Hash Tables

Bach. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería

Diccionario

- Es un tipo de dato abstracto (ADT) que almacena un conjunto de elementos. Cada elemento es representado por una clave (key) y una serie de datos que describen al elemento (data).
- También podemos decir que es un ADT que almacena pares (clave, datos).
- Los elementos tienen claves diferentes .

Diccionario

Debe soportar las siguientes operaciones

- Insertar un elemento
- **Buscar un elemento por clave**
- Eliminar un elemento

Diccionario

Student ID	Last Name	Initial	Age	Program
ST348-245	White	R.	21	Drafting
ST348-246	Wilson	P.	19	Science
ST348-247	Thompson	A.	18	Arts
ST348-248	Holt	R.	23	Science
ST348-249	Armstrong	J.	37	Drafting
ST348-250	Graham	S.	20	Arts
ST348-251	McFadden	H.	26	Business
ST348-252	Jones	S.	22	Nursing
ST348-253	Russell	W.	20	Nursing
ST348-254	Smith	L.	19	Business

Diccionario

Aplicación	Clave	Valor
Diccionario	palabra	definición, ...
Guía telefónica	nombre	teléfono, ..
Gestión de cuentas bancarias	número de cuenta	Titular de la cuenta,…

Diccionario

¿ Cómo lo implementamos eficientemente?

Diccionario-Implementación

Estructura	Insertar	Buscar	Eliminar
arreglo	$O(1)$	$O(n)$	$O(n)$
Arreglo ordenado	$O(n)$	$O(\log n)$	$O(n)$
Lista enlazada	$O(1)$	$O(n)$	$O(n)$
Árbol binario de búsqueda balanceado	$O(\log n)$	$O(\log n)$	$O(\log n)$

Tabla de Acceso Directo

- Un arreglo indexado por clave.

  Ejemplo: (dni, nombre)

  A[73433431] = “Carlos”

  A[45433437] = “Pedro”

- Todas los operaciones son $O(1)$

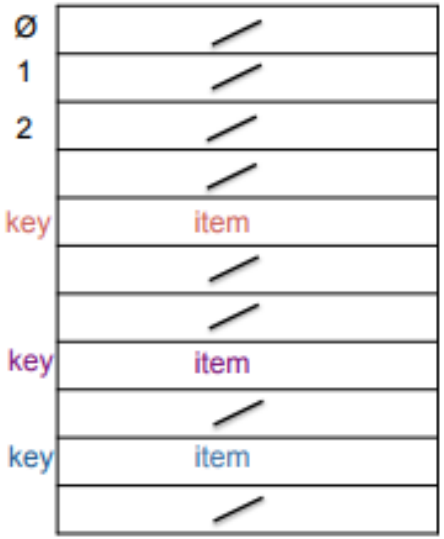


Tabla de Acceso Directo

- Tiene como limitación el tamaño de la tabla.
- Mucho espacio no usado.
- No permite manejar claves de otros tipos de datos como cadenas.

Hashing

- Técnica que permite realizar las operaciones del diccionario en $O(1)$ en promedio, manejando cualquier tipo de clave.
- Es una mejora de la tabla de acceso directo, añadiendo conceptos como “hash table” y “hash function”.

Hash Table

Tabla o arreglo de una tamaño razonable, con la misma idea de la tabla de acceso directo, pero ahora los índices serán siempre números pequeños, ya que serán el resultado del “hash function” de las claves.

Hash Function

- Función que mapea aleatoriamente las claves en un índice del hash table.

- Mapea enteros grandes o cadenas a enteros menores que pueden ser usado como índice del hash table.

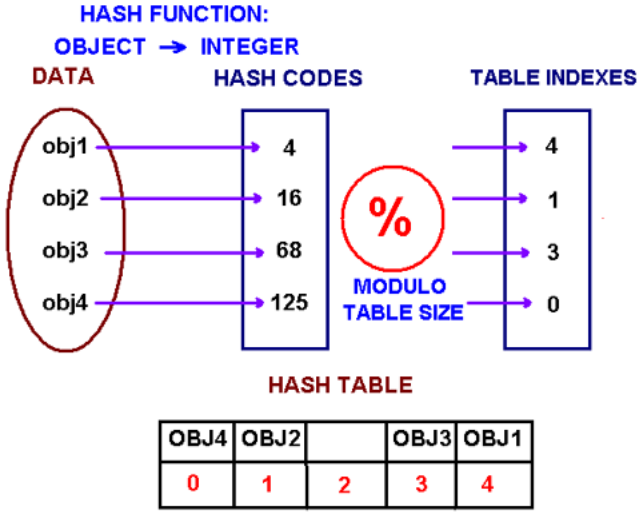
Ejm: $\text{hash}(x) = x \; mod \; 7$

Hash Function

Una buena función hash debe:

