УДК 004.4
DOI:

В.К. Гулаков, К.В. Гулаков, И.А. Савостин,
А.О. Трубаков, Е.О. Трубаков

ОЦЕНКА ЭФФЕКТИВНОСТИ МНОГОМЕРНОЙ ПИРАМИДЫ

Изучено одно из актуальных направлений в области больших данных – обработка многомерной информации. Рассмотрена одна из возможных структур данных – многомерная пирамида, которая позволяет выполнять многие операции над данными со сложностью O(1) либо O(logn). Приведены описание, анализ и оценка эффективности многомерной пирамиды.

Ключевые слова: многомерная пирамида, базовые операции, разновидности k-d пирамид, улучшение k-d пирамид, большая размерность данных.

V.K. Gulakov, K.V. Gulakov, I.A. Savostin, A.O. Trubakov, E.O. Trubakov

ASSESSMENT OF MULTIDIMENSIONAL PYRAMID EFFICIENCY

The paper deals with one of the urgent trends in the field of big data – multidimensional information processing. One of possible data structures – multidimensional pyramid which allows fulfilling many operations with the complexity O(1) or O(logn) is under consideration. For basic operations there are given thorough examples. The description, analysis and assessment of multidimensional pyramid efficiency are shown. The assessments of operation complexity with these data depending upon their dimensionality are given. The analysis of multidimensional pyramid efficiency for its different modifications is shown.

Key words: multidimensional pyramid, basic operations, sorts of k-d pyramids, k-d pyramid improvement, data large dimensionality.

Введение

В настоящее время в мире активно ведутся работы по проблеме больших данных (Big Data). Трудности связаны не только с большим объёмом данных, но и со сложностью объектов данных и их взаимоотношениями.

Объекты с многочисленными параметрами далеко не всегда можно свести к одному параметру с помощью свёртки либо уменьшения размерности объекта [2]. Альтернативой в этом случае является применение многомерной модели данных [1]. Отношения между данными существенно влияют на время выполнения основных операций над ними. Если алгоритм часто использует операции «вставить», «найти максимум/минимум», «удалить максимум/минимум», «удалить заданный элемент», «изменить заданный элемент» и некоторые другие, то здесь весьма удобными являются пирамидальные структуры данных [3]. Они позволяют выполнять большинство операций за один шаг, т.е. практически не зависят от размерности задачи, либо имеют логарифмическую сложность (в худшем случае). Одной из таких структур является многомерная пирамида.

Для эффективного представления многомерной приоритетной очереди используется k-d пирамида [7]. Базовая форма k-d пирамиды не использует дополнительного пространства, требует линейного времени на построение, логарифмического времени для вставки, удаления или изменения любого элемента очереди и обеспечивает постоянную сложность доступа к элементам, содержащим минимальный ключ любого измерения. Более того, структура может быть расширена до многомерной двусторонней приоритетной очереди. Для некоторых приложений оптимальным вариантом будут многомерные приоритетные очереди.

После разработки единой структуры отпала необходимость в дополнительной памяти для хранения связей, дополнительных операциях для поддержки различных приоритетных отношений. При использовании неявного представления k-d пирамида не требует дополнительной памяти

для указателей. Более того, k-d пирамида может поддерживать некоторые или все операции в любом количестве измерений, в особых случаях - содержать ранее изученные структуры (типа двусторонних очередей).

Эта пирамида связана с k-d деревьями, которые расширяют деревья бинарного поиска до нескольких ключей. Основным недостатком k-d деревьев являются трудные операции удаления и балансировки дерева. Представленная k-d пирамида автоматически балансируется и более проста в представлении.

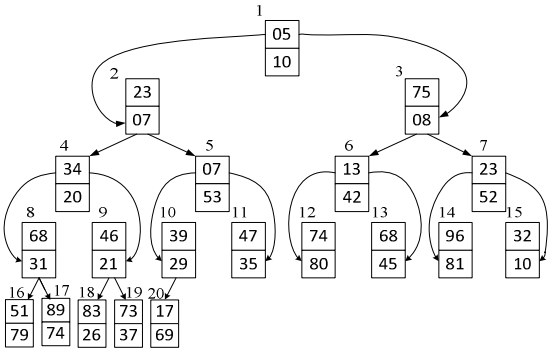


Рис. 1. 2-d пирамида из 20 элементов

Детальный анализ сложности операций над k-d пирамидой можно найти в [6].

Минимальная k-d пирамида H отражает множество элементов, каждый из которых содержит k ключей key_1, key_2, …, key_k, где key_i соответствует полностью упорядоченному множеству K_i.

