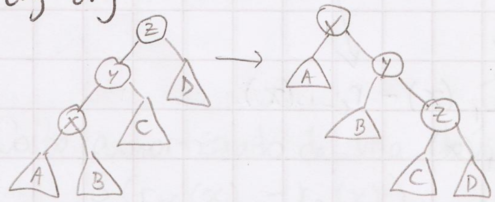
2015年10月27日 (火)

SPLAY TREES

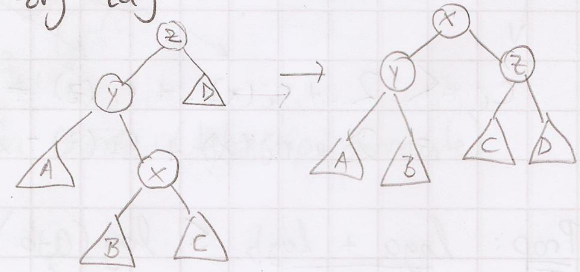
- Son árboles que se balancean de manera amortizada.
- Implica que en cada consulta el árbol cambia, en particular, al consultar un elemento, se harán balanceos (rotaciones) de tal forma que el elto quede en la raíz.
- Una secuencia de n operaciones tiene costo amortizado de $O(\log n)$ amortizado. si todos los eltos tienen prob uniforme $1/n$
- El costo es $O(H)$ entropía de Huffman.

Operaciones de rotación (recordar AVL)

zig-zig

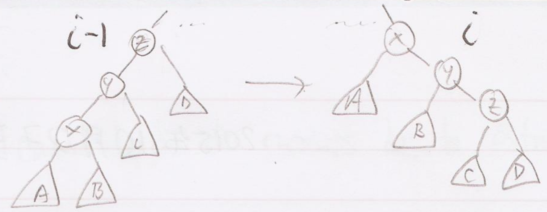


zig-zag



- zag-zig
- zag-zag
- zig } a dist 1 de
- zag } la raíz

Veamos costo de zig-zig ...



Fuimos a buscar x.

$S(x)$ = tamaño (# de nodos) del subárbol con raíz x (contando x)

$r_i(x) = \log_2 S(x)$ luego de la operación i

$$\phi_i = \sum_{x \in T} r_i(x)$$

En un zig-zig

$C_i = 2$ (dos rotaciones)

$\hat{C}_i = C_i + \phi_i - \phi_{i-1}$

los $r_i()$ de los nodos en A,B,C y D no cambian

$$\hat{C}_i = 2 + \cancel{r_i(x)} - r_{i-1}(x) + r_i(y) - r_{i-1}(y) + r_i(z) - \cancel{r_{i-1}(z)}$$

Notar que:
- $r_{i-1}(z) = r_i(x)$ (tienen el mismo $S()$)
- $r_{i-1}(y) > r_{i-1}(x)$
- $r_i(y) < r_i(x)$

$$\hat{C}_i < 2 + r_i(x) + r_i(z) - r_{i-1}(x) - r_{i-1}(x)$$
$$= 2 + r_i(x) + r_i(z) - 2r_{i-1}(x)$$

Prop: $\dfrac{\log a + \log b}{2} \leq \log\left(\dfrac{a+b}{2}\right)$

$\log ab \leq 2 \log\left(\dfrac{a+b}{2}\right)$

$\log ab \leq 2 \log(a+b) - 2$

$\log 4ab \leq \log (a+b)^2$

$4ab \leq a^2 + 2ab + b^2$

$0 \leq a^2 - 2ab + b^2$

$0 \leq (a-b)^2$

Luego $r_{i-1}(x) + r_i(z) = \log S_{i-1}(x) + \log S_i(z)$

$$\leq 2 \log \frac{S_{i-1}(x) + S_i(x)}{2} \quad \text{(por prop)}$$

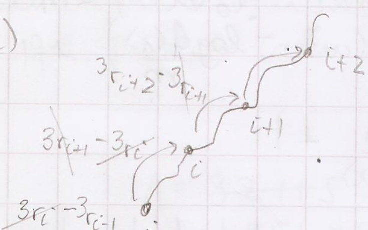
como $S_{i-1}(x) + S_i(z) < S_i(x)$

$$\Rightarrow \quad r_{i-1}(x) + r_i(z) < 2 \log \frac{S_i(x)}{2} = 2r_i(x) - 2$$

$$\therefore\ r_i(z) < 2r_i(x) - 2 - r_{i-1}(x)$$

$$\hat{c}_i < \cancel{2} + r_i(x) + 2r_i(x) \cancel{-2} - r_{i-1}(x) - 2r_{i-1}(x)$$

$$\hat{c}_i < 3\underbrace{(r_i(x) - r_{i-1}(x))}_{\text{huele a telescópica}}$$



Costo amortizado de una búsqueda suma (telescópicamente)

$$3(r_m(x) - r_0(x))$$

$$\leq 3\, r_m(x) = 3 \log n$$

Analizar para zig-zag, zag-zag, etc es análogo ... en zig y zag da como $3(\cdots) + 1$ o algo así pero solo 1 vez pues está debajo de la raíz.

Optimalidad estática.

