Algoritmos y Estructuras de Datos – Recuperatorios

Licenciatura en Tecnologías Digitales, UTDT

Segundo Semestre 2023

- No está permitido comunicarse por ningún medio con otros estudiantes ni con otras personas durante el examen, excepto con los docentes de la materia.
- Puede consultarse a los docentes solo por aclaraciones específicas del enunciado.
- El examen es a libro abierto: está permitido tener todo el material **impreso** y apuntes personales que deseen traer. **No está permitido el uso de dispositivos electrónicos para este fin.**
- **Cada ejercicio debe resolverse en hoja aparte.**

M4: Estructuras de Datos

Problema 1. (50 puntos) Estructuras de Datos
Se tiene la estructura de datos `Tira_ABB` que consiste en una lista simplemente encadenada de árboles binarios de búsqueda, junto con el invariante de representación de una `Tira_ABB n`:

```
struct Tira_ABB {
  ABB* arbol;
  int min, max;
  int cantidad;
  Tira_ABB* siguiente;
};
```

- n.arbol es un ABB no vacío.
- n.min y n.max almacenan el mínimo y máximo valor de n.arbol;
- n.cantidad almacena la cantidad de valores guardados en n.arbol.
- si `n.siguiente` no es nulo, `n.max < n.siguiente->min`, o sea, el ABB del siguiente nodo almacena números estrictamente más grandes que el ABB del nodo actual.

La búsqueda de valores en una `Tira_ABB` consiste en primero encontrar en qué ABB de la tira puede estar el elemento buscado, y luego hacer una búsqueda sobre el ABB. La inserción implica también encontrar el ABB donde corresponde insertar el valor, realizar la inserción, y reestablecer el invariante si es necesario.
Para insertar, se tienen los siguientes casos para un valor x y un nodo n:

- Si $x <$ `n.max`, se almacena en el nodo actual;
- Si $x \leq$ `2 * n.max` y `n` no tiene sucesor, se inserta en el nodo actual;
- En otro caso, se inserta en el nodo siguiente, creándolo si es necesario.

Implementar las siguientes operaciones:

(a) `bool buscar(int n, Tira_ABB* tira)` que devuelve `true` si el valor se encuentra en la tira.

(b) `Tira_ABB* insertar(int n, Tira_ABB* tira)` que agrega un nuevo valor a la `Tira_ABB` dada.

Las operaciones no deben recorrer la tira más de una vez, y tampoco deben hacer recorridas innecesarias sobre los ABBs. Se puede usar, como auxiliar, cualquier operación de ABB que haya sido definida en clase.

Problema 2. (50 puntos) Diseño de TADs
Se necesita diseñar un editor de archivos de configuración en formato INI:

```
[seccion 1]
v1 = 15
var2 = hola

[seccion 2]
var2 = chau
v3 = 5
```

```
class EditorINI {
  public:
  EditorINI();
  const set<string> & secciones() const;
  // Pre: true (si la seccion no existe, devuelve false)
  bool definida(const string& seccion, const string& clave) const;
  // Pre: definida(seccion,clave) es true
  string obtener(const string& seccion, const string& clave) const;

  // Pre: true (crea secciones si es necesario)
  void definir(const string& sección, const string& clave, const string& valor);
  // Pre: true
  void borrar_todas(const string& clave);

  // Pre: definida(x, clave) es true para alguna x en secciones()
  const set<string> & secciones_de(const string& clave) const;

private:
  map<string, map<string,string>> datos_secciones;
  /* completar */
};
```

En la parte privada, `datos_secciones` almacena, para cada sección, su diccionario de claves y valores asignados. Considere las siguientes restricciones de complejidad, donde S es la cantidad de secciones totales y N es la cantidad de claves totales. No se almacenan secciones vacías.

- `secciones` en $O(1)$, devuelve el conjunto de las secciones que existen,
- `definida` en $O(\log S + \log N)$, que devuelve si una clave está definida en una sección dada;
- `obtener` en $O(\log S + \log N)$, que devuelve el valor asociado a una clave;
- `secciones_de` en $O(\log N)$, que devuelve el conjunto de secciones donde aparece una clave;
- `definir` en $O(\log S + \log N)$, que asigna un valor a una clave en la sección correspondiente, creándola si es necesario;
- `borrar_todas`, sin requerimiento de complejidad, que elimina una clave de todas las secciones donde aparece.

(a) Agregar a la estructura interna lo que considere necesario para que se puedan implementar las operaciones en la complejidad deseada. Puede usar tipos de la `std` como `map`, `set`, `string`, `vector`, `list`...

(b) Escribir en español el invariante de representación de la estructura interna.

(c) Dar la implementación del método `borrar_todas` respetando el invariante propuesto. Calcular su orden de complejidad O en función de N y S.