HEAPS (Montículos)

Montículos binarios

- Arboles binarios casi-completos sesgados a izqda.
  Representación mediante un vector
- Montículos minimales: propiedad de ordenación
  Montículo vacío, inserción, acceso al mínimo, eliminar mínimo.
  Flotar y hundir
- Construcción recursiva (versus basada en insertar)
  Coste: Suma de alturas de un árbol completo.
- Heapsort: (selección de máximo + hundimiento) reiterado.
  Coste: N log N.

MONTÍCULOS SESGADOS (coste amortizado $4 \log N$)

Operaciones: Mezclar, Insertar, Eliminar Min.

Procedimiento de mezcla (demasiado) trivial: Recursión manteniendo la raíz menor y su hijo izqdo fijos.

Montículos sesgados: Se pasa al otro lado el hijo izqdo y se sigue recursivamente como antes con el derecho.

Complejidad amortizada

Nodo **pesado**: $|HD| > |HI|$; **ligero** = no pesado.

En una mezcla sólo puede cambiar el estado de los nodos en la rama más derecha de los dos árboles mezclados.

Las hojas son ligeros. Nodos ligeros en cualquier camino derecho desde un nodo es $\leq \lfloor \log N \rfloor + 1$

Tma 22.1: H_1 y H_2. Nodos en sus caminos derechos l_i, h_i

Función potencial: nº total de nodos pesados

Coste de la mezcla $\leq 2 \log N + (h_1 + h_2)$

Incremento de potencial $\leq 2 \log N - (h_1 + h_2)$.

Dem: $Ctm = l_1 + l_2 + h_1 + h_2$. Acoto l_i y uso el Tma 21.4.

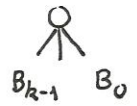
Incp: todos los pesados devienen ligeros. Uso de nuevo la acotación de l_i.

Tma 22.2: Coste amortizado de montículos sesgados

Dem: Cte mezcla + Inc. pot $\leq 4 \log N \Rightarrow \Sigma$ ctes $\leq 4M \log N$.

MONTICULOS BINOMIALES

Arboles binomiales

- $B_0 = 0$; $B_k =$  . $B_k =$ 

Props : $|B_k| = 2^k$; $h(B_k) = k$; $\binom{k}{i}$ nodes at depth i ;
The root is the node with greatest degree (k).

Binomial heaps (forests for the notion of binomial tree)

- Set of binomial trees such that
  1) Its trees are heaps. 2) No two trees have the same size
  Cor: A binomial heap with n nodes has at most $\lfloor \log n \rfloor + 1$ trees.
- Representation of binomial heaps
  - Left-child right-sibling with a pointer $p(x)$ to its parent.
    degree $(x) ::= \{ y \mid p(y) = x \}$.
  - Roots of the binomial tree form an ordered (by size) list.
  - Head (H) is a pointer to that list.
- Operations on binomial heaps
  Make-BH ; Min-BH by checking the list of roots ($O(\log(n))$) ;
  Union-BH by connecting the trees with the same size ($O(\log(n))$)
  When mixing the two lists we could find three trees with the same size ; in such a case we simply skip the first one !
  Insert-BH by means of Union-BH ($O(\log(n))$) ;
  Extract-min-BH : we get a bh after the removal and we unite it to the rest of the original heap ($O(\log(n))$) ;
  Decrease-key-BH : by bubbling up (flotando) the node ($O(\log(n))$) ;
  Delete-key-BH : by (artificially) bubling it up ($-\infty$) and then applying Extract-min-BH !! ($O(\log(n))$).

Note : The key idea is that binomial heaps generalize binomial trees in such a way that they become "square-like" : their width is $O(\log(n))$ exactly as the depth of their (binomial) trees !!

Without deleting we will get $O(1)$ amortized time. This is used at the graph algorithms that have to get minimums many times.
We get their good (amortized) behaviour by relaxing their structural constraints: these will be (sufficiently) recovered (on the fly!) when convenient !!

