Слабая пирамида

В.К. Гулаков, К.В. Гулаков

Статья посвящена одной из разновидностей пирамидальных структур данных, используемых при решении различных задач, слабой пирамиде. Рассмотрены некоторые задачи, где слабые пирамиды более эффективны. Краткое сравнение её и её разновидностей с наиболее популярными пирамидальными структурами показывает возможность выполнения основных операций над данными с минимальной сложностью. В статье даётся подробное описание структуры слабой пирамиды в виде массива и в виде связной структуры. Предпочтение отдаётся в силу наглядности связной структуре. Приводятся примеры выполнения различных операций над рассматриваемой структурой с помощью рисунков, и даётся оценка сложности их выполнения. Для увеличения эффективности операций вставки рассматривается подход с использованием буферов. Рассматривается связь слабой пирамиды с биноминальной пирамидой. Здесь используется понятие идеальной слабой пирамиды. В статье намечены пути экспериментального исследования слабой пирамиды в различных ситуациях и практических задачах.

Ключевые слова: **пирамидальные структуры данных, слабая пирамида, описание структуры, основные операции, ограничения, сложность алгоритма.**

I. Введение

Существуют некоторые тенденции в структурах данных, использование которых позволяет повысить эффективность операций над ними. К таким тенденциям можно отнести упрощение требований к идеальным структурам. Например, для бинарной пирамиды ослабление требования упорядоченности приводит к полуупорядоченной структуре, называемой слабой пирамидой.

Структура слабой пирамиды впервые была предложена в техническом докладе Петерсона [11]. Даттон [2] реализовал данную структуру и показал её практическую значимость при использовании для сортировки.

Слабая пирамидальная сортировка использует меньше сравнений, чем большинство других алгоритмов, близких к теоретическому нижнему пределу, поэтому особенно полезна, когда сравнение является дорогостоящим, например, при сравнении строк с использованием алгоритма сортировки на базе кодировки Unicode.

В дополнение к сортировке слабые пирамиды могут использоваться в следующих приложениях:

1. Построение слабой пирамиды и поиск наименьшего элемента у левого потомка правого поддерева корня, находится за оптимальное время как наименьший, так и второй наименьший элемент, используя $n+\lceil \log n \rceil-2$ сравнений элементов.

2. Построив слабую пирамиду и найдя максимальный элемент среди листьев (а если n нечетно, узел, который имеет одного потомка), мы находим за оптимальное время как самый маленький, так и самый большой элементы, используя $n+\lceil n/2 \rceil -2$ сравнений элементов.

Таблица 1. Сравнение различных пирамид (длина кода для различных пирамид, из библиотеки CPH STL [4] представлена приблизительно).

пирамиды	Длина кода (LOC)	Создание	Вставка	Удаление минимума
Бинарная пирамида (Binary heap) [16]	205	2n	$\lceil \lg n \rceil$	$2\lceil \lg n \rceil$
Бинарная пирамида на базе массива (array-based binary heap) [16]	205	2n	$\lceil \lg n \rceil$	$2\lceil \lg n \rceil$
Слабая пирамида weak heap [3]	565	n-1	$\lceil \lg n \rceil$	$\lceil \lg n \rceil$
Слабая пирамида на базе массива (array-based weak heap) [3]	214	n-1	$\lceil \lg n \rceil$	$\lceil \lg n \rceil$
Биномиальная пирамида (Binominal heap) [10]	317	n-1	2	2lgn+O(1)
Потоково - релаксационная слабая пирамида (Run-relaxed weak heap) [4]	1 021	n-1	3	2lgn+O(1)
Рангово - релаксационная слабая пирамида (Rank-Relaxed Weak heap) [7]	883	n-1	1	$\lceil \lg n \rceil$
Пирамида Фибоначчи (Fibonacci heap) [10]	296	n-1	1	$\lceil \lg n \rceil$
Парная пирамида (pairing heap) [11]	204	n-1	1	$\lceil \lg n \rceil$
2-3 пирамида (2-3 heap) [15]	546	n-1	1	$\lceil \lg n \rceil$
Пирамида с нарушениями (violation heap) [9]	459	n-1	1	$\lceil \lg n \rceil$
Дрожащая пирамида (Quake Heaps) [1]	383	n-1	1	$\lceil \lg n \rceil$
Однобуферная слабая пирамида (One-buffer weak heap) [4]		n-1	O(lgn)	lgn+1
Двубуферная слабая пирамида (Two-buffer weak heap) [6]		n-1	O(1)	lgn+O(1)

Таблица 2. Представление слабой пирамиды в форме двух массивов

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a_i	1	4	6	2	7	5	3	8	15	11	10	13	14	9	12
r_i	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1

3. Используя слабую пирамиду, мы можем немного улучшить ограничения многопутевого слияния n элементов до ограничения $n\lceil \log k \rceil + k - 1$, которое достигается при использовании турнирных деревьев. Это можно сделать за не более $n\lceil \log k \rceil + 0{,}086k - 1$ сравнений элементов [4].

