2-3 пирамида

К. В. Гулаков

Аннотация - Пирамидальные структуры данных получили последнее время существенное развитие и отличаются большим разнообразие. 2-3 пирамида является одной из представителей пирамид на базе леса деревьев. Преимущества 2-3 дерева заключаются в том, что жёсткое ограничение по высоте log(n) компенсируется балансировкой по ширине (горизонтальная балансировка). Это позволяет иметь минимальную высоту дерева и соответственно увеличить производительность некоторых операций. Тадао Такаоке предложил пирамидальную модификацию этих деревьев и объединение их в 2-3 пирамиду. Изучение этой работы оставляет много вопросов по реализации 2-3 пирамиды и операций над ней. При очевидных достоинствах этой структуры, возможно эти трудности помешали её популярности среди программистов. В настоящей работе проведено полное осмысление этой структуры, прежде всего в плане её реализации и операций над ней, что восполняет пробел в русскоязычной литературе.

Ключевые слова: **деревья, 2-3 пирамида, операции с деревьями, сложность алгоритма.**

Введение

Пирамидальные структуры данных получили последнее время существенное развитие и отличаются большим разнообразием [1]. 2-3 пирамида является одной из представителей пирамид на базе леса деревьев. Преимущества 2-3 дерева заключаются в том, что жёсткое ограничение по высоте log(n) компенсируется балансировкой по ширине (горизонтальная балансировка). Это позволяет иметь минимальную высоту дерева и соответственно увеличить производительность некоторых операций. Тадао Такаоке [4] предложил пирамидальную модификацию этих деревьев и объединение их в 2-3 пирамиду. Изучение этой работы оставляет много вопросов по реализации 2-3 пирамиды и операций над ней. При очевидных достоинствах этой структуры, возможно эти трудности помешали её популярности среди программистов. В настоящей работе проведено полное осмысление этой структуры, прежде всего в плане её реализации и операций над ней, что восполняет пробел в русскоязычной литературе.

Наиболее близкая по асимптотической сложности операций над 2-3 пирамидой является пирамида Фибоначчи [3]. Она может поддерживать n вставок, n удалений минимумов и m операций уменьшения ключа за время (m+n*log(n)). Релаксационные пирамиды Дрисколла [2] и другие имеют одинаковую общую сложность уменьшения ключа за O(1) время в худшем случае, но их трудно реализовать. 2-3 пирамида поддерживает n вставок, n удалений минимумов и m операций уменьшения ключа за O(m + n*log(n)) времени. Эксперименты показывают, что 2-3 пирамиды более эффективны.

Рассматриваемая структура данных может иметь широкое применение в алгоритмах с графами. Две характерные области применения - это проблема нахождения кратчайшего пути от одного источника и проблема минимальной сложности покрывающего дерева. Прямое использование пирамид Фибоначчи в алгоритмах Дейкстры и Прима решает эти две проблемы за время O(m + n*log(n)). Но когда значение ключа узла v уменьшается, поддерево с корнем v удаляется и связывается с другим деревом на корневом уровне. Если мы выполним эту операцию много раз, то форма дерева может стать неверной. То есть количество потомков узла может стать слишком большим из-за ссылок без балансировки. Если это произойдет на корневом уровне, мы столкнемся с трудностью, когда узел с минимумом удален и нам нужно найти следующий минимум. Чтобы предотвратить эту ситуацию, допускается потерять не более одного потомка из любого узла, чтобы остаться на текущей позиции. Если потребуется еще одна потеря, то это приведет к каскадному удалению. Для преодоления этой ситуации используется балансировка.

У дерева поиска есть две хорошо известные схемы балансировки дерева; дерево AVL и 2-3 дерево. Когда мы вставляем или удаляем элементы в или из бинарного дерева поиска, мы можем потерять баланс высоты дерева. Для предотвращения данной ситуации в AVL-дереве мы восстанавливаем баланс, изменяя форму дерева. Поскольку мы контролируем длину пути, мы можем рассматривать эту корректировку как «вертикальную балансировку». В 2-3 дереве применяется «горизонтальная балансировка».

Структура 2-3 пирамиды основана на размерах, которые являются более жесткими, чем в пирамиде Фибоначчи. Степень любого узла в 2-3 пирамидах ограничена log(n), что лучше, чем у пирамид Фибоначчи на постоянный коэффициент. В то время как пирамида Фибоначчи основана на бинарном связывании, 2-3 пирамида основана на троичном связывании: мы связываем три корня трех деревьев в порядке возрастания, в соответствии с ключевыми значениями. Мы называем этот путь из трех узлов ветвью. Разрешается уменьшать ветвь. Если есть требование дальнейшего сокращения, мы делаем корректировку, перемещая несколько поддеревьев с близлежащих позиций.

В дальнейшем рассматриваем только корневые деревья. Дерево состоит из узлов и ветвей, каждая ветвь соединяет два узла. Корень дерева T обозначается через root(T). Линейное дерево размером r является линейным списком из r узлов, так что его первый элемент

рассматривается как корень, и существует ветвь от узла к следующему.