Structure of Fibonacci heaps

- The trees at a FH are linked in a circular double list with no-order between its elements. These trees are in fact heaps.
- For each node of the ~~Heap~~ we have degree(x) giving us its number of children
- Another field mark(x) indicates weather x has lost some child since it was made the child of another node. This is defined for any node of the heap.
- We externally point out the root containing the minimum of the full structure, by means of min(H). n(H) gives us the total number of nodes in H.

Potential function to compute the amortized cost: $t(H)$ ::= trees in H; $m(H)$ ::= marked nodes in H; $\Phi(H) ::= t(H) + 2m(H)$.
Control of the maximum degree of any node $D(n) = \lfloor \log n \rfloor$, as far as only mergeable-heap operations are supported.

Mergeable-heap operations

- Unordered binomial tree: like binomial trees, but without maintaining the children of each node in any order. $U_0, U_1, \dots, U_{k-1}$
- Fibonacci heap: any collection of ubt's that can be generated using the operations of the type. We immediately get $D(n) = \lfloor \log n \rfloor$.
- Consolidation of the structure is delayed till Extract-Min application.
- Make-FH; Fib-Insert: we simply add a new tree, changing min(H) if needed $O(1)$; Min-FH, trivial $O(1)$, t, and then Φ, increased by 1; Union-FH, mainly concatenation, $O(1)$, $\Phi(H_1 \cup H_2) = \Phi(H_1) + \Phi(H_2)$;
  Extract-Min-FH: i) We remove the corresponding tree, leaving instead its children; ii) Consolidate recovers the structural invariant: no trees in the fh have the same degree. We mix two such trees into a single one preserving the heap property and unmark the root of the new child.
  Consolidate uses an array of size $D(H.n)$; we cross the fh and we mark the entry of the array which corresponds to the degree of the tree. When we find a tree whose entry at the array is marked, then we mix these two trees. Cost $O(D(n) + t(H))$
  Remark: Since $D(n)$ will be $\lfloor \log n \rfloor$ it is impossible to generate a tree with size greater or equal than it.

Decreasing a key in a FH

- Cut : removes the link between x and its parent y and introduces it as a new heap of the fh.
- Cascading - Cut : if this is the second child that y has lost (y.mark = T) then we also separate it (by Cut) from its parent, and iterate the procedure.
- We change H.min if the new key is smaller than it.
- Amortized cost : $O(1)$!!

  Actual cost : $O(c)$, where c is the number of cuts that were done.

  Change of potential : $t(H') = t(H) + c$; $m(H') \leq m(H) - (c-1) + 1$

  $$\Phi(H') - \Phi(H) = 4 - c$$

Remark : We bonus any unmarking by 2 : one pays for the cut and the other for the increasing of size of $t(H)$.

Deleting a node

- We simply change its key into $-\infty$ and then remove the min.
- Amortized time : $O(D(n))$.

Bounding the maximum degree (when deletions are allowed)

- We prove that $D(n) \leq \lfloor \log_\phi n \rfloor$, where ϕ is the golden ratio.
- For each node n we define size(n) ::= nodes in the tree rooted at n.

  We prove that size(n) remains expanential in n.degree

Lemma 1 : Let x.degree = k, $y_1, \dots y_k$ the children of x in the order they were added. Then y_i.degree $\geq i-2$ $\forall i \in 3..k$.

We only add a new child by applying Consolidate, so that y_i.degree = x.degree, when it was added. From them it can only lose a child, since otherwise it is Cut.

Fibonacci numbers : $F_{k+2} = 1 + \sum_0^k F_i$; $F_{k+2} \geq \phi^k$

Lemma 2 : Let x.degree = k, then size(x) $\geq F_{k+2}$.

- S_k is minimum possible size of a node x.degree
- $S_k \geq 2 + \sum_2^k S_{i-2} \Rightarrow S_k \geq F_{k+2} \geq \phi^k$.

Cor : The maximum degree D(n) of any node in a fh is $O(\log n)$

Remark : In Consolidate we should use the precise value above.