TAD Diccionario y Tablas de Hash.

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y T.I.

CI-2612: Algoritmos y Estructuras II

Plan

Sobre los Diccionarios

Definición Diccionario

Repertorio en forma de libro o en soporte electrónico en el que se recogen, según un orden determinado, las palabras o expresiones de una o más lenguas, o de una materia concreta, acompañadas de su definición, equivalencia o explicación [Diccionario RAE].

Características del TAD Diccionario

- Nos interesan Diccionarios del tipo catálogo
- Es un TAD Conjunto con las operaciones `Agregar`, `Eliminar` y `Buscar`
- Se quiere que las operaciones en promedio sean $O(1)$
- Permite almacenar objetos que poseen una clave y un valor
- Los objetos serán identificados unívocamente por la clave
- Las operaciones más frecuentes son `Agregar` y `Buscar`
- Se quiere que dada una clave, sea posible obtener el valor asociado en forma eficiente

Modelo abstracto de representación del TAD Diccionario

Especificación $\mathbb{A}$ **de TAD** $\mathit{Diccionario}\,(\mathit{T0}, \mathit{T1})$

Modelo de Representación

const MAX : int

var conoc : set $\mathit{T0}$

tabla : $\mathit{T0} \nrightarrow \mathit{T1}$

Invariante de representación del TAD Diccionario

Especificación $\mathbb{A}$ **de TAD** $\mathit{Diccionario}\,(\mathit{T0}, \mathit{T1})$

Modelo de Representación

const MAX : int

var conoc : set $\mathit{T0}$

tabla : $\mathit{T0} \nrightarrow \mathit{T1}$

Invariante de Representación

$$MAX > 0 \quad \wedge \quad \#\,\mathit{conoc} \leqslant MAX \quad \wedge \quad \mathit{conoc} = \mathrm{dom}\,\mathit{tabla}$$

Operaciones del TAD Diccionario parte 1

Operaciones

proc *crear* (**in** m : int ; **out** d : *Diccionario*)

{ **Pre** : $m > 0$ }

{ **Post** : $d.MAX = m \quad \wedge \quad d.conoc = \emptyset \quad \wedge \quad d.tabla = \emptyset$ }

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)

{ **Pre** : $c \notin d.conoc \quad \wedge \quad \# d.conoc < d.MAX$ }

{ **Post** : $d.conoc = d_0.conoc \cup \{ c \} \quad \wedge \quad d.tabla = d_0.tabla \cup \{ (c, v) \}$ }

TAD Diccionario | Especificación del TAD Diccionario

Operaciones del TAD Diccionario parte 2

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)

{ **Pre** : $c \in d.conoc$ }

{ **Post** : $d.conoc = d_0.conoc - \{ c \} \quad \wedge \quad d.tabla = d_0.tabla - \{ (c, d_0.tabla\ c) \}$ }

proc *buscar* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** v : *T1*)

{ **Pre** : $c \in d.conoc$ }

{ **Post** : $v = d.tabla\ c$ }

proc *existe* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** e : boolean)

{ **Pre** : true }

{ **Post** : $e \equiv (c \in d.conoc)$ }

Especificación $\mathbb{A}$ **de TAD** *Diccionario* (*T0*, *T1*)

Modelo de Representación

const MAX : int
var $conoc$: set $T0$
$tabla : T0 \nrightarrow T1$

Invariante de Representación

$MAX > 0 \;\; \wedge \;\; \# conoc \leqslant MAX \;\; \wedge \;\; conoc = \text{dom}\, tabla$

Operaciones

proc *crear* (**in** m : int ; **out** d : *Diccionario*)
{ **Pre :** $m > 0$ }
{ **Post :** $d.MAX = m \;\; \wedge \;\; d.conoc = \emptyset \;\; \wedge \;\; d.tabla = \emptyset$ }

proc *agregar* (**in-out** d : *Diccionario* ; **in** c : *T0* ; **in** v : *T1*)
{ **Pre :** $c \notin d.conoc \;\; \wedge \;\; \# d.conoc < d.MAX$ }
{ **Post :** $d.conoc = d_0.conoc \cup \{ c \} \;\; \wedge \;\; d.tabla = d_0.tabla \cup \{ (c, v) \}$ }

proc *eliminar* (**in-out** d : *Diccionario* ; **in** c : *T0*)
{ **Pre :** $c \in d.conoc$ }
{ **Post :** $d.conoc = d_0.conoc - \{ c \} \;\; \wedge \;\; d.tabla = d_0.tabla - \{ (c, d_0.tabla\ c) \}$ }

proc *buscar* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** v : *T1*)
{ **Pre :** $c \in d.conoc$ }
{ **Post :** $v = d.tabla\ c$ }

proc *existe* (**in** d : *Diccionario* ; **in** c : *T0* ; **out** e : boolean)
{ **Pre :** true }
{ **Post :** $e \equiv (c \in d.conoc)$ }

