UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA

# Guía de Ejercicios – parte 11 - Hashing

## Algoritmos y **Estructura de datos**

**Curso** Ing. **Gustavo Schmidt**

Conceptos a Implementar

De dificultad baja:

1) Crea una función de hashing que use el método de Módulo ($h(k) = k \pmod m$) para claves enteras ($k$). La capacidad $m$ debe ser un número primo.hashModulo(int k, int m)
2) Implementa una función de hashing para claves de tipo String. Utiliza la suma de los valores ASCII de los caracteres de la cadena y luego aplica el método del Módulo.hashSumaASCII(String key, int m)
3) Implementa el método del Cuadrado Medio (Middle Square). Para una clave numérica de 4 dígitos, elévala al cuadrado y extrae los 2 dígitos centrales para obtener el índice.hashCuadradoMedio(int k, int m)
4) Escribe una función de hashing basada en el método de Plegado (Folding/Folding Boundary). Divide un número de 8 dígitos en grupos de 2 dígitos y súmalos para obtener el índice.hashPlegado(int k, int m)
5) Diseña una función de hashing para String usando un método polinomial simple, donde cada carácter se multiplica por una potencia de una constante (ej: 31) antes de aplicar el Módulo.hashPolinomial(String key, int m)
6) Implementa un programa que calcule y muestre los índices de las claves $\{10, 21, 32, 43, 54\}$ usando la función Modulo con una capacidad $m=11$.Comparación de índices
7) Modifica el ejercicio F3 para que tome una clave de 6 dígitos y extraiga 3 dígitos centrales. Asegúrate de manejar el desbordamiento si el cuadrado es demasiado grande. Extracción de bits/dígitos
8) Escribe una función de dispersión (scramble) inicial. Esta función debe mezclar los bits de la clave usando operaciones XOR o shifts antes de pasarla a la función Modulo para reducir el riesgo de patrones. Manipulación de bits
9) Análisis de Colisiones: Dada la función $h(k) = k \pmod 10$, encuentra dos claves $k_1$ y $k_2$ diferentes (ambas mayores que 20) que resulten en una colisión. Pruebas de función de hashF
10) Implementa la primera función de la técnica de Doble Hashing ($h_1(k)$). Debe ser una función simple de Módulo. Función de doble hashing $h_1$

De dificultad media:

1) Implementa una tabla hash completa que utiliza Encadenamiento Separado (Separate Chaining). Utiliza un ArrayList de LinkedLists para el almacenamiento. Encadenamiento, put()
2) Para la estructura M1, implementa el método get(key) para buscar una clave. Debe recorrer la lista enlazada solo si encuentra el índice hash. Encadenamiento, get()
3) Implementa la resolución de colisiones por Sondeo Lineal (Linear Probing). Cuando ocurre una colisión en el índice $i$, busca el siguiente espacio libre en $i+1, i+2, \dots$ Sondeo Lineal, put()
4) Para la tabla hash con Sondeo Lineal (M3), implementa el método get(key). Debe continuar buscando más allá de una colisión hasta encontrar la clave o una celda nula. Sondeo Lineal, get()
5) Implementa el método remove(key) para la tabla con Sondeo Lineal. Introduce el concepto de Tombstone (marcador de borrado) para evitar romper las cadenas de búsqueda. Sondeo Lineal, remove()
6) Implementa la resolución de colisiones por Sondeo Cuadrático (Quadratic Probing). La secuencia de sondeo debe ser: $h(k), h(k) + 1^2, h(k) + 2^2, \dots$ Sondeo Cuadrático, put()
7) Implementa la técnica de Doble Hashing. Necesitarás dos funciones de hash: $h_1(k)$ (Módulo) y $h_2(k)$. La secuencia de sondeo es $h(k, i) = (h_1(k) + i \cdot h_2(k)) \pmod m$.Doble Hashing, put()
8) Inserta las claves $\{10, 20, 30\}$ en una tabla de tamaño $m=13$. Todas colisionan en el índice 10. Traza los índices utilizados por Sondeo Lineal y por Doble Hashing. Comparación de técnicas de sondeo
9) Implementa una tabla hash con Encadenamiento y añade una lógica para redimensionar (rehash) la tabla a una capacidad doblemente mayor cuando el factor de carga $\alpha$ exceda $1.0$.Redimensionamiento (Rehashing)
10) Diseña un registro (Entry) que almacene la clave, el valor y el campo booleano isDeleted (para manejar el Tombstone en el sondeo). Refactoriza tu implementación de Sondeo Lineal (M3) para usar este registro. Estructura de datos interna