Число потомков узла v называется степенью (рангом) v и обозначается deg(v). Степень дерева T, deg(T), определяется как deg(root(T)). Сумма двух деревьев T и S, обозначенная как S+T, — это просто набор из двух деревьев T и S. Полином деревьев определяется следующим образом. Поскольку произведение является ассоциативным, мы используем обозначение ri для произведения i r раз. Обратите внимание, что deg(ri) = i. r-арный полином деревьев степени k-1 определяется выражением

$$P = \mathbf{a}_{k-1}\mathbf{r}^{k-1} + \ldots + \mathbf{a}_1\mathbf{r} + \mathbf{a}_0 \qquad (1),$$

где ai – линейное дерево размера ai, которое называется коэффициентом в полиноме. Пусть |P| будет количеством узлов в P, и |ai|= ai. Тогда мы имеем |P| = ak-1 rk-1 +… + a1r + a0. Мы выбираем ai как 0 <= ai <= r – 1, чтобы можно было выразить n узлов вышеприведенным многочленом однозначно, так как k-разрядное выражение n уникально с k=⌈logr(n+1)⌉. ai=1 для биномиальных деревьев, ai=2 для 2-3 деревьев, ai=0 если дерево ранга i отсутствует. Член airi называется i-тым членом. Мы назовем ri полным деревом степени i.

Многочлен деревьев рассматривается как совокупность деревьев различных степеней, которые связаны основными ветвями. Определим полиномиальную очередь, r-номинальная очередь – это r-арный полином деревьев с меткой label(v), прикрепленной к каждому узлу v так, что если u является родителем v, то label(u) <= label(v). Биномиальная очередь – это 2-номинальная очередь.

Каждый airi член в формуле (1) – является деревом степени i+1, если ai>1. Одна дополнительная степень обусловлена коэффициентом. Слияние двух линейных деревьев r и s заключается в объединении двух списков по их меткам. Результат обозначается суммой r+s. Слияние двух членов airi и ai'ri – это слияние главных ветвей по их меткам. Когда корни совмещаются, деревья внизу перемещаются соответственно. Если ai + ai' <r, мы имеем объединенное дерево с коэффициентом ai + ai'. В противном случае у нас есть переносное дерево ri+1 и оставшееся дерево с главной веткой длиной ai + ai' – r. Два i-тых члена из обеих очередей объединяются, что приводит к возможному переносу в (i+1)-тые члены.

Вставка ключа в полиномиальную очередь должна объединить одиночный узел с меткой ключа в нулевом члене, занимая O(r*logr n) время возможного распространения переносов на более высокие члены. Таким образом, n вставок образуют полиномиальную очередь P за O(nrlogr n) время. Значение k в формуле (1) равно O(logr n), когда у нас есть n узлов в полиноме деревьев.

Сумма P + Q двух 2-3 пирамид P и Q определяется аналогично полиномиальным очередям. Эти операции с суммами включают вычислительный процесс на основе слияния.

Количество потомков вершины t произвольного дерева T назовем **степенью (degree)** этой вершины, а степенью дерева назовем степень его корня.

2-3 деревьями называются деревья T0, T1, T2, ..., определенные индуктивно. Дерево T0 состоит из единственной вершины. Под деревом Ti понимается дерево, составленное либо из двух либо из трех деревьев Ti-1: при этом корень каждого следующего становится самым левым потомком корня предыдущего (рис.1).

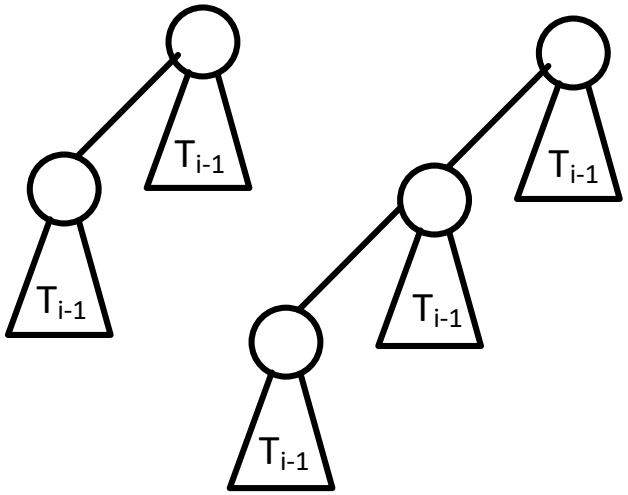

Рис. 1. Структура ветви 2-3 дерева

2-3 пирамида - это массив 2-3 деревьев T[i] (i=0..k) неодинаковой степени, для каждого из которых выполняется основное свойство пирамиды (для минимальной пирамиды ключ в корне не больше ключа потомка), определенных по индукции:

2-3 деревья образуются из одной, двух или трёх веток.

2-3 дерево - сбалансированное дерево, родительский узел которого может иметь как два, так и три сына (Рис. 2). 2-3 деревья являются идеально сбалансированными. Высота дерева из n вершин лежит в диапазоне $[\log 3n; \log 2n]$.

Пример 2-3 пирамиды приведен на рис. 2.