Características de tablas con direccionamiento directo

- Las claves de los objetos son distintas
- Cada clave proviene de un universo de posibles claves $U = \{0, 1, \ldots, m-1\}$
- La implementación consiste en arreglo en donde se almacenan las claves
- La tabla se representa con un arreglo $T[0, \ldots, m-1]$
- A cada clave de U le corresponde una casilla de T sin solapamiento.
- Un objeto con valor x y clave k puede ser almacenado en $T[k]$, también puede ser almacenado un apuntador a x
- Si no hay elementos en la tabla con clave k, la casilla $T[k]$ es NIL

Tablas con direccionamiento directo

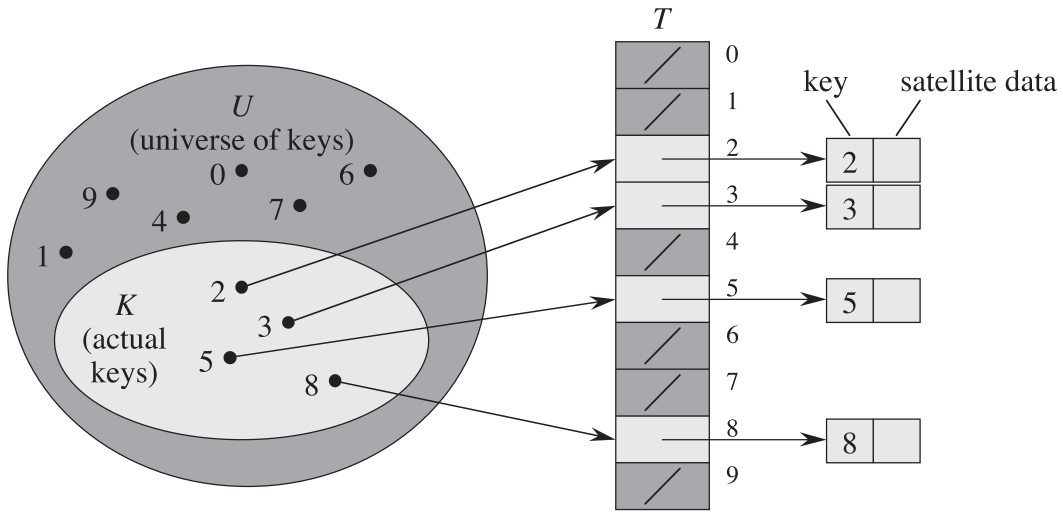

Figura: Ejemplo del almacenamiento de un conjunto de claves $k = \{2, 3, 5, 8\}$ del universo de claves $U = \{0, 1, \ldots, 9\}$. Fuente [1]

Operaciones de tablas con direccionamiento directo

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

1 $T[x.key] = $ NIL

Figura: Todas las operaciones son $O(1)$. Fuente [1]

Características de las tablas de hash

- En las tablas con direccionamiento directo si K es mucho más pequeño que U, hay una desperdicio de almacenamiento
- Se quiere reducir el almacenamiento de las tablas con direccionamiento directo, pero teniendo $O(1)$ en el caso promedio
- Se usa función de hash h para obtener la casilla en la tabla T que le corresponde a cada clave k.
- Para una tabla $T[0,..,m-1]$, se tiene una función $h : U \to \{0, 1, \ldots, m-1\}$
- Es decir, la función de hash h obtiene la casilla en $h(k)$ en T, donde se va almacenar el objeto con clave k y valor x.
- Hay una reducción del espacio usado porque el tamaño de la tabla T es m y no $|U|$

