Многоуровневые пирамиды

В.К. Гулаков, К.В. Гулаков

Аннотация **- Статья посвящена одной из разновидностей биноминальной пирамиды, - многоуровневой пирамиде, широко используемой при решении различных задач. Краткое сравнение её с наиболее эффективными пирамидальными структурами позволило сформулировать цели этой структуры и пути их достижения. В отличие от существующих, рассматриваемая пирамида отличается минимальным порядком сложности $O(\log\log n)$ операции удаления минимального элемента из пирамиды за счёт уменьшения количества операций сравнения. В статье рассматриваются различные варианты реализации многоуровневой пирамиды за счёт некоторого изменения биноминальных деревьев. Первый вариант предполагает наличие верхнего, нижнего уровней и хранилища пирамиды, за счёт которого происходит компенсация удалённых элементов и уменьшается количество операций по восстановлению пирамиды на нижнем уровне. Вторая реализация предполагает отказ от хранилища, но вместо биномиальных деревьев вводятся похожие на биноминальные F-деревья. Полученная F-пирамида позволяет выполнять операции удаления и удаления минимального элемента со сложностью $O(\log\log n)$. Показана возможность дальнейшего совершенствования этой структуры.**

Ключевые слова: **деревья, многоуровневая пирамида, пирамидальные структуры данных, сложность алгоритма.**

Введение

Многоуровневая пирамида (Layered Heap) была предложена в работе [8]. Эта структура используется для сокращения количества выполняемых операций сравнения в операциях удаления и удаления минимума в пирамидальных структурах. Она обеспечивает, в амортизированном смысле, постоянный порядок сложности на вставку, поиск минимума и уменьшения ключа, и логарифмический порядок сложности не более чем $1.44\log n + O(\log\log n)$ сравнений при удалении и удалении минимального элемента.

Известны несколько пирамидальных структур, которые обеспечивают постоянную сложность вставки и логарифмические затраты на удаление и удаление минимального элемента. Примерами таких пирамид, которые достигают этой оценки в амортизационном смысле [14], являются биноминальные очереди [4, 15] и парные пирамиды [9, 11]. Такая же оценка может быть достигнута в худшем случае со специальной реализацией биноминальных очередей. Пирамиды Фибоначчи [10] и тонкие пирамиды [12] достигают в амортизированном смысле постоянных затрат на каждую вставку, поиска минимума и уменьшения ключа, и логарифмическую сложность для удаления и удаления минимума. Другие структуры пирамид, которые достигают таких границ в худшем случае - это релаксационные пирамиды [7] и очереди с приоритетом [1, 2, 5, 6].

Среди приоритетных очередей, которые обеспечивают постоянную сложность вставки, количество сравнений, выполняемых в операции удаления минимума для биноминальных пирамид и парных пирамид, ограничено $2\log n + O(1)$. Константа, участвующая в логарифмическом коэффициенте, для других пирамид, упомянутых выше, больше чем 2.

Поскольку многоуровневые пирамиды используют биноминальные очереди в качестве базовой структуры, опираемся на операции для биноминальных пирамид, рассмотренные в [3].

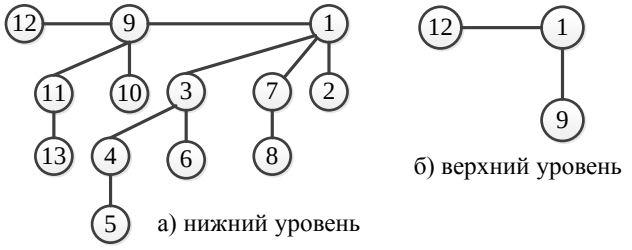
Для биноминальных пирамид существуют две основные процедуры, которые создают коэффициент умножения 2 при оценке количества сравнений в операциях удаления минимума: первая - это объединение деревьев с равными рангами, а вторая - сохранение нового минимального элемента.

Чтобы уменьшить количество сравнений, связанных с определением нового минимума после удаления текущего минимума, до $O(\log\log n)$, вводится новая структура, которая разделяется на верхний и нижний уровни.

Рассматриваются два варианта реализации многоуровневых пирамид. Первый вариант предназначен для выполнения всех стандартных операций кроме операции уменьшения с амортизационным порядком сложности O(1) для операций вставки, поиска минимума и логарифмическим порядком сложности для операций удаления и удаления минимума. Для уменьшения количества операций сравнения в операциях удаления и удаления минимума вводится промежуточная структура – хранилище.

