

Aux #7

2015年11月2日(月)

[PI]

- a) Tiempo $\Theta(n)$
- b) Estable \Rightarrow Orden de los "repetidos" se conserva
- c) In-place \Rightarrow usa espacio adicional cte.

Pedir los 3 es absurdo.

a) + b) = $\Theta(n) + \text{estable} = \text{Bucket Sort / Counting Sort}$. (clases)

→ tomar arreglo del tamaño del universo.

estable, $\Theta(n+U) = \Theta(n+2) = \Theta(n)$

espacio: $\Theta(n+U) \rightarrow$ arreglo para contar $\boxed{\square}$
→ arreglo del tamaño de la entrada (copia ordenada)

b) + c) = Bubblesort (y parece que Insertion Sort)

estable, tiempo $\Theta(n^2)$, espacio $\Theta(1)$

(al parecer hay un mergesort in-place que tiene tiempo estable $\Theta(n \log^2 n)$)

(Quicksort, es estable?)

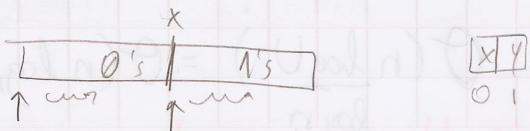
a) + c) { Hago una pasada por el arreglo contando cuántos 0's y

espacio $\Theta(1)$ 1's hay: $n_0 + n_1$ ($n_0 + n_1 = n$)

• Inicializa 2 punteros en 0 y n_0

• Ambos punteros avanzan hasta que el primero encuentra

tiempo $\Theta(n)$ { un 1 q el segundo un 0
• los intercambian / swap



P2

n números

$$U = n^2$$

Intentemos usar Radix Sort + 5 adv de atravesar

- Para cada dígito, desde el menos significativo al más significativo, ordeno usando BucketSort.

$$\Rightarrow \Theta(d \cdot (n+b))$$

\downarrow \downarrow \downarrow
nº dígitos elems base

n+b : counting/bucketsort

Si usamos $b = \log n$

\Rightarrow cada número tiene $\log(n^2)$ dígitos $\rightarrow \Theta(\log n)$

$$\Rightarrow$$
 tiempo $\Theta(\log n \cdot (n+2)) = \Theta(n \log n) \rightarrow \frac{n}{n}$

* Usamos base n

\Rightarrow cada número tendrá 2 dígitos

$$\Rightarrow$$
 radix sort toma $\Theta(2 \times (n+n)) = \Theta(2n) = \Theta(n)$

\uparrow \uparrow \uparrow
1 bucketsort por dígito elems buckets
0 0 1

Generalización

Si el universo es $[0, n^{k-1}]$

\Rightarrow en base n tienen a lo más k dígitos ...

$$\Rightarrow$$
 Radix Sort toma $\Theta(k(n+n)) = \Theta(kn)$

Aux #7

[P3]

si z es el límite teórico del número de bits necesarios para almacenar la información de una estructura, la estructura es **suscrita** si usa $\geq z + \Theta(z)$
↳ chica!
 $(\Theta(z) \Rightarrow$ compacta; $z + \Theta(1) \Rightarrow$ implícita)

se usan
nortos
comis
ops
básicas

{ Rank_b (B, i) : # de bits b en B[1, i]
Select_b (B, i) : Pos. de la i-ésima repetición de b.

a) • $\sum_{j=1}^r A[j]$

- el min valor de r tq $\sum_{j=1}^r A[j] \geq s$

Idea: usar representación unaria $\Leftrightarrow (4)_1 = 1111$

→ cada numero x pasa a ser una secuencia de x 1's

con un 0 al final.

=> Arreglo de $\Theta(n + t)$ bits.

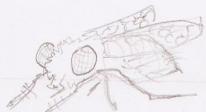
↳ tamaño arreglo

=> • Sum (r) = Rank₁ (select₀ (r))

• elegir el r-ésimo separador y contar 1's

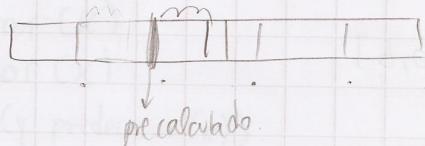
• Search (s) = Rank₀ (select₁ (s)) + 1

b) Como queremos una estructura suscrita entonces siempre vamos a trabajar en función de la cantidad de bits necesarios para representar la info.

