n(n-1)}{2} \rightarrow \frac{(n+2)(n-1)}{2}$$

Esercizio 2 (8 punti) Per $n > 0$, siano dati due vettori a componenti intere $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$. Si consideri la quantità $c(i, j)$, con $0 \leq i \leq j \leq n-1$, definita come segue:

$$c(i, j) = \begin{cases} a_i & \text{se } 0 < i \leq n-1 \text{ e } j = n-1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n-1, \\ c(i-1, j) \cdot c(i, j+1) & 0 < i \leq j < n-1. \end{cases}$$

Si vuole calcolare la quantità $M = \max\{c(i, j) : 0 \leq i \leq j \leq n-1\}$.

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di M .

2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

COMPUTE(A, B)

$N \leftarrow \text{length}(A)$

FOR $[s = 0 \text{ TO } N-1]$

$C[0, s] \leftarrow B[s]$

FOR $[i = 1 \text{ TO } N-1]$

$C[i, n-1] \leftarrow A[i]$

```
COMPUTE(a, b)
n <- length(a)
M <- -infinito
for i=1 to n-1 do
  C[i, n-1] <- a_i
  M <- MAX(M, C[i, n-1])
for j=0 to n-1 do
  C[0, j] <- b_j
  M <- MAX(M, C[0, j])
```

```
for i=1 to n-2 do
  for j=n-2 downto i do
    C[i, j] <- C[i-1, j] * C[i, j+1]
    M <- MAX(M, C[i, j])
return M
```

DO US RUMPI...

$$\textcircled{2} \sum_{i=1}^{n-2} \sum_{s=i}^{n-2} 1 = \sum_{i=1}^{n-2} (n-1-i) = \sum_{k=1}^{n-2} k$$

$$\frac{(n-2)(n-1)}{2}$$

$$\sim \rightarrow \frac{n(n-1)}{2}$$

GAUSS

Esercizio 2 (9 punti) Data una stringa di numeri interi $A = (a_1, a_2, \dots, a_n)$, si consideri la seguente ricorrenza $c(i, j)$ definita per ogni coppia di valori (i, j) con $1 \leq i, j \leq n$:

$$c(i, j) = \begin{cases} a_j & \text{if } i = 1, 1 \leq j \leq n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \leq n, \\ c(i-1, j) \cdot c(i, j+1) \cdot c(i-1, j+1) & \text{altrimenti.} \end{cases}$$

→ FORMULA

1. Si fornisca il codice di un algoritmo iterativo bottom-up $\text{COMPUTE_C}(A)$ che, data in input la stringa A , restituisca in uscita il valore $c(n, 1)$.
2. Si valuti il numero esatto $T_{CC}(n)$ di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

COMPUTE_C(A)

$N \leftarrow \text{LENGTH}(A)$

FOR ($j = 1$ TO N)

$C[1, j] = A[j]$

FOR ($i = 2$ TO N)

$C[i, N] = A[n+1-i]$

FOR ($i = 2$ TO N)

FOR ($j = N-1$ TO N)

$C[i, j] = \text{FORMULA}$

RETURN $C(N, 1)$

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione "reverse column-major", in cui calcoliamo gli elementi della tabella in ordine decrescente di indice di colonna e, all'interno della stessa colonna, in ordine crescente di indice di riga. Il codice è il seguente.

```
COMPUTE_C(A)
n = length(A)
for i=1 to n do
  c[1,i] = a_i
  c[i,n] = a_{n+1-i}
  for j=n-1 downto 1 do
    for i=2 to n do
      c[i,j] = c[i-1,j] * c[i,j+1] * c[i-1,j+1]
  return c[n,1]
```

→ PER EFFICIENZA ...

$$\left[\begin{array}{l} \text{for } j=n-1 \text{ downto } 1 \text{ do} \\ \quad \text{for } i=2 \text{ to } n \text{ do} \\ \quad \quad c[i,j] = c[i-1,j] * c[i,j+1] * c[i-1,j+1] \end{array} \right] \sim \sum_{j=1}^{n-1} \sum_{i=2}^n 2$$

Si osservi che un altro modo corretto di riempire la tabella è attraverso una scansione "reverse diagonal", che scansiona per diagonal parallele alla diagonale principale partendo da quella contenente solo $c[1, n]$.

