Programmazione 1	Esercitazione 6

Cognome: **Nome:** **Matricola:**

1. Scrivere un predicato `IsSet(As, P)` che controlla se la lista `As` rappresenta un insieme di oggetti, ovvero la lista non contiene nessuna coppia di elementi uguali. Due elementi 'x' e 'y' sono considerati uguali, se il predicato `P(x, y)` vale `True`.

   Qual è la complessità dell'algoritmo che avete usato per implementare questa funzione?

2. Scrivere una funzione `InsertAt(As, value, i)` che inserisce nella lista `As` l'elemento `value` in posizione `i`.

   Qual è la complessità dell'algoritmo che avete usato per implementare questa funzione?

3. Scrivere una funzione `Insert(As, z, Cmp=lambda x,y: x<y)` che inserisce nella lista (già ordinata) `As` l'elemento `value`, rispettando le seguenti regole: se l'elemento è già presente nella lista non viene aggiunto, altrimenti l'elemento viene aggiunto tra due valori 'x' e 'y' di `As` in modo tale che valga la relazione $x < z < y$, o più in generale `Cmp(x,z) == True` e `Cmp(z,y) == True`.

   Qual è la complessità dell'algoritmo che avete usato per implementare questa funzione?

4. Scrivere una funzione `Contains(As, z)` che prende in input una lista di numeri interi ordinata in ordine crescente e controlla se 'z' è contenuta nella lista As. Scrivere la funzione in modo che esegua in tempo $O(\log(n))$, dove n è la lunghezza della lista.

5. Si supponga di voler rappresentare un vettore di $\mathbb{R}^n$ come una lista di numeri `float`, e le matrici come liste di vettori, ovvero le righe della matrice. Usando questa rappresentazione, possiamo utilizzare delle operazioni sulle liste di liste per esprimere operazioni base tra vettori e matrici.

   Si chiede di implementare le seguenti operazioni fondamentali come operazioni tra liste, e liste di liste:

   (a) **DotProduct**: prodotto tra due vettori, componente per componente. Dati due vettori x e y in $\mathbb{R}^n$ calcolare lo scalare $p = \sum_{i=1,\dots,n} x_i y_i$.

   (b) **MatrixVector**: prodotto tra matrice e vettore. Data una matrice A di dimensione $m \times n$ e un vettore x in $\mathbb{R}^n$ calcolare il vettore t, in cui $t_i = \sum_{j=1,\dots,n} a_{ij} x_j$.

   (c) **MatrixMatrix**: prodotto tra due matrici. Date le due matrici A di dimensione $m \times n$ e B di dimensione $n \times m$ calcolare la matrice P, in cui $p_{ij} = \sum_{k=1,\dots,n} a_{ik} b_{kj}$.

   (d) **Transpose**: matrice trasposta. Data la matrice A di dimensione $m \times n$ calcolare la sua trasposta, ovvero la matrice B in cui $b_{ij} = a_{ji}$.

6. **CHALLENGE 4:** Il problema delle n-regine richiede di posizionare n regine su una scacchiera di dimensione $n \times n$, in modo tale che nessuna regina ne tenga un'altra sotto scacco. Si eseguano in ordine i punti seguenti:

   (a) Decidere che struttura dati usare per rappresentare una soluzione.

   (b) Modificare la funzione `DrawMatrix` vista per i frattali in modo tale che dia una rappresentazione grafica di una soluzione: il nero rappresenta la posizione di una regina, il bianco rappresenta una posizione vuota.

   (c) Scrivere una funzione, che dato un posizionamento delle regine rappresentato con la struttura dati scelta, controlli se il posizionamento è ammissibile, ovvero nessuna regina ne tiene un'altra sotto scacco.

   (d) Scrivere una funzione che prende in input un numero interno $n \geq 4$ e restituisce in output una soluzione al problema delle n-regine.

   (e) Qual è la complessità dell'algoritmo che avete usato per implementare la vostra soluzione?

   Per la **CHALLENGE**, potete mandare l'implementazione in python per email (facoltativo).