03AAX Algoritmi e Strutture Dati

Appello del 07/02/2023 - Prova di programmazione (18 punti)

1. (**18 punti**)

In un noto videogioco, il giocatore deve superare delle prove di *hacking* inserendo delle sequenze di token esadecimali di due caratteri in un buffer di lunghezza fissa per ottenere vari bonus. Si consideri la seguente descrizione del mini-gioco.

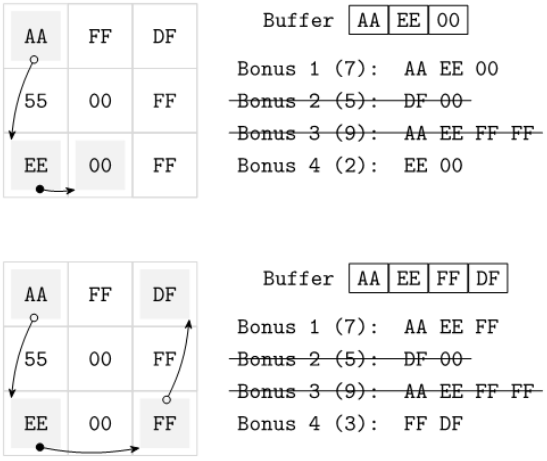
I token vengono scelti da una matrice quadrata di dimensione `N x N`. Ogni token della matrice può essere usato una singola volta, ma nella matrice possono apparire dei duplicati. La scelta dei token avviene alternando una serie di movimenti per riga e per colonna, partendo sempre con una scelta iniziale sulla prima riga. I movimenti possono interessare un qualsiasi token lungo una certa riga/colonna, indipendentemente dalla distanza dalla posizione corrente. Non sono mai ammessi movimenti in diagonale. Lo schema di movimento/scelta è quindi simile alla sequenza a seguire:

- Il primo token (scelta di indice zero) viene sempre scelto sulla prima riga
- Il secondo token (scelta di indice uno) viene scelto nella colonna in cui si sia scelto il token precedente
- Il terzo token (scelta di indice due) viene scelto nella riga in cui si sia scelto il token precedente
- E così via, alternando colonna/riga nelle mosse a seguire...

I bonus ottenuti dal giocatore sono associati a delle stringhe obiettivo, composte anch'esse da token esadecimali. Per semplicità, si assuma che il bonus sia un valore intero positivo. Il giocatore ottiene tutti i bonus la cui stringa associata appaia come sottostringa nel buffer riempito dal giocatore.

Il programma riceve in input la lunghezza `L` del buffer, un file contenente la matrice di gioco `grid.txt` e un file contente la lista di bonus disponibili `bonus.txt`. Il file `grid.txt` riporta sulla prima riga la dimensione `N` della matrice e a seguire `N` righe di `N` token esadecimali separati da un singolo spazio. Il file `bonus.txt` riporta sulla prima riga il numero di `B` bonus disponibili e a seguire `B` righe nel formato: `<numero_token> <valore_bonus> <sequenza_di_token>`

Esempi:



Si faccia riferimento all'immagine proposta, assumendo che per il primo esempio si abbia `L=3` e per il secondo esempio si abbia `L=4`. **(Es. 1, sopra)** il primo e quarto bonus sono entrambi ottenibili introducendo la sequenza `AA EE 00` (che corrisponde alla selezione dei token di posizione `[0,0],[2,0],[2,1]`), che rappresenta anche la sequenza a valore massimo che si possa introdurre. Il secondo bonus non è ottenibile poiché non c'è nessuna sequenza di movimenti, indipendentemente dalla lunghezza del buffer, che permetta di introdurre `DF 00` nel buffer stesso. Il terzo bonus non è ottenibile poiché il buffer ha lunghezza `L=3`. Buffer più lunghi avrebbero reso possibile ottenere quel bonus poiché la sequenza corrispondente esiste come sequenza di mosse valide nella griglia (`[0,0],[2,0],[2,2],[1,2]`). **(Es. 2, sotto)** il buffer di lunghezza quattro permetterebbe di completare tutte le sequenze, individualmente o meno, sempre con l'eccezione della seconda che non è realizzabile data la griglia di input. Nell'immagine è proposta una scelta sub-ottima, che copre il primo e il quarto bonus. L'ottimo sarebbe stato raggiunto preferendo la scelta dei quattro token `AA EE FF FF` (terminando in [1,2] anziché [0,2]) che coprirebbe contemporaneamente sia il Bonus 1 sia il Bonus 3, anziché Bonus 1 e Bonus 4 come invece si è scelto nella configurazione rappresentata.
Nelle immagini si sono distinte mosse obbligate per riga (colonna) con il pallino pieno (vuoto) come marker di inizio della freccia.