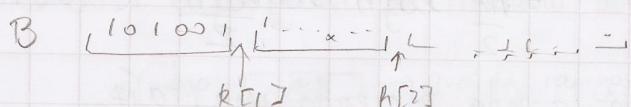


ÁRBOLES VENDEMANTE BOAS

La idea es dividir B , en bloques



Para cada bloque, guardaremos el valor de Rank "al final del bloque" en un arreglo R .



Si los bloques tienen tamaño b , cuando nos preguntén Rank (B, i), descompondremos $i = q \cdot b + r \Rightarrow$ el n^o de 1's hasta $q \cdot b$ es $R[q]$.

¿Qué b usar? $b = \lceil \log_2 n \rceil$ (en algunos controles lo han dado como hint)

¿Qué tamaño tiene R ? R tiene $\frac{n}{b}$ elem.

$$\Rightarrow |R| = \frac{n}{b} \cdot \log n = 2n \text{ bits}$$

$\underbrace{\text{elems}}_{\text{bits}}$ $\overbrace{\log n}^{\text{bits}}$
para representarlos.



Queda calcular el r (del $qb + r$):

O sea, el n^o de 1's en $B[qb+1, qb+r]$
Haremos una tabla con todos los posibles valores

$$\text{Si } b=4 : \quad T[1011, 3] = 2 \\ T[1011, 1] = 1$$

- $T[x, r] = n^{\circ}$ de bits en $x[1, r] \Rightarrow b \times 2^b$ elems
 ↑
 Patrón de bits de largo b
 $\in [0, 2^b - 1]$

- Espacio de T : $2^b \cdot b \cdot \log b = 2^{\frac{\log n}{2}} \cdot \log \frac{n}{2} \cdot \log \log \frac{n}{2}$ & chica
 $b = \frac{\log n}{2}$
 $(\textcircled{2}) \rightarrow$ ese 2 nos salvó la vida poniendo una raíz.

$$\text{Rank}(B, i) = R[q] + T[B[qb+1, qb+r], r]$$

$$\begin{cases} q = \lfloor \frac{i}{b} \rfloor \\ r = i \mod b \end{cases}$$

es $i = qb + r$ descomposición en tiempo constante?

$$\text{Espacio: } 2n + o(n)$$

- c) la tabla R es muy grande, por eso aparece $2n$

- Mantendremos T igual. Es $o(n)$
- Agregaremos un nivel más grande de bloques: superbloques \circlearrowleft
 de tamaño $S = (\frac{\log n}{2})^2$

- En $S[1, n/S]$ guardamos lo que antes iba en R , pero usando S en vez de b .

- Ahora R solo almacena el rank dentro del superbloque correspondiente.

$$R[q] = \text{rank}(B, qb) - S(\lfloor \frac{q}{\log n} \rfloor)$$

$|T| \leq o(n)$ igual que antes.

• $|S|: \frac{n}{S}$ elems, con valores hasta $n = \frac{n}{S} \log n = \frac{2n}{\log n} \in o(n)$

• $|R|: \frac{n}{b}$ elems, pero ahora los valores son más pequeños, solo hasta $O((\log n)^2)$

$$= \frac{n}{b} \cdot \log((\log n)^2) \text{ bits} = 2 \log \log n \cdot \frac{2n}{\log n} = 4n \cdot \frac{\log \log n}{\log n} \in o(n)$$

ÁRBOLES VAN EMDE BOAS

2015年11月3日 (K)

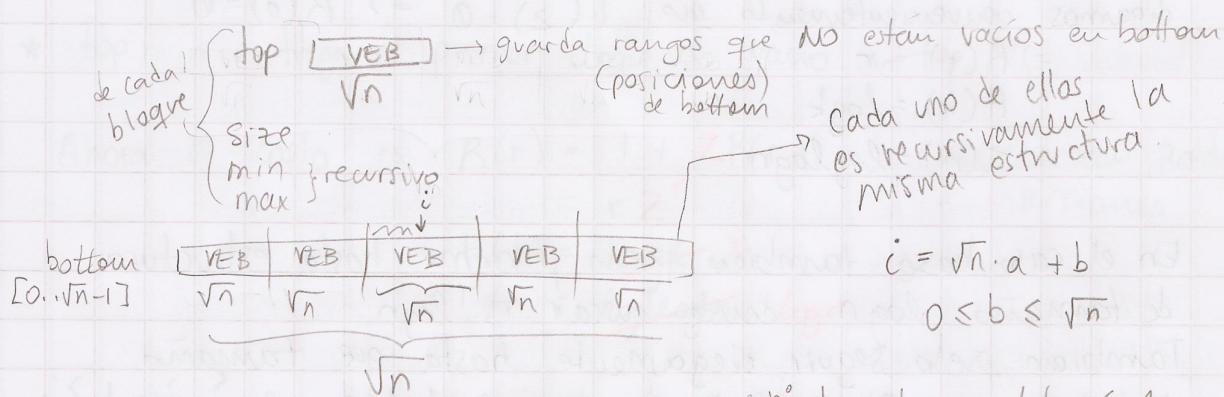
- insertar(x)
- borrar(x)
- sucesor(x)
(y predecesor(x))