Второй вариант предназначен для выполнения всех стандартных операций с амортизационным порядком сложности O(1) для операций вставки, поиска минимума, уменьшения элемента и логарифмическим порядком сложности для операций удаления и удаления минимума. В этом варианте хранилище не используется, но вместо стандартных биноминальных деревьев используются похожие на биноминальные F-деревья.

Рассмотрим первый вариант. Исходная очередь реализуется как биноминальная, при этом на верхнем



а) нижний уровень б) верхний уровень

Рис. 1. Представление верхнего и нижнего уровней пирамиды

уровне формируется еще одна структура биноминальной пирамиды, которая содержит только

элементы корней биноминальных деревьев исходной очереди (рис. 1).

Таким образом, минимальный элемент этого верхнего уровня является общим минимальным элементом. Размер верхнего уровня - $O(\log n)$, а для удаления минимума в этом уровне требуется $O(\log \log n)$ операций.

Задача состоит в том, как сохранить верхний уровень и эффективно выполнять операции над пирамидой на нижнем уровне (исходной пирамидой), чтобы уменьшить работу на верхнем уровне, тем самым достигнув заявленных ограничений. Если операция удаления минимума реализована так же, как и для стандартных биноминальных очередей, то будет логарифмическое число новых корней, которые необходимо вставить в верхний уровень. Следовательно, вводится новая реализация операции удаления минимума, которая не изменяет текущие корни деревьев. Рассмотрим, как различные операции над пирамидой реализованы для обоих уровней.

Удаление минимального элемента. На нижнем уровне задан указатель из дерева верхнего уровня, которое имеет корень с минимальным значением. Удаление минимума реализуется следующим образом (Рис. 2).

дерево теряет узел, ему по-прежнему присваивается один и тот же ранг. С каждым деревом поддерживается счетчик, указывающий количество удаленных узлов из этого дерева.

Когда биноминальное дерево ранга r теряет 2^{r-1} узлов (половина его полных узлов), дерево перестраивается за линейное время, образуя биноминальное дерево ранга r-1. Если в очереди существует другое дерево ранга r-1, эти два дерева сливаются, образуя дерево ранга r, а счетчики обновляются.

Удаление элемента. Когда узел удаляется с помощью операции удаления, его потомки объединяются с помощью процедуры возрастающего слияния. Счетчик, связанный с деревом, который включает удаление, уменьшается и, когда это необходимо, происходит перестройка, как в случае операции удаления минимума.

В рассмотренном примере (рис. 3) происходит удаление узла 3, пункт а). Полученные в результате удаления элемента новые деревья сливаются с помощью возрастающего слияния справа налево (пункт б)), как в случае удаления минимума. В пункте в) полученное дерево, в котором не хватает одного элемента, вставляется на место удаленного элемента.

Амортизированное количество сравнений,

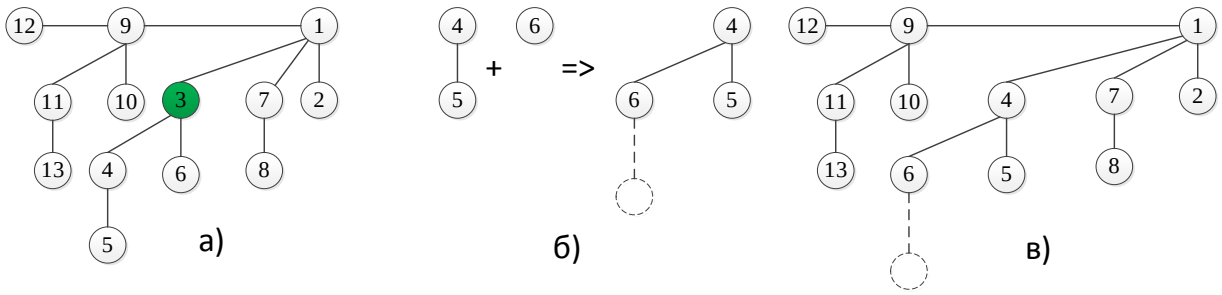


Рис. 3. Операция удаления элемента

После удаления минимального узла (Рис. 2а) его поддеревья последовательно объединяются справа налево (поддеревья с меньшими рангами), образуя новое дерево с одним наименьшим узлом (Рис. 2б, в, г). Другими словами, каждое поддерево сливается с деревом, являющимся результатом поэтапного слияния поддеревьев, находящихся справа от него. Слияние выполняется аналогично слиянию биноминальных деревьев; корень дерева, который имеет наибольшее значение, становится самым левым потомком другого корня. Эту процедуру обычно называют возрастающим или каскадным слиянием. Несмотря на то, что такое

выполняемое в операциях удаления и удаления минимума на нижнем уровне, ограничено $\log n + O(1)$.