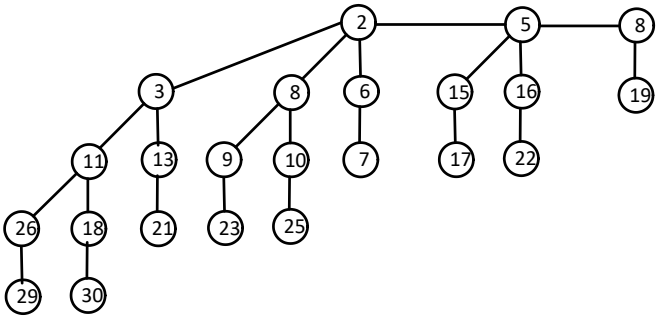

Рис. 2. 2-3 пирамида.

Если мы сохраняем n узлов в пирамиде, то верхняя граница для k задается при 2k-1 = n. Отсюда видно, что k <=⌈log (n+1)⌉.

Для балансировки 2-3 дерева используется понятие рабочего пространства. Пусть размерность узла v будет i-1, тогда веткой будет самое высокое измерение, на котором стоит узел v, это i-тое измерение и пусть v будет не главной ветвью. Мы определяем рабочее пространство для v как набор узлов на i-той ветке, на которой стоит v, (i+1)-тая ветка главного узла

v и другие i-тые ветки, чьи главные узлы находятся на (i+1)-той ветке. Рабочее пространство от 4 до 9 узлов. Пусть главный узел рабочего пространства будет узлом первой позиции в (i+1)-той ветке. Мы также определим tree(u) для u. В указанном выше рабочем пространстве должно быть дерево типа T(i-1) с корнем в u. В данной статье слова «ключ» и «метка» используются взаимозаменяемо.

Обозначим узел с меткой x как узел узел(x) (рис. 2). Размерность узла(6) и узла(7) равна 0, узла(8) равна 1, узла(3) равна 2. Под узлом(2) у нас есть три ветки размерностью 1, 2 и 3. Рабочее пространство для узла(18) это {узел(3), узел(13), узел(21), узел(11), узел(18), узел(30), узел(26), узел(29)}. Главный узел этого рабочего пространства - узел(3). Рабочее пространство для узла (9) имеет более высокую размерность и определятся как {узел(2), узел(8), узел(9), узел(3), узел(11), узел(26)}. Голова узла - узел(2).

Рассмотрим основные операции.

Слияние деревьев. Базовой операцией является операция слияния деревьев. Схемы слияний показаны на рис. 3.

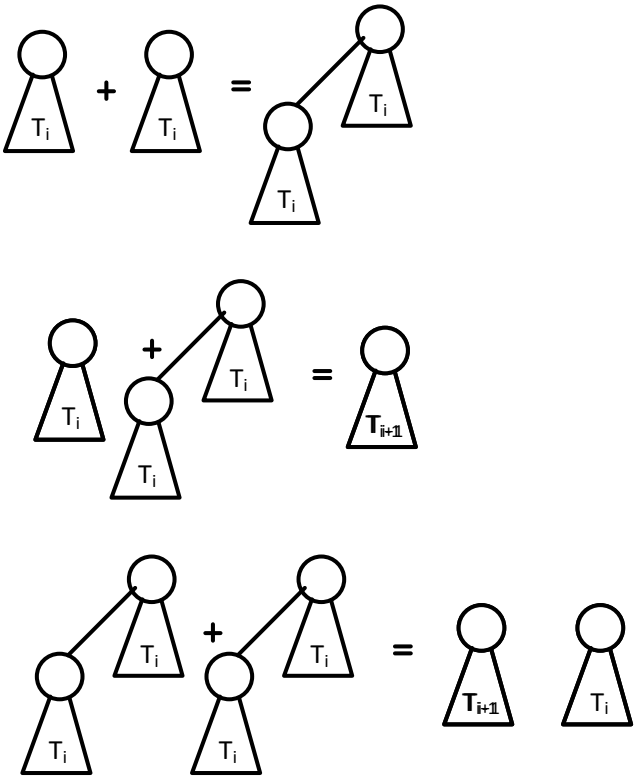

Рис. 3. Схема слияния 2-3 деревьев в 2-3 пирамиду

Вставка узла. Для добавления нового элемента создадим дерево нулевой степени, содержащее одну вершину, и производим операцию добавления этого дерева в пирамиду по следующему алгоритму:

1. Ищем дерево такой же степени или с двумя узлами;
2. В случае отсутствия таких деревьев, вставляемое дерево помещается в корневой список 2-3 пирамиды и завершаем вставку.
3. Сливаем вставляемое и найденное дерево в соответствии с основным свойством пирамид (корень не больше своих потомков) в новое дерево;
4. Рекурсивно проверяем наличие дерева со степенью равной степени нового дерева и сливаем их.

Примеры вставок одного узла приведены на рис.

