Фиббоначиевы кучи (Fredman, Tarjan -1987)

Фиббоначиева куча - это лес "фиббоначиевых"деревьев. *Фиббоначиево дерево* - это дерево, возникающее в процессе выполнения операций над кучей. Если нет удалений, то оно является неупорядоченным биномиальным деревом. После выполнения удалений (вырезаний) теряет свойства биномиального дерева, но при этом вершина степени k всегда является корнем поддерева, включающего 2^{ck} вершин.

Представление Фиббоначиевой кучи.
Каждая вершина x — запись со следующими полями:
$key[x]$ — ключ вершины,
$p[x]$ — ссылка на отца вершины,
$child[x]$ — ссылка на одного из сыновей вершины,
$right[x]$, $left[x]$ — ссылки на левого и правого соседей (братьев) вершины, все вершины, имеющие общего отца связаны в двусторонний циклический список,
$degree[x]$ — степень вершины (число детей),
$mark[x]$ — булева метка (= true, если x потеряла сына после того, как она в последний раз стала чьим-то ребенком).

Поля $right[x]$ и $left[x]$ связывают корни деревьев, входящих в кучу, в двусвязный циклический список. Доступ к куче H осуществляется через указатель $min[H]$ на корень дерева, содержащий минимальный ключ (эта вершина называется *минимальной*). Атрибут $n[H]$ содержит число вершин в куче H.

Через $D(n)$ обозначим максимальную степень вершины в куче с n вершинами (она ограничена $O(\log_2 n)$). При оценке сложности будут использоваться еще два параметра кучи: $t(H)$ – число деревьев в куче и $m(H)$ – число отмеченных вершин.

Метод потенциала

Пусть требуется оценить сложность $T(\sigma)$ выполнения последовательности операций $\sigma = op_1, op_2, \ldots, op_n$ над некоторой структурой данных D. Обозначим через c_i сложность (стоимость) выполнения i-ой операции. Тогда $T(\sigma) = \sum_{i=1}^{n} c_i$. Пусть $D_0 = D$ и D_i $(1 \le i \le n)$ – состояние структуры данных после выполнения i-ой операции op_i. *Потенциал* – это некоторая функция $\Phi(D)$, определенная на рассматриваемых структурах данных. *Учетная стоимость* $\hat{c}_i$ выполнения i-ой операции определяется формулой $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Тогда для суммарной учетной стоимости всех операций из σ имеем

$$\hat{T}(\sigma) = \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

Отсюда при $\Phi(D_n) \ge \Phi(D_0)$ получаем, что $T(\sigma) \le \hat{T}(\sigma)$, т.е. учетная стоимость операций σ является верхней оценкой их настоящей стоимости.

Мы будем использовать метод потенциала для оценки сложности операций над фиббоначиевыми кучами. *Потенциал кучи* H зададим выражением $\Phi(H) = c(t(H) + 2m(H))$, где c – константа, которая будет выбрана ниже.

Операции над фиббоначиевыми кучами

Создание новой кучи: **Make-Fib-Heap**: $min[H] := NIL; n[H] := 0;$
У такой пустой кучи $t(H) = m(H) = \Phi(H) = 0$.

Добавление вершины: Пусть в кучу нужно добавить вершину x с заданным полем $key[x]$. Алгоритм состоит в добавлении в кучу дерева степени 0, состоящего из этой вершины.

Fib-Heap-Insert(H, x):

1. $degree[x] := 0$;
2. $p[x] := NIL$;
3. $child[x] := NIL$; $left[x] := x$; $right[x] := x$;
4. $mark[x] := False$;
5. Вставить полученный список в список корней H;
6. **if** $min[H] = NIL$ OR $key[x] < key[min[H]]$ **then** $min[H] := x$;
7. $n[H] := n[H] + 1$.

Сложность – $O(1)$.

Поиск минимальной вершины: $min[H]$. Сложность – $O(1)$.

Соединение двух куч. Алгоритм состоит в объединении корневых списков куч.

Fib-Heap-Union(H_1, H_2):

1. H :=*Make-Fib-Heap()*;
2. $min[H] := min[H_1]$;

3. Соединить корневой список H_2 с корневым списком H;
4. **if** $(min[H] = NIL)$ OR $(key[min[H_2]] < key[min[H]]$ **then** $min[H] := min[H_2]$;
5. $n[H] := n[H_1] + n[H_2]$;
6. Освободить память, занятую заголовками H_1 и H_2.

Сложность – $O(1)$.

Удаление минимальной вершины. Алгоритм работает в два этапа: на первом удаляется корень с минимальным ключом и его сыновья добавляются в корневой список, на втором этапе процедура Consolidate объединяет деревья одинаковых степеней, оставляя не более одного дерева каждой степени.

Fib-Extract-Min(H):

1. $z := min[H]$;
2. **if** $(z \neq NIL)$
3. **then for-each** сына x вершины z;
4. **do** { добавить x в корневой список H]; $p[x] := NIL$};
5. удалить z из корневого списка H;
6. **if** $z = right[z]$ **then** $min[H] = NIL$ /* куча пуста
7. **else** { $min[H] := right[z]$;
8. Consolidate(H)};
9. $n[H] := n[H] - 1$.

