Дрожащая пирамида

В.К. Гулаков, К.В. Гулаков

Аннотация - Статья посвящена одной из разновидностей пирамидальных структур данных, широко используемых при решении различных задач. Обсуждаются вопросы повышения производительности программных средств за счёт применения эффективных пирамидальных структур данных. Краткий обзор развития знаний по пирамидальным структурам показывает закономерность появления таких структур как дрожащая пирамида, которая позволяет выполнять основные операции над данными с минимальной сложностью. В статье дается подробное описание структуры дрожащей пирамиды и приведены примеры выполнения операций над ней и их сложность. Указывается на особенность повышения эффективности за счёт использования соответствующего коэффициента. Сравнение теоретической сложности операций над различными пирамидальными структурами с операциями над дрожащей пирамидой указывает на то, что последняя не уступает по эффективности большинству популярных и эффективных пирамид, но отличается относительной простотой и возможностью после многочисленных операций восстанавливать свою эффективную структуру. В статье намечены пути её экспериментального исследования в различных ситуациях и практических задачах.

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

I. Введение

Связь структур данных и алгоритмов в первую очередь влияет на производительность программ, увеличивая их эффективность на порядки. Анализ основных операций алгоритма позволяет выбрать наиболее подходящие из известной структуры данных или предложить оригинальные. Пирамидальные структуры данных – вторые по важности после деревьев поиска. В основном они применяются в алгоритмах, где часто используются операции: найти минимум, удалить минимум, вставить, удалить, изменить элемент структуры, объединить структуры. К таким задачам и соответственно алгоритмам можно отнести различные варианты сортировок, алгоритмы на графах, алгоритмы в параллельном программировании, многочисленные задачи оптимизации, задача оценки релевантности в информационном поиске и т.д.

В литературе описано множество пирамид, с различными характеристиками. В зависимости от временных ограничений их можно разделить на две основные категории: в худшем случае или амортизационном случае. Большинство пирамид основаны на пирамидо-упорядоченных деревьях, то есть древовидные структуры, где элемент, хранящийся в узле, имеет ключ не меньше, чем ключ родителя (минимальная пирамида). Пирамидо-упорядоченные деревья дают реализацию пирамид, позволяющую достичь логарифмической сложности для всех операций. Сначала это были неявные бинарные пирамиды Вильямса [21], левосторонние пирамиды Крейна [6], измененные Кнутом, и биномиальные пирамиды Вилемина.

Появление пирамид Фибоначчи было прорывом, поскольку достигнута амортизационная сложность O(1) для всех операций, кроме удаления и удаления минимума, которые имеют амортизационную сложность O(lgn), где n-количество элементов в пирамиде и lg на базе двоичного логарифма. Недостаток пирамид Фибоначчи заключается в том, что они сложнее, по сравнению с существующими решениями и не так эффективны на практике, как другие, теоретически менее эффективные решения. Таким образом, пирамиды Фибоначчи открыли путь к дальнейшему развитию пирамид, и с тех пор были предложены многие решения исходя из амортизационного подхода, пытаясь соответствовать по временной сложности пирамиде Фибоначчи, будучи в то же время проще и эффективнее на практике. Парная пирамида, предложенная Фридманом, Седжвиком, Слетором и Тарьяном, была амортизационной версией биномиальной пирамиды, и в то же время позволяя достигать сложности как у пирамиды Фибоначчи, за исключением опять же операции уменьшения ключа, временная сложность которой оставалась неизвестной в течение многих лет. В 1999 году, Фридман доказал, что нижняя граница для операции уменьшения ключа в парной пирамиде есть Ω(lglgn); таким образом, амортизационная сложность операций над парной пирамидой не соответствует амортизационному исполнению пирамиды Фибоначчи. В 2005 году Питти доказал, что сложность уменьшения ключа имеет сложность 2O(√lg lg n). Позже, Элмасри предложил вариант парной пирамиды, который для

операции уменьшения ключа требует O(lglgn) амортизационного времени.