Верхний уровень реализуется как стандартная биноминальная пирамида, содержащая корни деревьев нижнего уровня.

Операция вставки реализована так же, как и в стандартных биноминальных пирамидах.

Когда узел с минимальным значением должен быть удален или когда операция удаления выполняется в корне дерева на нижнем уровне, этот узел также удаляется с верхнего уровня с дополнительной сложностью $O(\log \log n)$. Узел, который продвигается

вместо удаленного узла на нижнем уровне, вставляется на верхнем уровне с постоянной амортизированной сложностью. Когда новый узел вставлен на нижний уровень, это сопровождается постоянным количеством слияний (в амортизированном смысле). В результате операций слияния некоторые корни деревьев в нижнем уровне, связанные с другими корнями в верхнем уровне, должны быть удалены из него. На каждое из этих удалений тратится постоянное время. Применяется метод ленивых удалений, где удаляемый узел помечен до тех пор, пока не появится возможность для массовых удалений. Если любой из этих отмеченных узлов снова повышается в ранг корня на нижнем уровне (в результате удаления или удаления минимума), его метка на верхнем уровне удаляется.

Когда число отмеченных узлов достигает постоянного множителя для узлов верхнего уровня (скажем, половины), верхний уровень перестраивается за линейное время, избавляясь от отмеченных узлов. Сложность перестроения амортизационная с учётом что взятый k раз и k - константа, представляющая количество уровней. Вставка нового элемента приведет к постоянному амортизированному количеству вставок и меток на уровне. Количество уровней k должно быть постоянным для достижения постоянной амортизируемой сложности вставки.

Когда операции удаления и удаление минимума выполняются на нижнем уровне нашей структуры, биноминальные деревья теряют узлы, а их структура отклоняется от исходной биноминальной древовидной структуры. Наше амортизированное решение состоит в том, чтобы позволить любому дереву потерять половину своих потомков, а затем перестроить дерево как биноминальное дерево. Для достижения заявленного наихудшего ограничения, биноминальные деревья должны терять узлы более равномерным образом, чтобы сохранить их структурные свойства. Для этого вводится дополнительная структура данных - хранилище.

Основная идея состоит в том, чтобы сохранить одно дерево в хранилище и обработать его особым

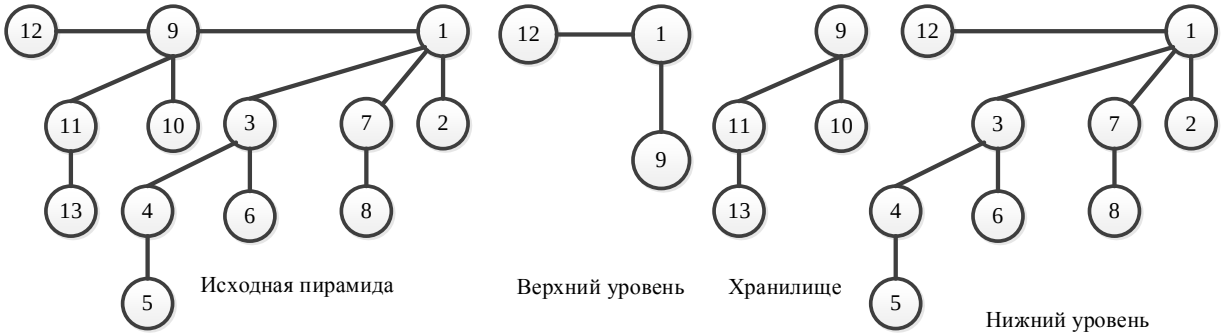


Рис. 4. Представление биноминальной пирамиде в виде структуры с хранилищем

слияния на нижнем уровне имеют постоянную сложность. Схема «ленивых» (отложенных) удалений работает потому, что не отмеченные узлы не могут стать минимальным узлом верхнего уровня, так как верхний уровень должен иметь немаркированный узел меньше каждого отмеченного узла.

Структура многоуровневой пирамиды обеспечивает в амортизированном смысле постоянные затраты на каждую вставку, нахождение минимума и логарифмическую сложность с максимальным количеством $\log n + O(\log \log n)$ сравнений при удалении и удалении минимума.