Пирамида H является бинарным деревом, удовлетворяющим следующим условиям:

1. H – полное бинарное дерево.

2. H поддерживает основные свойства k-d пирамиды:

- элемент в корне имеет наименьшее значение ключа key_1 в дереве;
- каждый уровень дерева, кратный соответствующему ключу key_i, содержит наименьший ключ key_i в своём поддереве (другими словами, в k-d пирамиде узел уровня i имеет наименьший ключ $key_{mod(i, k)+1}$ в поддереве).

Определим уровень узла как число узлов на пути от корня к данному узлу. Уровень $i=\lceil \log n \rceil$, где n – номер элемента внутри пирамиды. Высота пирамиды равна наибольшему уровню её узлов. На практике высота пирамиды обычно значительно больше, чем число измерений. Мы будем называть k размерностью пирамиды. Например, 2-d пирамида показана на рис. 1.

Очевидно, что k-d пирамида может быть представлена как явно, так и неявно (в виде массива). Если не указано иное, мы будем рассматривать явное представление. Легко заметить, что одномерная пирамида является бинарным деревом, а двухмерная с key_1=-key_2 для всех узлов – min-max пирамидой [4].

Рассмотрим операции на примере 2-d пирамид, затем обобщим изложенное до k-d пирамид.

Базовые операции над 2-d пирамидой

Самой важной операцией является восстановление основного свойства пирамиды. Предположим, что в 2-d пирамиде один узел изменён.

Алгоритм ВСПЛЫТИЕ (узел): для каждого предка узла начиная с корня проверить основное свойство пирамиды между предком и узлом; если оно нарушено, то поменять их местами. На рис. 2а изображена 2-d пирамида с изменённым (неправильным) узлом (5), на рис. 2б - результат работы процедуры *ВСПЛЫТИЕ*.

После такой процедуры все узлы, кроме одного, будут соответствовать основному свойству пирамиды, поэтому нам нужно всего лишь восстановить порядок в поддереве, корнем которого является неправильный узел. Это делает следующий рекурсивный алгоритм.

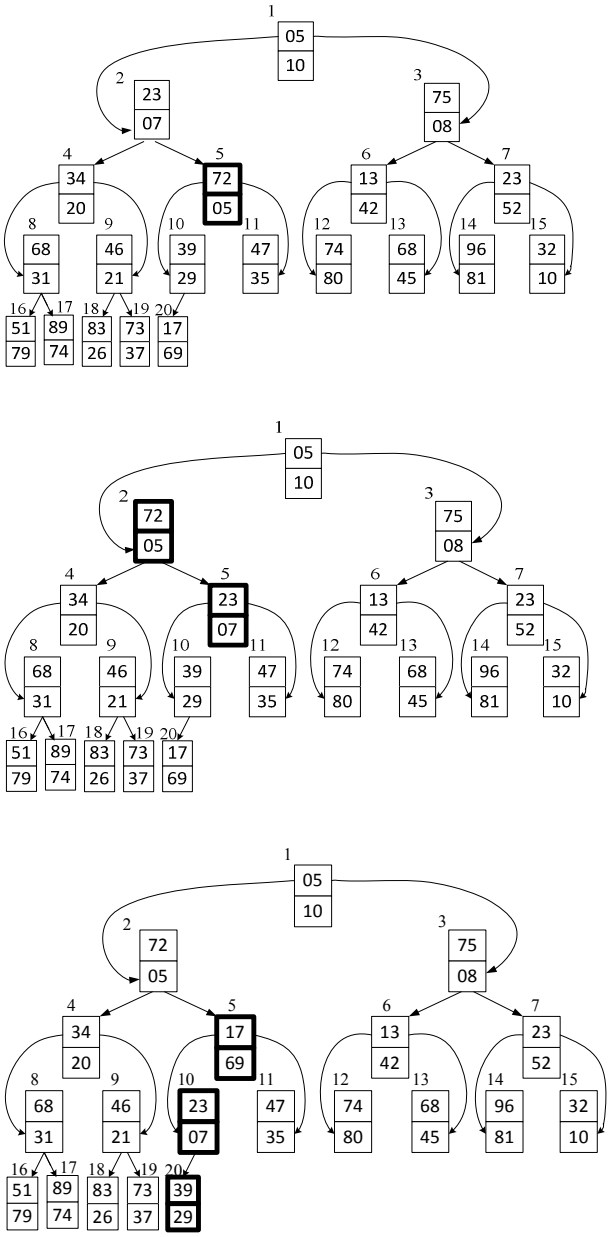
Алгоритм ПОГРУЖЕНИЕ (узел): для каждого потомка рассматриваемого узла проверяется основное свойство минимальной пирамиды; в случае нарушения этого свойства элементы меняются местами и процедура *ПОГРУЖЕНИЕ* выполняется рекурсивно.

