Tercer Parcial

21 de noviembre de 2020

Indicaciones generales

1. Este es un examen **individual** con una duración de **100 minutos: de 10:10am a 11:50am**.
2. En `e-aulas` puede acceder a las diapositivas y a la sección correspondiente a este parcial.
3. Solamente será posible tener acceso a `e-aulas.urosario.edu.co` y a los sitios web correspondientes a la documentación de C++ dispuestos por el profesor.
4. Maletas, morrales, bolsos, etc. deben estar ubicados al frente del salón.
5. Celulares y otros dispositivos electrónicos deben estar apagados y ser guardados dentro de las maletas antes de ser ubicadas en su respectiva posición.
6. El estudiante no debe intentar ocultar ningún código que no sea propio en la solución a la actividad.
7. El estudiante solo podrá disponer de hojas en blanco como borrador de apuntes (opcional).
8. El estudiante puede tener una hoja manuscrita de resumen (opcional). Esta hoja debe estar marcada con nombre completo.
9. `e-aulas` se cerrará a la hora en punto acordada. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de `e-aulas` será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0. **Se aconseja subir a e-aulas versiones parciales de la solución a la actividad.**
10. Todas las evaluaciones serán realizadas en el sistema operativo GNU/Linux.
11. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
12. La evaluación debe presentarse exclusivamente en uno de los computadores ubicados en el salón de clase y a la hora acordada. Presentar la evaluación desde otro dispositivo o en otro horario diferente al estipulado es causa de anulación.
13. **Cualquier incumplimiento de lo anterior conlleva la anulación del examen.**
14. Las respuestas deben estar totalmente justificadas.
15. **Entrega**: archivos con extensión `.txt`, `.cpp` o `.hpp` según el caso, conteniendo la demostración o el código. Nombre los archivos como `pY.Z`, con `Y = 1,2,3` y `Z = txt, cpp, hpp`. Comprima su código y demás archivos en *un único* archivo `parcial.zip` y súbalo a `e-aulas`. **Importante:** no use acentos ni deje espacios en los nombres de los archivos que cree.

En el desarrollo del examen, no olvide usar la plantilla para cada ejercicio.
Las implementaciones deben ejecutarse sin errores usando las funciones `main()` de cada plantilla.

1. [50 ptos.] Agregue a la implementación e interfaz de un árbol binario de búsqueda dos rutinas nuevas que no modifican el estado del árbol. Dado un árbol binario de búsqueda T, escriba los métodos

```
1 unsigned size() const;            // public
2 unsigned size(bstNode *T) const; // private
```

público y privado, respectivamente, que calculan el tamaño de T. Es decir, el número total de nodos que contiene T. La implementación no debe usar el atributo `count`.

2. [50 ptos.] Definimos un subvector de un vector `vec` como una sección de elementos contiguos de `vec` cuya longitud puede variar entre 1 y la longitud de `vec`. El problema del *subvector de valor absoluto* X consiste en encontrar todos los subvectores de un vector `vec` tales que el valor absoluto de la suma de sus elementos sea X.

   El archivo `hashtab.hpp` contiene la interfaz para la clase `HashTab`, que implementa una tabla hash con llaves tipo `vector<int>`.

```
1  struct KeyNode {
2    vector<int> key;
3    KeyNode *next;
4  };
5
6  class HashTab {
7  private:
8    KeyNode **table;
9    int tableSize;
10   int count;
11
12   bool compare_keys(const vector<int> &key1, const vector<int> &
       key2);
13   KeyNode* find_key(KeyNode* tmp, const vector<int> &key);
14   KeyNode* find_in_bucket(unsigned int i, const vector<int> &key
       );
15   void rehash(int max);
16
17   /* ********************************** */
18   unsigned int hash(const vector<int> &key);
19   /* ********************************** */
20
21 public:
22   HashTab();
23   ~HashTab();
24
25   int size();
26   bool empty();
27   void insert(const vector<int> &key);
28   void remove(const vector<int> &key);
29   bool find(const vector<int> &key);
30
31   /* ******************** */
32   void display_bucket(int i);
33   /* ******************** */
34 };
```

   El objetivo de esta estructura de datos es el de clasificar un vector que se le inserte de acuerdo al valor absoluto de la suma de sus elementos. Esto lo logra usando una función hash que

ubica a todos los elementos cuyo valor absoluto de su suma sea igual en el mismo bucket. Por ejemplo, los vectores `{5,-2}`, `{-1,0,1,5,-2,-3,3}` y `{-3}` son insertados en el bucket `3`. Note que la estructura no maneja valores, solo necesita llaves, que son los vectores a insertar en la tabla.

Dicha estructura puede ser usada para resolver fácilmente el problema del subvector de valor absoluto X, iterando sobre todos los posibles subvectores de cualquier vector e insertándolos en la estructura de manera que queden clasificados.

Implemente los métodos

a) [25 ptos.] `hash`: Ubica subvectores en la tabla hash durante la operación de insesrción en sus correspondientes *buckets*, tal como se muestra en el ejemplo de arriba.

b) [25 ptos.] `display_bucket`: Imprime al flujo de salida `std::cout` el conjunto de subvectores cuyo valor absoluto de la suma de sus elementos sea X.

en la estructura de datos `HashTab` con el fin de resolver el problema del subvector de valor absoluto X.