Instrucciones:

- ◇ Fecha de publicación: 12 de noviembre de 2019.
- ◇ Fecha de entrega: 19 de noviembre de 2019 hasta las 23:55.
- ◇ Medio de entrega: `https://e-aulas.urosario.edu.co` (no se reciben entregas por correo electrónico u otros medios).
- ◇ La actividad **debe** realizarse **en grupos de tres** estudiantes.
- ◇ Formato de entrega: implementación en C++14.
- ◇ Nombre archivos: definidos más abajo.
- ◇ Importante: no use acentos ni deje espacios en los nombres de los archivos que cree.
- ◇ Solamente un miembro del grupo debe realizar la entrega. Liste los miembros del mismo como un comentario en el encabezado de la implementación.

Protocolo para la evaluación:

Los siguientes lineamientos serán seguidos de forma estricta y sin excepción.

1. Los grupos pueden consultar sus ideas con los profesores para recibir orientación; sin embargo, la solución y detalles del ejercicio debe realizarlos **los integrantes de cada grupo**. Cualquier tipo de fraude o plagio es causa de anulación directa de la evaluación y correspondiente proceso disciplinario.
2. El grupo de trabajo debe indicar en su entrega de la solución a la actividad cualquier asistencia que haya recibido.
3. El grupo no debe consultar ninguna solución a la actividad que no sea la suya.
4. El grupo no debe intentar ocultar ningún código que no sea propio en la solución a la actividad (a excepción del que se encuentra en las plantillas).
5. Todas las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
6. E-aulas se cerrará a la hora en punto acordada para el final de la evaluación. 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.

No habrán excepciones a estas reglas.

Enunciado:

Resuelva los siguientes ejercicios sobre funciones y tablas hash. Utilice el estándar C++14 en la solución de los problemas. No olvide compilar con los *flags* apropiados para detectar *warnings* y errores: `-Wall -Wextra -Werror`.

Escriba su código solución a partir su implementación de tabla hash (hecha en el Taller 10) y de los archivos: `hashf_test_plantilla.cpp`, `load_test_plantilla.cpp` incluidos con esta Tarea. Asegúrese de seguir cuidadosamente las indicaciones del ejercicio.

1. La "eficiencia" de una función hash puede observarse, en parte, en la manera en la que la función distribuye elementos de un conjunto de llaves en una tabla hash. Una función hash eficiente distribuye de manera uniforme las llaves del conjunto, de esta forma las operaciones de diccionario: inserción, búsqueda y borrado, toman tiempo constante $\Theta(1)$ "en promedio".

   Con el fin de evaluar la eficiencia de una función hash, considere las siguientes funciones hash que distribuyen de forma diferente llaves tipo `string` en una tabla hash:

```cpp
unsigned int hash0(string key, int tableSize) {
    unsigned int hashVal = 0;
    for (char ch : key)
        hashVal += ch;
    return hashVal % tableSize;
}
```

```cpp
unsigned int hash64(string key, int tableSize) {
    unsigned int hashVal = 0;
    for (char ch : key)
        hashVal += 64 * hashVal + ch;
    return hashVal % tableSize;
}
```

   La idea es, entonces, usar estas funciones para obtener las posiciones, dentro de una tabla hash, de un conjunto de llaves tipo `string`. En este caso particular, el conjunto es una lista de palabras del diccionario español (para esto use el archivo `palabras.txt` incluido con el Taller 10). Suponga que el tamaño de la tabla hash es `tableSize = 101`.

   Para comparar su comportamiento, para cada función hash realice un histograma de frecuencias en donde se cuenten cuántas palabras se ubican en cada una de las 101 posiciones de la tabla. Recuerde que un histograma muestra la frecuencia con la que ocurren valores en una muestra de datos. En este caso los valores corresponden a la cantidad de elementos almacenados en una posición dada de la tabla hash. Grafique este número de elementos vs. posición en la tabla a manera de un diagrama de barras. Para realizar esta tarea use la plantilla `hashf_test.cpp` y obtenga los histogramas usando la librería `matplotlib` de Python. Analice cuál función hash es más eficiente y por qué.

2. La siguiente tarea consiste en comparar el tiempo de acceso promedio en tablas hash que usan las funciones hash presentadas en el punto anterior.

   Escriba dos implementaciones de `HashMap<VT>` con llaves tipo `string`: la primera usando la función `hash0` y la segunda usando la función `hash64`. Defina la variable de instancia correspondiente al tamaño de la tabla como `tableSize = 1013`. Estas implementaciones son similares a la realizada en el Taller 10. La implementación debe contener el método `get(string key)`, que recibe una llave y retorna el valor asociado a esta, si se encuentra, o un valor por defecto `notfound` que indica que la llave no se encontró en la tabla.

   Para poder medir el tiempo promedio de acceso a un elemento en el `HashMap`, cuando se usa cada una de las dos funciones hash, escriba un programa que calcule dicho tiempo promedio para diferentes valores del factor de carga `lambda = count / tableSize`. Implemente la función

   ```
   void ac_time(vector<string> keys, int repetitions, int
       size, minstd_rand0 &rng, double &av_time, double &
       sq_time);
   ```

   que recibe un vector con la lista de posibles llaves (palabras del diccionario español), un número de repeticiones, un número de elementos `size`, un generador de números aleatorios (de la librería `<random>`) y un par de variables por referencia: `av_time` y `sq_time`.

   a) Para cada una de las repeticiones, la función `ac_time` crea un `HashMap<int>` de enteros con `notfound = numeric_limits<int32_t>::max()` e inserta una cantidad `sz` de elementos con llaves tomadas aleatoriamente de las que se encuentran en el archivo `palabras.txt`.

   b) Ahora, con el mapa conteniendo `sz` elementos (tal que el factor de carga es `lambda = sz / tableSize`), calcule el tiempo t que tarda en acceder a cada uno de ellos, usando el método `get` y acumúlelo en `av_time`. Este se usará para calcular el tiempo promedio.

   c) Haga lo mismo para t^2 y acumúlelo en `sq_time`. Este se usará para calcular la desviación estándar del tiempo promedio.

   d) Divida `av_time` y `sq_time` entre el número de repeticiones para obtener el tiempo promedio y el tiempo cuadrado promedio, de acceso a un elemento dela tabla, respectivamente.

   e) Finalmente, repita el procedimiento para varios valores de `sz`, manteniendo `tableSize` constante, de manera que se pueda graficar `av_time vs. lambda`, con barras de error dadas por $\sqrt{\texttt{sq\_time} - \texttt{av\_time}^2}$. Asegúrese que `sz` sea tal que `lambda` tome valores menores a 1 y mayores o iguales también.

   En el desarrollo de esta parte de la Tarea, use la plantilla `load_test_plantilla.cpp` adjunta con este archivo.

   ENTREGABLES: Los archivos a entregar son los siguientes:

a) Versiones modificadas (funcionales) de las plantillas `hashf_test_plantilla.cpp` y `load_test_plantilla.cpp`.

b) Archivos de implementación `cpp` y de interfaz `hpp` con cada versión de `HashMap`, para los dos casos de las funciones hash trabajadas (`hash0` y `hash64`).

c) Archivo PDF con las gráficas de los histogramas del primer punto, para los dos casos de las funciones hash trabajadas (`hash0` y `hash64`).

d) Archivo PDF con las gráficas de tiempo vs. `lambda` para los dos casos de las funciones hash trabajadas (`hash0` y `hash64`).

Todo debe encontrarse en un estado que se pueda compilar y ejecutar en un archivo comprimido `zip`.