Эта процедура показана на рис. 2в, где неправильный узел (рис. 2б) восста-

навливается относительно его потомков. Процедура может быть представлена как процедура проталкивания неправильного узла вниз по пирамиде.

```
Алгоритм ВОССТАНОВЛЕНИЕ(узел)
 Вызвать ВСПЛЫТИЕ(узел);
 Вызвать ПОГРУЖЕНИЕ(узел);
Конец
```

Теперь процедуру восстановления основного свойства пирамиды можно представить следующим образом:



а) Исходное положение пирамиды с изменённым элементом;

рядом с элементом стоит его номер, внутри - ключи: key1 (сверху) и key2 (снизу).

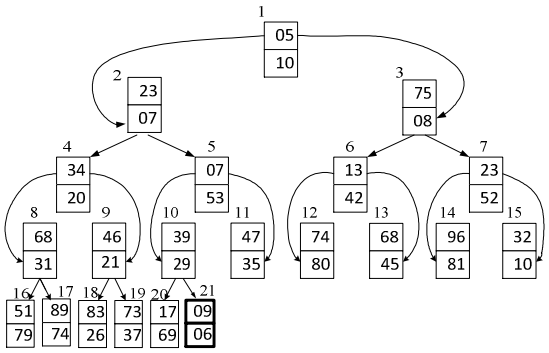
б) *ВСПЛЫТИЕ*
Шаг 1. Сравниваются элементы 5 и 1 по key1. Основное свойство пирамиды выполняется. Элементы местами не меняем.
Шаг 2. Сравниваются элементы 5 и 2 по key2. Основное свойство пирамиды не выполняется. Меняем элементы местами.

в) *ПОГРУЖЕНИЕ*
Шаг 1. Сравниваются элементы 5 и 20 по key1. Основное свойство пирамиды не выполняется. Меняем элементы местами.
Шаг 2. Сравниваются элементы 10 и 20 по key2. Основное свойство пирамиды не выполняется. Меняем элементы местами.

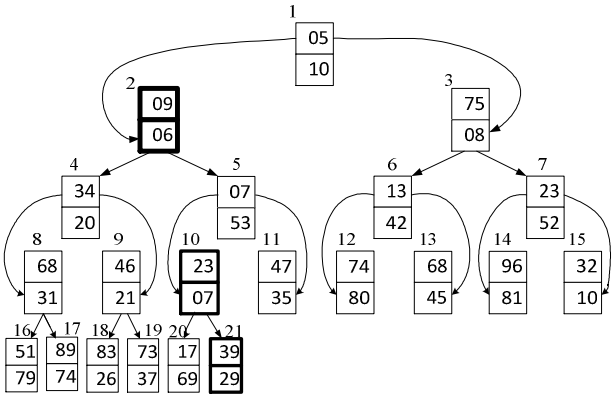
Рис. 2. Пример восстановления основного свойства 2-d пирамиды

Базирующиеся на этих алгоритмах операции над 2-d пирамидой довольно просты.

Для вставки элемента нам нужно добавить его в конец пирамиды и восстановить её основное свойство. Пример дан на рис. 3.



а) Исходная пирамида. Вставляется 21-й элемент с ключами key1=09, key2=06. Основное свойство пирамиды нарушено.



б) *ВСПЛЫТИЕ*
Шаг 1. Сравниваются элементы 21 и 1 по key1. Основное свойство пирамиды выполняется. Элементы местами не меняем.
Шаг 2. Сравниваются элементы 21 и 2 по key2. Основное свойство пирамиды не выполняется. Меняем элементы местами.
Шаг 3. Сравниваются элементы 21 и 5 по key1. Основное свойство пирамиды выполняется.
Шаг 4. Сравниваются элементы 21 и 10 по key2. Основное свойство пирамиды не выполняется. Меняем элементы местами.