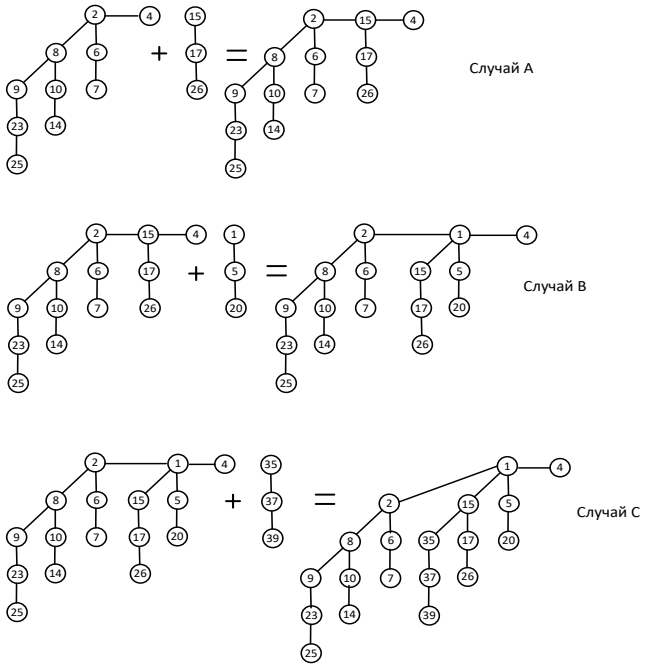

Рис. 4. Три случая вставки 2-3 деревьев в пирамиду

5 при построении 2-3 пирамиды.

Амортизационная сложность для вставки: мы вставляем новый узел в нулевой член на верхнем уровне. Таким образом, амортизационная сложность равна 0, то есть O(1). Фактическое время равно O(log n).

Вставка дерева. Такая вставка называется вставкой верхнего уровня. При добавлении дерева степени i в пирамиду возможны следующие варианты: предположим, что мы вставили деревья типа T(i) в член aiT(i) на верхнем уровне. Имеем три случая:

Случай A. ai = 0. в пирамиде нет дерева степени i, поэтому просто поместим его в правильное положение в пирамиде (рис. 4);

Случай B. ai = 1. Мы можем сформировать новое 2T(i) с одним сравнением. В пирамиде уже существует дерево степени i, тогда извлечем его и соединим с добавляемым, после чего добавим полученное дерево в пирамиду (рис. 4);

Случай C. ai = 2. Мы делаем перенос T(i+1) с двумя сравнениями. Затем переходим к вставке в ai+1T(i+1) (рис. 4).

Построение пирамиды. Построение пирамиды осуществляется путём вставки узла в 2-3 пирамиду и в случае необходимости восстановления её структуры. Пример построения 2-3 пирамиды путём вставки ключей от 1 до 10 приведен на рис. 5 и охватывает практически все возможные ситуации. В принципе 2-3

пирамида может быть построена путём слияния нескольких 2-3 пирамид.

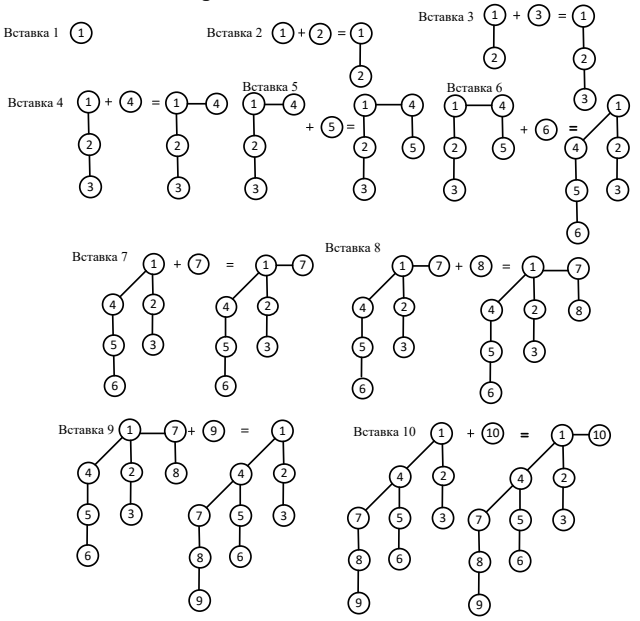

Рис. 5. Построение 2-3 пирамиды вставкой 10 чисел

Слияние пирамид сводится к слиянию корневых списков этих пирамид. Далее проверяется наличие одинаковых деревьев и в случае необходимости их последовательное рекурсивное слияние.

Удаление минимума. Найдём минимум, сканируя корневой список деревьев. Пусть T(i) имеет минимум в корне и Q будет полиномом, полученным из T(i), путем удаления корня. Тогда объединим P-T(i) и Q, т.е. (P-T(i)) + Q.