Оценка количества сравнений для операций удаления и удаления минимума может быть дополнительно уменьшена. Вместо двух уровней у нас может быть несколько уровней. Операция удаления минимума на втором уровне реализована так же, как и для первого уровня. На каждом уровне, отличном от самого высокого уровня, указатель для минимального элемента на этом уровне поддерживается для следующего более высокого уровня. За исключением самого высокого уровня, нам нужен постоянный логарифмический множитель, как оценка количества сравнений, выполняемых для удаления минимума на данном уровне. Поэтому оценка операций удаления и удаления минимума не более чем $\log n + \log \log n + \cdots + 2\log^{(k)} n + O(1)$ сравнений, где $log^{(k)}$ логарифм, образом (рис. 4). Всякий раз, когда узел удаляется в результате операции удаления или удаления минимума, мы берем узел из дерева в хранилище. Используя этот заимствованный узел, поддерево, которое потеряло свой корень, перенастраивается как биноминальное дерево той же структуры, что и до удаления. Рассмотрим это подробнее.

Чтобы заимствовать узел из хранилища, мы отделяем самого правого потомка от его корня, делая потомков следующих поддеревьев корня самыми правыми потомками корня в том же порядке. Если в хранилище остается только один узел, мы берем этот узел, отмечаем его как соответствующий узел на верхнем уровне и перемещаем другое биноминальное дерево из нижнего уровня в хранилище.

Важнейшим ограничением является то, что ранг дерева, перемещаемого в хранилище, не является самым большим среди деревьев нижнего уровня.

Особый случай - когда на нижнем уровне есть только одно дерево. В этом случае мы разбиваем это дерево, отрезая самое левое поддерево от его корня и перемещая это поддерево в хранилище, вставляем его корень в верхний уровень.

Всякий раз, когда корень биноминального дерева на нижнем уровне удаляется в результате операции удаления минимума, узел, который заимствован из хранилища, постепенно объединяется с поддеревьями

удаленного корня справа налево. Это приводит к биномиальному дереву с той же структурой, что и до удаления, и требует не более $\log n$ сравнений. На верхнем уровне, новый корень этого дерева вставляется, а соответствующий старому корню узел удаляется.

Если корень дерева в хранилище должен быть удален или если он имеет минимальное значение в операции удаления минимума, этот корень удаляется, и процедура поэтапного слияния выполняется на его потомках справа налево. На верхнем уровне вставляется новый корень хранилища, а соответствующий узел

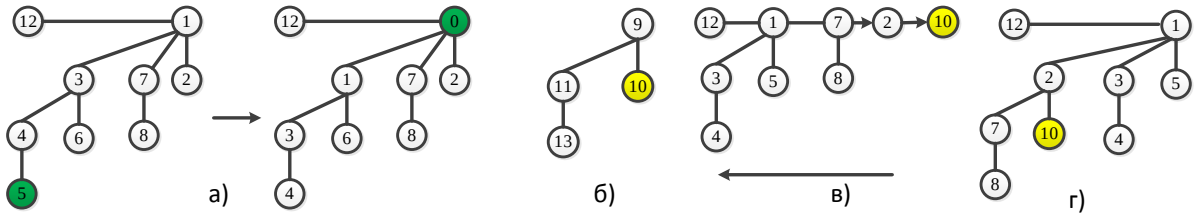


Рис. 5. Операция удаления с использованием хранилища

старого корня удаляется.

Рассмотрим операцию удаления с использованием хранилища на примере (рис. 5). В пункте а) удаляемый узел 5 «всплывает», как это было рассмотрено на рис. 2. После чего из хранилища берется самый правый потомок корня хранилища, элемент 10 (пункт б). После удаления минимума, образуются новые биномиальные деревья с рангами от 0 до 2. Заимствованный из хранилища узел сливается с образовавшимся деревом ранга 0, в результате чего получается дерево ранга 1, которое уже может слиться со следующим деревом того же ранга, т.е. происходит дальнейшая процедура возрастающего слияния (пункт в)). В результате заимствования узла из хранилища, после процедуры возрастающего слияния получается дерево того же ранга, что и до удаления элемента, в котором присутствуют все узлы (пункт в)). Полученное дерево вставляется в то же место в пирамиде, на котором оно было до удаления элемента.

Вставка. Подобно биномиальным пирамидам, вставка выполняется добавлением нового узла ранга 0 к нижнему уровню. Если в пирамиде есть два биномиальных дерева одного ранга r, два дерева должны быть объединены, образуя биномиальное дерево ранга r + 1. Мы не можем выполнить все необходимые слияния сразу. Вместо этого мы делаем постоянное количество слияний с каждой вставкой. Подобно нашему амортизированному решению, нам нужно сделать вставку и «ленивое» удаление (маркировку) на верхнем уровне.