- La función debe calcularse eficientemente.
- Debe distribuir las claves uniformemente sobre la tabla hash.
- Es determinística.

Hashing


HASH FUNCTION:
OBJECT → INTEGER
DATA
HASH CODES
TABLE INDEXES
obj1
obj2
obj3
obj4
4
16
68
125
%
MODULO
TABLE SIZE
4
1
3
0
HASH TABLE
OBJ4
OBJ2
OBJ3
OBJ1
0
1
2
3
4

Hash Function

Método de la División

hash(k) = k mod m, donde m es el tamaño del hash table

Hash Function

Método de la División

Qué pasa si :

- m es múltiplo de 2 y todas las llaves son pares.
- m es una potencia de 2
- Si m tiene un divisor pequeño.

Hash Function

Método de la División

Si claves son {0, 1, 2, … 50} y usamos un hash table de tamaño 12, entonces la clave múltiplos de 3, solo van a slot de índices múltiples de 3.

claves {0, 12, 24, 36, .. } van al slot 0

claves {3, 15, 27, 39,... } van al slot 3

claves {6, 18, 30, 42, .. } van al slot 6

claves {9, 21, 33, 45, .. } van al slot 9

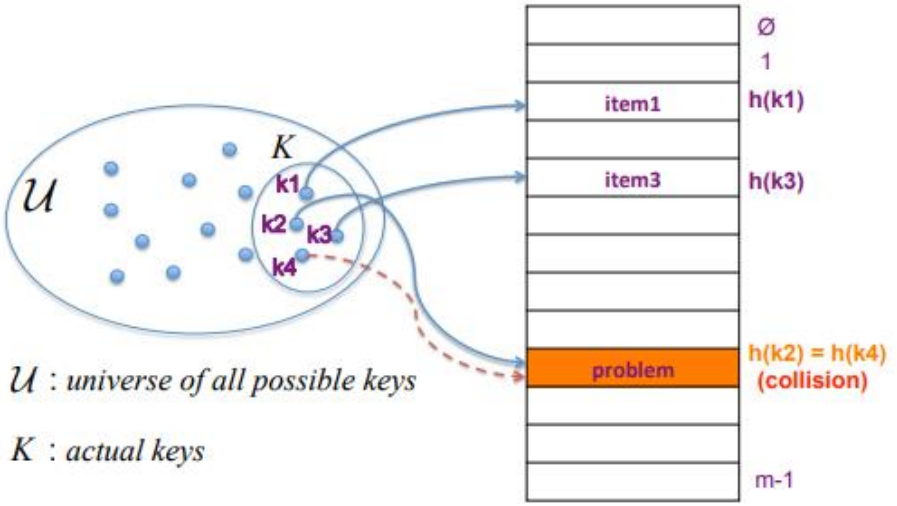
Hash Function

Método de la División

Escoger un m lo suficientemente grande y que sea primo, preferiblemente no cercano a una potencia de 2.

Colisiones

Cuando 2 claves distintas tiene el mismo valor de hash.