В работах [3,5,6] слабая пирамида получила развитие. Известны несколько разновидностей слабых пирамид:

1) Слабая пирамида, которая является бинарным деревом, полностью полуупорядоченной пирамидой;

2) Слабая очередь, которая является лесом идеальных слабых пирамид;

3) Релаксационная слабая очередь, которая расширяет слабые очереди, позволяющая некоторым узлам упорядочивать полупирамиды с нарушением основного свойства пирамиды. Имеются две разновидности:

-потоково-релаксационная слабая очередь (Run-Relaxed Weak Queue)

-рангово-релаксационная слабая очередь (Rank-Relaxed Weak Queue)

4) Пирамиды, использующие буфер:

- однобуферная слабая пирамида (One-buffer weak heap);

- двубуферная слабая пирамида (Two-buffer weak heap).

Сравнение слабой пирамиды и её разновидностей с наиболее популярными пирамидами, приведенное в таблице 1, позволяет сказать, что теоретические оценки основных операций над слабыми пирамидами, эффективнее, чем над бинарными пирамидами. Дальнейшее развитие структур на базе слабой пирамиды позволяет выполнять некоторые операции не хуже, чем наиболее продвинутые разновидности биномиальных, фибоначчиевых, парных и других пирамид. Например, рангово-релаксационные слабые пирамиды выполняют меньшее число сравнений, чем фибоначчиевы пирамиды. Но они сложнее (имеют большую длину кода).

Экспериментальные результаты указывают что слабые пирамиды достаточно эффективны на практике. Они конкурируют с другими структурами приоритетных очередей при рассмотрении количества выполненных сравнений элементов, и теряют с небольшим отрывом при рассмотрении фактического времени работы.

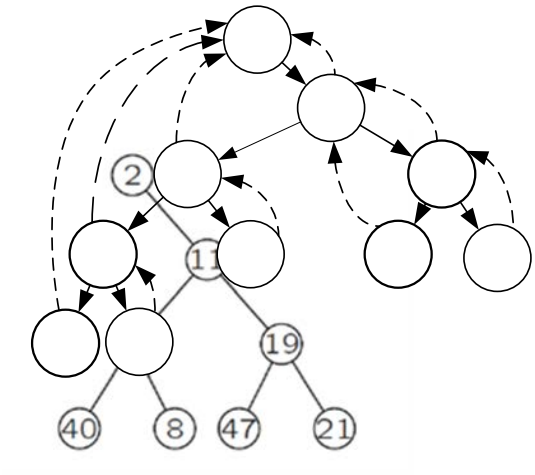
II. Структура слабой пирамиды и основные операции над ней

Слабая пирамида является бинарным деревом и получается уменьшением требований к бинарной пирамиде. Ей достаточно следующих трех условий:

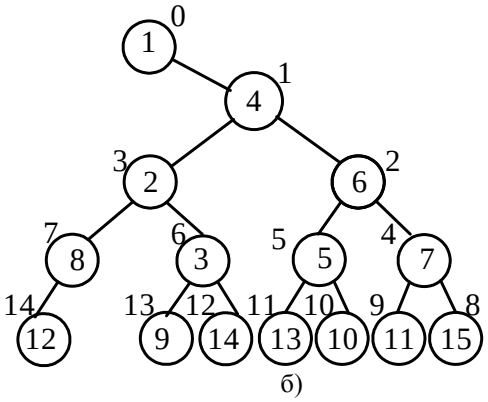
1. Корневое значение любого поддерева меньше или равно всем элементам в его правой ветви (основное свойство пирамиды);
2. У корня всей структуры нет левого потомка (оптимальное корневое свойство). Иногда такое дерево называют полуупорядоченным деревом (см., например [13]);
3. Листовые узлы находятся на последних двух уровнях (свойство баланса). Однако, последний уровень не обязательно заполняется слева направо.

Каждый узел в слабой пирамиде можно считать корнем меньшей слабой пирамиды, игнорируя её левого потомка. Узлы, не имеющие левого потомка, автоматически означают слабую пирамиду (рис. 1).

В ряде ситуаци



а)



б)

Рис. 2. Примеры слабых пирамид**.**

й используется понятие идеальной слабой пирамиды (*perfect weak heap*), у которой правое поддерево корня есть полное бинарное дерево (рис. 2а).

Слабая пирамида, как и бинарное дерево, может быть представлена в явной (в виде дерева – рис. 1) или в неявной (в виде массива – таблица 2) форме.

У слабой пирамиды в неявной форме есть дополнительный массив, который использует так называемые обратные биты r_i, i ∈ (0 до n-1). Индекс левого потомка располагается в $2i+r_i$, индекс правого потомка находится в $2i+1-r_i$. С этой целью r_i интерпретируется как целое число в {0,1}, инициализируемое со значением 0. Используя тот факт, что индексы двух потомков a_i обмениваются при переворачивании ri, поддеревья могут быть заменены установкой $r_i \leftarrow 1-r_i$, существенное свойство, чтобы реализация слияния двух слабых пирамид происходила за постоянное время.