Пирамиды, имеющие амортизационную сложность, сопоставляются с временной амортизационной сложностью пирамиды Фибоначчи. В частности, Дрисколем, Габоу, Шраирманом и Тарьяном [8], были предложены рангово-релаксационные пирамиды, Каплан и Тарьян [17] представили тонкие пирамиды, Чан [7] предложил дрожащие пирамиды, Хауплер, Сен и Тарьян ввели рангово-парные пирамиды [14], и Элмасри предложил пирамиды с нарушениями [11]. Элмасри усовершенствовал количество сравнений пирамиды Фибоначчи до постоянного фактора [9], а также рассмотрел варианты парной пирамиды, косой пирамиды и косой парной пирамиды [10]. Некоторые исследователи стремились достичь амортизационной границы пирамиды Фибоначчи более простым способом, двигаясь в разных направлениях. Петерсон [20] представил структуру, основанную на avl деревьях, а Хойер [15] представил несколько структур, в том числе на основе красно-черных деревьев, avl деревьях, и (a,b)-деревьях.

Рассмотрим теперь прогресс по этой проблеме, исходя из наихудшего случая. Цель в худшем случае это эффективные пирамиды, чтобы исключить непредсказуемость амортизированных пирамид, поскольку эта непредсказуемость вредна, например, в приложениях реального времени.

сложностью в худшем случае. Такой же результат был позже достигнут Капланом и Тарьяном [16] с толстыми пирамидами. Операция слияния была следующей целью для достижения постоянного времени в худшем случае для того, чтобы соответствовать временной сложности пирамид Фибоначчи. Бродал [3] достиг O(1) временной сложности в худшем случае для операции слияния на машине с указателями, но не для операции уменьшения ключа. Хотя операция уменьшения ключа и операция слияния могут быть достигнуты с O(1) временной сложностью в наихудшем случае отдельно, очень трудно достичь постоянного времени для обеих операций в одной структуре данных. Бродалу [4] удалось решить эту проблему, но его решение является очень сложным и требует использования (расширенных) массивов. Для моделей вычислений с указателями, проблема согласования временных границ пирамид Фибоначчи оставалась открытой, и прогресс в наихудших рамках был достигнут в других направлениях. В частности, Элмсри, Йенсен и Катайайнен представили двухуровневые релаксационные пирамиды [13], в которых количество сравнений ключей сводится к lgn + 3lglgn+ O(1) для временной сложности в операции удаления. Они также представили [12] новую идею для обработки операции уменьшения ключа путем введения структур с нарушениями, не нарушая порядка пирамиды.

Дрожащая пирамида (quake heap) [1,7,19] занимает такое же место среди всех пирамид, как и

