Биноминальная куча.

Евгений Пантелеев

Новосибирск
2021г.

Биномиальное дерево

Биномиальное дерево *(binomial tree)* B_k представляет собой рекурсивно определенное упорядоченное дерево.

1. Биномиальное дерево B_0 состоит из одного узла.
2. Биномиальное дерево B_k состоит из двух биномиальных деревьев B_{k-1}, связанных вместе: корень одного из них является крайним левым дочерним узлом корня второго дерева.

Биномиальное дерево

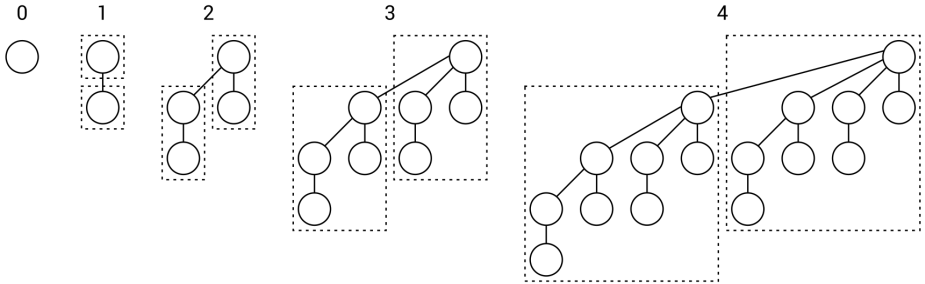


Figure: Биномиальные деревья

Свойства биномиальных деревьев

1. имеет $\mathbf{2^k}$ узлов;
2. имеет высоту $\mathbf{k}$;
3. имеет ровно $\binom{\mathbf{k}}{\mathbf{i}}$ узлов на глубине $i = 0, 1, \ldots, k$;

Биномиальная пирамида

Биномиальная пирамида (binomial heap) представляет собой множество биномиальных деревьев, которые удовлетворяют следующим свойствам:

1. Каждое биномиальное дерево в **H** подчиняется **свойству неубывающей пирамиды**: ключ узла не меньше ключа его родительского узла. Мы говорим, что такие деревья являются **упорядоченными в соответствии со свойством неубывающей пирамиды**.
2. Для любого неотрицательного целого **k** имеется не более одного биномиального дерева **H**, чей корень имеет степень **k**.

Биномиальная пирамида: рисунок

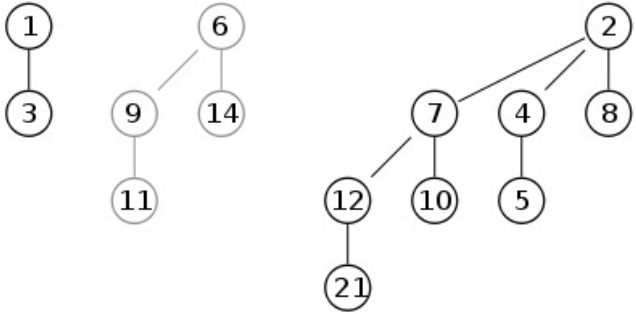


Figure: Биномиальная пирамида

- Минимальные элементы являются корнями деревьев.

Время выполнения операций

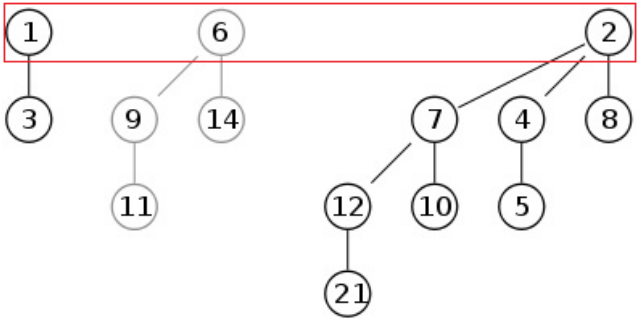
Процедура	**Бинарная пирамида** (наихудший случай)	**Биномиальная пирамида** (наихудший случай)
make	$\Theta(1)$	$\Theta(1)$
insert	$\Theta(\lg n)$	$O(\lg n)$
minimum	$\Theta(1)$	$O(\lg n)$
extract_min	$\Theta(\lg n)$	$\Theta(\lg n)$
union	$\Theta(n)$	$\Omega(\lg n)$
decrease_key	$\Theta(\lg n)$	$\Theta(\lg n)$
delete	$\Theta(\lg n)$	$\Theta(\lg n)$

Создание новой биномиальной пирамиды

- Для создания биномиальной пирамиды процедура **make** просто выделяет память и возвращает пустой объект **H**.
- Время работы этой процедуры составляет $\Theta(1)$.