Эквивалентное бинарное дерево показано на рис. 2б.

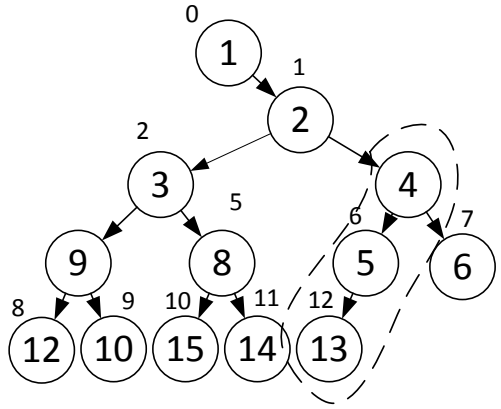


Рис. 4. Индексы массива узлов записываются в верхнем левом углу узла. Узел с ключом 2 является GParent узлов с ключами 4, 5, 13.

Пирами да должна быть полност ью динами ческой, оба эти массива должны быть изменяе мыми.

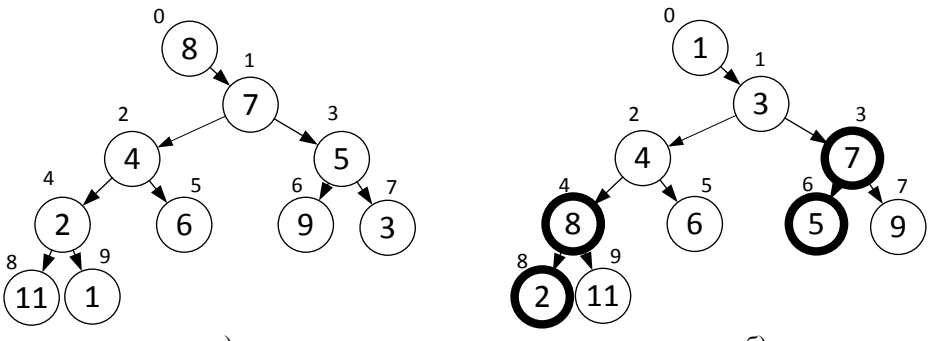
В ысота слабой пирами ды, содержащей n элементов, равна $\lceil \log n \rceil + 1$.

Для выполнения операций над слабыми пирамидами требуются две процедуры **GParent (i) и Swap (i; j).** Расмотрим их подробней. Определим GParent(i) как Gparent(i)=gparent(parent(i)) в случае, если i является левым потомком, и Gparent(i)=parent(i) если i является правым потомком. В слабой пирамиде, Gparent(i) относится к индексу самого глубокого элемента, который, как известно, меньше или равен элементу в i. Рис. 3 представляет визуальную связь с GParent. На рис. 4 приведен пример.

Все узлы должны быть больше их GParent. Мы также определяем GChildren(i), который возвращает все элементы, у которых предок GParent(i).

Исходная реализация слабой пирамиды использует представление массива. Поскольку слабая пирамида построена вначале, мы можем узнать есть ли узел в левом или правом поддереве простой проверкой, если индекс узла четный или нечетный. Возвращаем родительский узел, назначая $i = \lfloor i / 2 \rfloor$. Это полезно с точки зрения временной сложности, потому что мы можем просто использовать правило сдвига бита до тех пор, пока самый правый бит не будет равен 1. Следовательно, сложность вычисления GParent может быть O(1) на современных процессорах и не требует каких-либо сравнений узлов. Определим также путь минимальных элементов. Это путь, состоящий из элементов GChildren(root).

Определим вспомогательную функцию, называемую Swap(i,j). Эта функция меняет местами элементы a_i и a_j, если $a_i > a_j$, меняет левое / правое поддерево i и изменяется r_i. Если $a_i < a_j$, это функция



а) б)

Рис. 5. Иллюстрация работы процедуры SWAP: а) исходное множество; б) результирующая слабая пирамида

ничего не делает. Поэтому порядок элементов i и j важен!

На рис. 5а представлено исходное множество ключей для наглядности в виде дерева и результирующая слабая пирамида (рис. 5б). Для её получения первым шагом восстанавливаем основное свойство слабой пирамиды, т.е. меняем местами узлы 4 и 9 и узлы 3 и 7, а затем меняем у узлов 4 и 3 правые и левые поддеревья с помощью процедуры Swap. Следующим шагом вычисляем GParent для всех узлов с проверкой условия GParent(i)<a_i. В нашем примере узел 4 меняется с узлом 0, узел 3 меняется с узлом 1 путём всплытия.

III. Операция поиска минимального элемента в слабой пирамиде

Операция FIND-MIN просто возвращает элемент корня слабой пирамиды. Эта операция требует 0 сравнений.