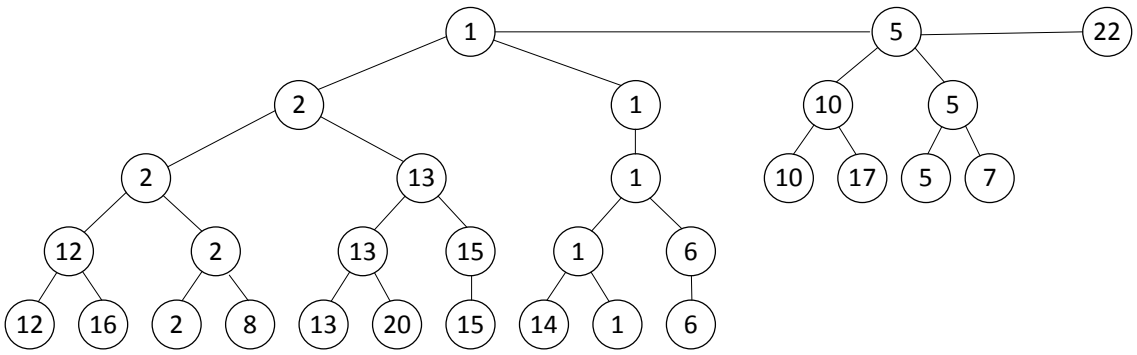

Рис.1. Пример дрожащей пирамиды, состоящей из трех турнирных деревьев

Целевые установки для наихудшего случая были даны пирамидой Фибоначчи, т.е. временные ограничения пирамид Фибоначчи в идеале должны быть подобраны в самом худшем случае. Следующие усовершенствования после биномиальных пирамид сделаны с неявными пирамидами Карлсона, Мунро и Поблете [5] поддерживающие в худшем случае O(1) временную сложность для вставок и O(lgn) для удалений на простой пирамиде, хранящейся в виде массива. Потоково-релаксационные пирамиды [8] достигают амортизационной границы, заданные пирамидой Фибоначчи в худшем случае, за исключением операции слияния, которая реализуется с O(lgn) временной пирамида Фибоначчи. Она состоит из коллекции турнирных деревьев (Рис.1).

Турнирное дерево является корневым деревом со следующими свойствами:

1. Все элементы множества представлены в листьях этого дерева.
2. Все листья имеют одинаковую высоту, т.е. все пути от корня к листу имеют одинаковую длину.
3. Все внутренние узлы имеют одного или двух потомков.
4. Элемент каждого внутреннего узла определяется как минимальный элемент его потомков.

В турнирных деревьях удобно считать, что меньший потомок в каждом узле всегда левый потомок и, если узел имеет ранг 1, его единственный потомок является левым потомком. Корни всех деревьев объединены в упорядоченный список. Доступ к пирамиде осуществляется ссылкой на первый корень в списке корней.

Каждый узел содержит:

- ключ элемента;
- указатель на родителя узла;
- указатель на левого потомка узла;
- указатель на правого потомка узла;
- высота узла.

II. Основные операции над пирамидой и их сложность.

Для выполнения основных операций над дрожащей пирамидой используются две процедуры: связывания (Link) и вырезания (Cut).

Процедура Link - связывает две пирамиды в одну. Сначала объединим корневые списки пирамид в один корневой список, поддерживая упорядоченность по высоте. Храним по указателю на начало списков и в результирующий список записываем минимальный из них, откуда только что записали, сдвигаем на следующий. Далее проходим от начала до конца новый полученный корневой список и сливаем деревья одинаковой высоты в одно турнирное дерево. Могут быть случаи:

1) Только 2 дерева одинаковой высоты. Тогда объединяем их.

2) 3 дерева одинаковой высоты. Объединяем 2 последних. Пример объединения двух деревьев представлен на Рис. 2.

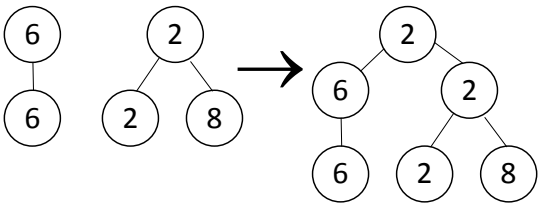

Рис. 2. Объединение двух деревьев высоты 2

Процедура вырезать (Cut) получает указатель на такой узел X, в котором X отличается от ключа в родительском узле. Процедура убирает связь между X и его родительским узлом. Пример вырезания узла с ключом 2 представлен на Рис. 3.

Операция **create** - создать пустую пирамиду. Создается пустой список root list. Сложность операции O(1).

Операция Insert. Вставка элемента может быть сделана тривиально, создав новое дерево размера 1. Число деревьев в коллекции увеличивается на 1, но может быть уменьшено путем объединения в удобное время позже (Рис. 4). Сложность алгоритма вставки O(1).

Операция **find-min** найти минимум в пирамиде. Очевидно, минимум находится в корневом списке, то есть, чтобы его найти нужно пройтись по корневому списку. Сложность: O(root_list.length) = O(log n).