Richieste:

- **(Strutture Dati)** Definire e implementare opportune strutture dati reputate necessarie a modellare le informazioni del problema secondo quanto richiesto e le funzioni di acquisizione dei dati stessi. In caso di organizzazione delle strutture dati su più file, indicare esplicitamente il modulo di riferimento.
- **(Verifica)** Data una sequenza di scelte sulla griglia, valutare che questa rispetti i vincoli di movimento esposti e determinare la composizione del buffer. Se valida, valutare il valore della somma dei bonus ottenibili. Il formato dei dati in input per questa richiesta è a discrezione del candidato, che è tenuto a fornirne anche una breve descrizione.
- **(Ottimizzazione)** Identificare la scelta di token con cui riempire il buffer che massimizzi la somma dei bonus ottenibili.

03AAX Algoritmi e Programmazione

Appello del 07/02/2023 - Prova di programmazione (12 punti)

1. (2 punti)

Sia data una matrice `M` di dimensione `r x c` contenente elementi interi. Scrivere una funzione che generi una matrice `M'` di dimensione `r x c` derivata da `M` sostituendo il valore di ogni cella con la media di tutti i suoi vicini (diagonali incluse) e la cella stessa. Per gli angoli la media sarà quindi tra 4 valori, per le celle lungo i bordi (angoli esclusi) tra 6 elementi e per tutte le altre celle tra 9 elementi. La matrice `M'` sia allocata dentro alla funzione. Completare opportunamente il prototipo in modo che la nuova matrice sia disponibile al chiamante.

```
void f(int **M, int r, int c, ...);
```

Esempio:

$$M = \begin{Bmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{Bmatrix} \qquad M' = \begin{Bmatrix} 1.25 & 1.33 & 1.25 \\ 1.33 & 1.33 & 1.33 \\ 1.25 & 1.33 & 1.25 \end{Bmatrix}$$

2. (4 punti)

Si scriva una funzione wrapper `int f(T t)` (e la relativa funzione ricorsiva) che ricevuto in input un albero n-ario di interi `t` di tipo `T` conti la lunghezza del cammino più lungo contenente solo valori positivi o nulli. Fornire la definizione del tipo `T` e del tipo nodo al suo intero, come ADT di prima classe e come quasi ADT rispettivamente, indicando esplicitamente la divisione in moduli adottata. Per rappresentare l'albero richiesto, ogni nodo tiene traccia dei figli mediante un vettore di puntatori a nodo e il numero di figli.

Non è ammesso l'utilizzo di funzioni di libreria.

3. (6 punti)

Una matrice `M` quadrata di dimensione `N x N` rappresenta le distanze dirette tra `N` città. Ogni cella contenente un valore positivo `d` indica che la coppia di città `(i,j)` dista `d`. Un valore pari a zero indica l'assenza di una connessione diretta tra le due città. Scrivere una funzione ricorsiva in grado di suddividere le città nel minor numero di gruppi possibili facendo in modo che tutte le città in un gruppo siano mutualmente raggiungibili (direttamente o transitivamente) attraversando al massimo una città intermedia. È ammesso che ci siano gruppi composti da una singola città.

PER ENTRAMBE LE PROVE DI PROGRAMMAZIONE (18 o 12 punti):

- ***indicare nell'elaborato e nella relazione nome, cognome e numero di matricola.***
- ***se non indicato diversamente, è consentito utilizzare chiamate a funzioni standard, quali ordinamento per vettori, funzioni su FIFO, LIFO, liste, BST, tabelle di hash, grafi e altre strutture dati, considerate come librerie esterne.***
- ***gli header file devono essere allegati all'elaborato (il loro contenuto riportato nell'elaborato stesso). Le funzioni richiamate, inoltre, dovranno essere incluse nella versione del programma allegata alla relazione. I modelli delle funzioni ricorsive non sono considerati funzioni standard.***
- ***consegna delle relazioni (per entrambe le tipologie di prova di programmazione): entro il 10/02/2023, alle ore 14:00, mediante caricamento su Portale. Le istruzioni per il caricamento sono pubblicate sul Portale nella sezione Materiale). QUALORA IL CODICE CARICATO CON LA RELAZIONE NON COMPILI CORRETTAMENTE, VERRÀ APPLICATA UNA PENALIZZAZIONE. Si ricorda che la valutazione del compito viene fatta, senza la presenza del candidato, sulla base dell'elaborato svolto in aula. Non verranno corretti i compiti di cui non sarà stata inviata la relazione nei tempi stabiliti.***