IV. Операция слияния двух слабых пирамид.

Фундаментальная операция на слабых пирамидах слияние двух пирамид одинаковой высоты h (или высоты отличающейся на единицу), чтобы сделать слабую пирамиду высотой h + 1. Для этого требуется ровно одно сравнение между корнями. Какой корень меньше (при условии минимальной пирамиды) тот и является окончательным корнем. Его правый потомок является корнем второго дерева, который сохраняет своих потомков (правое поддерево). Левое поддерево состоит из потомков корня первого поддерева (с меньшим корнем). На рис.6 представлен пример слияния двух слабых пирамид.

V. Операция вставки элемента в слабую пирамиду.

Операция INSERT добавляет элемент i в слабую пирамиду как новый лист. Если этот лист является единственным потомком его родителя, мы делаем его левым потомком, обновляя обратный бит у родителя, (это экономит одно ненужное сравнение элемента) и рекурсивно вызываем Swap (GParent (i); i) до тех пор, пока GParent < a_i. Чтобы восстановить упорядочение слабой пирамиды, мы вызываем процедуру всплытия sift-up, начиная с положения a_i.

Процедура sift-up для слабой пирамиды предназначена для восстановления основного свойства в правой ветви пирамиды путём обмена вставляемого элемента с его родителем в случае необходимости.

Временная сложность зависит от реализации и единицы измерения. Если используется реализация массива и подсчитывается количество сравнений элементов, то требуется от 1 (новый элемент - самый левый узел в слабой пирамиде) до максимум logn -1 сравнений (новый элемент является самым маленьким, вставленным в самый правый узел слабой пирамиды).

На рис.7 представлена слабая пирамида после операции вставки ключа «2».

Для вставки заданного ключа k мы начинаем с последнего неиспользованного индекса x в массиве A и помещаем k в A[x]. Затем мы поднимаемся вверх по Gparent до тех пор, пока свойство слабой пирамиды не будет удовлетворено. В среднем, длина пути Gparent от листового узла до корня составляет примерно половину глубины дерева.

VI. Операция построения слабой пирамиды.

Parent(a_i), i ≠ 0, является родителем элемента a_i, если a_i является правым потомком и GParent(a_i), если a_i - левый потомок.

Мы используем процедуру GParent(i) для обозначения индекса такого предка. В упорядоченной слабой пирамиде ни один элемент не будет меньше элемента у его GParent. Следовательно, для построения пирамиды требуется O(n) времени в худшем случае, более того, n- 1 сравнений элементов и не более n - 1 элементарных обменов.

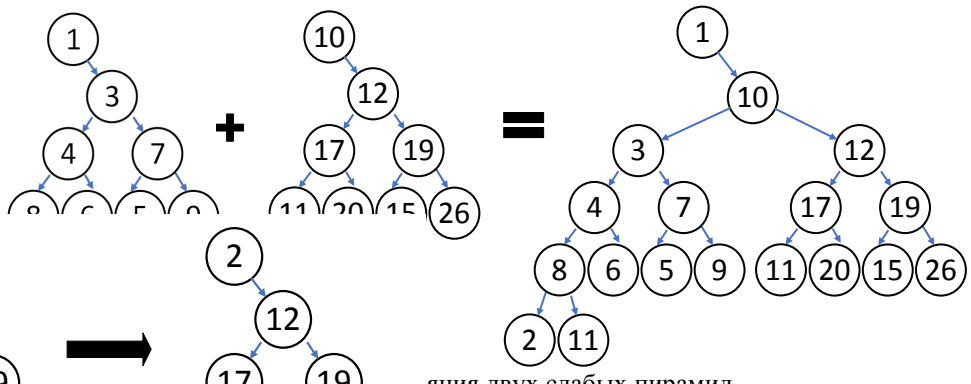
Построение слабой пирамиды можно осуществлять двумя способами. Первый способ (рис. 8а) состоит в заполнении структуры слабой пирамиды по уровням, а затем восстановление её основных свойств.

Второй способ (рис. 8б) - построение слабой пирамиды путём вставки.

Для примера возьмем последовательность чисел: 5, 12, 6, 14, 3, 8, 1, 17

Операция удаления минимального элемента из слабой пирамиды. Операция DELETE-Min работает следующим образом:

1. Удаляется корневой элемент (минимальный элемент) и заменяется фиктивным элементом со значением ∞,