De dificultad alta:

1) Problema de las Dos Sumas (Two Sum): Dada una lista de números enteros y un objetivo $T$, encuentra los dos números que suman $T$. Resuélvelo en complejidad $O(n)$ usando un HashMap para almacenar los "complementos" necesarios. $O(n)$ búsqueda
2) Contador de Frecuencia: Escribe una función que tome una cadena larga y devuelva el carácter que aparece con la mayor frecuencia. El uso de un HashMap<Character, Integer> es obligatorio. Conteo de frecuencia
3) Detección de Subarray Suma Cero: Dado un array de enteros, determina si existe un subarray (secuencia contigua) cuya suma sea $0$. Utiliza un HashSet para almacenar las sumas prefijo acumuladas. Sumas acumuladas (Prefix Sum)
4) Agrupación de Anagramas: Dada una lista de cadenas, agrupa las cadenas que son anagramas entre sí (por ejemplo, ["oído", "odio"] deben agruparse). Usa un HashMap donde la clave sea la versión ordenada de la cadena. Clave Hash Canónica
5) Implementación de un Set con Hashing: Implementa tu propia clase MyHashSet sin usar java.util.HashSet. Usa tu implementación de tabla hash con Sondeo Lineal donde solo almacenas las claves. Estructura de Datos Abstraída
6) Implementación de un Caché LRU (Least Recently Used): Diseña una estructura de caché que soporte put y get en $O(1)$. Debe ser capaz de eliminar el elemento menos usado cuando se llena. Usa un HashMap y una LinkedList doblemente enlazada. Estructura Híbrida (Hash + Lista)
7) Validar una Baraja de Póker: Dada una lista de 52 cartas como cadenas (ej: "A-Espadas"), usa un HashSet para verificar que la baraja sea válida (no hay cartas duplicadas y tiene 52 cartas).Verificación de Unicidad
8) Clave Hash Personalizada: Crea una clase Punto2D(int x, int y) y úsala como clave en un HashMap. Sobrescribe correctamente los métodos hashCode() y equals() de Punto2D para asegurar que el HashMap funcione. Contratos hashCode() y equals()
9) Secuencia Consecutiva Más Larga: Dada un array desordenado de enteros, encuentra la longitud de la secuencia de elementos consecutivos más larga (ej: en $[100, 4, 200, 1, 3, 2]$, la secuencia es $1, 2, 3, 4$, longitud 4). Resuélvelo en $O(n)$ usando un HashSet. Optimización de conjuntos
10) Ejercicio Conceptual: Hashing Consistente: Investiga qué es el Hashing Consistente (Consistent Hashing) y escribe un ensayo corto (200 palabras) explicando cómo resuelve el problema de la redistribución masiva de datos cuando se añade o quita un servidor en un sistema distribuido. Aplicación en Sistemas Distribuidos

11) Ventajas de la Capacidad Prima: Explica por qué es una práctica común elegir un número primo como la capacidad $m$ de una tabla hash cuando se usa el método de Módulo.Teoría de Números y Hashing

12) Comparación de Complejidad: Compara la complejidad temporal ($O$) de buscar un elemento en el peor caso en una tabla hash con Encadenamiento Separado ($\alpha > 1$) y una lista enlazada simple. ¿Por qué se sigue prefiriendo el hash? Análisis de Complejidad