Операция **Decrease-key(X,K)** – уменьшает ключ X в элементе на K. Рассмотрим высший элемент, который хранит ключ X. Было бы слишком дорого, чтобы обновить элементы всех предков X. Вместо этого мы вырезаем поддерево с корнем в точке X. После этого уменьшим величину X в отдельном новом дереве.

Теперь мы должны обратиться к одному ключевому вопросу: после многих операций уменьшения и вставки турнирные деревья могут вырождаться. Пусть ni обозначает число узлов на высоте i. (В частности, n0 = n = | S |.). Поддерживается следующее условие с помощью некоторой фиксированной постоянной $\alpha \in (1/2, 1)$:

$$n_{i+1} \leq \alpha n_i \quad (1)$$

Для примера мы можем установить $\alpha = 3/4$. Из рассматриваемого условия (1) следует, что максимальная высота будет не больше, чем $\log 1/\alpha n$. Когда условие (1) нарушается по i, происходит "дрожащее" событие, и мы удаляем все с высоты i+1 и поднимаемся, чтобы снова перестроить дерево (Рис. 5).

Проверка условия (1) представлена для дрожащей пирамиды на Рис. 6. Есть два дерева высотой 0, одно высотой 2 и дерево высотой 4. Нулевой уровень имеет 11 узлов (n0=11), уровень 1 имеет 6 узлов (n1=6), уровень 2 имеет 4 узла (n2=4), 3-ий уровень имеет 2 узла (n3=2), а 4-ый уровень имеет 1 узел. Условие (1) выполняется на всех уровнях.

Интуитивно кажется что, события большой "величины" (то есть, события на малых высотах i) должны происходить редко.

Операция delete min. Мы просто удаляем путь с узлами, которые хранят минимальный элемент. Число деревьев в наборе растет, и в это время можно сделать повторяющие операции связывания, чтобы сократить

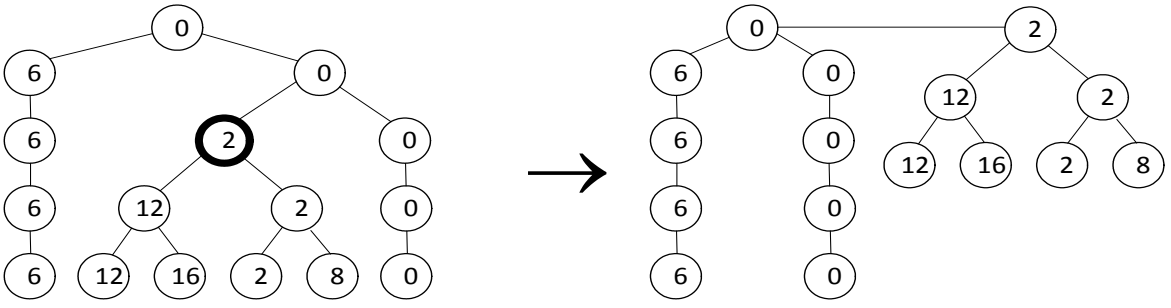

Рис. 3. Процедура вырезания элемента с ключом 2

количество деревьев. А именно, каждый раз, когда есть два дерева одинаковой высоты, мы связываем их.