Глобальная перестройка верхнего уровня. Чтобы достичь требуемых оценок в наихудшем случае, используем метод глобальной перестройки. Если количество немаркированных узлов меньше m/c, где m - количество узлов на верхнем уровне и c≥2 - некоторая константа, начинаем перестраивать весь уровень.

По-прежнему используем и обновляем наш оригинальный верхний уровень, но параллельно мы также создаем новую структуру пирамиды. Если узел, который должен быть отмечен, также существует в новой структуре, он также должен быть помечен. Всякий раз, когда новый узел вставлен в текущий верхний уровень, мы вставляем его и в новую структуру. Всякий раз, когда мы отмечаем узел для удаления или вставляем новый узел в текущий верхний уровень, мы копируем два немаркированных узла из текущей структуры в новую структуру.

Из этого следует, что в течение следующих не более m/2c операций все немаркированные узлы должны быть скопированы в новую структуру. На этом этапе можно убрать текущую структуру и использовать новую. Новая структура будет содержать не менее половины узлов.

Наша модифицированная структура пирамиды обеспечивает постоянную сложность каждой вставки и поиска минимума и логарифмическую сложность с максимальным количеством $\log n + O(\log \log n)$ сравнений при удалении и удалении минимума.

Второй вариант реализации многоуровневой пирамиды основан на понятии F-пирамиды, которая используется в качестве нашей основной структуры. Пирамида, состоящая из F-деревьев, позволяет убрать хранилище, т.к. при операции удаления в F-деревьях не нужно заимствовать узлы, чтобы сохранить свою структуру, и позволяет поддерживать операцию уменьшения ключа.

F-ПИРАМИДЫ.

F-дерево рекурсивно определяется следующим образом. F-дерево ранга 0 является единственным узлом. F-дерево ранга r состоит из корневого узла, который имеет r или r - 1 поддеревьев. Эти поддеревья, справа налево, являются F-деревьями с последовательными рангами 0, 1, … , r -1 или 0, 1, … , r - 2. Назовем F-дерево ранга r, тяжелым F-деревом, если его корень имеет r поддеревьев, и легким, если его корень имеет r - 1 поддерево.

Каждый узел в F-пирамиде реализуется тремя указателями, указывающими на левого брата, правого брата и левого потомка. Левый указатель самого левого потомка указывает на родителя вместо его несуществующего левого брата.

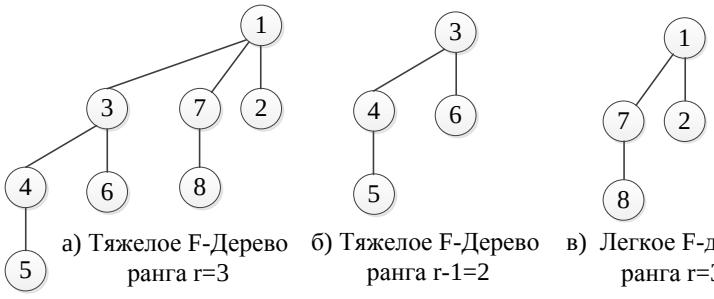


Рис. 6. Операция разделения в F-дереве

F-пирамида - это лес F-деревьев, в каждом из которых выполняется основное свойство пирамиды, т.е. значение каждого узла меньше или равно значению его потомков. Основные F-деревья все тяжелые, но это условие не обязательно верно для поддеревьев. Для этих деревьев вводятся две дополнительные операции: разделение и слияние.

Разделение. Тяжелое F-дерево ранга r можно разбить на два F-дерева, вырезав самое левое поддерево корня из остальной части дерева. Это самое левое поддерево образует F-дерево ранга r - 1 (тяжелое или легкое), а остальная часть дерева образует легкое F-дерево ранга r (рис. 6). В этой операции сравнения не выполняются.

Слияние. Два F-дерева одного ранга r могут быть объединены за постоянное время и за одно сравнение, что приводит к F-дереву ранга r+1. Пусть x - корень дерева, имеющего наибольшее значение.