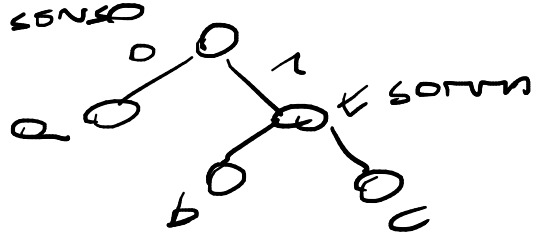
$$= \sum_{j=1}^{n-1} 2(n-1-j) = 2(n-1)(n-1) = 2(n-1)^2$$

2. Ogni iterazione del doppio ciclo dell'algoritmo esegue due operazioni tra interi, e quindi

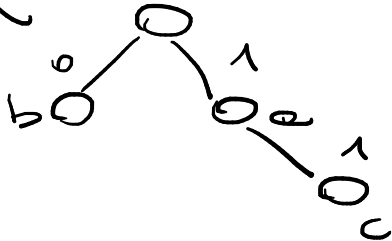
$$2(n-1)(n-1) = 2(n-1)^2$$

Esercizio 2 (9 punti) Si consideri un file definito sull'alfabeto $\Sigma = \{a, b, c\}$, con frequenze $f(a), f(b), f(c)$. Per ognuna delle seguenti codifiche si determini, se esiste, un opportuno assegnamento di valori alle 3 frequenze $f(a), f(b), f(c)$ per cui l'algoritmo di Huffman restituisce tale codifica, oppure si argomenti che tale codifica non è mai ottenibile.

1. $e(a) = 0, e(b) = 10, e(c) = 11$
2. $e(a) = 1, e(b) = 0, e(c) = 11$
3. $e(a) = 10, e(b) = 01, e(c) = 00$



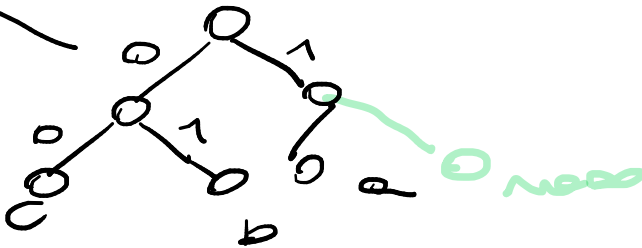
(S)



CODIFICA

NON È LIBERO
DA ERRORESI

(NO)

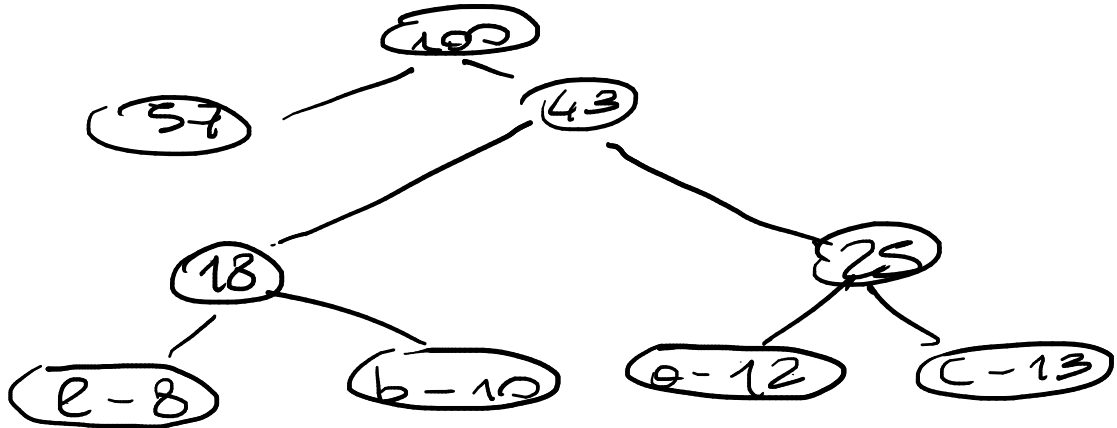


(NO)

Domanda B (5 punti) Indicare, in forma di albero binario, il codice prefisso ottenuto tramite l'algoritmo di Huffman per l'alfabeto $\{a, b, c, d, e\}$, supponendo che ogni simbolo appaia con le seguenti frequenze:

| a | b | c | d | e |
|----|----|----|----|---|
| 12 | 10 | 13 | 57 | 8 |

Spiegare il processo di costruzione del codice.