Si el elemento x se busca $q(x)$ veces (de las m) entonces el Costo amortizado es $\Theta\left(\sum_{x\in T} \frac{q(x)}{m}\log\frac{m}{q(x)}\right)$

Le daremos un peso $w(x) = \frac{q(x)}{m}$ a x.

$$w = \sum_{x\in T} w(x) = \sum_{x \in T} \frac{q(x)}{m} = 1$$

$$S(x) = \sum_{V \text{ desc de } x} w(x) \quad , \quad r_i(x) = \log_2 S_i(x)$$

$$\begin{aligned}\hat{c}_i &\leq 3\,(r_m(x) - r_0(x)) + 1 \\ &= 3\left(\log(w) - \log\frac{q(x)}{m}\right) + 1 \\ &= 3\log\frac{m}{q(x)} + 1\end{aligned}$$

Si promediamos sobre todos los x, con su probabilidad

$$3\,\frac{q(x)}{m}\log\frac{m}{q(x)} + 1$$

$$1 + 3H.$$

Universos Discretos

2015年10月29日(木)

- Ordenar en $O(n)$
- Predecesor en $O(\log\log \underset{\text{universo}}{U})$
- Tries y árboles de sufijos   n strings $O(m)$

- Counting Sort
  - las claves están en $[1 \ldots U]$
  - n claves   no hay registros asociados
  - no hay nada más que las claves (no hay punteros. Dos números con el mismo valor son indistinguibles)

  → inicializar contadores $C[i] \leftarrow 0$
  → acumular el # de ocurrencias de cada clave
  → output el # de veces que aparece cada clave.

```
for i ← 1 to U
    C[i] ← 0
for j ← 1 to n
    C[A[i]] ← C[A[i]]+1
j ← 1
for i ← 1 to U
    for k ← 1 to C[i]
        A[j] ← i
        j ← j+1
```

A = ~~3 2 1 1 5 2 1 3 1 2 2~~
1 1 1 1 2 2 2 2 3 3 5

C:
1 □
2 □
3 L
4
5 1

Funciona porque los números iguales son indistinguibles.
Los elementos SON las claves
Complejidad $O(n+U)$
conviene solo cuando U es razonable. Gasta mucho espacio extra.

ll

Cuando los elementos sí son distinguibles cuando tienen = valor uso (los elementos son algo más que solo claves)

- Bucket Sort

  1° cuento

  2° inicializo punteros a la zona de A' donde se escriben los distintos clones

  3° paso por A copiando cada valor a su posición definitiva en A'

A = 3 2 1 1 5 2 1 3 1 2 2

P

C:
1 □ pos ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5
2 □ pos ~~5~~ ~~6~~ ~~7~~ ~~8~~ 9
3 L pos ~~9~~ ~~10~~ 11
4 pos 11
5 1 pos ~~11~~ 12

A' = 1 1 1 1 2 2 2 2 3 3 5
1 2 3 4 5 6 7 8 9 10 11

```
for i ← 1 to U
    C[i] ← 0
for j ← 1 to n
    C[A[i]] ← C[A[i]] + 1
P[1] ← 1
for i ← 2 to U
    P[i] ← P[i-1] + C[i-1]
for j ← 1 to n
    A'[P[A[j]]] ← A[j]
    P[A[j]] ← P[A[j]] + 1
```

Conserva la identidad de los eltos y el alg. es <u>estable</u> (preserva el orden relativo original entre claves iguales)

Complejidad $\Theta(n + U)$ también

↳ inicializar contadores

Luego podemos ordenar en tiempo lineal si $|U|$ es $\Theta(n)$

Bajo un modelo NO basado en comparaciones

¿Podemos ir más lejos?

Aux #7

17	1 0 0 0 1
5	0 0 1 0 1
14	0 1 1 1 0
2	0 0 0 1 0
6	0 0 1 1 0
8	0 1 0 0 0

Vamos a ordenar por el último bit usando bucket sort ($|U| = 2$!)

14	0 1 1 1 0
2	0 0 0 1 0
6	0 0 1 1 0
8	0 1 0 0 0
17	1 0 0 0 1
5	0 0 1 0 1

Ahora por el segundo

8	0 1 0 0 0
17	1 0 0 0 1
5	0 0 1 0 1
14	0 1 1 1 0
2	0 0 0 1 0
6	0 0 1 1 0

ahora están ordenados por los 2 últimos bits AL MISMO TIEMPO

pues bucket sort es ESTABLE

→

8	0 1 0 0 0
17	1 0 0 0 1
2	0 0 0 1 0
5	0 0 1 0 1
14	0 1 1 1 0
6	0 0 1 1 0

→

17	1 0 0 0 1
2	0 0 0 1 0
5	0 0 1 0 1
6	0 0 1 1 0
8	0 1 0 0 0
14	0 1 1 1 0

→

2	0 0 0 1 0
5	0 0 1 0 1
6	0 0 1 1 0
8	0 1 0 0 0
14	0 1 1 1 0
17	1 0 0 0 1

✓

$\Theta(n \log U)$ → cuántos bits se ocupan para escribir los elementos

↳ por qué no mejor elegir chunks más grandes que 1?

Puedo elegir chunks de $\log n$ bits para que el universo sea n (Recordar que Bucketsort funciona en $\Theta(n+U)$ si $|U|$ es $\Theta(n)$

$$\Rightarrow \Theta\left(n \frac{\log U}{\log n}\right) = \Theta(n \log_n U)$$

ej: $U = \Theta(n^6)$

$$\rightarrow \Theta\left(\frac{n \log n^6}{\log n}\right) = \Theta(6n) = \Theta(n)$$

esa constante puede llegar a competir con Quicksort.

OJO cuando U es muy grande