Universo $[1 \dots n]$

tiempos $\Theta(\log \log n)$
espacio $\Theta(n)$

Es razonable usar esta
estructura cuando la cantidad de
elementos es similar al tamaño del universo

Lineal en el tamaño del
UNIVERSO



Cómo buscamos?

- **succ(i)** devuelve el primer alto mayor igual que i en

$$\text{Sea } i = a \cdot \sqrt{n} + b$$

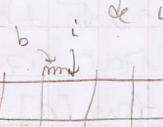
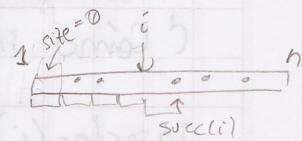
if $\text{bottom}[a].\text{size} > 0 \& \text{bottom}[a].\text{max} \geq b$ \leftarrow Significa que el sucesor

return $a\sqrt{n} + \text{bottom}[a].\text{succ}(b)$

$$c \leftarrow \text{top}. \text{succ}(a+1)$$

$$\text{return } c \cdot \sqrt{n} + c.\text{min}$$

? encontrar
el primer
bloque NO
vacío



bloque a (b es el 'offset')

podría haber un caso

base donde la estructura es

un árbol binario o algo, uso búsqueda binaria (en una hoja de tamaño Θ también puedo llegar a universo de tamaño 1).

conveniente

$\log n$, luego buscar en $\log \log n$ también)



size, min, max

$$\sqrt[n]{a} = a^{\frac{1}{n}} \rightarrow 1$$

$n \rightarrow \infty$

ARMAR BLOQUE

¿Cuánto toma? todo el costo a parte de la llamada recursiva

$$T(n) = 1 + T(n)$$

$$n=2^k \Rightarrow T(2^k) = 1 + T(2^{k/2})$$

$$H(k) = T(n) \Rightarrow H(k) = 1 + H(\frac{k}{2})$$

$$k=2^r$$

$$R(r) = H(k) \Rightarrow R(r) = 1 + R(r-1)$$

digamos convenientemente que $T(2) = 0 \Rightarrow R(0) = 0$

$$\Rightarrow R(r) = r$$

$$H(k) = \log k$$

$$T(n) = \log \log n$$

En el caso base también puedo pasarme hasta estructura de tamaño $\log n$, donde buscar es $\log n$. También puedo seguir ciegamente hasta que tamaño universo es 1, pero no es recomendable.

¿Cómo insertarlos? si el bloque estaba vacío, además de insertar en bottom hay que insertar en top.

insertar(i)

caso base, cualquier

sea $c = a\sqrt{n} + b$

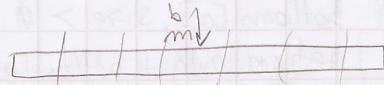
actualizar size, min, max

if $\text{bottom}[a].size = 0$

top.insertar(a)

bottom[a].insertar(b).

de toda
la estructura



top.size es el número de bloques.

borrar(i)

{ caso base cualquiera }

Sea $i = a\sqrt{n} + b$

bottom[a].borrar(b)

if bottom[a].size = 0

top.borrar(a)

actualizar size, min, max.

$$\min \leftarrow \text{top}.min + \sqrt{n} + \text{bottom}[\text{top}.min].min$$

* top.min entrega el primer bloque no vacío en top. (i) correcto

Ahora el costo es $\cdot R(r) = 1 + 2R(r-1)$

$$= r2^r$$

Cada inserción produce
2 inserciones

$$\cdot H(k) = \log k$$

$$\cdot T(n) = \log \log n$$

¿Solución? Cambiar invariantes de la estructura

Ahora \rightarrow el min de la estructura NO se almacena en los bottom (ni top), no guardo copias extras.

Ahora en SVCC(c) se agrega línea al comienzo

SVCC(i)

if $i \leq \min$ return \min

:

insertar(i)

{ if size = 0: min ← i, max ← i, size = 1
else if i < min: i ← min
sea $i = \sqrt{n} + b$
actualizar size, min, max
if bottom[a].size = 0
 top.insertar(a)
bottom[a].insertar

borrar(i)

{ if size = 1: size ← 0
else if i = min:
 min ← top.min, $\sqrt{n} + \text{bottom}[\text{top}.min].min$
 i ← min
 sea $i = \sqrt{n} + b$
 bottom[a].borrar(b)
if bottom[a].size = 0
 top.borrar(a)
actualizar size, max.