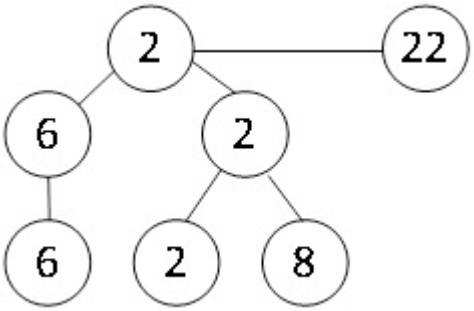

Рис. 4. Пример операции вставки элемента с ключом 22

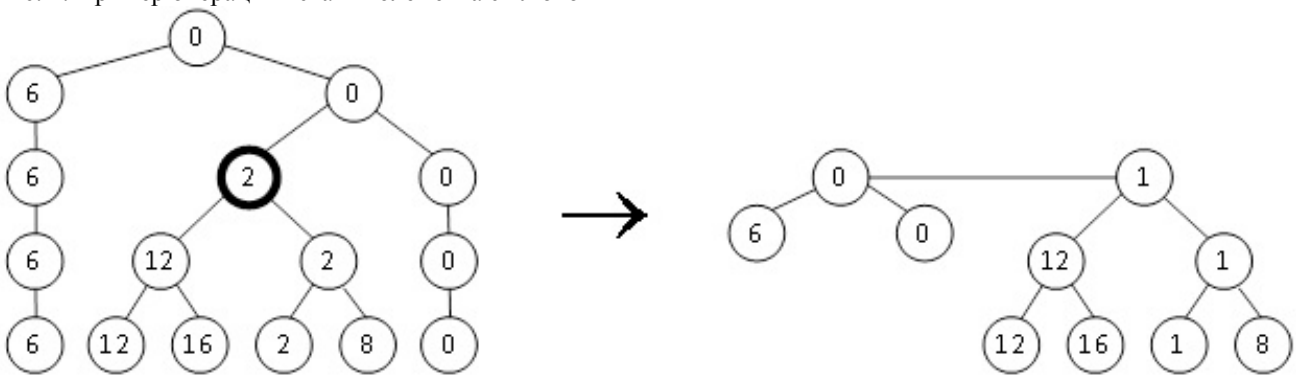

Рис. 5. Пример операции decrease-key(2,1)

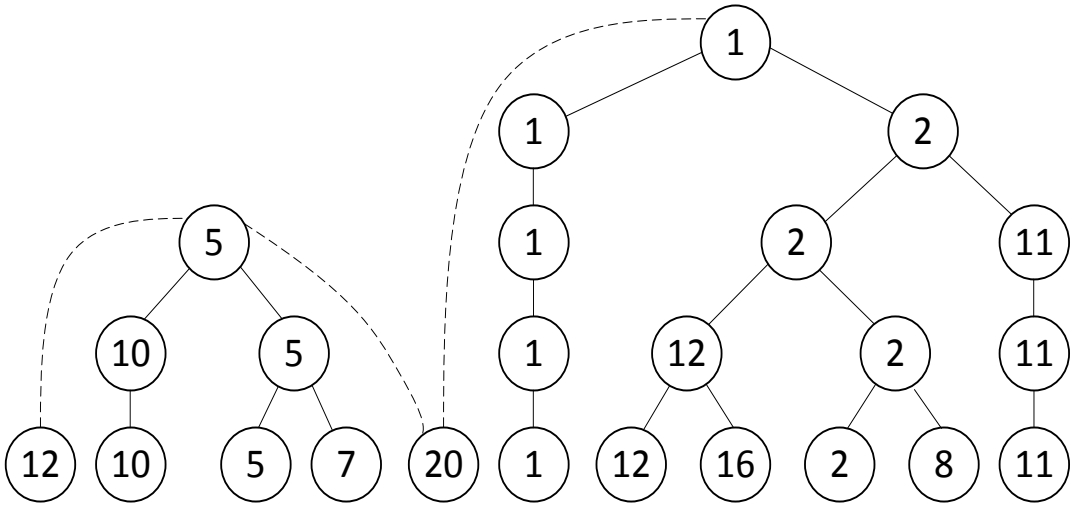

Рис. 6. Дрожащая пирамида с четырьмя турнирными деревьями.

Для эффективной реализации delete-min необходимо иметь дополнительную информацию о дрожащей пирамиде. В дополнение к турнирному дереву мы вводим 2 массива:

Массив T. Запись T[i] хранит список листьев всех деревьев с высотой i.

Массив n. Запись n[i] хранит число узлов с высотой i.

Массивы T и n поддерживаются в актуальном состоянии с константными затратами. Кроме того, после каждой операции используется свойство (1).