Поиск минимального ключа

- Минимальный ключ должен находиться в корне одного из деревьев.
- Для поиска минимума надо обойти корни всех деревьев. Их число не превышает $\lg n + 1$.
- Поскольку надо проверитьне более $\lg n + 1$ корней, время работы процедуры **minimum** составляет $O(\lg n)$.

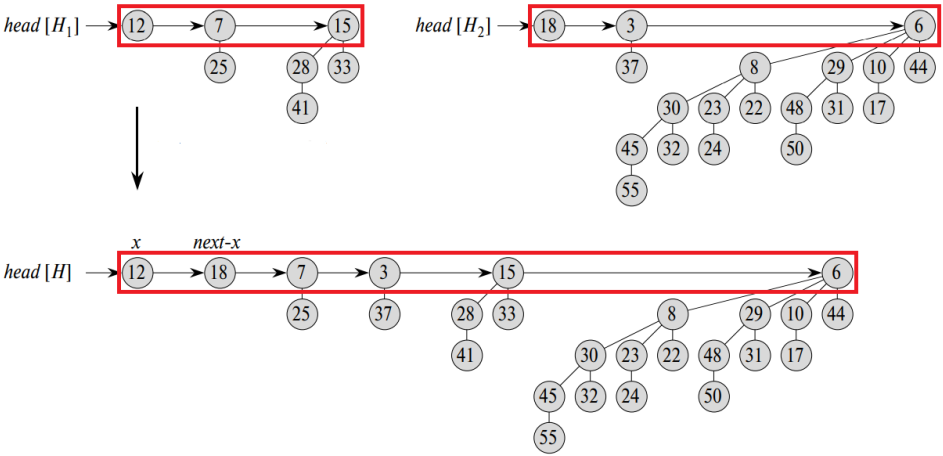


Слияние двух биномиальных пирамид

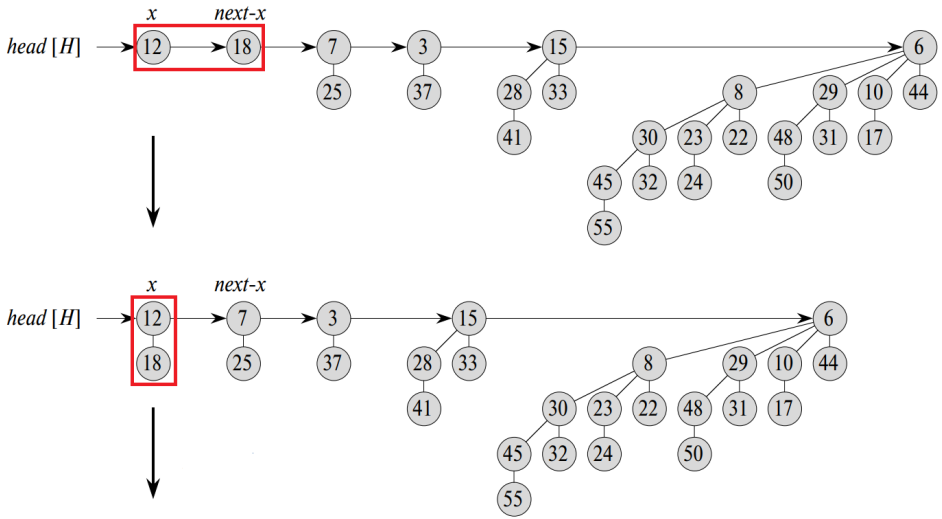
Процедура **union** имеет две фазы:

- объединяем списки корней биномиальных пирамид H_1 и H_2 в единый связанный список H, который отсортирован по степеням корней в монотонно возрастающем порядке.
- постепенно объединяем мелкие биномиальные деревья в более крупные. В получившемся списке могут встречаться пары соседних вершин одинаковой степени. Поэтому мы начинаем соединять деревья равной степени и делаем это до тех пор, пока деревьев одинаковой степени не останется. Операция выполняется за $\Omega(\lg n)$.