Tablas de hash

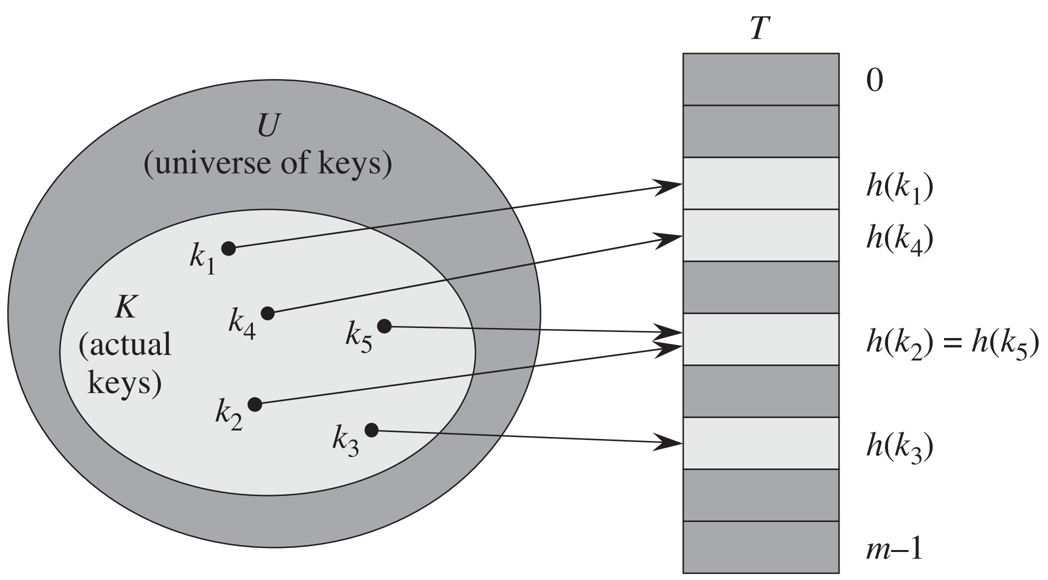

Figura: Ejemplo del uso de una función de hash para asignar claves a las casillas de una tabla. Las claves k_2 y k_5 colisionan. Fuente [1]

Colisiones de las tablas de hash

- Una colisión ocurre cuando una función de hash asigna una misma casilla a dos claves
- Una buena función de hash es aquella que evita colisiones
- Si $|K| \leq m$ el chance de una colisión es bajo si se tiene una buena función de hash
- Si $|K| > m$ la colisión es inevitable
- Se quiere diseñar tablas de hash que manejen efectivamente las colisiones
- Estrategias para manejar las colisiones:
  - Tablas de hash usando encadenamiento
  - Tablas de hash basadas en direccionamiento abierto

Tablas de hash usando encadenamiento

- Se colocan los elementos que están asociados a una casilla en una lista enlazada (simple o doble)
- La casilla i almacena un apuntador a la cabeza de una lista enlazada.
- Los elementos de la lista enlazada son aquellos para los cuales sus claves obtienen el mismo valor de la función de hash.
- Las colisiones se resuelven colocando a los elementos con un mismo valor de h en una lista enlazada.