Процедура уплотнения кучи Consolidate(H). Использует вспомогательный массив $A[0..D(n[H])]$. Вначале он пуст. В процессе работы $A[i]$ указывает на корень дерева степени i. Если обнаруживаются 2 дерева степени i , то они сливаются в одно дерево степени $i + 1$. При этом корень y с большим ключом становится сыном корня x с меньшим ключом, степень x увеличивается, а y теряет метку, если ее имела.

Consolidate(H)

1. **for** $i = 0$ **to** $D(n[H])$ **do** $A[i] := NIL$;
2. **for-each** вершины w из корневого списка H
3. **do** { $x := w$; $d := degree[x]$;
4. **while** $A[d] \neq NIL$
5. **do** { $y := A[d]$;
6. **if** $key[x] > key[y]$ **then** поменять x и y местами;
7. Fib-Heap-Link(H, y, x);
8. $A[d] := NIL$; $d := d + 1$};
9. $A[d] := x$};
10. $min[H] := NIL$;
11. **for** $i = 0$ **to** $D(n[H])$ **do** /* вставка полученных деревьев в кучу
12. **if** $A[i] \neq NIL$
13. **then** { вставить $A[i]$ в корневой список H;
14. **if** $(min[H] = NIL)$ OR $(key[A[i]] < key[min[H]])$ **then** $min[H] := A[i]$}.

Процедура **Fib-Heap-Link**(H, y, x) объединяет деревья с корнями y и x:
1. Удалить y из корневого списка H;
2. Сделать y сыном x, увеличив $degree[x]$;
3. $mark[y] := False$.

Сложность процедуры Consolidate можно оценить как $O(D(n[H])) + c_1 *$*(число вызовов Fib-Heap-Link)* (почему ?). При каждом вызове Fib-Heap-Link потенциал $c * (t(H) + 2m(H))$ уменьшается по крайней мере на c. Выберем c так, чтобы учетная стоимость Fib-Heap-Link стала отрицательной. Тогда учетная стоимость Consolidate и Fib-Extract-Min оценивается как $O(D(n[H]))$.

Уменьшение ключа вершины x до k.

Fib-Heap-Decrease-Key(H, x, k) :

1. **if** $k > key[x]$ **then** return ERROR; /* новый ключ > старого
2. $key[x] := k$;
3. $y := p[x]$;
4. **if** $(y \neq NIL)$ AND $(key[x] < key[y])$
5. **then** { Cut(H, x, y);
6. Cascading-Cut(H,y) };
7. **if** $key[x] < key[min[H]]$ **then** $min[H] := x$.

Перенос x в корневой список H.
Cut(H, x, y) :

1. Удалить x из списка детей y, уменьшив $degree[y]$ на 1.
2. Добавить x в корневой список H.
3. $p[x] := NIL$;
4. $mark[x] := False$.

Перенос отмеченной вершины в корневой список H.
Cascading-Cut(H, y) :

1. $z := p[y]$;
2. **if** $z \neq NIL$
3. **then if** $mark[y] = False$;
4. **then** $mark[y] := True$; /* неотмеченная вершина y, потерявшая сына, метится
5. **else** { Cut(H, y, z); /* отмеченная вершина y, потерявшая сына,
6. Cascading-Cut(H,z) }. /* переносится в корневой список H.

Сложность процедуры Fib-Heap-Decrease-Key можно оценить как $O(r)$, где r – это длина цепочки рекурсивных вызовов пары процедур Cut, Cascading-Cut. При каждом таком вызове одна вершина добавляется в корневой список и исчезает одна метка. Таким образом, потенциал уменьшается на c. Выбрав c достаточно большим, получим учетную стоимость Fib-Heap-Decrease-Key $O(1)$.

Удаление вершины из H : ключ вершины уменьшается до $-\infty$ (достаточно уменьшить до $key[min[H]] - 1$), а затем удаляется минимальная вершина.
Fib-Heap-Delete(H, x) :

1. Fib-Heap-Decrease-Key$(H, x, -\infty)$
2. Fib-Heap-Extract-Min(H).

Учетная стоимость Fib-Heap-Delete – $O(D(n[H]))$.
Оценка $D(n[H])$
Обозначим для вершины x через T_x поддерево с корнем x, а через $size(x)$ – число вершин этого дерева.

Лемма 1. *Пусть $degree[x] = k$. Тогда детей x можно упорядочить так, что их степени будут не меньше $0, 0, 1, 2, 3, \ldots, k-2$.*

Числа Фиббоначчи F_k определяются следующими реккурентными соотношениями:
$F_0 = 0,\ F_1 = 1,\ F_k = F_{k-1} + F_{k-2}\ (k > 1)$.

Лемма 2. *Для любого $k \geq 0$*

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_k.$$

Пусть $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$ — "золотое сечение".

Задача 1. *Докажите, что для любого $k \geq 0 \quad F_{k+2} \geq \varphi^k$.*

Лемма 3. *Пусть x — вершина фиббоначчиевой кучи со степенью $degree[x] \geq k$. Тогда $size(x) \geq F_{k+2} \geq \varphi^k \geq 1.6^k$.*

Следствие. *Максимальная степень $D(n)$ вершин в фиббоначиевой куче с n элементами ограничена $O(\log n)$.*

Задача 2. *Объясните, каким образом помеченная вершина может появиться в корневом списке.*

Задача 3. *Предложите эффективную реализацию операции изменения ключа у вершины Fib-Heap-Change-Key(H, x, k), в результате которой $key[x]$ получит значение k. Оцените учетную стоимость этой операции при $k < key[x], k = key[x], k > key[x]$.*