Espacio

$$\begin{aligned} T(n) &= 1 + (\sqrt{n} + 1) T(\sqrt{n}) \\ T(n) &\leq 2n - 3 \\ T(n) &\leq 1 + (\sqrt{n} + 1)(2\sqrt{n} - 3) \\ &= 1 + 2n - \sqrt{n} - 3 \\ &\leq 2n - 3 \end{aligned}$$

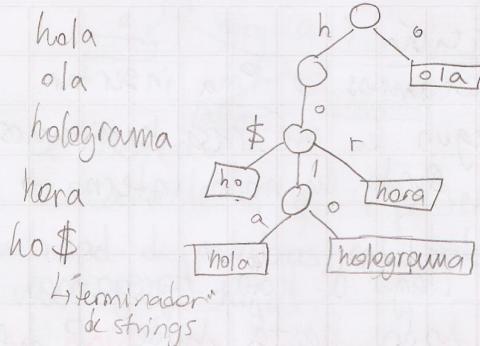
2015年11月5日 (木)

TRIES (árboles digitales)

Conjunto de n strings
Largo total L .

- espacio $\Theta(L)$
- + tiempo de búsqueda de string de largo m
- $\Theta(m)$
- o $\Theta(m \log n)$

tamaño alfabeto



- Para buscar $P[1..m]$, $P[m+1] = \$$
voy bajando desde la raíz por las letras $P[1], P[2], \dots$
hasta que
 - 1) no existe un hijo con label $P[i]$
→ P no está en el conjunto.
 - 2) llego a una hoja
→ comparo el resto de P con el resto del string en la hoja.

* Esto toma tiempo $\Theta(m)$... ¿cómo busco por cuál hijo bajar?

hash de hijos? ABB? lista ordenada y búsqueda bin?

↳ $\Theta(m)$ o $\Theta(m \log n)$

- Búsqueda de prefijos $P[1..m]$ (no le agrego "\$")
voy bajando desde la raíz por $P[1], P[2], \dots$ hasta que
 - 1) = a 1) anterior
 - 2) llego a una hoja → P solo puede ser prefijo de la cadena en esa hoja ≠ verificar
 - 3) se termina P en un nodo interno (como "ho" en el trié anterior)
→ P es prefijo de todas las hojas que descienden de ese nodo (predo guardar el númer de hijos en cada nodo, o recorrerlo entero)

鳥 イリ ポリヨ

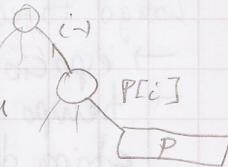
フライ
Fu rai

Inserción

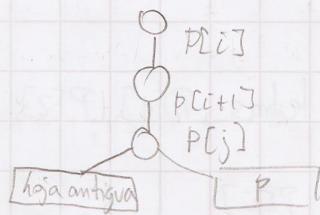
Buscamos el P a insertar.

Según en qué caso terminemos.

1) Quedo en nodo interno \rightarrow agregamos una hoja



2) Llego a hoja. Agregamos una secuencia de nodos hasta donde P difiere de la hoja a la que habíamos llegado.



proporcional al largo
de lo que inserto/borro
pues estoy a lo más "m"
pisos más abajo de la raíz

Borrar

- Encuentro P en una hoja hija de V

- borro la hoja

- mientras V tenga un solo hijo y sea hoja,

$u \leftarrow \text{padre}(v)$

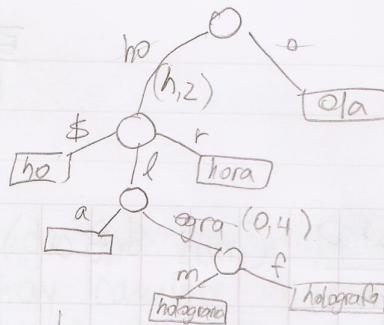
$\text{padre}(\text{hoja}) \leftarrow u$

borrar V

$v \leftarrow u$

} me gustaría hacer esto de una

hola
ola
holograma
hora
ho\$



holograma?
hiforma?

Blind Trie

- los nodos con un solo hijo se borran
- cada arista guarda el # de caracteres que representa.
- Espacio $\Theta(n)$ n cantidad de palabras, cada nodo tiene al menos 2 hijos.