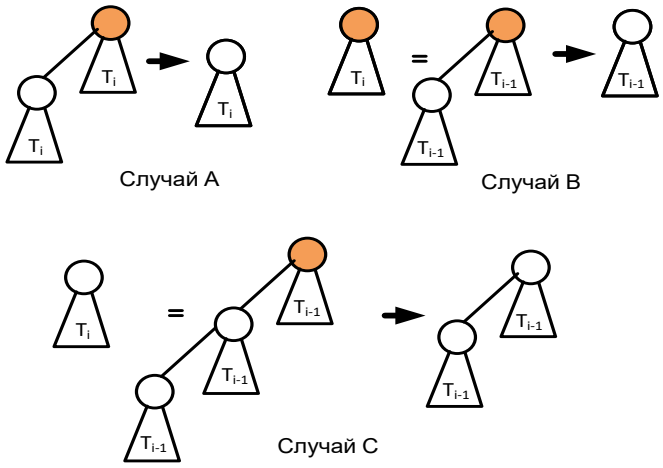

Рис. 6. Три случая удаления дерева

После того, как мы удаляем корень дерева $\mathbf{a}_i T(i)$, у которого минимальный ключ, дерево разбивается на поддеревья $\mathbf{b}_0 T(0)$, …, $\mathbf{b}_i T(i)$, где каждый $\mathbf{b}_j$ равен **1** или **2** для $j = 0, \ldots, i\text{-}1$, ($\mathbf{b}_i$**=1**, если $\mathbf{a}_i$**=2**, и $\mathbf{b}_i$**=0**, если $\mathbf{a}_i$**=1**. Мы сливаем эти деревья с оставшимися деревьями в пирамиде для j = 0, …, *i*. Этот процесс слияния очень похож на сложение двух троичных чисел. В общем случае мы объединяем $\mathbf{a}_j T(j)$, $\mathbf{b}_j T(j)$, и $\mathbf{c}_j T(j)$, где $\mathbf{c}_j T(j)$ – это перенос из $(j\text{-}1)$-той позиции. Если $\mathbf{c}_j$ **= 1**, есть перенос, если $\mathbf{c}_j$ **= 0**, то нет переноса. Существует 18 комбинаций ($\mathbf{a}_j$**,** $\mathbf{b}_j$**,** $\mathbf{c}_j$**).** Кроме симметрии, мы рассмотрим следующие случаи:

- Случай (1, 0, 0). Охватывает вышеупомянутый случай A (рис. 4);
- Случай (**1, 1, 0**). Включает вышеупомянутый случай B(рис. 4);
- Случай (**1, 1, 1**). Проведя три сравнения, мы можем сделать перенос $T(j+1)$.
- Случай (**2, 1, 0**). Включает вышеупомянутый случай C(рис. 4);
- Случай (**2, 1, 1**). Включает вышеупомянутый случай C(рис. 4);
- Случай (**2, 2, 1**). Включает вышеупомянутый случай C(рис. 4);

Случай (**2, 2, 0**). Сравнивая ключи двух корней и помещая меньшие, друг над другом, мы можем создать дерево типа $T(j+1)$, которое является переносом на следующую позицию. Мы производим одно сравнение.

Амортизационная сложность для удаления минимума: требуется не более $\lceil \log(n+1) \rceil$ сравнений для нахождения минимума. После этого мы разбиваем поддеревья под корнем с помощью минимума. Это приводит к потере потенциала максимум $2\lceil \log(n+1) \rceil$. Процесс слияния этих поддеревьев на верхнем уровне занимает $O(\log\ n)$ фактического времени. Амортизационная сложность данного процесса равна 0. Таким образом, амортизационная сложность одной операции по удалению минимума ограничена $3\lceil \log(n+1) \rceil$. Фактическое время равно $O(\log\ n)$.

Удаление дерева: мы не удаляем *tree*(*v*) для узла v, если он является корнем на верхнем уровне. Предположим, что мы удаляем *tree*(*v*) типа $T(i\text{-}1)$ для узла v. Рассмотрим 2 случая, когда размер рабочего пространства больше 4 и когда он равен 4. Начнем с первого случая. Пусть *i*-тая ветка узла *v* будет (*u, v, w*) или (*u, w, v*). Тогда удалим *tree*(*v*) и сожмем ветку. Если ветка равна (*u, v*), тогда удалить *tree*(*u*). И мы потеряем эту ветку. Чтобы предотвратить эту потерю, мы перемещаем несколько деревьев типа $T(i\text{-}1)$ в рабочем пространстве.

Пусть размер рабочего пространства равен 4 узлам. Удалите *tree*(*v*) и переставьте остальные три узла таким образом, что два из них попадут под главный узел рабочего пространства для формирования i-той ветки. Затем мы восстанавливаем i-тую ветку длиной 2, то есть из 3 узлов, но теряем $(i+1)$ ветку. Наша работа переходит в более высокое измерение, если оно существует. Иначе, стоп. Во время балансировки пирамиды, необходимо сравнивать значения ключей узлов, чтобы сохранить пирамидальную упорядоченность при размещении деревьев.

Уменьшение ключа. Предположим, что ключ узла *v* был уменьшен. Если *v* не на верхнем уровне, удалим *tree*(*v*) и вставим его в *j*-тый член на верхнем уровне, с помощью нового ключа, где *j* – это размерность *v*. Если *v* на верхнем уровне, мы ничего не делаем после уменьшения ключа.