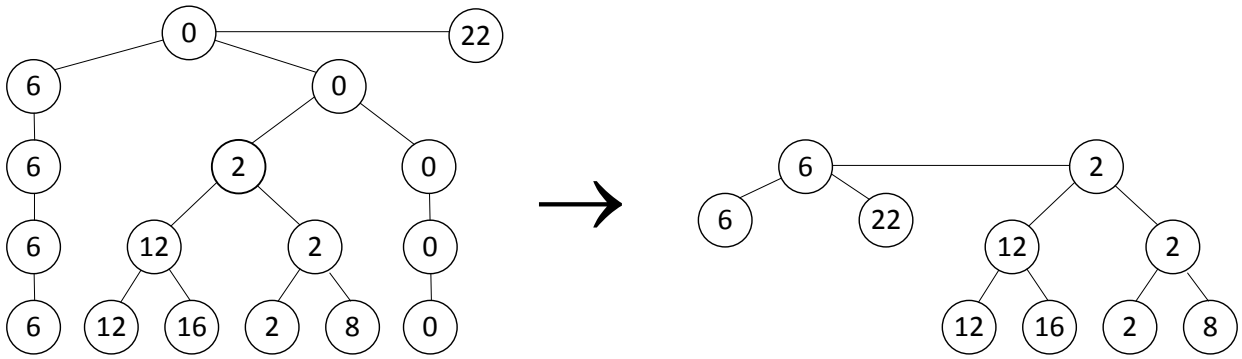

Рис. 7. Пример операции delete-min

1. Нахождение минимального элемента. Перебор всех корней и нахождение элемента x с минимальным ключом.

2. Удаление минимального элемента. Удаление пути, соответствующего x. (получаем несколько новых деревьев)

3. Пока есть 2 дерева одинаковой высоты, соединяем 2 дерева (уменьшаем количество деревьев на 1)

4. Дрожание. После выполнения операций Insert и decrease-key проверяем свойство (1). Найти на всех уровнях самый низкий уровень i, для $n_{i+1}>3/4*n_i$ (нарушение свойства (1)). Если уровень i существует, удалить все узлы с большей высотой, чем у i. Иначе, ничего не делать.

Операции insert(x) и decrease-key() в трясущейся пирамиде выполняются за амортизационное время O(1), а операция delete-min() выполняется за время O(logn).

Операция delete - удалить произвольный элемент. Сначала уменьшим значение в вершине до минимально возможного значения. А затем удалим минимальный элемент в пирамиде.

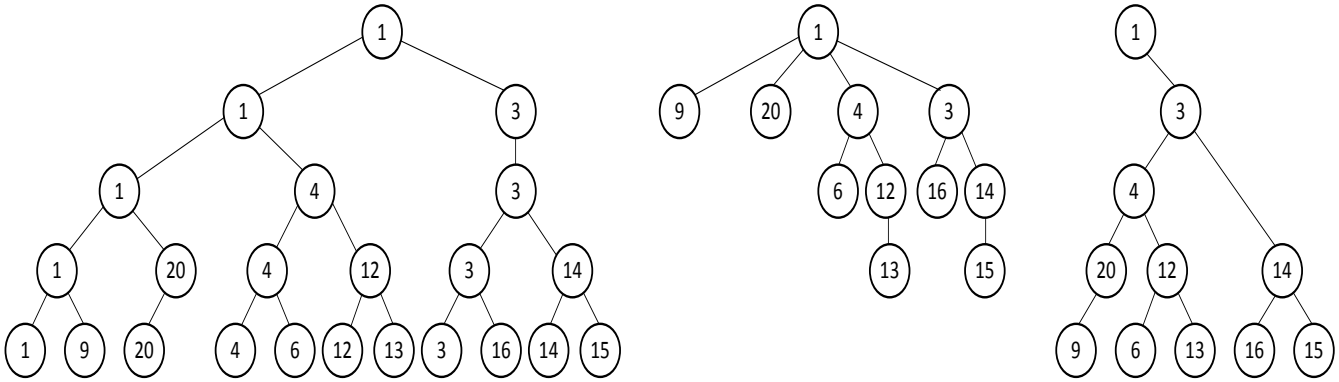

Рис. 8. Преобразование турнирного дерева в пирамидо- упорядоченное дерево или полуупорядоченное дерево.

