Trabajo Práctico # de C++

Estructuras de Datos, Universidad Nacional de Quilmes

27 de octubre de 2018

1. Introducción

Ejercicio 1

Definir el tipo abstracto de datos Pokemon como un número entero que representa la vida y un nombre. Su interfaz es la siguiente:

- `Pokemon crearPokemon(string nombre, int vida)`
- `string getNombre(Pokemon p)`
- `int getVida(Pokemon p)`
- `void cambiarNombre(Pokemon& p, string nombre)`
- `bool estaVivo(Pokemon p)`
- `void restarVida(Pokemon& p)`
  Le resta una unidad a la vida.
- `void lucharN(int n, Pokemon& p, Pokemon& r)`
  Le resta vida "n" veces a los dos pokemones. Pensar una solución que sea $O(n)$ y otra que sea $O(1)$.
- `void destruir(Pokemon& p)`
  Libera memoria

Implementar esta interfaz destructiva, utilizando memoria dinámica.

Ejercicio 2

Definir un tipo abstracto para representar fracciones, con la siguiente interfaz:

- `Fraccion fraccion(int x, int y)`
- `Fraccion sumar(Fraccion f1, Fraccion f2)`
- `Fraccion restar(Fraccion f1, Fraccion f2)`
- `Fraccion multiplicar(Fraccion f1, Fraccion f2)`
- `Fraccion dividir(Fraccion f1, Fraccion f2)`
- `void invertir(Fraccion& f1)`
  Invierte numerador y denominador.
- `void sumplificar(Fraccion& f1)`
- `void destruir(Fraccion& f1)`

Ejercicio 3

Definir el tipo abstracto Maybe utilizando una interfaz destructiva (operaciones nothing, just, fromJust e isNothing).

2. Listas, Set y Queue

Ejercicio 4

Implementar como usuario de LinkedList las siguientes funciones:

- `int sumar(List xs)`
- `int length(List xs)`
- `List mapSucc(List xs)`
- `List take(int n, List xs)`
- `List drop(int n, List xs)`

Ejercicio 5

Definir como usuario de listas iterables las primeras 12 funciones del punto 2.1. de la Práctica 1.

Ejercicio 6

Definir el tipo abstracto Set con la siguiente interfaz destructiva:

- `Set emptyS()`
- `Set singleton(int x)`
- `bool belongs(int x, Set s)`
- `void addS(int x , Set& s)`
- `void removeS(int x, Set& s)`
- `int size(Set s)`
- `Set unionS(Set s1, Set s2)`
- `Set insersectS(Set s1, Set s2)`
- `List setToList(Set s)`
- `void destroySet(Set& s)`

Ejercicio 7

Definir el tipo abstracto Queue con la siguiente interfaz destructiva:

- `Queue emptyQ()`
- `bool isEmptyQ(Queue q)`
- `void queue(Queue& q, int x)`
- `int firstQ(Queue q)`
- `void dequeue(Queue& q)`

- `Queue copiar(Queue q)`
- `void destroyQ(Queue& q)`

Todas las operaciones deben costar $O(1)$, a excepción de `destroyQ`. ¿Esto era posible en Haskell? Luego como usuario de esta estructura definir las funciones del 1.1 de la Práctica 4.

3. Árboles

Ejercicio 8

Definir una representación de árbol binario en C++ e implementar los primeros 7 items del ejercicio 1 de la Práctica 3, utilizando recursión.

Ejercicio 9

Definir las mismas funciones del apartado anterior pero de forma iterativa (sin recursión).

Ejercicio 10

Implementar el ejercicio de expresiones aritméticas en C++, utilizando una interfaz funcional como en Haskell.

4. Arrays

Ejercicio 11

Implementar las primeras 15 funciones del punto 2.1 de la Práctica 1 pero utilizando arrays en lugar de listas.

Ejercicio 12

Definir el tipo abstracto ArrayList, una lista que posee la misma interfaz que las listas recorribles pero cuya representación es un array.

Ejercicio 13

Definir el tipo abstracto Set utilizando un array de booleanos. Luego como usuario de esta estructura definir las funciones del 3.1 de la Práctica 4.

Ejercicio 14

Definir el tipo abstracto Multiset utilizando una técnica muy similar a la del ejercicio anterior.

Ejercicio 15

Implementar una versión eficiente del ejercicio de MiniTablero de la práctica 5, pero esta vez implementando un tablero de 2 dimensiones.

5. Heaps

Ejercicio 16

Definir el tipo abstracto BinaryHeap visto en la teórica.

6. Hashing

Ejercicio 17

Definir el tipo abstracto HashMap, que es un Map cuya implementación utiliza una tabla de Hash.

Ejercicio 18

Definir el tipo abstracto HashSet, que es un Set cuya implementación utiliza una tabla de Hash.

7. Sorting

Ejercicio 19

Implementar los distintos sortings vistos en la teórica, incluyendo heapsort.