- la distancia entre un nodo y su hoja no se pone. No es necesario.
- Para buscar bajo confiando en que calzan las letras entre medio. Al llegar a una hoja, debo comparar todo pues puede que no sea: Por ESO hay que guardar el string entero (en otro lado, asumimos que siempre es así) pues las aristas, el trie, no guarda información suficiente.

Buscar:

- igual que en trie, saltando los caracteres que no veo
- al llegar a una hoja, verificar todo P.

Buscar prefijo

- idem a buscar-prefijo de trie, pero en caso 3) comparar P con cualquier hoja que desciende del nodo. Si es prefijo de esa, es prefijo de todas, y sino, no lo es de ninguna.
- ★ Siempre pude arreglármelas para tener un puntero a una hoja desde cada nodo.

車 くろ 犬 いぬ テニス
te ni su

• Insertar P. (más difícil)

2 pasos

- 1) descubrir la posición de la primera diferencia
buscar P (y no encontrarlo)

caso a) buscar una hoja descendiente ~~cualquier~~
y encontrar 1º diferencia

caso b) buscar la 1º diferencia con esa hoja

- 2) colgar la nueva hoja a la profundidad correcta

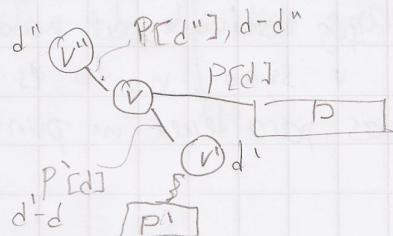
Volver a la raíz y bajar nuevamente hasta llegar a la profundidad buscada. Sea d' la profundidad deseada.

(caso a) hay un nodo explícito v a profundidad d'
→ a agregar una nueva hoja a v

La distancia entre ~~hijo~~
nodo y hoja no se pone

caso b) llego a un nodo v' de profundidad $d' > d$ y
su padre era v'' de profundidad $d'' < d$
(quedo en medio de una arista.)

→ corto la arista con un nuevo nodo v



Ojo que puede que haya que arreglar esto por error de ±1
en d, d', d''

馬

パトリシア

Borrar (más fácil. Puede que fuga que borrar padre, pero No abuelo(s), pues no hay caminos variados)

(Morrison 1968) ÁRBOLES PATRICIA = Blind Tries con $\underline{f=2}$; las cadenas son secuencias de bits.



→ Caminos de a lo más largo.
a seguir para Normalmente 8 veces si son 256 caracteres
buscar es hasta menos.

⇒ Tiene exactamente $n-1$ nodos. El anterior
tendrá un poquito menos, pero en ambos
es a lo sumo n .

ÁRBOL DE SUFIJOS (Suffix tree) (Weiner, 1973)

1 2 3 4 5 6 7 8 9 10 11 12
abracadabra \$

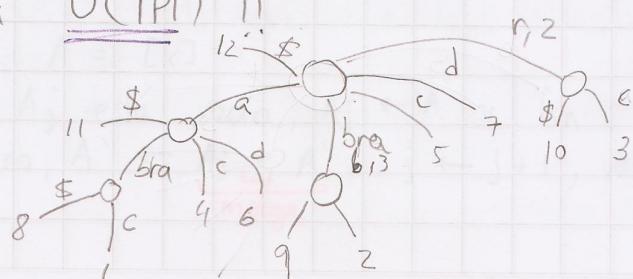
- Un string de largo n , tiene n sufijos (considera los strings con \$)
- todo substring en patron de largo n es un PREFIJO de un SUFIJO.

→ Substring = prefijo de un sufijo
 todos los substrings = P \equiv todos los sufijos que empiezan con P.
 → inserto todos los sufijos en un Patricia tree y hago búsqueda de prefijos por P.

→ obtengo los sufijos que empiezan con P
 ⇒ los substrings = P del texto.

En orden
y no en orden
del tamaño
del texto

$O(|P|)$



espacio $\Theta(n)$

Patricia tiene
 $O(n)$ nodos y
un string de tamaño
 n tiene n sufijos

PAT arrays (Mambré y Myers 1990)
Gonnet y Baeza-Yates 1993

Arreglo de fijos (suffix array) Solo las hojas son suficientes!

12, 11, 8, 1, 4, 6, [9, 2], 5, 7, 10, 3. Orden lexicográfico

buscar: $\Theta(m \log n)$

↳ aquí sí importa el largo del texto
Pues hay búsqueda binaria

Para construir se puede

hacer con quicksort !! (ordenar sufijos) Salvo si hay mucha repetición de patrones largos. Las comparaciones se vuelven muy costosas.