Tablas de hash usando encadenamiento

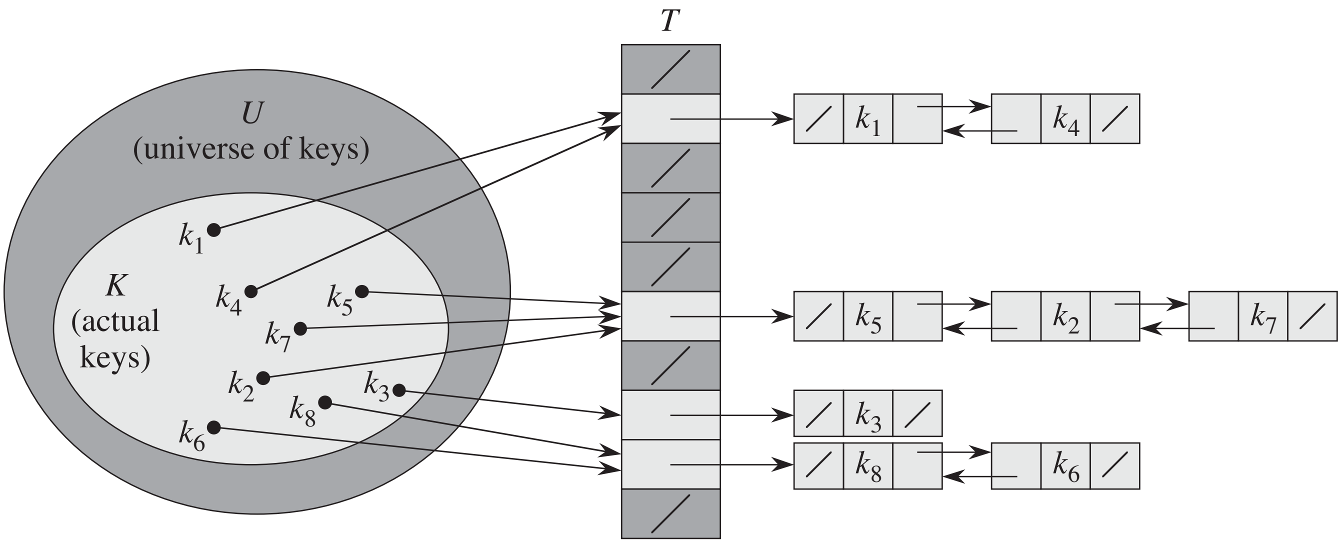

Figura: Ejemplo de una tabla de hash donde las colisiones se resuelven por encadenamiento. Fuente [1]

Evitando colisiones en las tablas de hash usando encadenamiento

- El tamaño de la tabla es generalemnte $1/5$ a $1/10$ del numeros de elementos a almacenar
- Los elementos de la lista enlazada no estan ordenados
- Se usa lista doblemente enlazada para en caso de querer remover los elementos rápidamente

Operaciones de las tablas de hash usando encadenamiento

CHAINED-HASH-INSERT(T, x)

1 insert x at the head of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

1 search for an element with key k in list $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

1 delete x from the list $T[h(x.key)]$

Figura: Fuente [1]

Análisis de las tablas de hash usando encadenamiento

- Peor caso de insertar es $O(1)$. Se asume que los elementos no están en la lista
- Peor caso de buscar depende del número de elementos en la lista enlazada.
- Si todos los elementos n están en la misma casilla, la búsqueda es $O(n)$
- Peor caso de eliminar depende de la búsqueda del elemento en la lista enlazada.
- Si todos los elementos n están en la misma casilla, la eliminación es $O(n)$ si la lista es simplemente enlazada y $O(1)$ si es doblemente enlazada

Análisis de las tablas de hash usando encadenamiento

Definición de factor de carga α

Sea m el número de casillas de una tabla de hash y n el número de elementos en una tabla de hash. El factor de carga es $\alpha = \frac{n}{m}$

Teorema

La búsqueda de una clave k que **no** se encuentra en la tabla de hash, tiene un tiempo experado de $\Theta(1 + \alpha)$

Teorema

La búsqueda de una clave k que **si** encuentra en la tabla de hash, tiene un tiempo experado de $\Theta(1 + \alpha)$

- Si $n = O(m)$, entonces $\alpha = \frac{n}{m} = O(1)$, lo que indica que la búsqueda toma tiempo constante en promedio.

¿Qué hace que una función hash sea buena?

- Debe de fácil y eficiente computación
- Satisface que cada clave es igualmente probable de ser asignada a cada casilla de la tabla (*simple uniform hashing*)
- La probabilidad de que una clave sea asignada a una casilla no depende de que la casilla este o no asignada
- Los valores que se obtienen deben ser idependientes de cualquier patrón que exista en los datos de entrada
- En algunos casos debe tener propiedades adicionales, por ejemplo: el valor obtenido es cercano a la clave obtenida

Interpretando claves como números naturales

- La mayoría de funciones de hash asume que las claves son números naturales
- Si una clave no es un número natural, se busca la forma de interpretarla como un número natural
- En muchos casos se puede realizar un método para trasformar una clave cualquiera en número natural
- Se va a asumir que las claves son números naturales

Método de la división

- Hace la asignación de una clave k a una de las m casillas por tomar el resto de dividir k entre m, esto es

$$h(k) = k \mod m$$

- La ventaja es que la computación de esta función es rápida y fácil de implementar
- Se debe evitar el uso de ciertos valores de m, por ejemplo potencias de 2 y números que no sean primmos
- Se recomienda usar números que sean primos y que no sean un número cercano a una potencia de 2

Método de la multiplicación

- Consiste en los siguientes pasos:
  - Se multiplica la clave k por una constante A, donde $0 < A < 1$
  - Se extrae la parte fracional de kA
  - Se multiplica la parte fraccional por m
  - Se toma toma el resultado del operador piso (floor) del resultado
  - $$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor = \lfloor m(kA \mod 1) \rfloor$$
- La desventaja de este método es que es más lento que el método de la división
- El tamaño m de la tabla no es importante, como en el método de división
- Generalmente se usa una potencia de 2 como valor de m

Referencias

T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
Introduction to Algorithms.
McGraw Hill, 3ra edition, 2009.