Рис. 3. Вставка элемента в 2-d пирамиду

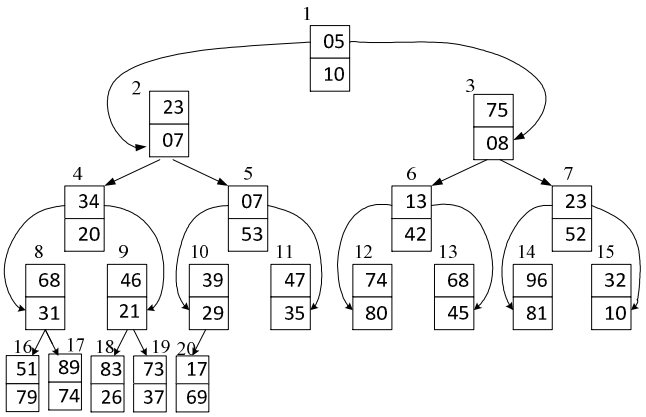
Для удаления минимального элемента по ключу key_1 мы должны поменять местами корень с этим элементом и восстановить порядок пирамиды. В этом случае необходимо выполнить только погружение. Пример дан на рис. 4б и 4в для пирамиды с рис. 4а.

При удалении минимального элемента по ключу key_2 мы сначала располагаем его среди первых трёх элементов пирамиды. Затем мы меняем его местами с последним элементом пирамиды и восстанавливаем основное свойство пирамиды.

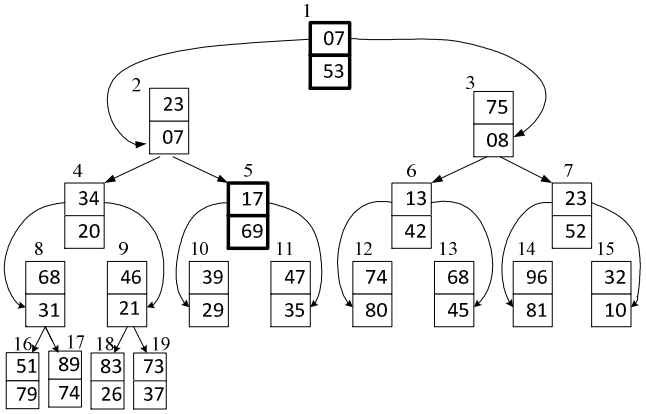
Для удаления произвольного элемента (с известной позицией) мы просто меняем его местами с последним элементом в пирамиде и восстанавливаем основное свойство. Чтобы изменить произвольный элемент, нужно выполнить изменение элемента, а затем восстановить основное свойство пирамиды.

Создание 2-d пирамиды похоже на создание бинарной пирамиды, т. е. оно выполняется снизу вверх. Мы описываем его в рекурсивной форме, хотя рекурсия не является необходимой. Она упрощает объяснение, так как имеет параметр, показывающий ключ уровня.

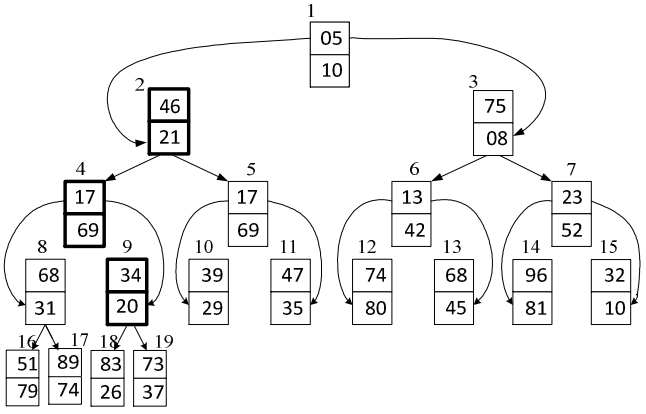
```
Алгоритм СОЗДАТЬ(пирамида, key_i)
 Вызвать СОЗДАТЬ(левое_поддерево, key_mod(i, 2)+1);
 Вызвать СОЗДАТЬ(правое_поддерево, key_mod(i, 2)+1);
 Вызвать ПОГРУЖЕНИЕ(корень);
Конец
```



а) Исходное положение 2-d пирамиды.