Сначала подробно опишем операцию уменьшения ключа. Предположим, мы выполняем

уменьшение ключа в узле v размерности *i*-1. После уменьшения значение ключа v, мы удаляем *tree*(*v*) и затем вставляем *tree*(*v*) на верхний уровень (корневой список). Для удаления дерева давайте классифицировать ситуацию, используя части структуры дерева. На следующих рисунках (рис. 7-15) показано только рабочая область. Каждый узел может рассматриваться как дерево той же степени, по общим соображениям. Левая и правая стороны до и после преобразования. Ветки собираются слева внизу от ветки *i*-той размерности (*i*-тая ветка для краткости), в одной из которых стоит v, а то, что справа внизу называется веткой (*i*+1)-ой размерности. У нас на рисунках есть 2 или 3 ветки *i*-той размерности. Удаляя узел и сжимая ветку у нас, будет свободная позиция, которая может или не может быть отрегулирована. Проверяя эти ветки, мы классифицируем ситуацию на несколько случаев, в зависимости от размера *w* рабочей области. На следующих рисунках *i*-тые ветки расположены в неубывающем порядке длин для более простого объяснения. Мы называем это стандартной схемой. Другие случаи похожи.

Случай 9. *w* = 4. Все ветки рабочего пространства определены с помощью *i*-того и (*i*+1)-той размерности, имеющего один потенциал. Мы делаем ветвь *i*-той с потенциалом 3 для главного узла, и ситуация становится связанной с потерей узла (*i*+1)-той ветке. Структура будет сделана в рабочем пространстве и определена посредством (*i*+1)-той и (*i*+2)-той ветками. Процесс балансировки может повторяться несколько раз и остановиться в одном из случаев 1-8 или, если нет более высокой ветки.

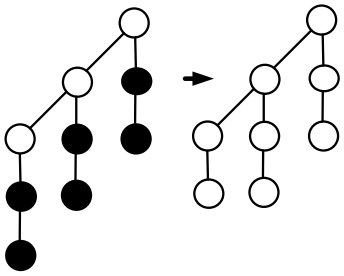

Рис. 7. Случай 1. *w*=9. Удаление любого из шести черных узлов принесет такую же новую форму в пределах стандартной схемы.

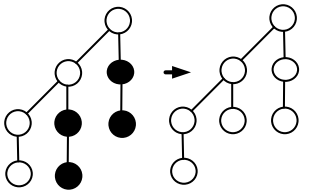

Рис. 8. Случай 2. *w* = 8. Первый случай *w* = 8 аналогичен случаю 1. Мы не делаем сравнений.

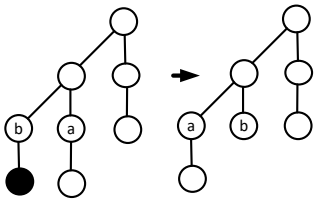

Рис. 9. Случай 3. *w* = 8. В этом случае мы можем переставить пирамиду в рабочей области без сравнения. Для визуализации процесса метки *a* и *b* приведены на рисунках. В некоторых случаях подобные перестановки сделаны.

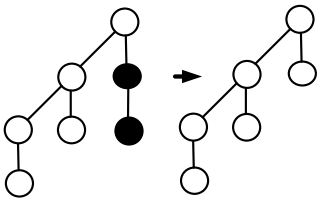

Рис. 10. Случай 4. *w*=7. Мы не производим сравнения.

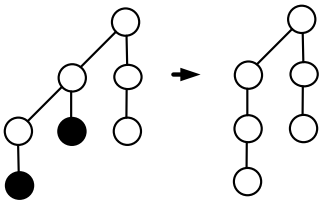

Рис. 11. Случай 5. *w* = 7. Мы производим не более одного сравнения.

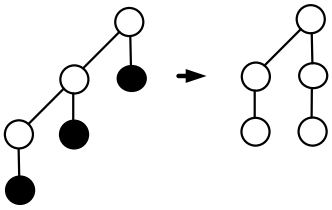

Рис.12. Случай 6. *w* = 6. Мы производим не более одного сравнения.

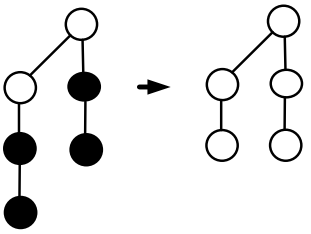

Рис.13.Случай 7. w=6. Другой случай при w=6

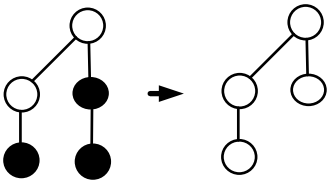

Рис. 14. Другой случай 8 при w = 5

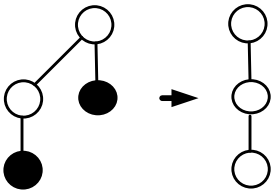

Рис.15. Случай 9 при w=4

В пирамиде на рис. 2, мы называем узел с ключём *x* как *узел*(*x*). Предположим, мы уменьшаем ключи с 6 до 4, с 29 до 14. Тогда мы имеем следующее $T = \mathbf{1}T(3) + \mathbf{1}T(0)$.

