Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

FICH
UNL

Algoritmos y Estructuras de Datos.
Parcial 2. [2018-11-13]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 TORVALDS, LINUS

[Ej. 1] [CLAS2 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[set-vb]:** Dada la siguiente interfaz para un conjunto (set) por vector de bits que contiene pares de letras mayúsculas (de `"AA"` a `"ZZ"`, representadas como strings de longitud 2):

```
typedef string elem_t;
int index(elem_t t);
elem_t element(int j);
typedef int iterator_t;

class set {
private:
    vector<bool> v;
    iterator_t next_aux(iterator_t p);
public:
    set();
    pair<iterator_t,bool> insert(elem_t x);
    elem_t retrieve(iterator_t p);
    void erase(iterator_t p);
    bool erase(elem_t x);
    iterator_t find(elem_t x);
    iterator_t begin();
    iterator_t end();
    iterator_t next(iterator_t p);
};
```

implemente los métodos **`begin`**, **`end`**, **`next`** e **`insert`** (si dentro de alguno de ellos utiliza otro método de set, también deberá implementarlo) y las funciones **`index`** y **`element`**.

b) **[set-abb]:** Dada la siguiente interfaz para un conjunto (set) por árbol binario de búsqueda:

```
template<class T>
class set {
private:
    typedef btree<T> tree_t;
    typedef typename tree_t::iterator node_t;
    tree_t bstree;
    ...
public:
    class iterator {
            note_t node;
            tree_t &bstree;
    private:
                friend class set;
                iterator(tree_t &t, node_t n) : bstree(t), node(n) {}
        ...
```

```
16      };
17      ...
18      pair<iterator,bool> insert(T x);
19      iterator find(T x);
20      int erase(T x);         //<- implementar
21      iterator begin();       //<- implementar
22      iterator end();         //<- implementar
23      node_t next(node_t  ); //<- implementar
24      ...
25  };
```

implemente los métodos **begin**, **end**, **next** y **erase** (si dentro de alguno de ellos utiliza otro método de set, también deberá implementarlo).

[Ej. 2] [OPER2 (W=20pt)]

a) **[heap-sort (5pt)]** [heap-sort]: Dados los enteros { 7, 11, 5, 13, 9, 4, 1, 4 } ordenarlos por el método de ordenamiento por montículos. Mostrar el montículo inicial y el resultante luego de extraer cada mínimo parcial.

b) **[quick-sort (5pt)]** Dados los enteros { 6, 1, 5, -2, 6, 11, -9, 4, -4, 3, 12, -7 } ordenarlos por el método de ordenamiento rápido (quick-sort). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber, el mayor de los dos primeros elementos distintos.

c) **[hash (5pt)]** Se implementa un diccionario a través de una tabla de dispersión cerrada con B=10 cubetas, con función de dispersión h(x) = x/(10 B) y redispersión lineal. Insertar los enteros { 710, 193, 225, 818, 27, 353, 179 } . Luego eliminar el elemento 353 e insertar el elemento 179, en ese orden. Mostrar la tabla resultante. ¿Que rango de números enteros es capaz de gestionar el diccionario propuesto?

d) **[2d-game (5pt)]** En el desarrollo de videojuegos suele ser importante determinar de forma eficiente la colisión entre dos objetos. En los juegos actuales las escenas constan de realmente muchos objetos (>10000) distribuidos en todo el espacio y para aportar realismo se debe generar cada cuadro muy rápido (mínimo 60fps). Una estrategia para evitar comprobar la colisión *todos contra todos* consiste en particionar el espacio de forma recursiva, tal como presenta la Figura 1, de tal manera de obtener una distribución de objetos casi constante por región. ¿Qué estructura de datos utilizaría para contener los objetos? Presentelá para el caso de la Figura y mencione las ventajas de utilizar esta estrategia en términos de sus tiempos de ejecución.

[Ej. 3] [PREG2 (W=20pt)]

a) En el proceso de construcción de una condificación binaria con árboles de Hufmann se van tomando dos árboles **T1** y **T2** y se combinan en uno nuevo donde **T1** y **T2** quedan como hijos derecho e izquierdo de la raíz, es decir **(Z T1 T2)**, donde **Z** es cualquier valor. Discuta si es relevante cuál de ellos se pone como hijo izquierdo e hijo derecho. Es decir, ¿da lo mismo reemplazar por **(z T1 T2)** ó **(z T2 T1)**? Justifique.

b) ¿Cuál es el tiempo de ejecución **T(n)** de la operación **splice** par árboles binarios?

c) Explique el proceso necesario para borrar un elemento en una tabla de dispersión cerrada. ¿Se puede reemplazar el elemento en la cubeta por el elemento indefinido **undef**? Justifique.

d) ¿Cuál es el tiempo de ejecución para las operaciones binarias de conjunto en la representación de conjuntos con listas ordenadas?

e) ¿Se puede definir un árbol binario simplemente como un subconjunto de los árboles ordenados orientados limitando a que la cantidad de hijos sea a lo sumo 2? Justifique.

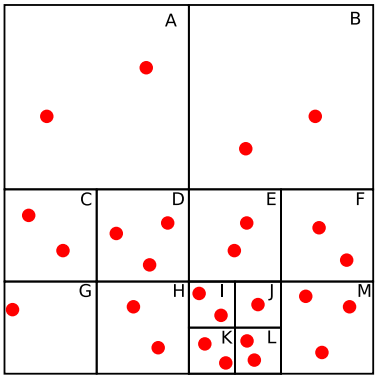

Figura 1: Escena 2d particionada en subregiones. Los puntos indican la ubicación de cada objeto. Las letras son las etiquetas de cada subregión.

f) Defina las propiedades para que un árbol sea un **montículo**. Indique que tipo de árbol se utiliza (AB o AOO) y que propiedades debe tener.

g) Discuta las propiedades de estabilidad de los diferentes métodos de ordenamiento. Indique si son estables o no y justifique brevemente porqué.

mstorti@galileo/aed-3.1-308-gc31be5e3/Thu Nov 15 13:08:58 2018 -0300