Colas de Prioridade [Priority Queues]

Note Title 9/3/2009

Prefacio: Incremento de Enteros

o "Analisis Amortizado"

* Cual es la complejidad de "n+1"?

- $\lg n$ en el peor caso si n es "grande"
- $O(1)$ siempre sin es "pequeño"

* Ejemplo

0000 0000 → +1 tiempo "O(1)"

0000 0001 → +1

0000 0010 (1)

(- - -)

0011 1111 → +1 tiempo "1+lg n"

0100 0000 → +1 tiempo "1"

0100 0001

* Cuál es la complejidad de n incrementaciones?

- $O(n \lg n)$
- $\lg 1 + \lg 2 + \lg 3 + \dots + \lg n/2 + \lg n/2 + 1 + \dots + \lg n$

$\Rightarrow \frac{n}{2} \lg n/2 \in O(n \lg n)$

- ⚠ $O(n)$ amortizado:

La mitad de las operaciones son $0 \to 1$ costo 1

$1/4$ $01 \to 10$ 2

$1/8$ $011 \to 100$ 3

$(- - -)$

En total se suman a $O(n)$

① Analisis Amortizado

- Funcion Potencial

$\varphi_0 = 0$

$\varphi_i =$ "Potencial" de la instancia ($\Rightarrow$ Costo de la op)

$\Delta\varphi_i = \varphi_{i+1} - \varphi_i$

$c_i \Rightarrow$ costo de operacion i

Costo amortizado $\bar{c}_i = c_i + \Delta\varphi_i$

$\sum \bar{c}_i = \sum c_i + \varphi_n - \varphi_0 \geq \sum c_i$

- Caso de incrementación binaria

φ_n = # unos en $\overline{n}^2$ ($\equiv$ n en notación binaria)

$\varphi_0 = 0$

$c_i = 1 +$ 1 $\neq$ # unos a la derecha

$\Delta\varphi_i = \begin{cases} -k+1 & \text{si } i = 2^k \\ \end{cases}$

$\sum \overline{c_i} = \underbrace{\sum c_i}_{\leq n} + \underbrace{\varphi_n - \varphi_0}_{\leq n}$

$\rightsquigarrow$ $2n$ en costo amortizado

Otra aplicaciones

Vector en Java

= un arreglo de tamaño variable

- iniciar con tamaño de 4
- doblar el tamaño cada i^{th} adición

$\Rightarrow O(n)$ peor caso

$\Rightarrow O(1)$ amortizado

② Colas de Prioridades (Priority Queues)

* Dominio

  Conjunto de llaves con prioridades (ordenadas)

* Operaciones Usuales

  M. Build $(e_1, \ldots, e_n)$

  M. insert (e)

  M. min ()

  M. deleteMin

* Mas operaciones (Colas con identificadores)

M. insert(e) : devolviendo un identificador h

M. remove(h)

M. decreaseKey(h, k) k = nueva prioridad

M. merge(Q)

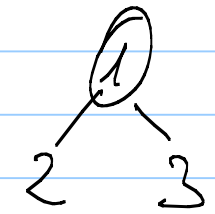
Soluciones Conocidas

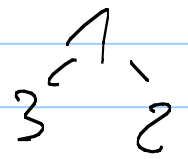
	Lista desorden	Lista orde	Arbol binario	(Min) Heap Binary Heap
Insert	$O(1)$	$O(n)$		$O(\lg n)$
min	$O(n)$	$O(1)$	Tarea	$O(1)$
deletemin	$O(n)$	$O(1)$		$O(\lg n)$
delete	$O(1)$	$O(1)$		$O(\lg n)$
merge	$O(1)$	$O(n)$		$O(m \lg(n+m))$
build	$O(n)$	$O(n)$		$O(n)$

Trade off

Flexibilidad de Representacion

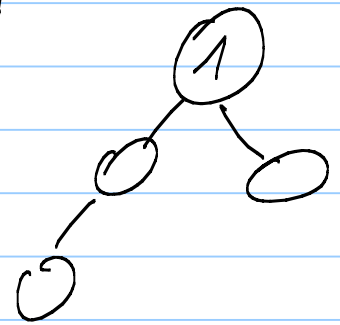
$\{1, 2, 3\} \longrightarrow$





$\{1, 2, 3, 4\} \longrightarrow$ #posibilidades?

$2 * 2 * 1$



$\rightsquigarrow$ Buscando estructuras con mas flexibilidad

$\Longrightarrow$ "Pairing Heaps"

③ Colas de Prioridades Binomiales (Binomial Heaps)

Descripción

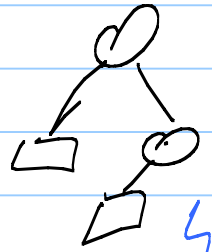
Arbol binomial

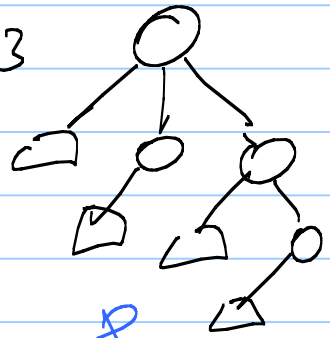
(de orden k) tiene exactamente k hijos y los hijos tienen orden distintos

Ej

$k=0$  1

$k=1$  2

$k=2$  4

$k=3$  8

Bosque binomial

una lista de arboles binomieles de orden distintos

Propriedades

- hay no mas que un arbd de orden i ↑
- Para n fijado, Hay exactamente 1 bosques
- Este descomposición // descomposición en binaria

⇒ Hay a lo maximo $\lg n+1$ arboles

Definicion Una "cola binomial" es un bosque binomial con la condicion de heap en los arboles.

Operadores

Union

* dos arboles de mismo orden k:
  compara la raiz y
  crea un arbol de orden k+1

* dos bosques binomiales:

Como la addicion de dos enteros exprimidos en base dos

Insert

Como +1 en enteros:
crea un arbol binomial de orden 0
y hace la union si y hay uno.

Minima: busque a dentro de los raices de los $\log n$ arboles

DeleteMin: Divida un arbol en dos,
y recursivamente ($O(\lg n)$) correga el bosque

Decrease key : en el arbol, $O(\lg n)$

Delete : Decrease key + Delete Min.

④ Fibonacci Heap

mas relajacion sobre la estructura de los arboles, para una mejor performancia amortizada.

5) Resumen

	Lista desorden	Lista orde	Arbol binario	(Min) Heap Binary Heap	Binomial Heap	Fibonacci Heap
Insert	$O(1)$	$O(n)$		$O(\lg n)$	$O(\lg n)$	$O(1)$
min	$O(n)$	$O(1)$	Foreo	$O(1)$	$O(1)$	$O(1)$
deletemin	$O(n)$	$O(1)$		$O(\lg n)$	$O(\lg n)$	$O(\lg n)^*$
delete	$O(1)$	$O(1)$		$O(\lg n)$	$O(\lg n)$	$O(\lg n)^*$
merge	$O(1)$	$O(n)$		$O(m \lg(n+m))$	$O(\lg n)$	$O(1)$
build	$O(n)$	$O(n)$		$O(n)$	$O(n)$	$O(n)$
decreasekey	$O(1)$	$O(n)$		$O(\lg n)$	$O(\lg n)$	$O(1)^*$

$* =$ Amortizado