яния двух слабых пирамид

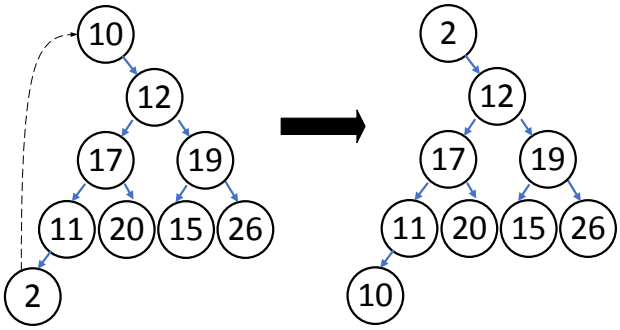


Рис 7. Слабая пирамида после вставки элемента с ключом 2

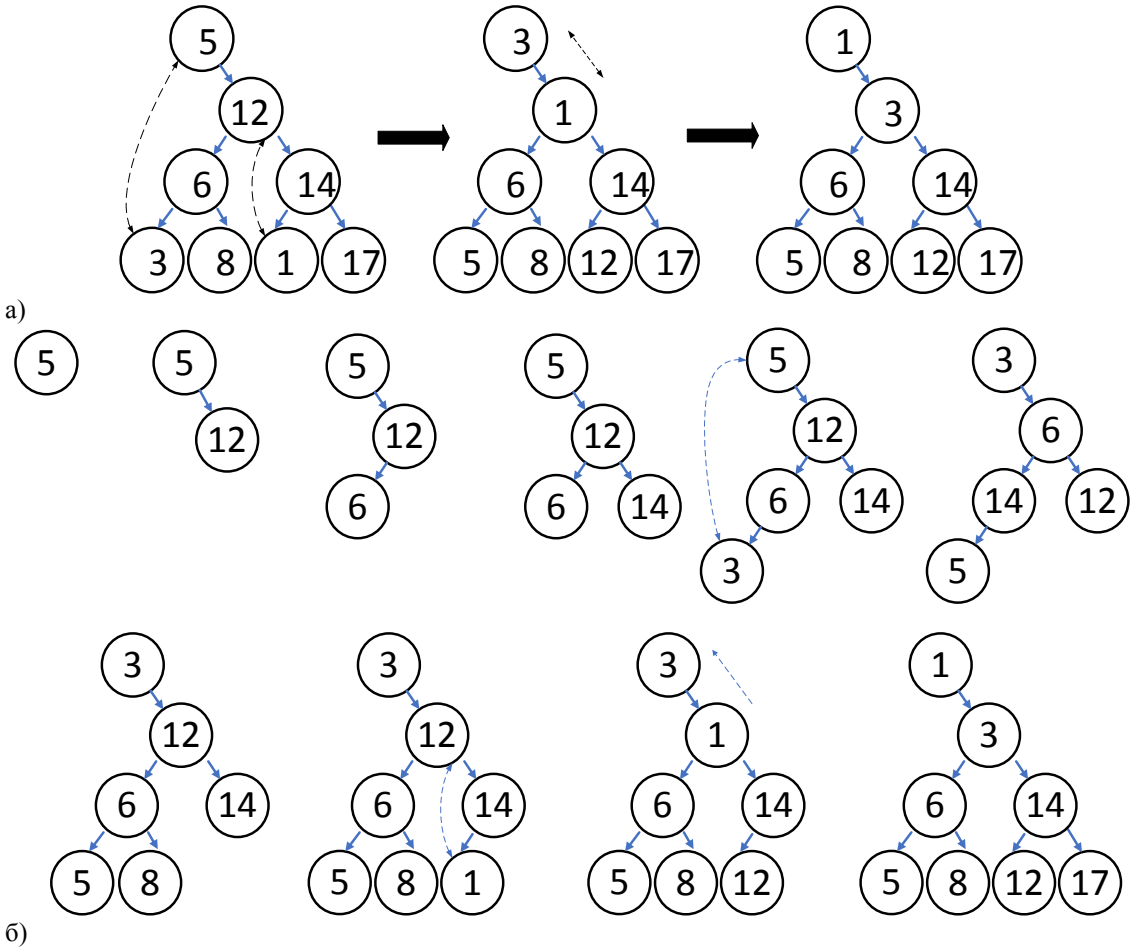


Рис. 8. Построение слабой пирамиды

2. Вызывается Swap (∞; x), где x - второй по величине элемент в слабой пирамиде,

3. Продолжается использование левых поддеревьев (x = 2x) и рекурсивно вызывается Swap (∞; i) пока не будет достигнут лист,

4. Удаляется элемент ∞

При рекурсивном вызове функции Swap на 3-м шаге мы никогда не должны забывать обменять левое / правое поддерево. Это обеспечивает свойство упорядоченности слабой пирамиды.

Операция DELETE-Min требует время от $\lceil \log n \rceil$ ($\log n$ требуется для нахождения наименьшего элемента на пути минимальных элементов и 0, если следующий наименьший элемент - последний) до $\lceil 2\log n \rceil$ сравнений ($\log n$ для поиска наименьшего элемента на пути наименьших элементов и еще один $\log n$, чтобы заполнить все рекурсивные нарушения, пока мы не достигли последнего элемента на пути наименьших элементов).

Таким образом, удаление минимума имеет сложность O($\log n$) и включает не более $\lceil \log n \rceil$ сравнений элементов. На рис. 9 приведен пример удаления минимального элемента.