1. Если два дерева тяжелые: корень x вставляется в качестве крайнего левого потомка другого корня, образует тяжелое F-дерево ранга r+1, т.е. обычное слияние биноминальных пирамид.
2. Если два дерева легкие, то x связывается как самый левый потомок другого корня, образуя легкое F-дерево, т.е. обычное слияние биноминальных пирамид, только а результате получается легкое F-дерево

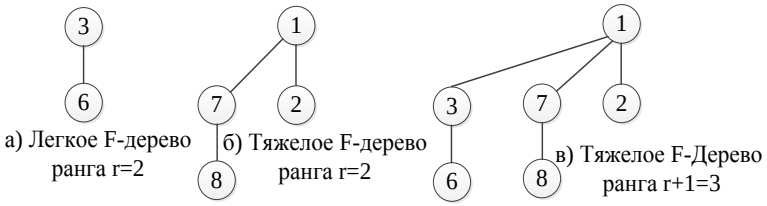


Рис. 7. Операция слияния тяжелого и легкого F-деревьев. Легкое F-дерево имеет большее значение корня

3. Если одно дерево легкое (рис. 7а), а другое тяжелое (рис. 7б):
   - 3.1. Если дерево x является легким деревом, то ссылка на x как на самый левый потомок другого корня, образует тяжелое F-дерево (рис. 7в).
   - 3.2. Если дерево x является тяжелым деревом (рис. 8а), то нужно разделить дерево x на два F-дерева рангов r - 1 и r (рис. 8б и 8в). Полученное лёгкое дерево ранга r=2 (рис. 10в) сливается с исходным лёгким деревом (рис. 8г) в результате чего получается легкое дерево ранга r+1 (рис. 8д). После этого, полученное в результате разделения, тяжёлое дерево ранга r также становится самым левым потомком результирующего легкого дерева ранга

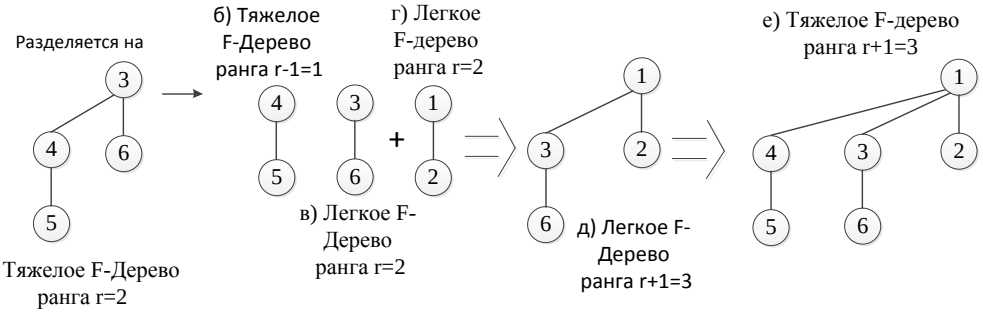


Рис. 8. Операция слияния тяжелого и легкого F-деревьев, где F-дерево имеющее большее значение корня - тяжелое

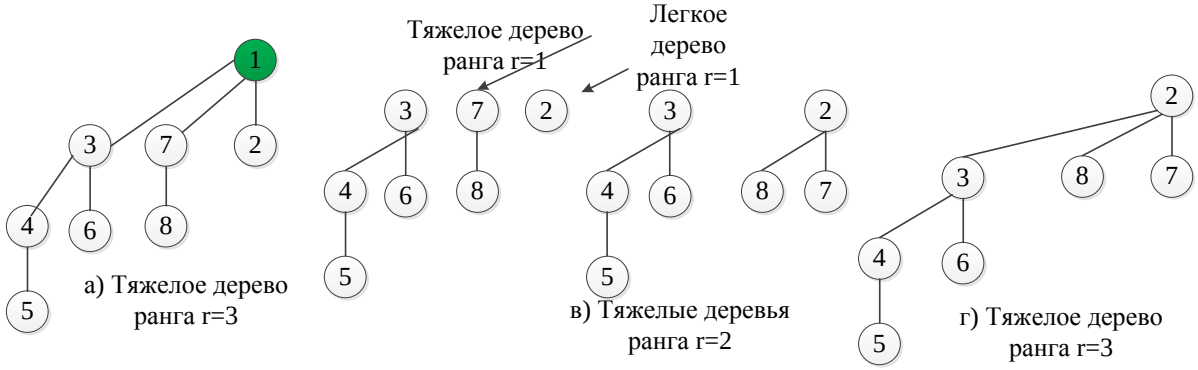


Рис.9. Удаление минимума в F-дереве

r+1, тем самым образует тяжелое F-дерево ранга r+1(рис. 8e).