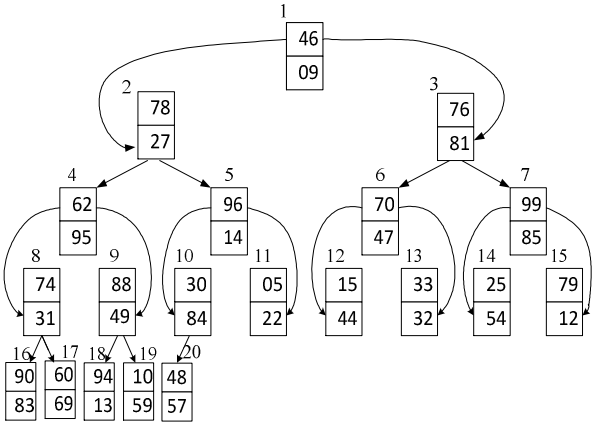
б) Удаление min key1:
Шаг 1. Удаляется элемент 1, на его место вставляется элемент 20. Сравниваются элементы 1 и 5 по key1. Основное свойство пирамиды не выполняется. Меняем элементы местами.
Шаг 2. Сравниваются элементы 10 и 5 по key1. Основное свойство пирамиды выполняется.



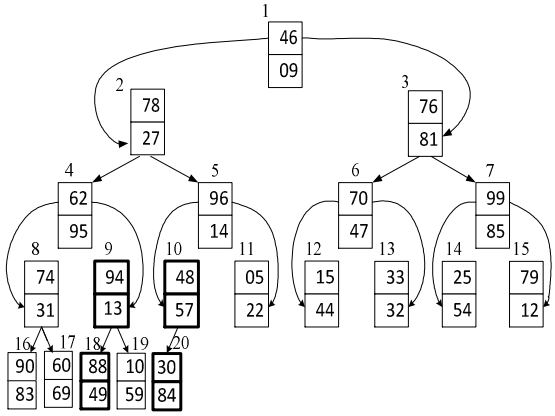
в) Удаление min key2:
Шаг 1. Удаляется элемент 2, на его место вставляется элемент 20. Сравниваются элементы 2 и 9 по key2. Основное свойство пирамиды не выполняется. Меняем элементы местами.
Шаг 2. Сравниваются элементы 4 и 9 по key1. Основное свойство пирамиды не выполняется. Меняем элементы местами.
Шаг 3. Сравниваются элементы 18 и 9 по key2. Основное свойство пирамиды выполняется.

Рис. 4. Пример удаления элемента с минимальными ключами из 2-d пирамиды

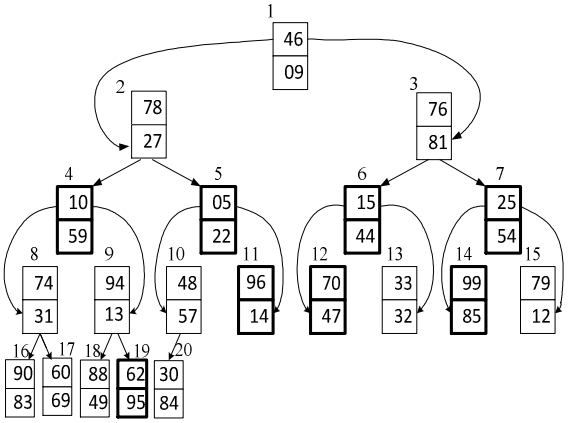
Пример показан на рис. 5.



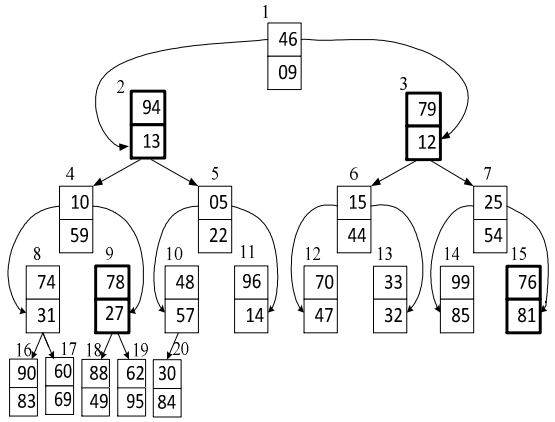
Бинарное дерево, представляющее множество из 20 2-d элементов



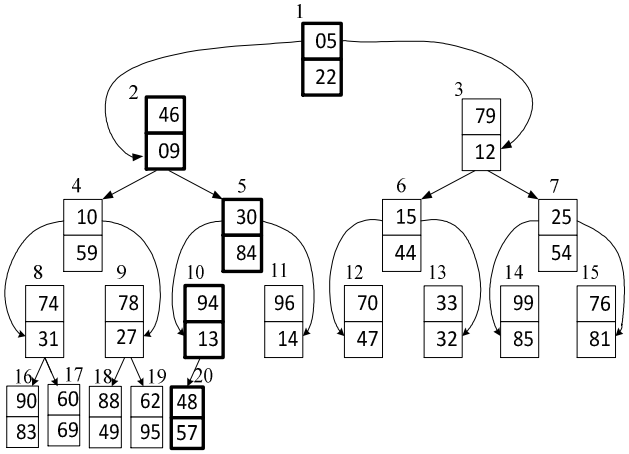
Восстановление основного свойства пирамиды на 4-м уровне по key2



