Universidad de Granada

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIA

DECSAI ETSII Informática y de Telecomunicación, C/ Periodista Daniel Saucedo Aranda s/n- 18071- Granada (España)

Estructuras de Datos
Curso 2013-2014. Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

1. (1 punto) Dado el *TDA Montón de Cartas*, con N cartas de la baraja española, y con las siguientes operaciones:
   - **Barajar:** Dispone las N cartas en orden aleatorio
   - **CogerCarta:** Devuelve la carta en el tope del montón
   - **EliminarCarta:** Elimina la carta en el tope del montón
   - **InsertarCarta**: Inserta una carta al final del montón

   a) Deducir la eficiencia de las cuatro funciones suponiendo que representamos el *TDA Montón de Cartas como: 1)Vector, 2)Lista, 3) Pila y 4) Cola*
   b) *Implementar la función Barajar suponiendo que el TDA Montón de Cartas se representa como: 1)Vector, 2)Lista, 3) Pila y 4) Cola. Nota: Se puede usar si es necesario objetos contenedores auxiliares para su implementación.*
2. (1 punto) Dados dos multiset con elementos enteros, implementar la función:
   multiset<int> multi_interseccion (const multiset<int> & m1, const multiset<int> &m2)
   que calcula la intersección de dos multiset: *elementos comunes en los dos multiset repetidos tantas veces como aparezcan en el multiset con menor número de apariciones del elemento.*
   Por ejemplo siendo m1={2,2,3,3} y m2={1,2,3,3,3,4} entonces m1∩m2={2,3,3} ó si m1={2,2,2,3,3} y m2={1,2,2,2,2,3,3,3,4} entonces m1∩m2={2,2,2,3,3}
3. (2 puntos) Suponed que tenemos el T.D.A. Tabla Hash abierta (unordered_set) (class TH), en la que la resolución de colisiones se hace utilizando para cada cubo una lista.

```
1. #include <vector>
2. #include <list>
3. using namespace std;
4. class TH{
5. private:
6.  struct info{
7.   int key;//clave
8.   int di;//dirección
9.  };
10. vector <list<info> > data;
11. int fhash(int k)const;
12. bool recolocar()const;
13. public:
14.       ...
15./* k la clave, d la direccion
   asociada para esa clave*/
16.void insertar(int k, int d);
17.  ...
18. class iterator{
19. private:
20. list<info>::iterator it_cub;
21. vector<list<info> >::iterator it;
22.           ...
23. }
24. iterator begin();
25. iterator end();
26.           ...
27.};
```

   a) Implementar la función ***insertar*** suponiendo que tenemos implementada la función hash (***fhash***). Después de añadir la nueva clave k y su dirección asociada d, la función ***insertar*** debe comprobar si es necesario redimensionar la tabla hash. Para ello se supone que tenemos implementada la función ***recolocar***. Esta función devuelve verdadero en el caso que sea necesario pasar todos los datos a un nueva tabla de mayor tamaño, y falso en caso contrario. El tamaño de la nueva tabla tiene que ser el primo más cercano a 2M por exceso, siendo M el tamaño original.
   b) Implementar la ***clase iterator*** (un iterador sobre todos los elementos de la tabla hash) de la

tabla hash, así como las funciones ***begin*** y ***end*** de la clase TH.

4. (2 puntos)
   a) Dado un árbol binario de enteros (positivos y negativos) implementar una función que obtenga el número de caminos, en los que la suma de las etiquetas de los nodos que los componen sumen exactamente **k.**

      int NumeroCaminos(bintree<int> & ab, int k)

   b) Construir el AVL y el APO que resultan de insertar (en ese orden) los elementos del conjunto de enteros {45,23,12,20,15,22,24,55,52}.
5. (1 punto) Describid las similitudes y diferencias (razonando la respuesta) en el funcionamiento del método básico de inserción de un elemento el los siguientes tipos de datos abstractos : 1) vector, 2) list, 3) map, 4) set, 5) priority_queue, 6) tabla hash abierta (unordered_set).

Tiempo: 3 horas