ELO320 – Tarea 1 Fase 2
Prof. Mauricio Araya
2do semestre del 2019

IMPORTANTE: Las tareas son **individuales**. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría o la facilitación de esto para otros. Es considerado aceptable discutir -en líneas generales- los métodos y resultados con sus compañeros, pero se prohíbe explícitamente realizar las tareas en conjunto o compartir código de programación.

Manejo de colisiones en Tablas Hash

El manejo de colisiones en una Tabla Hash por encadenamiento es una de las estrategias posibles a implementar. Otra estrategia muy usada es el **open addressing**, donde todo elemento ingresado se encuentra en la Tabla Hash. En términos generales en caso de ocurrir una colisión, se busca alguna otra casilla de la Tabla Hash que se encuentre disponible.

Para realizar un `insert` utilizando **open addressing** la Tabla Hash se prueba sucesivamente hasta encontrar una casilla disponible en la cual almacenar la llave `key` a insertar. *La secuencia de posiciones probadas depende del valor de la llave a insertar.* Por lo que, la función hash es ahora modificada para incluir el índice a utilizar como segunda entrada:

$$h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}. \tag{1}$$

```
1: function HASH-INSERT(T,k)
2:     i = 0
3:     repeat
4:         j = h(k, i)
5:         if T[j] == empty then
6:             T[j] = k
7:             return j
8:         else
9:             i = i + 1
10:    until i == m
11:    error "hash table overflow"
```

Por lo que con este algoritmo, se generará la secuencia de prueba: $\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$.

Tres son las típicas técnicas utilizadas para calcular la secuencia de posiciones probadas necesarias para **open addressing**: *linear probing*, *quadratic probing* y *double hashing*. Todas estas técnicas garantizan que $\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$ es una permutación de $\langle 0, 1, \ldots, m-1\rangle$ para cada llave k. Las funciones hash utilizadas en cada una de estas técnicas se encuentran descritas en la Tabla 1, donde $h', h_1 y h_2$ son funciones hash auxiliares del estilo $h', h_1, h_2 : U \rightarrow \{0, 1, \ldots, m-1\}$.

Linear probing	$h(k,i) = (h'(k) + i) \mod m$
Quadratic probing	$h(k,i) = \left(h'(k) + c_1 i + c_2 i^2\right) \mod m$
Double hashing	$h(k,i) = (h_1(k) + i h_2(k)) \mod m$

Cuadro 1: Funciones hash para técnicas de **open addressing**.

Trabajo a realizar

1. Eligiendo funciones hash $h'(k) = k \mod m$, $h_1(k) = k \mod m$, $h_2(k) = 1 + (k \mod (m-1))$, $c_1 = 3$ y $c_2 = 5$. Implemente un código que le permita llenar una Tabla Hash de tamaño variable, manejando las colisiones por **open addressing** (las tres estrategias).

2. Genere tres tablas hash, cada una de ellas de tamaño 1500, inicialmente vacías. Para cada tabla hash, genere un archivo de texto de salida de 100 columnas y 15 filas donde se representarán los mapas de ocupación por cada uno de los métodos de **open addressing**. Represente por un espacio `' '` el slot disponible y con `'X'` el slot utilizado.

3. Genere 1500 números aleatorios entre 0 y 1000000 para ser almacenados en cada una de las tablas hash. Agregue en su código la opción de ingreso que le permite controlar la cantidad de elementos a insertar en cada una de sus tablas hash. Su código debe generar como salida tres archivos de texto que representan el mapa de ocupación para cada caso. Los archivos deben ser guardados con el siguiente formato: `hash_tipo_xxxx.txt`, donde `tipo` puede ser $\{linear, quadratic, double\}$ y `xxxx` representa el número de casillas utilizadas en la tabla hash, indicada con 4 dígitos.

4. Genere mapas de ocupación para 100, 500, 1000 y 1500 elementos almacenados en las tablas hash. Describa en su archivo `README.txt` comentarios sobre los patrones de llenado obtenidos y comente lo que ocurre en cada caso.

Condiciones de Entrega

Utilice `/*comentarios*/` para describir lo que se hace en cada etapa de su código. Prepare un archivo adicional README.txt donde debe especificar sus datos (nombres, apellidos, ROL USM, y emails) y además que es cada archivo de su programa, como debe ser compilado y como debe ser ejecutado. Además, debe explicar cualquier particularidad que pueda tener su programa. El código deberá estar escrito según el estándar de codificación GNU y ser compilado y ejecutado en el servidor Aragorn **SIN ERRORES** para ser evaluado. El código escrito debe estar indentado de forma coherente y consistente. Para la entrega, envíe un solo archivo (.zip, .rar, .tar.gz, etc.) que contenga su código, es decir, sus archivos .c, .h utilizados, mas el archivo README.txt. El entregable debe ser subido a la plataforma AULA antes de las 23:50 del viernes 27 de marzo 2020. Se aplicara descuento de Fibonacci por hora (i.e., 1 punto la primera hora, 1 punto la segunda, 2, 3, 5, 8, 13, 21, 33, 54, etc..).