Сложность: O(log n) + O(log n) = O(log n)

Как и пирамида Фибоначчи, дрожащая пирамида может легко поддерживать слияние. Сложность операции O(1). Затрачивается время только для конкатенации списков деревьев.

Возможны разные варианты применения слияния. Слияние равных по высоте деревьев может быть сделано в любое время, например, сразу после вставки или после удаления минимума, не затрагивая амортизационной сложности. Кроме того, мы можем выполнять слияние деревьев до тех пор, пока их количество превышает O(log(n)), где n – число узлов в пирамиде.

Турнирные деревья требуют линейное число дополнительных узлов, но для эффективного использования памяти возможны другие варианты представления турнирного дерева, где каждый элемент хранится только в одном узле. Одним из вариантов является преобразование каждого турнирного дерева T в пирамидо- упорядоченное дерево T' ранга O(log n): потомок x в T' есть правый потомок всех узлов хранения x в T. (рис. 8 в центре). Биномиальные пирамиды и пирамиды Фибоначчи, как правило, описываются деревьями этого вида.

Другой вариант заключается в преобразовании T в бинарное дерево T" следующим образом: после быстрого сокращения одноранговых узлов в T, правый потомок x в T" является правым потомком самого высокого узла хранения x в T; левый ребенок x в T" это брат правого потомка самого высокого узла хранения x в T. (рис. 8 справа). Полученное дерево T" это полуупорядоченное бинарное дерево: значение каждого узла x меньше, чем значение любого узла в правом поддереве x. Хойер [15] выступал за использование таких деревьев при реализации.

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

Таблица 1. Асимптотическая сложность операций

	Длина логических строк кода	Операции						
		Create	Find_min	Insert	Meld	Decrease	Delete	Delete_min
Бинарная пирамида (неявная)	184	O(n)	O(1)	O(logn)	O(nlogn)	O(logn)	O(logn)	O(logn)
Биномиальная пирамида (неявная)	317	O(1)	O(logn)	O(logn)	O(logn)	O(logn)	O(logn)	O(logn)
Фибоначчиева пирамида (неявная)	282	O(1)	O(1)	O(1)	O(1)	O(lglgn)	O(logn)	O(logn)
Парная Пирамида (неявная)	186	O(1)	O(1)	O(1)	O(1)	O(lglgn)	O(logn)	O(logn)
Рангово-парные пирамиды	376	O(1)	O(1)	O(1)	O(1)	O(lglgn)	O(logn)	O(logn)
Пирамида Бродала	824	O(1)	O(1)vv	O(1)	O(1)	O(1)	O(logn)	O(logn)
Слабые рангово-релаксационные пирамиды	638	O(1)	O(1)	O(1)		O(1)	O(logn)	O(logn)
Дрожащая пирамида	383	O(1)	O(log n)	O(1)	O(1)	O(1)	O(logn)	O(log n)
Пирамида с нарушениями	481	O(1)	O(1)	O(1)	O(1)	O(1)	O(logn)	O(logn)

Сравнение асимптотической сложности операций для наиболее характерных пирамид (таблица 1) показывает, что эти показатели у дрожащей 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 дрожащих пирамид является отдельной задачей. Некоторые попытки были сделаны в работе [18]. Однако, эта работа требует продолжения с целью выработки рекомендаций по применению этой структуры данных в различных алгоритмах.

СПИСОК ЛИТЕРАТУРЫ

1. Гулаков, В.К. Многомерные структуры данных. Монография / В.К. Гулаков, А.О. Трубаков – Брянск: БГТУ, 2010. – 387 с.

