Algoritmos y Programación II – Cátedra Lic. Gustavo Carolo

Evaluación Final – 2008-03-05

--- Entregar teoría y práctica por separado --- Leer bien el enunciado ---

Nombre:

Padrón:

Mail:

Cuatrimestre cursado: T: P: F:

Teoría Tema 1

Dado el siguiente lote de datos:

130, 276, 16, 279, 359‡, 339, 72, 92‡, 89‡, 321, 341‡, 5, 214, 161, 48, 220‡, 154, 94‡, 110, 269, 322‡

Se pide:

a) Mostrar cómo queda el B-tree de orden 2 resultante de ingresar los datos en el orden dado. Mostrar los resultados parciales luego de insertar un elemento marcado con ‡.
   Mostrar como queda el B-tree obtenido tras eliminar los siguientes elementos en el orden dado:

   214, 161, 269

   En este caso no es necesario mostrar el árbol para cada eliminación.
b) Desarrollar el algoritmo Heap sort en orden descendente (los elementos quedan ordenados de mayor a menor) hasta congelar 4 elementos.
c) Desarrollar el algoritmo de selección de reemplazo con un buffer de tamaño 3.
d) Desarrollar todas las fases del algoritmo de merge polifásico para 3 vías de entrada con las particiones obtenidas en el punto c.
e) Tanto el algoritmo Heap sort como el Insertion sort son del tipo de inserción (para cada elemento buscan el mínimo/máximo y lo congelan). Sin embargo el algoritmo Heap sort es más veloz bajo ciertas condiciones. Deduzca que condiciones son y por qué se vuelve más veloz.

a) Insertar: 130, 276, 16, 279, 359, 339, 72, 92, 89, 321, 341, 5, 214, 161, 48, 220, 154, 94, 110, 269, 322. Luego eliminar el 214, el 161 y finalmente el 269

Resolucion:
Primero inserto 5 elementos hasta obtener una raiz: 130, 276, 16, 279, 359

(276)
(16 | 130) (279 | 359)

Luego inserto: 339, 72, 92

(276)
(16 | 72 | 92 | 130) (279 | 339 | 359)

Insertando: 89 obtengo mi segundo elemento de la raiz

(89 | 276)
(16 | 72) (92 | 130) (279 | 339 | 359)

E insertando: 321, 341 logro un tercer elemento en la raiz

(89 | 276 | 339)
(16 | 72) (92 | 130) (279 | 321) (341 | 359)

Llegamos al cuarto elemento de la raiz insertando: 5, 214, 161, 48, 220

(89 | 161 | 276 | 339)
(5 | 16 | 48 | 72) (92 | 130) (214 | 220) (279 | 321) (341 | 359)

Completamos un poco más con: 154, 94

(89 | 161 | 276 | 339)
(5 | 16 | 48 | 72) (92 | 94 | 130 | 154) (214 | 220) (279 | 321) (341 | 359)

Y finalizamos el arbol obteniendo un nivel mas con: 110, 269, 322

(161)

(89 | 110) (276 | 339)

(5 | 16 | 48 | 72) (92 | 94) (130 | 154) (214 | 220 | 269) (279 | 321 | 322) (341 | 359)

Ahora debemos eliminar: 214, 161, 269.
Eliminar 214 no tiene complicacion. Eliminamos 161 y obtenemos.

(89 | 110 | 276 | 339)
(5 | 16 | 48 | 72) (92 | 94) (130 | 154 | 220 | 269) (279 | 321 | 322) (341 | 359)

y sin el 269 obtenemos el resultado final

(89 | 110 | 276 | 339)

(5 | 16 | 48 | 72) (92 | 94) (130 | 154 | 220) (279 | 321 | 322) (341 | 359)

c) Selección de reemplazo con buffer de tamaño 3:

Buffer: XX XX XX
No sale nada, Entra 130

Buffer: 130 XX XX
No sale nada, Entra 276

Buffer: 130 276 XX
No sale nada, Entra 16

Buffer: 130 276 16
Sale 16, Entra 279

Buffer: 130 276 279
Sale 130, Entra 359

Buffer: 359 276 279
Sale 276, Entra 339

Buffer: 359 339 279
Sale 279, Entra 72

Buffer: 359 339 72
Sale 339, Entra 92

Buffer: 359 92 72
Sale 359, Entra 89

Buffer: 89 92 72
Nueva particion por buffer lleno!
Se cierra la particion p1: 16, 130, 276, 279, 339, 359
Comienza saliendo 72, Entra 321

Buffer: 89 92 321
Sale 89, Entra 341

Buffer: 341 92 321
Sale 92, Entra 5

Buffer: 341 5 321
Sale 321, Entra 214

Buffer: 341 5 214
Sale 341, Entra 161

Buffer: 161 5 214
Nueva particion por buffer lleno!
Se cierra la particion p2: 72, 89, 92, 321, 341
Comienza saliendo 5, Entra 48

Buffer: 161 48 214
Sale 48, Entra 220

Buffer: 161 220 214
Sale 161, Entra 154

Buffer: 154 220 214
Sale 214, Entra 94

Buffer: 154 220 94
Sale 220, Entra 110

Buffer: 154 110 94
Nueva particion por buffer lleno!

Se cierra la particion p3: 5, 48, 161, 214, 220
Comienza saliendo 94, Entra 269

Buffer: 154 110 269
Sale 110, Entra 322

Buffer: 154 322 269
Sale 154, No entra nada

Buffer: XX 322 269
Sale 269, No entra nada

Buffer: XX 322 XX
Sale 322, No entra nada

Salida final:
p1: 16, 130, 276, 279, 339, 359
p2: 72, 89, 92, 321, 341
p3: 5, 48, 161, 214, 220
p4: 94, 110, 154, 269, 322

d) Merge polifásico para

p1: 16, 130, 276, 279, 339, 359
p2: 72, 89, 92, 321, 341
p3: 5, 48, 161, 214, 220
p4: 94, 110, 154, 269, 322

Tabla inicial
0 0 1 = 1
1 1 1 = 3
1 2 2 = 5

Con 4 particiones hay un dummy y una distribución (1,2,2).

I1	I2	I3	**O1**
D	p1	p2	**p1,p2**
p3	p4	p1,p2	**p1,p2,p3,p4**

e) Ambos algoritmos buscan el máximo/mínimo N veces y lo congelan, la diferencia principal es que Insertion sort hace N búsquedas secuenciales (de orden N) y Heap sort hace N búsquedas binarias (de orden $\log_2(N)$). El resultado es un orden $O(N^2)$ para el Insertion y un $O(N.\log_2(N))$ para el Heap sort. La condición para que Heap supere en tiempo a Insertion es que N sea lo suficientemente grande como para que la diferencia en el orden se haga significativa ($N^2 >> N.\log_2(N)$) y justifique hacer el procedimiento de heapify.