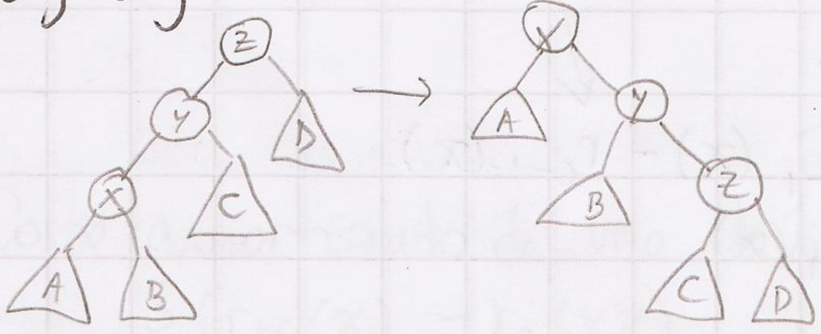
SPLAY TREES

2015年10月27日 (火)

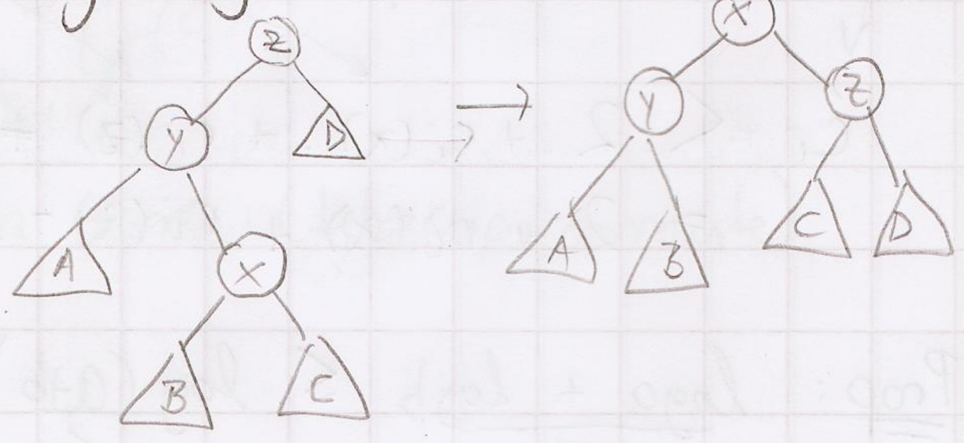
- Son árboles que se balancean de manera amortizada.
- Implica que en cada consulta el árbol cambia, en particular, al consultar un elemento, se harán balanceos de tal forma que el elto quede en la raíz. rotaciones
- Una secuencia de n operaciones tiene costo amortizado de $\theta(\log n)$ amortizado,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

Operaciones de rotación (recordar AVL)

zig-zig

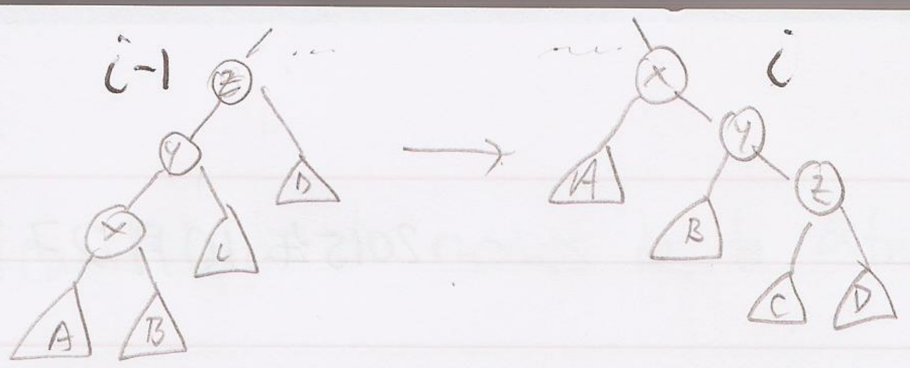


zig-zag



- 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 notaciones)
$\hat{c}_i = c_i + \phi_i - \phi_{i-1}$

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

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

Notar que: • $r_{i-1}(z) = r_i(z)$ (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: $\frac{\log a + \log b}{2} \leq \log\left(\frac{a+b}{2}\right)$

$$\log ab \leq 2\log\left(\frac{a+b}{2}\right)$$
$$\log ab \leq 2\log(a+b) - 2$$
$$\log 4ab \leq \log(a+b)^2$$
$$4ab \leq a^2 + 2ab + b^2$$
$$0 \leq a^2 - 2ab + b^2$$
$$0 \leq (a-b)^2$$