2. Brodnik. A., Lopez-Ortiz, A., Raman, V., Viola, A.. Space-Efficient Data Structures, Streams, and Algorithms / Springer Heidelberg Dordrecht London NewYork 2013 Library of Congress Control Number: 2013944678, 363p
3. Brodal, G. S.. Fast meldable priority queues. In Proc. 4th International Workshop Algorithms and Data Structures, volume 955 of Lecture Notes in Computer Science, Springer, 1995. p. 282–290.
4. Brodal, G. S.. Worst-case efficient priority queues. In Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 1996. p 52–58.
5. Carlsson, S., Munro J. I., Poblete P. V.. An implicit binomial queue with constant insertion time. In Proc. 1st Scandinavian Workshop on Algorithm Theory, volume 318 of Lecture Notes in Computer Science, Springer, 1988. p. 1–13.
6. Crane, C. A. Linear lists and priority queues as balanced binary trees. PhD thesis, Stanford University, Stanford, CA, USA, 1972.
7. Chan, Timothy M.. Quake Heaps: A Simple Alternative to Fibonacci Heaps Manuscript, 2009.
8. Driscoll, J. R. Gabow, H. N. Shrairman, R. Tarjan. R. E. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. Communications of the ACM, 31(11):, 1988. p. 1343–1354
9. Elmasry, A. Layered heaps. In Proc. 9th Scandinavian Workshop on Algorithm Theory, volume 3111 of Lecture Notes in Computer Science, Springer, 2004. P. 212–222.
10. Elmasry, A. Parameterized self-adjusting heaps. J. Algorithms, 52(2):103–119, 2004.
11. Elmasry, A The violation heap: A relaxed Fibonacci-like heap. In Proc. 16th Annual International Conference on Computing and Combinatorics, volume 6196 of Lecture Notes in Computer Science, pages 479–488. Springer, 2010.
12. Elmasry, A Jensen, C. Katajainen, J. On the power of structural violations in priority queues. In Proc. 13th Computing: The Australasian Theory Symposium., v. 65 of CRPIT,. Australian Computer Society, 2007. p. 45–53
13. Elmasry, A., Jensen, C., Katajainen, J. Two-tier relaxed heaps. Acta Informatica, 45(3):, 2008. p. 193–210
14. Haeupler, B. Sen, S. Tarjan. R. E. Rank-pairing heaps. In Proc. 17th Annual European Symposium on Algorithsm, v. 5757 of Lecture Notes in Computer Science, Springer, 2009. SIAM Journal of Computing, to appear. p. 659–670.
15. Høyer. P. A general technique for implementation of efficient priority queues. In Proc. Third Israel Symposium on the Theory of Computing and Systems, , 1995. p. 57–66
16. Kaplan, H. Tarjan, R. E.. New heap data structures. Technical report, Department of Computer Science, Princeton University, 1999.
17. Kaplan, H. Tarjan, R. E.. Thin heaps, thick heaps. ACM Transactions on Algorithms, 4(1), 2008.
18. 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
19. Mulzer, W., Quake Heaps Erdbebenhaufen http://page.mi.fu-berlin.de/mulzer/
20. 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.
21. William J. Algorithm 232: Heapsort. Communications of the ACM, 7(6):, 1964. p. 347–348

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

Quake heap

V.K. Gulakov, K.V. Gulakov

Abstract— **the article is devoted to one of the varieties of pyramidal data structures widely used in solving various problems. Issues improving of software productivity due to use effective pyramidal data structures are discussed. A brief overview of pyramidal structures knowledge development shows regularity of the appearance such structures as a quake heap, which allows performing basic operations on data with minimal complexity. The article gives a detailed description of quake heap structure and gives examples of operations on it and its complexity. It is indicated to feature of improving efficiency due to use the corresponding coefficient. Comparison of the theoretical complexity operations over various pyramidal structures with operations over a quake heap indicates that it is not inferior in efficiency to most popular and effective pyramids, but differs relative simplicity and the ability to restore its effective structure after many operations. The article outlines the ways of its experimental research in various situations and practical problems.**

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