29/Oct

Esteban

Función de Hash

- Queremos

  Uniformidad Simple

$$\forall j \sum_{k \in K : h(k) = j} P(k) \approx \frac{1}{|N|}$$

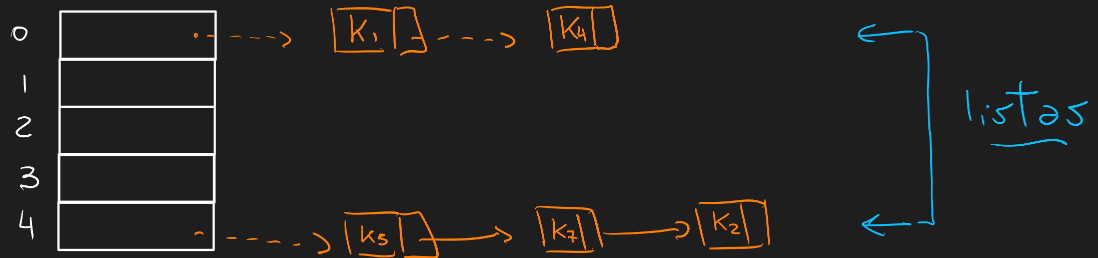
- Difícil de Construir

  Buscamos independencia de la distribución de los datos

Hashing por Concatenación

ej: $h(x) = x \mod 5$

← tamaño de la tabla



Insert(elem, k):

- calcula hash $h(k)$ y en esa posición de la tabla inserta al principio de la lista que allí existe.
- Costo: $O(1)$

Buscar (K) :

Búsqueda lineal en la lista asociada a h(K)

Costo : O (long de la lista asociada)

Delete (K) :

Lo mismo que buscar

Tamaño de las listas

n = # elem en tabla

$\alpha = \frac{n}{|T|}$: factor de carga

- Teorema: bajo la hipótesis de simplicidad uniforme de la función de hash, si las colisiones se resuelven por concatenación, en promedio
  - una búsqueda fallida requiere tiempo $\Theta(1+\alpha)$
  - una búsqueda exitosa requiere tiempo $\Theta(1+\alpha/2)$

Como $\alpha = \frac{n}{|T|}$

si $n \approx |T| \Rightarrow \alpha = cte \Rightarrow$ Costo : O(1)

(siempre que la función de hash esté bien distribuida)

Direccionamiento Abierto

Todos los elem. se almacenan EN la tabla

Dirección = h(k, i)

i-ésimo intento

```
insertar (el, k, T) es
  i ← 0;
  mientras (T[h(k, i)] está ocupada e (i<|T|))
      incrementar i;
  si (i < |T|), hacer T[h(k, i)] ← (el,k)
  en caso contrario <overflow>
```

```
buscar (k,T)
  i=0;
  mientras ((k!= T[h(k, i)].clave) y T[h(k,i)]!=null
  e (i<|T|)) incrementar i;
  si (i < |T|) y T[h(k, i)]!=null entonces T[h(k,i)]
  en casocontrario <no está>
```

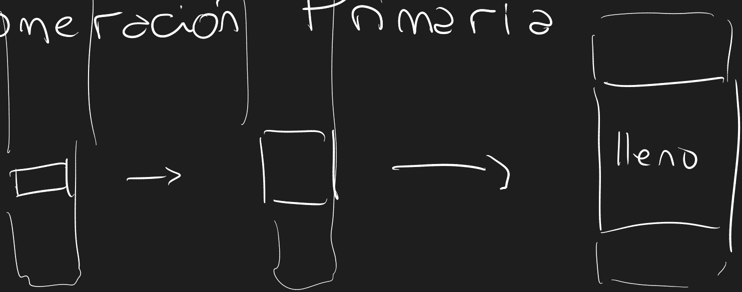
Para borrar

Dejamos una etiqueta, pues null puede romper todo

Barrido linear

- h(k, i) = (h'(k)+i) mod |T|, donde h'(k) es una función de hashing
- Se recorren todas las posiciones en la secuencia T[h'(k)], T[h'(k)+1], ... T[|T|], 0, 1, ..., T[h'(k)-1]
- Posibilidad de aglomeración <u>primaria</u>: dos secuencias de barrido que tienen una colisión, siguen colisionando
- Los elementos se aglomeran por largos tramos

Aglomeración Primaria



Barrido cuadrático

- $h(k, i) = (h'(k)+c_1i+c_2i^2) \bmod |T|$, donde $h'(k)$ es una función de hashing, c_1 y c_2 son constantes
- Ejemplos:
  - $h(k, i) = h'(k)+i^2$, $h(k, i+1) = h'(k)+(i+1)^2$, $i=1,\ldots, (|T|-1)/2$
  - $h(k, i) = h'(k)+i/2+i^2/2$, $|T|=2^x$
- Posibilidad de aglomeración secundaria: si hay colisión en el primer intento....sigue habiendo colisiones ($h'(k_1)= h'(k_2) \rightarrow h'(k_1,i)= h'(k_2,i)$)
- Describir $h(k, i)$ cuando $h'(k)=k \bmod |5|$

Hashing Doble

- Idea: que el barrido también dependa de la clave
- $h(k, i) = (h_1(k)+ih_2(k)) \bmod |T|$, donde $h_1(k)$ y $h_2(k)$ son funciones de hashing
- El hashing doble reduce los fenómenos de aglomeración secundaria
- Y no tiene aglomeración primaria

Construcción de funciones de Hash

División

- $h(k)=k \bmod |T|$
- Baja complejidad
- Aglomeraciones
  - No potencias de 2: si $|T| = 2^p$ entonces todas las claves con los p bits menos significativos iguales, colisionan
  - No potencias de 10 si las claves son números decimales (mismo motivo)
  - En general, la función debería depender de todas las cifras de la clave, cualquiera sea la representación
  - Una buena elección en la práctica: un número primo no demasiado cercano a una potencia de 2 (ejemplo: $h(k)=k \bmod 701$ para $|K|=2048$ valores posibles)

Partición

- Particionar la clave k en $k_1,k_2,\ldots,k_n$
- $h(k)=f(k_1,k_2,\ldots,k_n)$
- Ejemplo: la clave es un No. de tarjeta de crédito. Posible función hash:

$$\underbrace{477}\underbrace{2\,6}\underbrace{453}\underbrace{\,734}\underbrace{8} \rightarrow \{477, 264, 537, 348\}$$

$$f(477,264,537,348) = (477+264+537+348) \bmod 701 = 224$$

Extracción

- Se usa solamente una parte de la clave para calcular la dirección
- Ejemplo: Las 6 cifras centrales del número de tarjeta de crédito
  - 4772 6453 7348 → 264537