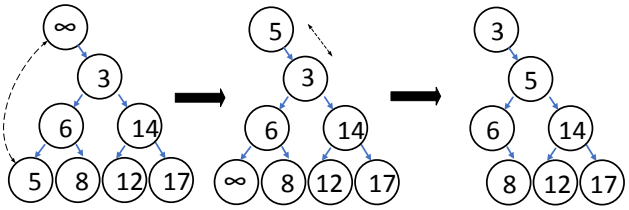


Рис 9. Слабая пирамида после удаления элемента с минимальным ключом.

VII. Операция уменьшения элемента в слабой пирамиде.

Операция DecreaseKey стартует в узле x, величину которого надо уменьшить. В начале изменяется значение заданного элемента, а затем восстанавливаются основные свойства пирамиды в случае необходимости также как это делалось в предыдущих операциях. В примере на рис. 10 показано уменьшение элемента с ключом 8 на ключ 4. Далее восстанавливаются основные свойства слабой пирамиды: проверяется условие $GParent(a_i) \leq a_i$ и в случае необходимости ключи меняются местами (в нашем случае меняются местами ключи 4 и 6, а затем меняются местами поддеревья узла с ключом 4.

Сложность выполнения операции DecreaseKey O($\log n$).

VIII. Слабые пирамиды с использованием буферов.

Сложность операции вставки может быть уменьшена до амортизируемой константы. Ключевая идея - использовать буфер, поддерживающий постоянное время вставки. Буфер может быть реализован как отдельный массив изменяемого размера или как расширение массива элементов a [4].

Кроме того, сохраняется указатель на минимальный элемент в буфере. Максимальный размер

буфера установлен равным ⌈log n⌉ +2 где n общее количество сохраненных элементов. Новый элемент вставляется в буфер, если его размер равен нижнему пороговому значению. Как только пороговое значение достигнуто, вставка массива выполняется путем перемещения всех элементов буфера в слабую пирамиду. Для операции удаления минимума минимум буфера сравнивается с минимумом слабой пирамиды и, соответственно, операция выполняется либо в буфере, либо в слабой пирамиде. Удаление минимума из буфера выполняется путем удаления минимума и сканирования буфера для определения нового минимума. Удаление минимума из буфера имеет сложность O⌈logn⌉ и включает не более ⌈logn⌉ сравнений элементов.

Рассмотрим, как выполнить массовую вставку за время O(logn). Сначала перемещаются элементы буфера к первым свободным местам массива элементов. Упорядоченность слабой пирамиды восстанавливается всплытием по уровням. Начиная с родителей новых

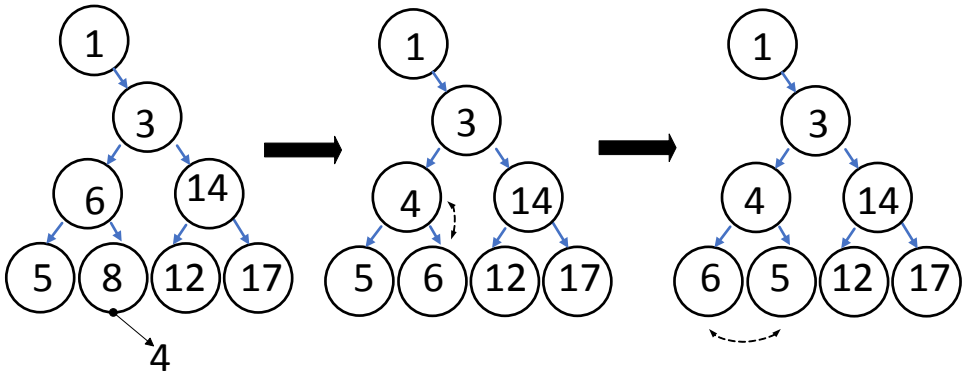


Рис 10. Слабая пирамида после уменьшения элемента с ключом 8 на ключ 4.

узлов, для каждого узла мы выполняем операцию всплытия для восстановления порядка слабой пирамиды между элементом в этом узле и теми, что находятся в его правом поддереве. Затем мы рассматриваем родителей этих узлов на следующем верхнем уровне, восстанавливая порядок слабой пирамиды до этого уровня. Это повторяется до тех пор, пока число узлов, которое нам нужно обработать на уровне, не станет равным двум. На этом этапе мы должны перейти к более эффективной стратегии. В противном случае амортизационная стоимость одной вставки будет логарифмической (из-за стоимости повторных операций погружения)!

Во второй фазе мы восстанавливаем порядок слабой пирамиды на двух путях от этих двух узлов вверх. Для этого мы идентифицируем Gparent для каждого из двух узлов, затем выполняем операцию погружения, после чего следует операция всплытия, начиная с каждого из двух Gparent.

Для слабой пирамиды с массовыми вставками время работы вставки равно O(1) в амортизационном смысле. Число сравнений элементов, выполненных для каждой вставки, равно 5 + O(1).