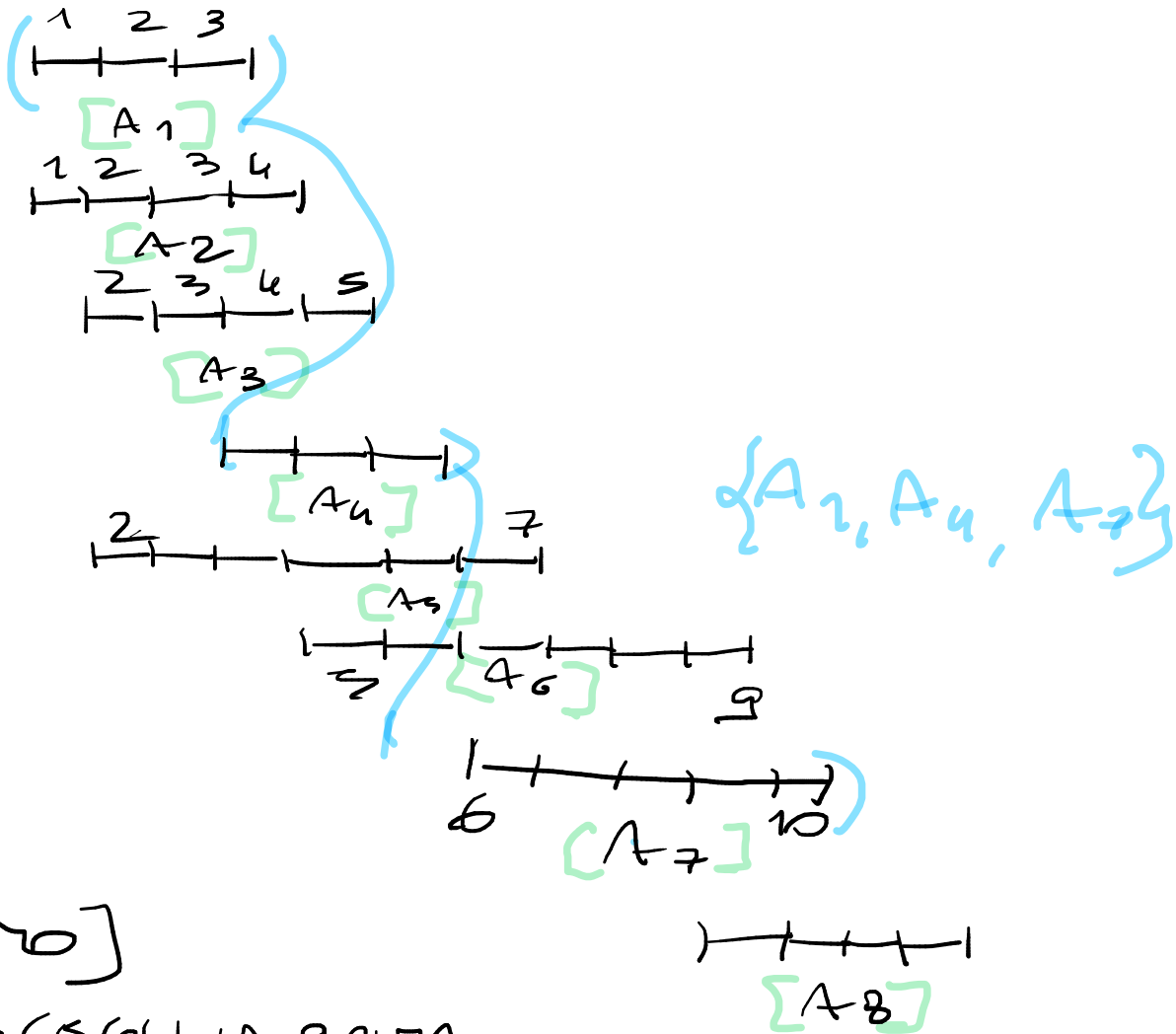


Domanda B (6 punti) Si consideri un insieme di 8 attività $a_i, 1 \leq i \leq 8$, caratterizzate dai seguenti vettori s e f di tempi di inizio e fine:

$$s = (1, 1, 2, 4, 2, 5, 6, 9)$$

$$f = (3, 4, 5, 6, 7, 9, 10, 12)$$

Determinare l'insieme di massima cardinalità di attività mutuamente compatibili selezionato dall'algoritmo greedy GREEDY_SEL visto in classe. Motivare il risultato ottenuto descrivendo brevemente l'algoritmo.



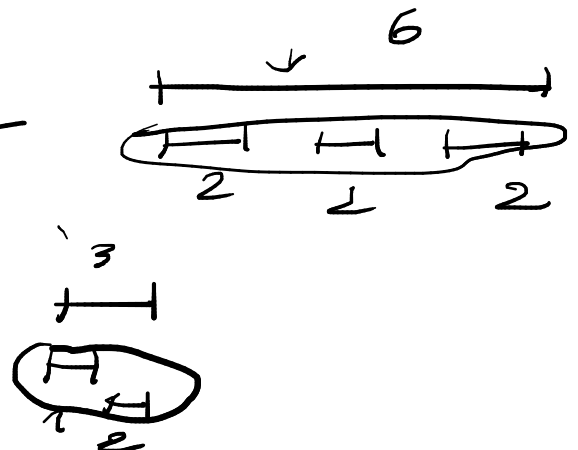
[OTTIMO]

→ SCELTA LA PRIMA / ULTIMA

[NON OTTIMO]

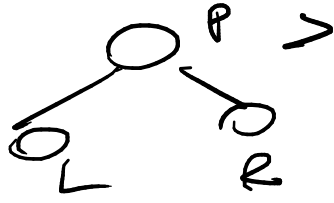
→ ATTIVITÀ PIÙ LUNGA

→ ATTIVITÀ PIÙ CORTA



Domanda 23 Scrivere una funzione `IsMaxHeap(A)` che dato in input un array di interi $A[1..n]$ che verifica se A è organizzato a max-heap e ritorna un corrispondente valore booleano. Valutarne la complessità.

A \downarrow $L = 2I$ / $R = 2I + 1$
 \curvearrowright $\text{HEAPIFY} \dots$
 $\text{ISMAXHEAP}(A)$
 $\text{WHILE } (1 \leq N/2)$
 $\text{IF } (A[1] > A[2I] \text{ AND } A[1] > A[2I+1])$
 \Rightarrow (11 FOR CONVERSION) $\uparrow \uparrow \uparrow$
 $(A[1] \leftarrow A[1/2])$
 GOTO
 $\uparrow \uparrow \uparrow$
 $\text{IF } (1 = 2N/2) \text{ RETURN TRUE}$
 RETURN FALSE

$N/2 \rightarrow \begin{cases} 2 \times I \\ 2 \times I + 1 \end{cases}$


Soluzione: Versione ricorsiva

`IsMaxHeap(A,i,j)` // verifica se l'array $A[i,j]$ e' un max-heap
 (o meglio, se contiene un max-heap radicato in i ,
 dato che al passo ricorsivo non tutto $[i,j]$
 conterra' il sottoalbero scelto

```

l = 2*i
r = 2*i+1
if l < j
  leftOk = [A[i] >= A[l]] and IsMaxHeap(l, j)
else
  leftOk = true
if r < j
  rightOk = [A[i] >= A[r]] and IsMaxHeap(r, j)
else
  rightOk = true

```

\rightarrow return leftOk and rightOk

Per quanto riguarda la complessità si osservi che $T(n) = c + 2T(n/2)$ per un'opportuna costante c e quindi, utilizzando il master theorem, si deduce che la complessità è $O(n)$.

Versione iterativa

```

IsMaxHeap(A,n) // verifica se l'array A[1..n] e' un max-heap
i=2
while (i <= n) and (A[i] <= A[i/2])
  i++
return (i==n+1)

```