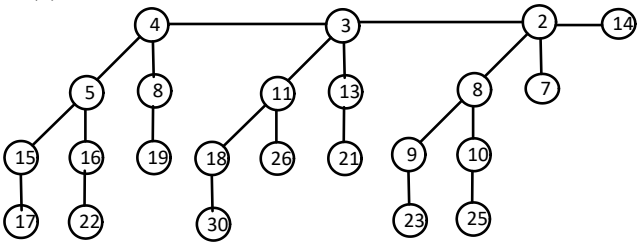

Рис. 16. 2-3 пирамида после двух операций уменьшения ключа

Сначала мы удаляем узел(6), вызывая перемещения узла(7) в положение потомка узла(2). Новый узел(4) вставляется в 2*T*(0), в результате чего *T*(1) с *узлом*(8) и *узел*(19) переносятся на 2*T*(1), по причине вставки. Тогда вновь сформированный *T*(2) будет перемещен в 2*T*(2), в результате чего образуется

T(3). Для второго уменьшения ключа у нас есть узел(14). Так как ветка больше не сужается, две ссылки от *узла*(11) до *узла*(18) и *узел*(26) меняются местами (рис. 16).

На рисунке 17а удаленный узел обозначен черным кружком на первой картинке. У нас есть два случая при $w = 4$ с последующим случаем при $w = 6$, результаты на рисунке 17б.

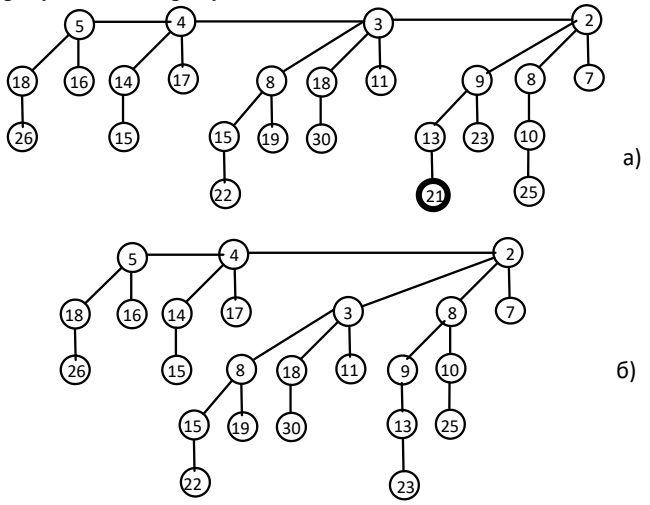

Рис. 17. Изменение 2-3 пирамиды в случае 9 (с 4 узлами)

Амортизационная сложность для уменьшения ключа. Для уменьшения значения ключа требуется время $O(1)$. После этого мы выполняем различные операции, описанные выше, расходы которых ограничены 2. Амортизационная сложность для случая 9 и случаев A, B, C выше 0. Таким образом, амортизационная сложность для этой части составляет 2.

Реализация. Вставка будет разделена на два блока: интерфейс вставки и слияние. Интерфейс вставки будет отвечать за выборку двух сливаемых деревьев (вставляемое дерево + дерево, занимающее позицию), и обработку результата модуля слияния. В случае, когда позиция свободна модуль будет размещать дерево в нужной ячейке, в обратном же случае он будет отдавать управление модулю слияния. Модуль слияния отвечает за выбор алгоритма слияния, и приведение его в исполнение. На выход будут отдаваться ссылки на вставленное дерево (если нет, то NULL), и дерево, которое необходимо вставить (если нет, то NULL). Второй случай возникает тогда, когда в результате слияния степень дерева изменилась.

Если мы выполним n операций вставки, n удалений минимума и m операций по уменьшению ключа, то общее время теперь ограничено $O(m + n \log n)$. С точки зрения количества сравнений, у нас есть $2m + 3n \log n$ через амортизационный анализ, описанный выше. Таким образом, мы можем решить проблему кратчайшего пути из одного источника и проблему минимальной сложности связующего дерева за время $O(m + n \log n)$.

Структуру данных 2-3 пирамиды можно реализовать с помощью указателей. Для деревьев на верхнем уровне мы готовим массив указателей размером $k = \lceil \log(n+1) \rceil$, который мы назовем массивом верхнего уровня. i-тый элемент массива указателей указывает на корень дерева i-того члена при $i = 0, \ldots, k - 1$, если член существует. В противном случае он нулевой. Пусть верхняя ветка узла v будет i-той. Структура данных для узла v состоит из целочисленных переменных *key* и $dim = i$, указателя на голову i-той ветки и массива размером k, элементами которого являются пары (*Второй, Третий*). Мы называем такой массив – массивом узлов. *Второй* из j-того элемента массива указывает на корень второго узла j-той ветки узла v. *Третий* – это третий узел. Пока *второй* не нулевой, третий может быть нулем, в этом случае нет третьего узла на данной ветке. С такой структурой данных мы можем исследовать рабочее пространство для v. Предположим, мы выполняем уменьшение ключа узла v размерности i. Изучая i-тую и $(i+1)$-тую ветки с помощью массива, мы можем знать, в каком случае пространство падает. Если мы подготовим фиксированный размер k для массивов всех узлов, нам понадобится $O(n \log n)$ пространства.

