

## Algoritmi e Strutture Dati – 17/01/23 – Parte A

**Esercizio -1** Iscriverti allo scritto entro la scadenza. In caso di inadempienza, -1 al voto finale.

**Esercizio 0** Scrivere correttamente nome, cognome, numero di matricola, riga e colonna su tutti i fogli consegnati. Consegnare foglio A4 e foglio protocollo di bella. In caso di inadempienza, -1 al voto finale.

### Esercizio A1 – Punti $\geq 8$

Si consideri questo algoritmo:

```

int P(int[] A, int n)
return R(A, 1, n)

int R(int[] A, int i, int j)
    int n = j - i + 1
    if n < 4 then
        return sum(A, i, j)
    int d = ⌊n/4⌋
    int r1 = R(A, i, i + 3d - 1)
    int r2 = R(A, i + d, j)
    return r1 - r2
  
```

dove  $\text{sum}(A, i, j)$  somma i valori di  $A$  compresi fra  $i$  e  $j$ . Scrivere l'equazione di ricorrenza associata a questo algoritmo e calcolare la complessità computazionale che ne deriva.

### Esercizio A2 – Albero massimale – Punti $\geq 10$

Scrivere un algoritmo

TREE maxTree(int[] A, int n)

che prenda in input un vettore contenente  $n$  interi distinti e restituisca un *albero binario massimale*, così definito:

- La radice contiene il valore massimo di  $A$ , che supponiamo essere nella posizione  $j$ ;
- il sottoalbero sinistro è un sottoalbero massimale definito sui valori contenuti nel sottovettore a sinistra di  $A[j]$ ;

- il sottoalbero destro è un sottoalbero massimale definito sui valori contenuti nel sottovettore a destra di  $A[j]$ .

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale nei casi ottimo e pessimo.

Per esempio, se  $A = [2, 3, -1, 8, 4, 5]$ , l'albero massimale avrà 8 nella radice, i valori  $[2, 3, -1]$  nel sottoalbero di sinistra, i valori  $[4, 5]$  nel sottoalbero destro, entrambi disposti come alberi massimali.

### Esercizio A3 – Alberello – Punti $\geq 12$

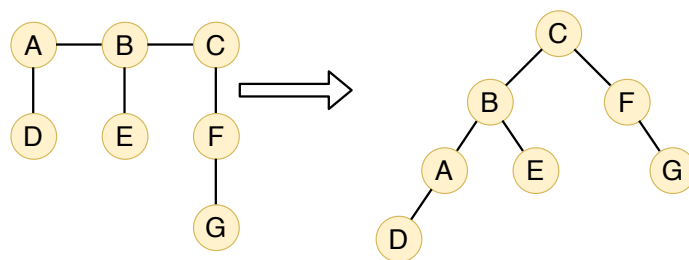
Scrivere un algoritmo:

int minHeigh(GRAPH G)

che prenda in input un grafo non orientato e connesso che rappresenta un albero **binario** non radicato, e restituisca l'altezza minima che si ottiene scegliendo uno dei nodi quale radice dell'albero.

Discutere informalmente la correttezza dell'algoritmo proposto e la sua complessità computazionale, discutendo anche casi ottimi e pessimi se sono diversi nel vostro caso.

Nella figura seguente è mostrato un possibile grafo di input e una sua "alberizzazione" di altezza 3. Altre "alberizzazioni" hanno altezze superiori, quindi l'algoritmo deve restituire 3.



Nota: non è richiesto di gestire alcun modo il caso in cui il grafo di input non sia rappresentabile tramite un albero binario.

## Algoritmi e Strutture Dati – 17/01/23 – Parte B

**Esercizio -1** Iscriverti allo scritto entro la scadenza. In caso di inadempienza, -1 al voto finale.

**Esercizio 0** Scrivere correttamente nome, cognome, numero di matricola, riga e colonna su tutti i fogli consegnati. Consegnare foglio A4 e foglio protocollo di bella. In caso di inadempienza, -1 al voto finale.

### Esercizio B1 – Griglia lineare – Punti $\geq 8$

Scrivere un algoritmo

```
int count(int n, int s)
```

che restituisca il numero di modi in cui è possibile riempire una griglia di dimensione  $1 \times n$  utilizzando pezzi di dimensione di dimensione  $1 \times 1, 1 \times 2, \dots, 1 \times s$ .

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

Per esempio, se  $s = 4$ , i pezzi a disposizione sono i seguenti:



Se  $n = 4$ , i modi per riempire la griglia sono i seguenti:



In altre parole, ci sono 8 modi per riempire una griglia  $1 \times 4$  con pezzi con dimensioni che vanno da 1 a 4.

Altri esempi:

- $s = 2, n = 4$ : 5 modi
- $s = 3, n = 6$ : 24 modi
- $s = 8, n = 4$ : 8 modi

### Esercizio B2 – Tape – Punti $\geq 10$

Sia  $f_1 \dots f_n$  una sequenza di  $n$  file memorizzati su un nastro magnetico. Siano  $d_1 \dots d_n$  le lunghezze (dimensioni) di ognuno dei file e siano  $p_1 \dots p_n$  le probabilità di utilizzo dei file. Le probabilità di utilizzo sono tali che  $p_i \leq 1, \forall i, 1 \leq i \leq n$  e  $\sum_{i=1}^n p_i = 1$ .

Poiché il nastro viene riavvolto fino all'inizio dopo ogni lettura, il *costo di lettura del file  $i$ -esimo*  $C_i$  è pari al costo necessario per raggiungere l'ultimo byte del file, cioè somma delle lunghezze di tutti i file che lo precedono più la lunghezza del file stesso:

$$c_i = \sum_{k=1}^i d_k$$

Il *costo medio* di lettura è dato dalla media dei costi di lettura dei file, pesati per la loro probabilità di utilizzo:

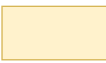

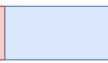

$$C = \frac{1}{n} \sum_{i=1}^n c_i \cdot p_i$$

L'ordinamento dei file influenza il costo medio; l'obiettivo è minimizzare tale costo. Leggendo su Internet, avete trovato due possibili strategie greedy per ordinare i file sul nastro:

- per ordine decrescente di probabilità di utilizzo;
- per ordine crescente di lunghezza;

Sapete bene però che le strategie greedy a volte funzionano e a volte no; per ognuna delle strategie, il vostro compito è dimostrare che sono ottime rispetto alla minimizzazione del costo medio, oppure trovare un controesempio (piccolo).

Per esempio, si consideri la disposizione dei file nella figura sottostante.

Probabilità	0.25	0.25	0.25	0.25
				
Lunghezza	100	100	100	100
Costo	100	200	300	400

Il costo medio di lettura è pari a:

$$\begin{aligned} C &= (100 \cdot 0.25 + 200 \cdot 0.25 + 300 \cdot 0.25 + 400 \cdot 0.25)/4 \\ &= (25 + 50 + 75 + 100)/4 \\ &= 250/4 = 62.5 \end{aligned}$$

### Esercizio B3 – Cardinalità – Punti $\geq 12$

Scrivere una funzione

```
minCardinality(int n)
```

che prenda in input un intero positivo  $n$  e stampi tutti i sottoinsiemi non vuoti di  $\{1, \dots, n\}$  il cui minimo è uguale alla dimensione dell'insieme.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

Per esempio, se  $n = 6$ , la funzione deve stampare (non necessariamente in quest'ordine):

$\{1\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{2, 6\}, \{3, 4, 5\}, \{3, 4, 6\}, \{3, 5, 6\}$