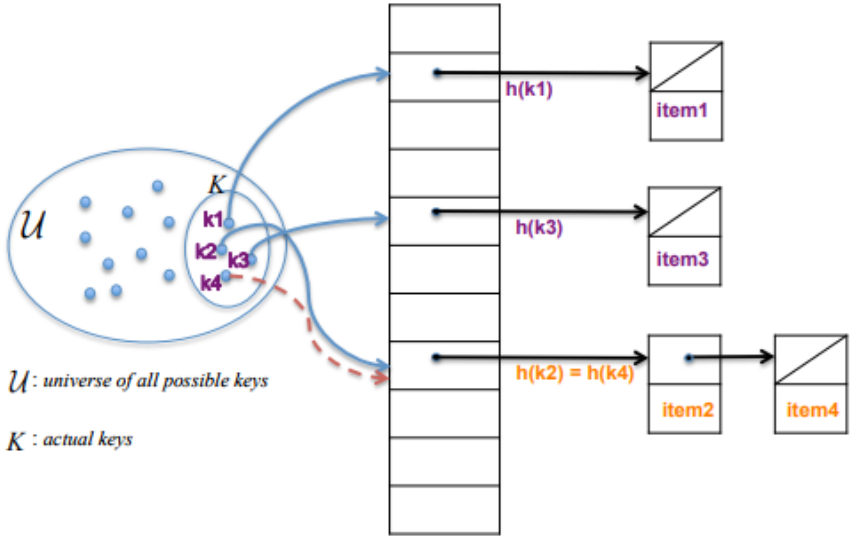
Manejando Colisiones

1.- Chaining

2.- Open Addressing

Chaining

Cada slot del hash table lleva consigo una lista enlazadas de todos los elementos con llaves de igual valor hash.



Chaining

Ventajas

Fácil de implementar.
Hash Table nunca se llena.

Desventajas

Usamos memoria extra.
Si la cadena es muy larga, la búsqueda se vuelve lineal.

Chaining

Complejidad de la Búsqueda

n = número de llaves guardadas en la tabla
m = número de slots

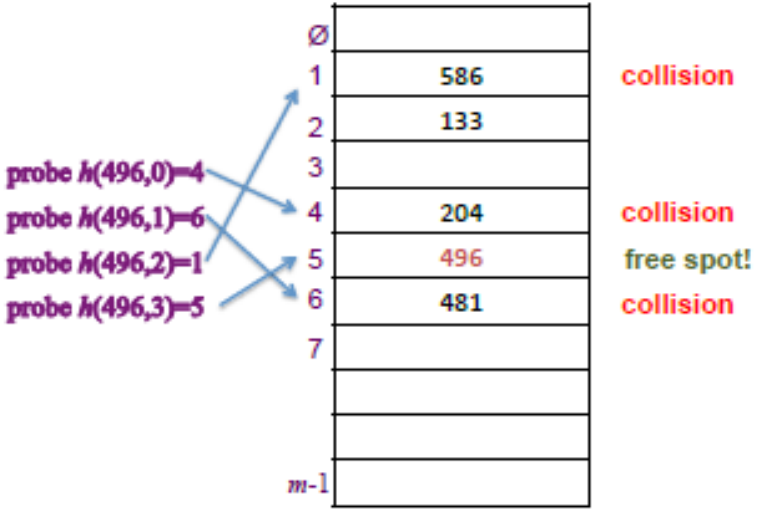
Load factor (promedio de llaves por slot): $\propto = n/m$

$O(1 + \propto)$ → hash + recorrer la lista

Open Adressing

- Se usa cuando el tamaño de la tabla (m) es mayor al número de claves (n).
- Todos los elementos son guardados en la misma tabla hash.
- Si ocurre una colisión se busco el slot hasta encontrar uno vacío.

Open Adressing



Open Adressing

Linear Probing

hi(x) = (hash(x) + i) % m

Quadratic Probing

hi(x) = (hash(x) + i*i) % m

Double Hash

hi(x) = (hash1(x) + i * hash2(x)) % m

Open Adressing

Complejidad de Búsqueda

$$\propto = \frac{n}{m} < 1$$

$$O(\frac{1}{1-\propto})$$

Unordered Map

Implementación de un hash table en C++.

```cpp
int main(){
    unordered_map<string, int> H;
    //insertar O(1)
    H["pedro"] = 123;
    H["luis"] = 456;
    // eliminar O(1)
    H.erase("pedro");
    //buscar O(1)
    if( H.find("pedro") != H.end() ) cout << "encontrado" << endl;
    else cout << "no encontrado" << endl;
    return 0;
}
```

String Hashing

Muchas veces necesitamos comparar strings eficientemente y a pesar de que el **hashing** no es completamente determinístico, nos ayuda a esta tarea debido a que las probabilidades de colisiones son muy pequeñas.