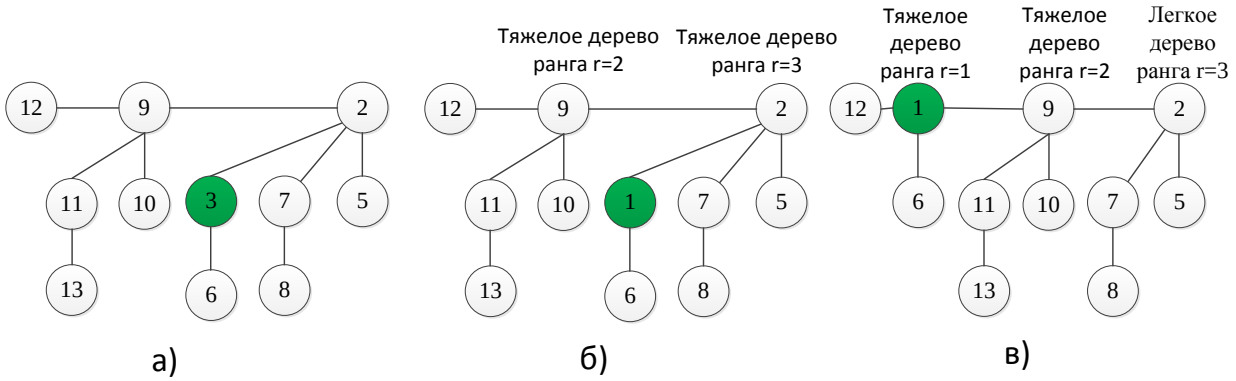


Рис. 10. Операция уменьшения ключа

Удаление минимального элемента. При указании на верхнем уровне на дерево с минимальным значением корня, удаление минимума реализуется следующим образом. После удаления этого минимального узла мы выполняем процедуру возрастающего слияния с потомками справа налево. Начинаем с единственного узла, который является самым правым потомком этого корня (дерево ранга 0) в качестве легкого дерева ранга 1. Отсюда следует, что результирующее дерево по-прежнему будет F-деревом того же ранга (рис. 9). Если полученное F-дерево легкое, делаем его тяжелым, уменьшив его ранг. На верхнем уровне новый корень этого дерева вставлен и удаляется соответствующий узел со старым корнем. Затем продолжается слияние любых двух F-деревьев, имеющих один и тот же ранг, до тех пор, пока не будет двух деревьев одного ранга. После каждого этого слияния соответствующий узел на верхнем уровне помечается для ленивого удаления. Амортизированное количество сравнений, выполняемых этой операцией, составляет не более $1.44\log n + O(\log\log n)$.

Уменьшение ключа. Пусть x - узел, значение которого уменьшается. Если x является корнем дерева на нижнем уровне, операция уменьшения ключа выполняется на соответствующем узле на верхнем уровне, и процедура завершается. В противном случае поддерево x будет вырезано и создано новое дерево. Если это дерево легкое, оно делается тяжелым, уменьшив его ранг (рис. 10). Затем узел x вместе вставляется в верхний уровень. Амортизированная сложность этой операции постоянна.

Удаление. Чтобы удалить узел, его значение уменьшается, чтобы стать наименьшим из узлов пирамиды. Затем он удаляется путем применения операции удаления минимума.

Наша модифицированная структура пирамиды обеспечивает в амортизированном смысле постоянные затраты на каждую вставку, нахождение минимума и уменьшение ключа, и логарифмическую сложность при максимальном сравнении $1.44\log n + O(\log\log n)$ при удалении и удалении минимума.

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.

На практике эффективность пирамид зависит не только от её теоретической оценки, но и от многих факторов, таких как основные операции в алгоритме решаемой задачи и их количественное сочетание, длина кода программы, кэширование, ветвление программы и другие метрики. Экспериментальное исследование многоуровневых пирамид является отдельной задачей. Некоторые попытки были сделаны в работе [13]. Однако, эта работа требует продолжения с целью выработки рекомендаций по применению этой структуры данных в различных алгоритмах.

