

Introducción a la Programación

Segundo Parcial

1) Intercalar [2 puntos]

A la hora de jugar juegos de cartas, como el truco, el tute, o el chinchón, es importante que la distribución de las mismas en el mano sea aleatoria. Para esto, al comienzo de cada mano, antes de repartir las mismas se realizan mezclas sucesivas. Una técnica de mezclado es la denominada "mezcla americana" que consiste en separar el mazo en (aproximadamente) dos mitades e intercalar las cartas de ambas mitades. Implementar la función intercalar que dadas dos listas $s1$ y $s2$ con igual cantidad de elementos devuelva una lista con los elementos intercalados. Esto es, las posiciones pares de res tendrán los elementos de $s1$ y las posiciones impares los elementos de $s2$, respetando el orden original.

```
problema intercalar (in s1: seq<Z>, in s2: seq<Z>) : seq<Z> {  
  requiere: {|s1| = |s2| }  
  asegura: {|res| = 2 * |s1|}  
  asegura: {res[2*i] = s1[i] y res[2*i+1] = s2[i], para i entre 0 y |s1|-1}  
}
```

TIP: realizar la iteración mediante índices y no mediante elementos

Por ejemplo, dadas

$s1 = [1, 3, 0, 1]$

$s2 = [4, 0, 2, 3]$

se debería devolver $res = [1, 4, 3, 0, 0, 2, 1, 3]$

2) Posición de n-ésima aparición [2 puntos]

Guido y Marcela son dos estudiantes de IP, nerviosos con el parcial de Python. Con el objetivo de tener un rato antes del parcial para preguntarse algunas dudas deciden encontrarse en el colectivo y viajar juntos. Para poder coordinar de forma exacta en qué colectivo se tienen que subir, Marcela usa sus habilidades de programación aprendidas en IP para acceder de forma poco legítima a la base de datos de colectivos de todas las empresas. Con esto, arma una lista de todos los colectivos que van a pasar por la parada de Guido alrededor del horario acordado y le indica a Guido que se tiene que subir en el 3er colectivo de la línea 34. Por desgracia, Guido se olvida sus lentes antes de salir y no es capaz de distinguir a qué línea pertenece cada colectivo que llega a la parada. Por lo que solo puede contar cantidad total de colectivos que pasan.

Implementar la función `pos_nesima_aparicion` que dada una secuencia de enteros s , y dos enteros n y $elem$ devuelve la posición en la cual $elem$ aparece por n -ésima vez en s . En caso de que $elem$ aparezca menos de n veces en s , devolver -1.

```

problema pos_nesima_aparicion (in s: seq<Z>, in n: Z, in elem: Z) : Z {
  requiere: {n>0}
  asegura: {Si el elemento aparece menos de n veces en la secuencia, res= -1 }
  asegura: {Si el elemento aparece al menos n veces en la secuencia, s[res] = elem }
  asegura: {Si el elemento aparece al menos n veces en la secuencia, elem aparece n-
1 veces en s entre las posiciones 0 y res-1 }
}

```

Por ejemplo, dadas
 $s = [-1, 1, 1, 5, -7, 1, 3]$
 $n = 2$
 $elem = 1$
 se debería devolver $res = 2$

3) Matriz espejada [3 puntos]

Implementar la función `matriz_espejada` que dada una matriz devuelve `True` si cada una de sus filas es capicúa. Es decir, si cada fila es igual leída de izquierda a derecha o de derecha a izquierda. Definimos a una secuencia de secuencias como matriz si todos los elementos de la primera secuencia tienen la misma longitud.

```

problema matriz_espejada(in m:seq<seq<Z>> ) : Bool {
  requiere: {todos los elementos de m tienen igual longitud (los elementos de m son
secuencias)}
  asegura: {(res = true) <=> todos los elementos de m son capicúa}
}

```

Por ejemplo, dada la matriz
 $m = [[1,2,2,1],[-5,6,6,-5],[0,1,1,0]]$
 se debería devolver $res = true$

4) En el hipódromo: posiciones de los caballos [3 puntos]

Además de recitales de artistas de renombre internacional (ej: Bizarrap), en el hipódromo de Palermo se realizan cotidianamente carreras de caballos. Por ejemplo, durante el mes de Octubre se corrieron 10 carreras. En cada una de ellas participaron alrededor de 10 caballos.

Implementar la función `cuenta_posiciones_por_caballo` que dada la lista de caballos que corrieron las carreras, y el diccionario que tiene los resultados del hipódromo en el formato `carreras:posiciones_caballos`, donde `carreras` es un `String` y `posiciones_caballos` es una lista de strings con los nombres de los caballos, genere un diccionario de `caballos:#posiciones`, que para cada caballo devuelva la lista de cuántas veces salió en esa posición.

Tip: para crear una lista con tantos ceros como caballos se puede utilizar la siguiente sintaxis `lista_ceros = [0]*len(caballos)`

```
problema cuenta_posiciones_por_caballo(in caballos: seq<String>, in carreras:  
dict<String,seq<String>>: dict<String,seq<Z>> {  
  requiere: {caballos no tiene repetidos}  
  requiere: {Los valores del diccionario carreras son permutaciones de la lista caballos (es  
  decir, tienen exactamente los mismos elementos que caballos, en cualquier orden posible)}  
  asegura: {res tiene como claves los elementos de caballos}  
  asegura: {El valor en res de un caballo es una lista que indica en la posición i cuántas  
  veces salió ese caballo en la i-ésima posición.}  
}
```

Por ejemplo, dados

```
caballos= ["linda", "petisa", "mister", "luck" ]  
carreras= {"carrera1":["linda", "petisa", "mister", "luck"],  
           "carrera2":["petisa", "mister", "linda", "luck"]}  
se debería devolver res = {"petisa": [1,1,0,0],  
                           "mister": [0,1,1,0],  
                           "linda": [1,0,1,0],  
                           "luck": [0,0,0,2]}
```