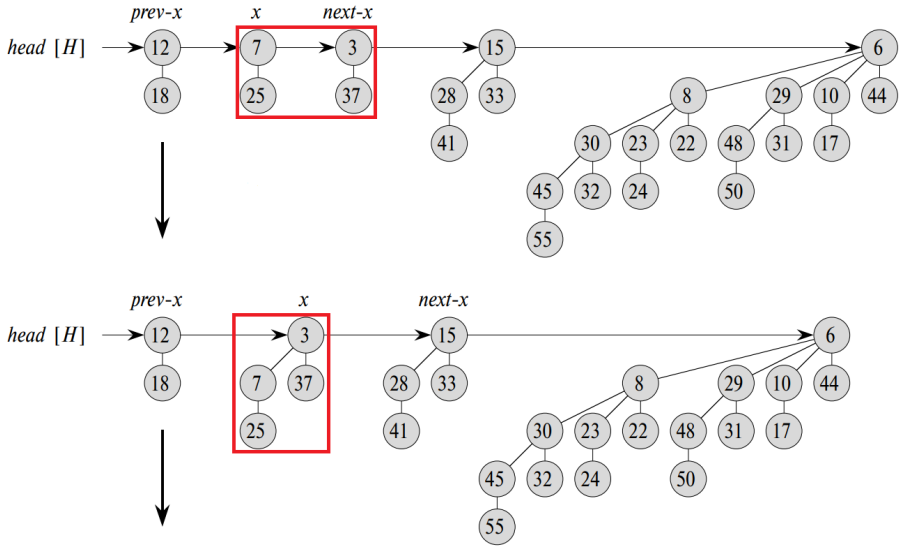
Слияние двух биномиальных пирамид



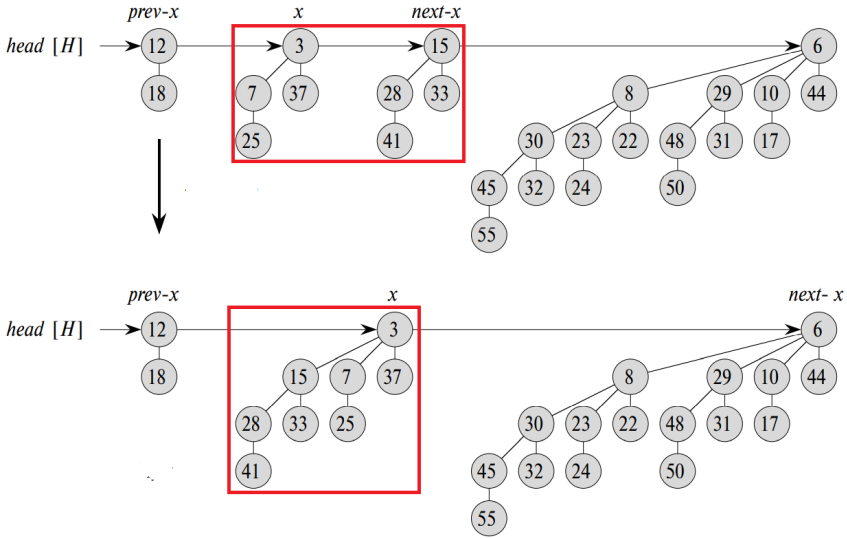
Слияние двух биномиальных пирамид



Слияние двух биномиальных пирамид



Слияние двух биномиальных пирамид



Вставка узла

Процедура **insert**:

- создает биномиальную пирамиду **H'** с одним элементом за время $O(1)$.
- объединяет ее с биномиальной пирамидой **H**, содержащей n узлов, за время $O(\lg n)$.

Вызов **union** должен освободить память, выделенную для временной биномиальной пирамиды **H**.

