Problem Set 3

docente: Luciano Gualà

Esercizio 1
Sia T un albero binario di n nodi con radice r in cui ogni nodo ha un valore non negativo associato. La *profondità* di un nodo v è il numero di archi del cammino da v alla radice. I nodi che si incontrano lungo tale cammino (v compreso) sono detti *antenati* di v. Diremo che un nodo v è *generazionalmente profondo* se la sua profondità è strettamente maggiore del valore di un suo antenato di valore minimo.

Si assuma che T è mantenuto attraverso una struttura collegata e che ogni nodo v abbia associato i seguenti campi: puntatori al padre e ai figli ($v.p$, $v.s$, $v.d$) e valore del nodo ($v.val$). Si progetti un algoritmo con complessità temporale $O(n)$ che, preso T, restituisca il numero di nodi generazionalmente profondi di T. Si fornisca lo pseudocodice dettagliato dell'algoritmo.

Esercizio 2 *(Aggiungendo un'operazione a una Pila)*
Progettare una struttura dati che implementa un tipo di dato *Pila* che mantiene una sequenza di elementi con chiave e che, oltre le classiche operazioni di `Top`, `Pop` e `Push`, consente un'operazione aggiuntiva chiamata `Min`. Tale operazione restituisce il puntatore all'elemento di chiave minima contenuto nella pila. Tutte le operazioni devono avere complessità temporale $O(1)$ nel caso peggiore.

Esercizio 3 *(Un oracolo per il problema del* Minimum Range Query*)*
Sia A un vettore di n valori reali. Progettare un algoritmo che, dato A, costruisca un *oracolo* (ovvero una struttura dati) che sia in grado di rispondere in tempo $O(1)$ a *query* (ovvero domande) del seguente tipo: dati due interi i, j, calcolare l'indice dell'elemento di valore minimo nella porzione $A[i; j]$ del vettore.

Si noti che una soluzione semplice al problema è quella di precalcolare tutte le risposte alle $\Theta(n^2)$ query e memorizzarle in una matrice. In questa soluzione, però, l'oracolo (ovvero la matrice delle risposte) ha dimensione $\Theta(n^2)$. Vogliamo, invece, fare meglio in termini di memoria occupata dall'oracolo la cui dimensione richiediamo essere $O(n \log n)$. Non imponiamo invece nessun vincolo sulla complessità temporale necessaria per costruire l'oracolo.

Suggerimento: un'idea potrebbe essere quella di memorizzare solo le risposte a un sottoinsieme delle $\Theta(n^2)$ query. Tale sottoinsieme deve avere dimensione $O(n \log n)$ e deve comunque consentire di rispondere a una generica query in tempo costante.

Esercizio 4 *(Un algoritmo di programmazione dinamica per aiutare George Martin)*
La prossima stagione del Trono di Spade durerà n puntate e George R. R. Martin vuole aumentare la propria popolarità. Sa come fare: farà morire alcuni dei suoi personaggi. Sa che se fa morire un personaggio nella puntata i, guadagnerà $p_i \geq 0$ punti in popolarità. Se ne facesse morire due, sempre nella puntata i, l'effetto drammatico sarebbe più forte, ma non il *doppio* più forte; infatti guadagnerebbe $p_i + \frac{p_i}{2}$ punti in popolarità. In generale, se facesse morire j personaggi nella puntata i il suo guadagno in punti di popolarità sarebbe

$\sum_{t=1}^{j} \frac{p_i}{2^{t-1}}$. Come a dire, l'effetto drammatico della violenza decresce esponenzialmente all'aumentare dei personaggi che muoiono. Ci si abitua a tutto, del resto. Inoltre Martin ha un altro problema: sa che il pubblico, ormai affezionato al suo sadismo, smetterebbe di guardare la serie se ci fossero più di due puntate consecutive senza un morto e, come se non bastasse, questa non sarà l'ultima stagione della serie TV, quindi è bene che qualche personaggio a cui il pubblico è affezionato resti in vita. Ancora per un po', almeno. Martin, a tal proposito, ha stimato che, nell'economia della narrazione, può sacrificare fino a k personaggi. Ma quando? Aiutatelo progettando un algoritmo di programmazione dinamica che calcoli la strategia che gli faccia guadagnare il più possibile in popolarità.