Prima esercitazione

La scadenza per la consegna dell'esercitazione è fissata per il **4 dicembre 2022 alle ore 23:59**.

La consegna delle soluzioni avviene attraverso **hackerrank**. Il link per accedere è https://hr.gs/es1asd2022-23 (password Es1ASD2022-23). **Iscrivetevi utilizzando la vostra email dell'università, la password qui sopra specificata e il numero di matricola.**
A prescindere dal tempo a disposizione specificato dal timer, il test terminerà una volta raggiunta la deadline specificata sopra.
Avete a disposizione tre bottoni per ogni esercizio:

- Run code, per eseguire il vostro codice su un caso di test fornitovi;

- Run tests, per eseguire i test che dovete superare per poter ottenere il punto dell'esercizio. Superare i test è una condizione necessaria ma non sufficiente per ottenere il punto dell'esercizio, tuttavia poi verrà esaminato manualmente il codice e i commenti relativi alla complessità.

- Submit, per sottomettere la vostra soluzione dell'esercizio. Potete comunque sottomettere altre soluzioni per il medesimo esercizio sempre riutilizzando il pulsante Submit. Verrà considerata l'ultima submission per la valutazione.

Nella home, dove vengono elencati i vari esercizi con il loro nomi, c'è il pulsante *Submit test*. Come vi verrà ricordato dal popup, se effettuate la submission dell'intera esercitazione, poi non potrete più modificare le vostre soluzioni. Fate attenzione.

Analisi della complessità: la vostra analisi delle complessità delle funzioni deve essere dettagliata, giustificata opportunamente (se per esempio utilizzate il teorema master, specificate i valori di a, b e così via) e posta come commento nel codice. Analisi e risultati non motivati opportunamente o scorretti renderanno scorretta la soluzione dell'esercizio e verranno attribuiti 0 punti.

Ad ogni esercizio viene assegnato un punto. Il punto viene attribuito solo se TUTTE le implementazioni e TUTTE le relative complessità sono corrette. Se risulta impossibile testare la soluzione per via di errori di sintassi nel codice, verranno attribuiti 0 punti alla soluzione.

Per qualsiasi domanda, chiedete al tutor via email o all'incontro settimanale di tutorato.

Esercizio 1

Scrivere una funzione **EFFICIENTE blackHeight(u)** che dato in input la radice **u** di un albero binario, i cui nodi **x** hanno, oltre ai campi **key**, **left** e **right**, un campo **col** che può essere **'B'** (per “black”) oppure **'R'** (per “red”), verifica se *per ogni nodo*, il cammino da quel nodo a qualsiasi foglia contiene lo stesso numero di nodi *neri* (*altezza nera* del nodo x). In caso negativo, restituisce -1, altrimenti restituisce l’*altezza nera* della radice.

Il prototipo della funzione è:

```
int blackHeight(PNode u)
```

Valutare la complessità della funzione, indicando eventuali relazioni di ricorrenza.

Il tipo PNode è così definito:

```
struct Node{
  int key;
  char col;
  Node* left;
  Node* right;
  Node(int k, char c, Node* sx = nullptr, Node* dx = nullptr)
    : key{k}, col{c}, left{sx}, right{dx} {}
};

typedef Node* PNode;
```

Esercizio 2

Sia **T** un albero generale i cui nodi hanno campi: **key**, **left-child** e **right-sib**. Scrivere una funzione **EFFICIENTE** che dato in input la radice dell'albero **T** e un intero **k** (**k>=2**) verifica se **T** è un albero **k-completo**.

Il prototipo della funzione è:

```
bool k_Completo(PNodeG r, int k)
```

Analizzare la complessità della funzione.

Il tipo PNodeG è cosi' definito:

```
struct NodeG{
  int key;
  NodeG* left_child;
  NodeG* right_sib;
  NodeG(int k, NodeG* sx = nullptr, NodeG* dx = nullptr)
    : key{k}, left_child{sx}, right_sib{dx} {}
};

typedef NodeG* PNodeG;
```

Esercizio 3

Sia **T** un albero ternario completo i cui nodi sono colorati di **bianco** (carattere **'w'**) o **nero** (carattere **'b'**). L'albero è rappresentato tramite un **vettore posizionale**. Scrivere una funzione **EFFICIENTE** che ritorni la *lunghezza* del *cammino più lungo* all'interno dell'albero con nodi che presentano tutti lo *stesso colore* (il cammino può partire da un nodo qualsiasi, non necessariamente dalla radice) e il *colore* dei nodi del cammino più lungo.

Il prototipo della funzione è:

```
int max_l_cammino_monocolore(const vector<char>& tree, char& colore_max_cammino)
```

Analizzare la complessità della funzione.

Esercizio 4

N barre di marmo di lunghezza **L = [L_0, L_1, ... , L_{N-1}]** in metri sono posizionate una accanto all'altra.
Ci sono **K** macchine tagliatrici, posizionate una accanto all'altra, in grado di trasformare una barra in cubetti di marmo.
Ogni macchina elabora barre di marmo a velocità diverse **S = [S_0, S_1, ..., S_{K-1}]** in metri/sec.

Le macchine tagliatrici funzionano con i seguenti vincoli:

- una barra può essere elaborata da *una sola* macchina;
- una macchina può elaborare *più barre, una dopo l'altra*;
- una macchina, dopo aver completato una barra, può elaborare solo la *barra adiacente alla sua destra* (quindi una macchina può elaborare *SOLO sequenze consecutive di barre*);
- le macchine possono lavorare *in parallelo*;
- l'ordine di barre e macchine *non può essere modificato*.

Scrivere una funzione minTime che restituisce il tempo minimo richiesto per elaborare tutte le barre e inserisce nel vettore **ass** l'assegnazione delle barre alle macchine (ovvero **ass[i] = j** quando la barra **i** è assegnata alla macchina **j**)

Il prototipo della funzione è:

```
int minTime(const vector<int>& len, const vector<int>& speed, vector<int>& ass)
```

Analizzare la complessità.

Esercizio 5

Ai campionati mondiali di maratona il numero di partecipanti è così alto che occorre organizzare la gara in due turni in giorni diversi.
Si costruiscono così le due classifiche, una per turno, ciascuna delle quali riporta i partecipanti a quel turno in ordine di arrivo e il tempo corrispondente.

Dopo le gare, *SENZA unificare* le due classifiche, si vuole avere una tecnica efficiente per determinare, dato una posizione **k**, l'atleta che si è piazzato in quella posizione nella classifica generale. ovvero che ha ottenuto il **k-mo** tempo *più* basso.

Il prototipo della funzione è:

```
int estrai(const vector<int>& classifica1, const vector<int>& classifica2, int k)
```

Restituire -1 se k non indica un elemento presente all'interno delle sequenze. Analizzare la complessità della funzione.

Ultime modifiche: martedì, 22 novembre 2022, 11:28

◄ Registrazione delle lezioni

Vai a...

Terzo Appello ►