Вместо одного буфера можно использовать k буферов, каждый из которых имеет размер k. Мы сохраняем минимум каждого буфера в его корне и сохраняем указатель на общий минимум буферов. Вставленный элемент хранится в буфере, пока у него есть пустой слот. После введения минимума этого буфера, а также общего минимума буферов при необходимости он корректируются. Для этого требуется не более двух сравнений элементов. Если все буферы полны, мы применяем групповую вставку к элементам k^2 в буферах таким же образом, как и раньше. Это составляет $(2+O(1))k^2+2\lceil \log n \rceil$ сравнений элементов в целом. Для операции удаления минимума общий минимум буферов сравнивается с минимумом слабой пирамиды. Если минимум находится в буфере, после удаления мы находим новый минимум в этом буфере, а затем новый общий минимум буферов. Это включает в себя не более 2k-3 сравнений элементов. Если минимум находится в слабой пирамиде, удаление выполняется способом, описанным ранее.

IX. Слабая очередь как лес идеальных слабых пирамид.

Слабое полупирамидальное упорядоченное дерево возникает естественным образом, когда многоарное дерево рассматривается как бинарное дерево. Идеальная слабая пирамида, в которой хранится ровно 2^r элементов, представляет собой пирамидальное упорядоченное биномиальное дерево ранга r. С другой стороны, пирамидальное упорядоченное биномиальное дерево является компактным представлением совершенного турнирного дерева. Таким образом, деревья турниров, биномиальные очереди и слабые

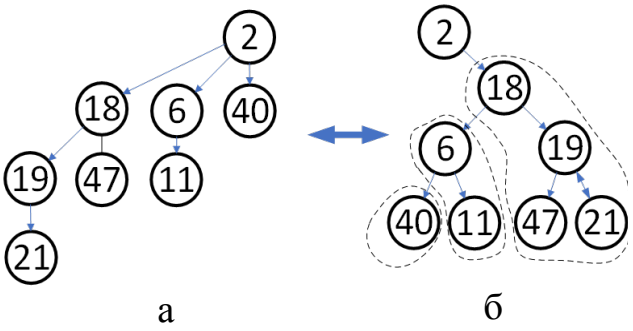


Рис. 11. Пирамидальное упорядоченное биномиальное дерево ранга 3 (Рис. 11а) и соответствующая идеально слабая пирамида (Рис. 11б) высоты 3.

пирамиды являются тесно связанными структурами данных. На рис.11 показан переход от биномиального дерева к слабой пирамиде.

Переход от биномиального дерева к идеально слабой пирамиде осуществляется следующим образом. Вся левая ветвь биномиального дерева становится правой ветвью идеально слабой пирамиды. Левой ветвью каждого узла (кроме корня) являются его правые братья.

Если слабая пирамида не идеальна, т.е. элементы на последнем уровне не идут справа налево, либо последний уровень не полный, то пирамида представляется несколькими биномиальными деревьями.

Другие отличительные свойства слабой пирамиды:

1) она может быть несовершенной (в отличие от биномиального дерева);

2) это одно дерево (в отличие от биномиальной очереди, представляющей собой коллекцию идеальных деревьев);

3) она достаточно сбалансирована (в отличие от, например, парных пирамид).

Вставка элемента выполняется, начиная с листа вверх до восстановления порядка пирамиды, а удаление минимального значения начинается с корня. Поддерживаются для динамической пирамиды в виде массива операция поиск минимума за O(1) в наихудшем случае, а вставка и удаления минимума за O(logn) в наихудшем случае, используя не более $\lceil \log n \rceil$ сравнений элементов. Слабую пирамиду размера n можно построить с использованием n-1 сравнений элементов.

Ранги биномиальных деревьев соответствуют высотам слабой пирамиды.

Операции на идеальной слабой пирамиде – легкие и более гибкие чем на биноминальных деревьях. Кроме того, бинарные деревья обеспечивают лучшее использование памяти, так как необходимы две ссылки, чтобы отразить отношение потомка и родителя. А слабая очередь, хранящая n элементов, является набором непересекающихся идеальных слабых пирамид основанных на двоичном представлении $n = \sum_{i=0}^{\lfloor \log n \rfloor - 1} b_i 2^i$. В канонической форме, и слабая очередь содержит идеальные слабые пирамиды H_i размера 2^i только если $b_i = 1$.

Как правило, слабые пирамиды определены в более общем виде, где число хранящихся элементов не должно быть степенью двойки.

Обычно слабые пирамиды представляют с помощью массива с дополнительными битами, но представление на основе деревьев более гибкое, когда подпирамиды часто перемещаются, что имеет место в наших алгоритмах. Вместо операций над одной слабой пирамидой, мы сохраняем множество непересекающихся совершенных слабых пирамид таким же способом, как можно сохранить множество пирамидальных упорядоченных биномиальных деревьев в биномиальной очереди.

Хотя метод прост, понятен концептуально, нельзя утверждать, что он привел бы к самому короткому коду, или к самому быстрому внедрению на практике, по сравнению с существующими методами. Это связано с тем, какие операции и их сочетание преобладают в соответствующем алгоритме.

