Auxiliar 9 - Hashing

Profesores: Nelson Baloian
Jeremy Barbay
Patricio Poblete
Auxiliares: Gabriel Flores, Sven Reisenegger
Juglar Díaz, Gabriel Norambuena
Cristóbal Muñoz

P1. Promociones

Usted es el dueño de un negocio de venta de comida. Para atraer más clientes se decide realizar promociones especiales para compradores frecuentes.

Cada mes se le ofrecerán descuentos a grupos diferentes de personas basado en cuánto gastan en su local. Por ejemplo, en un mes aquellos clientes que gasten entre $\$5000 - \$15000$ obtendrán una bebida gratis, en otro mes los clientes que gasten entre $\$15000 - \$25000$ obtienen un cupón para un almuerzo 2x1.

Los rangos de gasto cambian mes a mes, y quieres lograr encontrar el grupo correcto de personas de cada mes en tiempo $O(K + log(N))$, en donde K es el tamaño del grupo y N es el número total de clientes de su negocio.

¿Que estructura(s) de dato(s) escogería, y como haría la solución a este problema? Comente brevemente.

Puede asumir que el tiempo de ejecución no incluye el costo de armar la base de datos de los clientes y que ésta existe antes de saber los intervalos de gastos. Además cada cliente compra en su local en múltiplos de 1000.

P2. Funciones de Hash

¿Cuál de las siguientes funciones de hash sobre enteros distribuirá las llaves más uniformemente sobre una tabla de Hash con 10 espacios numerados del 0 al 9? Considere un i que va en el rango $[0, 2020]$.

(a) $h_0(i) = i^2 \ mod \ 10$

(b) $h_1(i) = i^3 \ mod \ 10$

(c) $h_2(i) = (11 * i^2) \ mod \ 10$

(d) $h_3(i) = (12 * i) \ mod \ 10$

Comente qué pasaría si se cambiara el modulo aplicado a las funciones.

P3. Hashing con Direccionamiento Abierto

Una de las maneras para resolver las colisiones en una tabla de Hash es utilizar direccionamiento abierto, es decir, guardar los valores que chocan en la misma tabla pero en otra posición. Una técnica para encontrar esa posición es *Linear Probing*.

Una tabla de hashing con Linear Probing es un tipo de hashing en donde para solucionar el problema de las colisiones se avanza un espacio en la tabla, es decir, si un valor cae en un espacio que ya está ocupado, se intenta de insertar en el siguiente hasta que se encuentra un espacio vacío.

(a) Implemente las operaciones para crear, insertar y contiene elementos en esta estructura de datos. Asuma que existe la función de hash $h(e)$.

La operación de eliminar un elemento en la tabla es más compleja debido a los "hoyos" que se pueden generar entre el hash de un elemento y su posición final. Existen 2 formas distintas de resolver esto:

- Se puede hacer una eliminación *lazy*, esto quiere decir que el elemento no se elimina directamente, sino que es marcado y luego en la búsqueda si un elemento está marcado no será considerado.
- Otra forma es usando una estrategia *eager*, esto quiere decir que eliminamos directamente el elemento y luego buscamos en las posiciones siguientes un elemento para colocar en el espacio libre, repitiendo la operación cuántas veces sea necesario.

(b) Comente acerca de la implementación, las ventajas y desventajas de cada estrategia.

(c) Implemente una función `public static void eliminarEager(int[] HT, int element)` que utilice la estrategia *eager*, asuma que existe la función de hash $h(e)$.