Tablas de hash 2

Fernando Schapachnik[1,2]

[1]En realidad... push('Fernando Schapachnik', push('Esteban Feuerstein', autores))
[2]Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II,
segundo cuatrimestre de 2018

(2) Repasemos tablas de hash

- Queríamos buscar en algo así como $O(1)$.
- El problema eran las colisiones.
- Vimos que podíamos manejarlas con hashing cerrado o abierto.
- En hashing abierto teníamos que definir $h(c, i)$ como la función que nos indicaba en qué posición de A debíamos intentar ubicar la clave c en el intento i.
- Hoy vamos a ver distintas formas de definir esa función.

(3) Barrido (y limpieza?)

- ¿Tendría sentido una función $h(c, i)$ que deje posiciones de la tabla sin explorar? No.
- Por eso se habla de *barrido*:
  - Lineal.
  - Cuadrático.
  - Hashing doble (o rehashing).
- Veamos...

(4) Barrido lineal

- La idea es ir recorriendo todas las posiciones en secuencia:
- $h(c, i) = (h'(c) + i) \bmod |A|$, donde $h'(c)$ es otra función de hashing.
- Veamos la secuencia que se genera:
  $A[h'(c)]$, $A[h'(c) + 1]$, ..., $A[|A|]$, $A[0]$, ...
- Ejemplo:
  - $h(c, i) = (h'(c) + i) \bmod 101$, $h'(c) = c \bmod 101$
  - Insertemos 2, 103, 104, 105, etc., en ese orden.
- Cada inserción colisiona varias veces antes de poderse realizar.
- Eso se llama aglomeración primaria, y si bien es un caso extremo, puede darse.
- Cuando hay aglomeración primaria, si dos secuencias colisionan en algún momento, siguen colisionando.

(5) Barrido cuadrático

- $h(c, i) = (h'(c) + a\ i + b\ i^2) \bmod |A|$, donde $h'(c)$ es un función de hash y a y b son constantes.
- Es una forma de evitar la aglomeración primaria, ya que el polinomio varía dependiendo del número de intento.
- Sin embargo, puede producirse aglomeración secundaria: si hay colisión en el primer intento, sigue habiendo colisiones ($h'(c_1) = h'(c_2) \ \Rightarrow \ h(c_1, i) = h(c_2, i)$).

(6) Hashing doble o rehashing

- Idea: que el barrido también dependa de la clave.
- $h(c, i) = (h_1(c) + i\ h_2(c)) \bmod |A|$, $h_1(c)$ y $h_2(c)$ son dos funciones de hashing.
- ¿Qué pasa con la aglomeración primaria y la secundaria?
- El hashing doble es muy poco probable que tenga aglomeración primaria.
- Y reduce los fenómenos de aglomeración secundaria.

(7) No todo es natural...

- Si las claves no son naturales, ¿cómo construimos buenas funciones de hash?
- La idea general es asociar a cada clave un número entero, en una forma que dependerá del contexto.
- Ejemplo: código ASCII de cada caracter, multiplicado por cierta base elevada a la posición.
- Ie, ascii(prim(s)) 2^0+ascii(prim(resto(s))) $2^1 + \ldots$

(8) Más allá de mod

- Una vez que nuestras claves son numéricas tenemos que ver cómo terminar de definir la función.
- Hay varias alternativas que varían en complejidad y en comportamiento con respecto a la aglomeración.

(9) Basadas en división

- $h(c) = c \bmod |A|$
- Tiene baja complejidad pero...
- Si $|A| = 10^p$ para algún p todas las claves cuyos últimos digitos coincidan colisionarán...
- Y si $|A| = 2^p$ para algún p lo mismo sucederá con los p bits menos significativos.
- 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 (eg, $h(c) = c \bmod 701$ para $|A| = 2048$ valores posibles).

(10) Partición y extracción

- Pueden ser útiles cuando la clave es muy larga.
- La idea es pensar a c como compuesto por segmentos $c_1 c_2 \ldots c_n$.
- En el caso de partición, se busca calcular $h(c) = h'(c_1, c_2, \ldots, c_n)$
- Ejemplo: partir el número de la tarjeta de crédito en cuatro partes y luego hacer $(c_1 + c_2 + c_3 + c_4) \bmod 701$.
- En el caso de extracción, la idea es quedarse sólo con algunos de los c_i.

(11) No sólo de tablas vive la función de hash

- Las funciones de hash tienen interés más allá del mundo de las estructuras de datos.
- Se usan mucho en seguridad.
- En cripto se utilizan hashes *de una vía*.
- Es decir, la idea es que sea prácticamente imposible obtener la preimagen.
- Se suele pedir que cumplan con:
  - Resistencia a la preimagen. Dado h debería ser difícil encontrar un m tal que $h = \text{hash}(m)$.
  - Resistencia a la segunda preimagen. Dado m_1 debería ser difícil encontrar un $m_2 \neq m_1$ tal que $\text{hash}(m_1) = \text{hash}(m_2)$.
  - Etc.
- Muy útiles para almacenar contraseñas (conviene que las contraseñas no se puedan leer). ¿Cómo hago?
- ¿Y para asegurar que el software bajado es el correcto?

(12) Bonus: un problema de seguridad

Ejemplo:

- Requerimiento: antender 1000 conexiones por segundo.
- Almacenamiento de las conexiones: tabla de hash, h= IP mod 1000
- Preparo el ataque: genero montones de pedidos de conexión que hasheen al mismo bucket.
- Complejidad: pasó de $O(1)$ a $O(n)$ para las búsquedas.
- Ataque: fuerzo a la aplicación a realizar búsquedas en el bucket ($O(n)$), consumiendo todo el tiempo de CPU.

(13) Más bonus: Universal Hashing

Concepto teórico: *Universal Hashing*

Si k la cantidad de celdas de la tabla de hash y $H = \{h_1 : \alpha \to [1 \dots k], \dots, h_n : \alpha \to [1 \dots k]\}$ es un conjunto de funciones de hash, definimos la cantidad de colisiones de H en los valores distintos i y j ($\#C_{i,j}^H$) como $\#\{h \in H \mid h(i) = h(j)\}$, es decir, la cantidad de funciones de H que hashean al mismo valor los elementos i y j.

H es *universal* ssi $\forall i \neq j \in \alpha : \#C_{i,j}^H \leq \frac{|H|}{k}$.

(14) Más bonus: Universal Hashing (cont.)

H es *universal* ssi $\forall i \neq j \in \alpha : \#C_{i,j}^H \leq \frac{|H|}{k}$.
¿Qué ventaja tiene que H sea universal?

Analicemos la probabilidad de que dos funciones de H colisionen en todos sus valores:

$$\frac{favorables}{totales = |H|} \leq \frac{min_{i,j}\ \#C_{i,j}^H}{|H|} \leq \frac{\frac{|H|}{k}}{|H|} = \frac{1}{k}$$

Cuando $k \to \infty \ \Rightarrow\ \frac{1}{k} \to 0$.

Entonces: selecciono la función de hash de una familia universal, al azar en tiempo de ejecución. El atacante no sabe cuál es la seleccionada (aunque conozca H). Aunque sus "chances" de adivinar la función son $\frac{1}{n}$, la probabilidad de que todos sus pedidos de conexión hasheen a la misma celda tiende a cero con k.