БИБЛИОГРАФИЯ

1. Гулаков В.К., Гулаков К.В. Дрожащая пирамида // International Journal of Open Information Technologies ISSN: 2307-8162 vol. 6, no.1, 2018
2. Гулаков В.К., Гулаков, К.В. Слабая пирамида // International Journal of Open Information Technologies ISSN: 2307-8162 vol. 6, no.7, 2018
3. Кормен Т., Лейзер Ч., Ривест Р. Алгоритмы: построение и анализ. — 2-е изд. М.: Издательский дом «Вильямс», 2007.— С.1296.
4. Brown M. Implementation and analysis of binomial queue algorithms. SIAM J. Comput. 7 (1978), 298-319.
5. Brodal G. Fast meldable priority queues. 4th WADS, LNCS 955 (1995), 282-290.
6. Brodal G. Worst-case efficient priority queues. 7th ACM SODA (1996), 52-58.
7. Driscoll J., Gabow H., Shairman R., Tarjan R. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. Comm. ACM 31(11) (1988), p. 1343-1354.
8. Elmasry A. Layered Heaps Algorithm Theory - SWAT 2004 p. 212-222
9. Fredman M., Sedgewick R., Sleator D., Tarjan R. The pairing heap: a new form of self adjusting heap // Algorithmica 1(1) (1986), 111-129.
10. Fredman M., Tarjan R. Fibonacci heaps and their uses in improved network optimization algorithms. // J. ACM 34(3) (1987), 596-615.
11. Iacono J. Improved upper bounds for pairing heaps. // 7th SWAT, LNCS 1851 (2000), 32-45.
12. Kaplan H. Tarjan R. New heap data structures. // TR-597-99, Princeton University. 1999.

13. 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
14. Tarjan R. Amortized computational complexity. / SIAM J. Alg. Disc. Meth. 6 (1985), 306-318.
15. Vuillemin J. A data structure for manipulating priority queues. / Comm. ACM 21(4) (1978), 309-314.

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

Layered Heap

V.K. Gulakov, K.V. Gulakov

***Abstract*— The article is devoted to one of the varieties of the binomial heap, a multilevel heap widely used in solving various problems. A brief comparison of it with the most effective pyramidal structures made it possible to formulate the goals of this structure and the ways to achieve them. Unlike the existing ones, the heap is characterized by the minimum complexity order $\mathbf{O}(\log\log n)$ of the operation of removing the minimum element from the heap due to the reduction in the number of comparison operations. The article discusses various options for implementing a multi-level heap due to some change in binomial trees. The first option assumes the presence of upper, lower levels and a heap storage, due to which compensation of deleted elements occurs and the number of operations to restore the heap at the lower level is reduced. The second implementation involves the abandonment of the store, but instead of binomial trees, similar to binomial F-trees are introduced. The resulting F-Heap allows you to perform the removal and deletion of a minimal element with complexity $\mathbf{O}(\log\log n)$. The possibility of further improvement of this structure is shown.**

***Keywords*: trees, multilevel heap, pyramidal data structures, algorithm complexity**.

REFERENCES

1. Gulakov V.K., Gulakov K.V. Drozhashhaja piramida // International Journal of Open Information Technologies ISSN: 2307-8162 vol. 6, no.1, 2018

2. Gulakov V.K., Gulakov, K.V. Slabaja piramida // International Journal of Open Information Technologies ISSN: 2307-8162 vol. 6, no.7, 2018

3. Kormen T., Lejzer Ch., Rivest R. Algoritmy: postroenie i analiz. — 2-e izd. M.: Izdatel'skij dom «Vil'jams», 2007.— S.1296.

4. Brown M. Implementation and analysis of binomial queue algorithms. SIAM J. Comput. 7 (1978), 298-319.

5. Brodal G. Fast meldable priority queues. 4th WADS, LNCS 955 (1995), 282-290.

6. Brodal G. Worst-case efficient priority queues. 7th ACM SODA (1996), 52-58.

7. Driscoll J., Gabow H., Shairman R., Tarjan R. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. Comm. ACM 31(11) (1988), p. 1343-1354.

8. Elmasry A. Layered Heaps Algorithm Theory - SWAT 2004 p. 212-222

9. Fredman M., Sedgewick R., Sleator D., Tarjan R. The pairing heap: a new form of self adjusting heap // Algorithmica 1(1) (1986), 111-129.

10. Fredman M., Tarjan R. Fibonacci heaps and their uses in improved network optimization algorithms. // J. ACM 34(3) (1987), 596-615.

11. Iacono J. Improved upper bounds for pairing heaps. // 7th SWAT, LNCS 1851 (2000), 32-45.

12. Kaplan H. Tarjan R. New heap data structures. // TR-597-99, Princeton University. 1999.

13. 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 R. 61-73

14. Tarjan R. Amortized computational complexity. / SIAM J. Alg. Disc. Meth. 6 (1985), 306-318.

15. Vuillemin J. A data structure for manipulating priority queues. / Comm. ACM 21(4) (1978), 309-314.