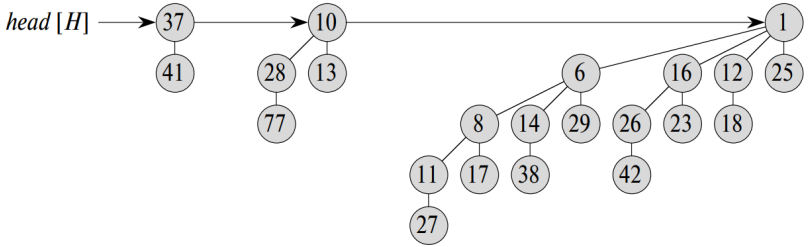
Извлечение вершины с минимальным ключом

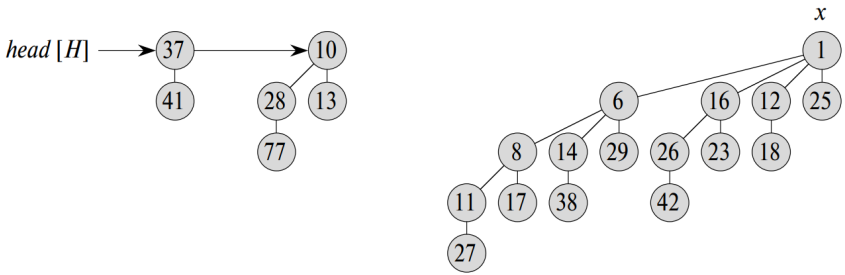
Процедура **extract_min**:

1. находим элемент при помощи **minimum**.
2. удаляем его из корневого списка. Из перевернутого списка его детей делаем корневой список для новой кучи H_1.
3. объединяем исходную кучу с H_1.

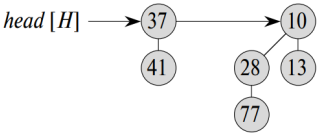
Сложность $\Theta(\lg n)$.