String Hashing

Sea $\boldsymbol{S}$ una string de tamaño $\boldsymbol{n}$.

$$\boldsymbol{hash(S) = \left(S[0].B^{n-1} + S[1].B^{n-2} + \cdots + S[n-1].B^{0}\right) \% m}$$

$\boldsymbol{B}$ $(29, 31, 53, \ldots)$ debe ser un primo cercano al número de carácteres del alfabeto y $\boldsymbol{m}$ $(10^9 + 7, 10^9 + 9, ..)$ un primo lo suficientemente grande.

String Hashing

```
string s;
cin >> s;
int n = s.size();
int B = 29, m = 1000000007;
long long pot[ n ];
// calculando potencias
pot[ 0 ] = 1;
for(int i = 1; i < n; ++i){
    pot[ i ] = (pot[ i - 1 ] * B ) % m;
}
// hashing
int hash = 0;
for( int i = 0; i < n; ++i){
    hash += ( (s[ i ] -'a' + 1) * pot[ n - 1 - i] ) % m;
}
```

Hashing Prefijos

Sea $\boldsymbol{S}$ un string de tamaño $\boldsymbol{n}$.

$$hPref[\ i\ ]: hash\ del\ substring\ S[0, i]$$

$$\boldsymbol{hPref[i] = (\ hPref[i-1] * B + S[i]\ )\%\ m}$$

Hashing Prefijos

```
string s;
cin >> s;
int n = s.size();
int B = 29, m = 1000000007;
long long hPref[ n ];
hPref[ 0 ] = s[ 0 ] - 'a' + 1;
for(int i = 1; i < n; ++i){
    hPref[ i ] = (hPref[ i - 1] * B + (s[ i ] - 'a' + 1)) % m;
}
```

Hashing Substrings

Sea $\boldsymbol{S}$ un string de tamaño $\boldsymbol{n}$.

$$hPref[\,i\,]: hash\ del\ substring\ S[0, i]$$

$$\boldsymbol{hSub_{i,j} = \left(hPref[j] - hPref[i-1].B^{j-i+1}\right) \% m}$$

Hashing Substrings

```
int hSub(int i, int j){
    if( i == 0 ) return hPref[ j ];
    return ((hPref[j] - hPref[i - 1] * pot[j - i + 1]) % m + m) % m;
}
```

Algoritmo de Rabin-Karp

Problema de String Matching

Dado un patrón $\boldsymbol{P}$ y un texto $\boldsymbol{T}$, determinar si P aparece en T y enumerar las posiciones de todas las ocurrencias.

Algoritmo de Rabin-Karp

Solución Trivial

Fuerza Bruta $\boldsymbol{O}(|\boldsymbol{T}||\boldsymbol{P}|)$

Algoritmo de Rabin-Karp

Solución Eficiente

$O(|T| + |P|)$

```
int hSub(int i, int j, long long hPref[], long long pot[], int M){
    if( i == 0 ) return hPref[ j ];
    return ((hPref[j] - hPref[i - 1] * pot[j - i + 1]) % M + M) % M;
}

void rabinKarp(string T, string P){
    int n = T.size(), m = P.size();
    int B = 29, M = 1000000007;
    long long pot[ n ], hPref[ n ];
    if( n < m ) return;
    pot[ 0 ] = 1;
    for(int i = 1; i < n; ++i){
        pot[ i ] = (pot[ i - 1 ] * B ) % M;
    }
    hPref[ 0 ] = T[ 0 ] - 'a' + 1;
    for(int i = 1; i < n; ++i){
        hPref[ i ] = (hPref[ i - 1] * B + (T[ i ] - 'a' + 1)) % M;
    }
    int hPatron = 0;
    for( int i = 0; i < m; ++i){
        hPatron += ((P[i] - 'a' + 1) * pot[m - 1 - i]) % M;
    }
    for(int i = 0; i <= n - m; ++i){
        if( hSub(i, i + m - 1, hPref, pot, M) == hPatron ) cout << i << endl;
    }
}
```

Aplicaciones del Hashing

- Cantidad de strings distintas en un arreglo.
- Cantidad de substrings distintas de un string.
- Cantidad de substrings palindrómicas de un string.

Problemas

Codeforces 7D – Palindrome Degree

SPOJ EPALIN – Extend to Palidrome

SPOJ LPS – Longest Palindromic Substring

Referencias

- Cormen, Introduction to Algorithms
- E-maxx, String Hashing

¡ Good luck and have fun !