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

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

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

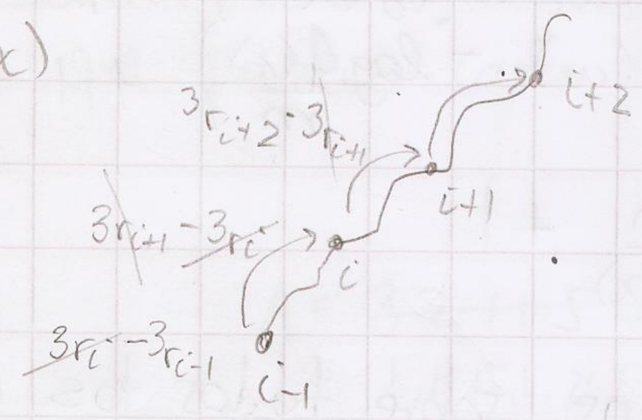
$\Rightarrow r_{i-1}(x) + r_i(z) < 2 \log \frac{S_i(x)}{2} = 2r_i(x) - 2$

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

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

$\hat{c}_i < 3(r_i(x) - r_{i-1}(x)$

suele 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.

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

$$S(x) = \sum_{y \text{ desde } x} w(x), \quad r_i(x) = \log_2 S_i(x)$$

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

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

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 $O(\log\log U)$ ← universo
- Tries y árboles de sufijos n strings $O(m)$

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

  → 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
```

$A = \cancel{3\,2\,1\,5\,2\,3\,1\,2\,2}$
$1\,1\,1\,2\,2\,2\,2\,3\,3\,5$

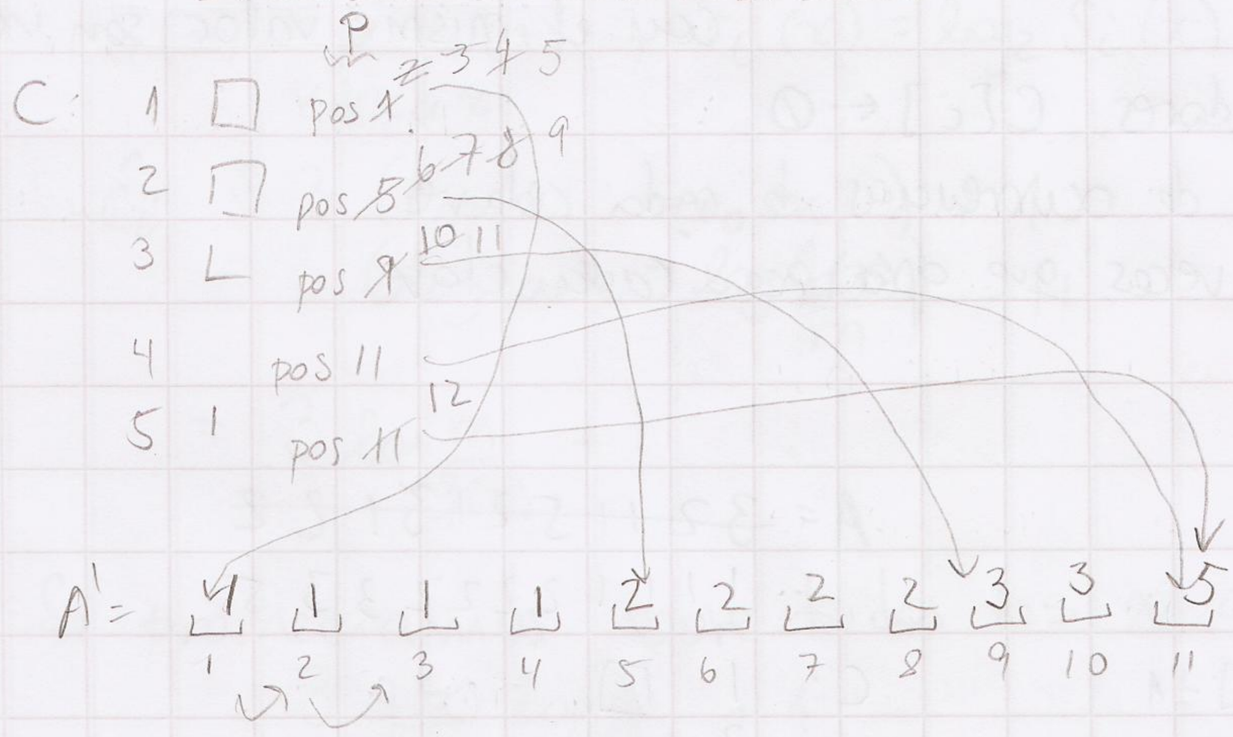
C: 1 □
2 □
3 ⌊
4
5 (1)

Funciona porque los números iguales son indistinguibles. Los elementos SON las claves
Complejidad $\Theta(n+U)$
Conviene solo cuando U es razonable. Gasto mucho espacio extra.

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




```
for i ← 1 to U
    C[i] ← 0
for j ← 1 to n
    C[A[j]] ← C[A[j]] + 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 estable (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?

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 ALMISMO 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 $\log_2 n$ bits para que el universo sea n

(Recordar que Bucketsort funciona en $\Theta(n+U)$ si $|U|$ es $\Theta(n)$

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

OJO cuando U es muy grande

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

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

esa constante puede llegar a competir con Quicksort.