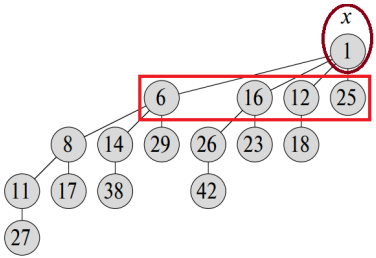
Извлечение вершины с минимальным ключом

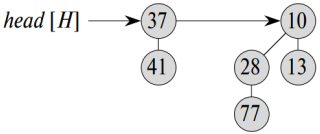


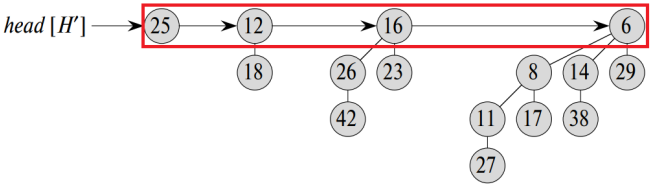


Извлечение вершины с минимальным ключом

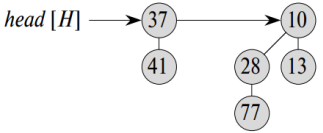


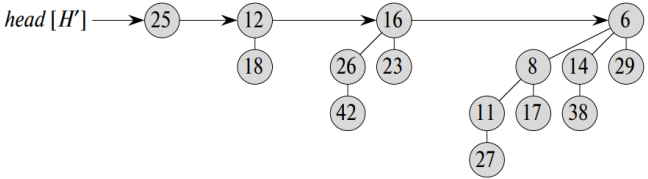


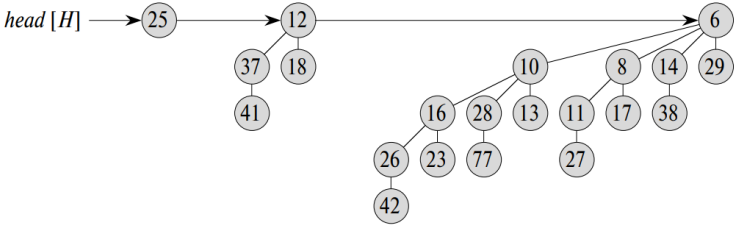




Извлечение вершины с минимальным ключом





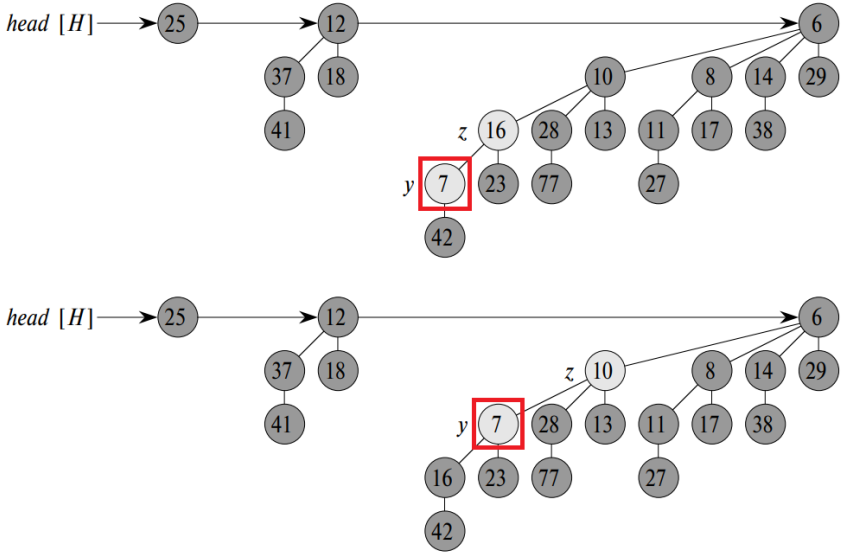


Уменьшение ключа

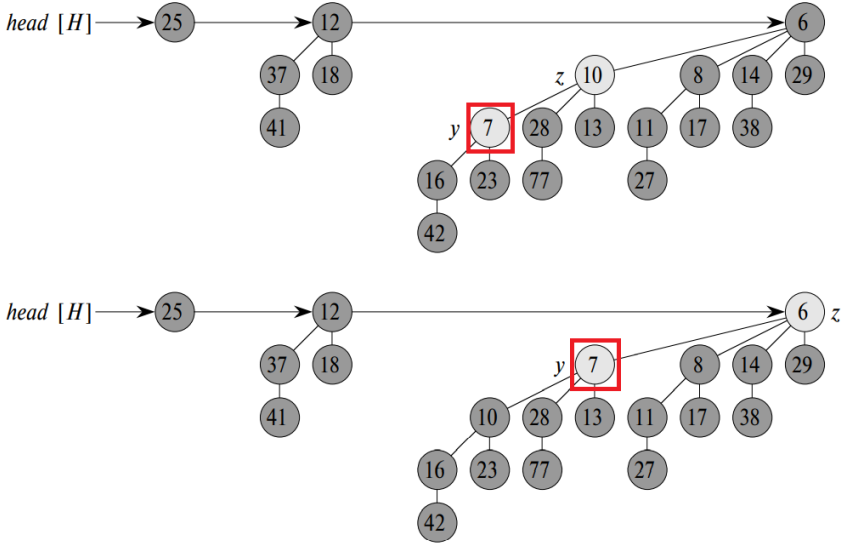
Процедура **decrease_key**:

- уменьшает значение ключа узла **x** в биномиальной пирамиде **H**, присваивая ему новое значение **k**.
- В случае, если **k** превышает текущий ключ **x**, процедура сообщает об ошибке.

Уменьшение ключа



Уменьшение ключа



Удаление ключа

Операция **delete** сводится к **decrease_key** и **extract_min**:

- сначала нужно уменьшить ключ до минимально возможного значения, а затем извлечь вершину с минимальным ключом.
- узел всплывает вверх, откуда и удаляется **extract_min**.

Процедура выполняется за время $\Theta(\lg n)$.

Использование

1. Применяется для реализации **очереди с приоритетом**.
2. Так же подходит для создания lock-free concurent queue:
   Gavin Lowe. ”Lock-Free Concurrent Binomial Heaps”, 2018

Comparison concurrent priority queue implementations

Operation probabilities			Initial	This paper	Sundell &	Lindén &
insert	deleteMin	minimum	size		Tsigas	Jonsson
0.5	0.5	0.0	10K	4012K±47K	2904K±40K	1386K±137K
0.5	0.5	0.0	100K	3270K±29K	2912K±37K	1797K±78K
0.5	0.5	0.0	1000K	1574K±42K	2499K±42K	1854K±195K
0.4	0.4	0.2	100K	3531K±46K	3581K±33K	2859K±234K
0.3	0.3	0.4	100K	3827K±43K	4475K±232K	4233K±624K
0.6	0.4	0.0	0	3525K±67K	2144K±277K	1953K±70K
0.7	0.3	0.0	0	2289K±97K	1569K±140K	1690K±67K

Figure : Experimental results, giving throughput (in operations per second).

- 64 threads, 2.1GHz Intel(R) Xeon(R) E5-2683 CPUs with 256GB of RAM
- two million randomly chosen operations
- 10 executions and 95% condence intervals.