Мы можем реализовать нашу структуру данных с пространством $O(n)$, как это сделано в пирамиде Фибоначчи, хотя эта версия будет менее экономичной. Мы используем тот же самый массив верхнего уровня. Узел немного отличается. В частности, узел v, размерности i состоит из переменных, похожих на приведенную выше версию, за исключением массива узлов. Теперь он заменен указателем на второй узел ветки, самой большой размерности i. Второй и третий узел, если такие имеются, указывают на каждый другой. Вторые узлы имеют родительские указатели на v. Из каждого третьего узла мы можем перейти к главному узлу, косвенно, через второй узел. Вторые узлы образуются двусвязной круговой структурой, которая обеспечивает время $O(1)$ для вставки и удаления, а также перехода из j-той ветки на $(j+1)$-тую ветку для $1 <= j <= i-1$ за O(1) время.

Заключение.

Измерение сложности n удалений минимумов, n вставок, m уменьшений ключей по количеству сравнений показало, что оно составляет в 2-3 пирамиде $O(m + n \log n)$. Анализ показывает, что количество сравнений ограничено $2m + 3n \log n$. Чтобы сравнить 2-3 дерево и пирамиду Фибоначчи, мы можем модифицировать последнюю в тех же рамках, что и первое, то есть, ограничение операций на удаление минимума, вставку и уменьшение ключа. В оригинальной статье [3], количество деревьев в пирамиде может быть поднято до n. Когда выполняется операция по удалению минимума, деревья одного ранга (степень в данной статье), объединяются путем сравнения корней. В этом процессе минимум также найден. Чтобы изменить пирамиду Фибоначчи, мы можем ограничить количество корней до 1.44 log n, и чтобы найти минимум, мы можем отсканировать корни. В этой версии мы можем показать посредствам амортизационного анализа, что количество сравнений ограничено примерно 2m + 2.88n log n, где мы берем

одну единицу сложности для отмеченного узла. В [3] они взимается 2 единицы за отмеченный узел для учета сложности уменьшения каскадных вычислений. Это принесло бы член 3m, вместо 2m в вышеуказанной сложности. В конце концов пирамида Фибоначчи и 2-3 пирамида находятся почти на одном уровне при анализе количества сравнений в худшем случае. Фактическая реализация обеих пирамид для алгоритма Дейкстра для некоторого класса случайных графов показала, что 2-3 пирамида лучше, чем пирамида Фибоначчи примерно на 20% как в количественном сравнении, так и по процессорному времени. Причина в том, что у нас есть несколько случаев, когда амортизационная сложность уменьшения ключа равна 0 или 1, а не 2. Также при удалении минимума мы уменьшаем потенциалы ветвей под минимальным узлом, но потеря составляет 1 или 2, не всегда 2, на каждой из ветвей.

Метод балансировки пирамиды при удалении узла не является уникальным. Чтобы утверждать эффективность этого подхода, необходимы дополнительные исследования. Можно определить аналогичные пирамиды, такие как 2-3-4 пирамиды, 3-4 пирамиды и т.д. и операции над ними. Вопрос о том, является ли производительность этих пирамид лучше, остается открытым.

СПИСОК ЛИТЕРАТУРЫ

1. Гулаков В. К., Гулаков К. В. Монопирамидальные структуры данных. – М.: Горячая линия –Телеком, 2019. –148 с.: ил.
2. Driscoll, J.ft., H.N. Gabow, ft. Shrairman, and R.E. Tarjan, An alternative to Fibonacci heaps with application to parallel computation, Comm. ACM, 31(11) (1988) 1343-1345.
3. Fredman, M.L. and R,E, Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Jour. ACM 34 (1987) 596-615
4. Takaoka, T.. Theory of 2-3 Heaps.— Cocoon (1999), p.41-50

2-3 Heap

Konstantin V. Gulakov

Abstract - Heap data structures have recently received significant development and are very diverse. 2-3 heap is one of the representatives of the pyramids at the base of the forest of trees. The advantage of a 2-3 tree is that the hard height constraint log(n) is compensated by balancing in width (horizontal balancing). This allows to have a minimum tree height and thus increase the performance of some operations. Tadao Takaoka proposed a pyramidal modification of these trees and combining them into a 2-3 heap. The study of this work leaves many questions on the implementation of 2-3 heaps and operations on it. While the merits of this structure are obvious, it is possible that these difficulties prevented its popularity among programmers. In the present work, a complete understanding of this structure is carried out, primarily in terms of its implementation and operations on it, which fills the gap in the Russian-language literature.

Keywords: trees, 2-3 heap, trees operations, algorithm complexity.

REFERENCES

1. Gulakov V. K., Gulakov K. V. Monopiramidal'nye struktury dannyh. – M.: Gorjachaja linija –Telekom, 2019. – 148 s.: il.
2. Driscoll, J.ft., H.N. Gabow, ft. Shrairman, and R.E. Tarjan, An alternative to Fibonacci heaps with application to parallel computation, Comm. ACM, 31(11) (1988) 1343-1345.
3. Fredman, M.L. and R,E, Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Jour. ACM 34 (1987) 596-615
4. Takaoka, T.. Theory of 2-3 Heaps.— Cocoon (1999), p.41-50