Сравнение асимптотической сложности операций для наиболее характерных пирамид (таблица 1) показывает, что эти показатели у слабой пирамиды не хуже, чем у наиболее популярных пирамид. Однако, на практике эффективность пирамид зависит не только от её теоретической оценки, но и от многих факторов, таких как основные операции в алгоритме решаемых задач и их количественного сочетания, длины кода программы, кэширования, ветвления программы и других метрик. Экспериментальное исследование слабых пирамид является отдельной задачей. Некоторые попытки были сделаны в работе [12]. Однако, эта работа требует продолжения с целью выработки рекомендаций по применению этой структуры данных в различных алгоритмах.

СПИСОК ЛИТЕРАТУРЫ

1. Гулаков, В.К. Дрожащая пирамида / В.К. Гулаков, К.В. Гулаков. International Journal of Open Information Technologies ISSN: 2307-8162 vol. 6, no.1, 2018

2. Bruun, A., Edelkamp, S., Katajainen, J., Rasmussen, J. Policy-based benchmarking of weak heaps and their relatives, in: Proceedings of the 9th International Symposium on Experimental Algorithms, vol. 6049 of Lecture Notes in Computer Science, Springer-Verlag, 424-435, 2010.

3. Dutton, R. D. Weak-heap sort, BIT 33 (3) (1993) 372-381.

4. Edelkamp, S., Elmasry, A., Katajainen, J. The weak-heap data structure: Variants and applications, Journal of Discrete Algorithms 16 (2012) 187-205.

5. Edelkamp, S., Elmasry, A., Katajainen, J. A catalogue of weak-heap programs, CPH STL Report 2012-2, Department of Computer Science, University of Copenhagen, 2012.

6. Edelkamp S., Elmasry A., Katajainen J.i. Weak Heaps Engineered 23rd International Workshop on Combinatorial Algorithms held in Tamil Nadu, India, in July 2012.

7. Edelkamp, S. Rank-relaxed weak queues: Faster than pairing and Fibonacci heaps., Technical Report 54, TZI, Universit¨at Bremen, 2009.

8. Edelkamp, S., Elmasry A., Katajainen, J. The Weak-Heap Family of Priority Queues in Theory and Praxis Proceedings of the Eighteenth Computing: The Australasian Theory Symposium (CATS 2012), Melbourne, Australia, 2012.

9. Elmasry, A. Violation heaps: A better substitute for fibonacci heaps. CoRR, abs/0812.2851, 2008.

10. Floyd, R. W. Algorithm 245: Treesort 3, Communications of the ACM 7 (12) (1964) 701.

11. Fredman, M. L., Sedgewick, R., Sleator, D. D., Tarjan, R. E. The Pairing Heap: A New Form of Self-adjusting Heap, Algorithmica, 1 (1986), 111–129.

12. Larkin, D.H., Sen, S., Tarjan, R.E. A Back-to-Basics Empirical Study of Priority Queues / 16th Workshop on Algorithm Engineering and Experiments 2014 (ALENEX14) Portland, Oregon, USA 5 January 2014. P. 61-73

13. Peterson, G. L. A balanced tree scheme for meldable heaps with updates. Technical Report GIT-ICS-87-23.

School of Information and Computer Science, Georgia Institute of Technology (1987).
14. Rasmussen, J. Implementing run-relaxed weak queues. Technical Report CPH STL 2008-1, Department of Computing, University of Copenhagen, 2005.
15. Takaoka, T. Theory of 2-3 Heaps. In Proc. of COCOON 99, vol. 1627 of Lecture Notes in Computer Science, pp.41-50, 1999.
16. Vuillemin, J. A data structure for manipulating priority queues, Communications of the ACM 21 (4) (1978) 309-315.

Сведения об авторах:
Гулаков Василий Константинович
кандидат технических наук
Брянский государственный технический университет
профессор кафедры Информатика и программное обеспечение, (БГТУ)
Эл. почта: gvk10@yandex.ru
Телефон: +79103328612

Гулаков Константин Васильевич
кандидат технических наук
Брянский государственный технический университет
доцент кафедры Информатика и программное обеспечение, (БГТУ)
Эл. почта: gulakov32@yandex.ru
Телефон: +79103386234

Weak heap

V.K. Gulakov, K.V. Gulakov

Abstract— the article is devoted to one variety of pyramidal data structures - a weak pyramid. Some problems where weak pyramids are more effective are considered. A brief comparison of it and its species with the most popular pyramid structure shows the ability to perform basic operations on the data with minimum complexity. The article gives detailed description of the weak pyramid structure in the array form and in the coherent structure form. Preference is given to the connected structure visibility. Examples of performing various operations on the structure under consideration are given with the help of figures, and an assessment of its implementation complexity is given. To increase the efficiency of insertion operations, the approach using buffers is considered. The connection between a weak pyramid and a binomial pyramid is considered. Here the concept of an ideal weak pyramid is used. The article outlines the ways of experimental research of a weak pyramid in various situations and practical problems.

Keywords— pyramidal data structures, weak heap, description of the structure, basic operations, constraints, algorithm complexity.