Восстановление основного свойства пирамиды на 3-м уровне по key1



Восстановление основного свойства пирамиды на 2-м уровне по key2



Восстановление основного свойства пирамиды на 1-м уровне по key1

Рис. 5. Создание 2-d пирамиды

2-d пирамида из n элементов может быть создана за время O(n). В такой пирамиде поиск элемента с минимальным значением любого ключа имеет сложность O(1). Вставка, удаление элемента с минимальным значением любого ключа, удаление любого элемента, позиция которого известна, и изменение известного элемента имеют сложность O(log n) (в худшем случае).

2-d пирамида может быть представлена как min-max пирамида [5], если мы установим $key_2 = -key_1$ для всех узлов, где key_1 - исходный минимальный ключ, а key_2 - новый максимальный ключ. Однако операции для 2-d пирамиды были разработаны

так, что key_1 и key_2 могут быть полностью не связаны. Для этого случая более эффективно будет использование операций, специфичных для min-max пирамид. Для случая, в котором узел нормализован по отношению к потомкам, в общей 2-d пирамиде минимальный ключ может быть в любом из его детей и внуков. Однако в min-max пирамиде, если в ней имеется какой-либо внук, минимальный ключ будет в одном из внуков. Поэтому функция погружения может лишь по очереди проверять внуков, пока они не кончатся. Точно так же функция всплытия может использовать перевёрнутый подход и идти только через минимальные и максимальные уровни. Это снизит число сравнений на 50%. В целом операции на 2-d пирамиде могут быть улучшены, если результаты некоторых сравнений содержатся в результатах некоторых других сравнений, базирующихся на известных отношениях между двумя приоритетами.

Базовые операции над k-d пирамидами

Выводы из ранее изложенного могут быть легко обобщены для случая k-d пирамиды при любом постоянном k. Извлечение минимального элемента по ключу key_i представлено поиском среди постоянного числа узлов первых i уровней, поэтому требует O(1) времени. Если извлечение часто используется, позиции могут быть сохранены в памяти для обеспечения быстрого доступа. Функция всплытия остается такой же (независимо от k), функция погружения может быть представлена следующим образом (обобщение случая с 2-d пирамидой). Пусть key_i будет ключом уровня *узла*. Необходимо найти узел v с наименьшим ключом key_i среди потомков *узла* в пределах k уровней.

```
Алгоритм ПОГРУЖЕНИЕ(узел)
 Если узел имеет меньший ключ keyi, чем v,
      то остановка,
      иначе поменять местами узел  и v;
Вызвать ВСПЛЫТИЕ(v) для поддерева, корнем которого является
узел;
 Вызвать ПОГРУЖЕНИЕ(v);
Конец
```

Создание k-d пирамиды аналогично случаю с 2-d пирамидой. Отличие лишь в том, что любой уровень описан по отношению к модулю k вместо 2. Поэтому k-d пирамида из n элементов, где k - константа, может быть создана за время O(n). При этом требуется O(1) времени для извлечения элемента с минимальным значением любого из k ключей; для вставки, удаления элемента с минимальным значением любого из k ключей и удаления или изменения известного элемента требуется время O(log n).

На рис. 6 показаны 3-d пирамида и результат удаления корня. Пошагово процедура выполняется следующим образом:

Шаг 1. Удаляется элемент 1, на его место вставляется элемент 20. Сравниваются элементы 1 и 9 по key1. Основное свойство пирамиды не выполняется. Меняем элементы местами.

Шаг 2. Сравниваются элементы 2 и 9 по key2. Основное свойство пирамиды выполняется.

Шаг 3. Сравниваются элементы 4 и 9 по key3. Основное свойство пирамиды не выполняется. Меняем элементы местами.

Шаг 4. Сравниваются элементы 9 и 19 по key1. Основное свойство пирамиды не выполняется. Меняем элементы местами.

Следует заметить, что скрытая константа при удалении экспоненциальна по k, что делает операции неэффективными, когда значение k слишком велико. (С другой стороны, трудоемкость операции вставки не зависит от k, а трудоемкость создания пирамиды пропорциональна k^2 [7].) Далее показано, что экспоненциальная зависимость от константы k может быть устранена после легкой модификации структуры.

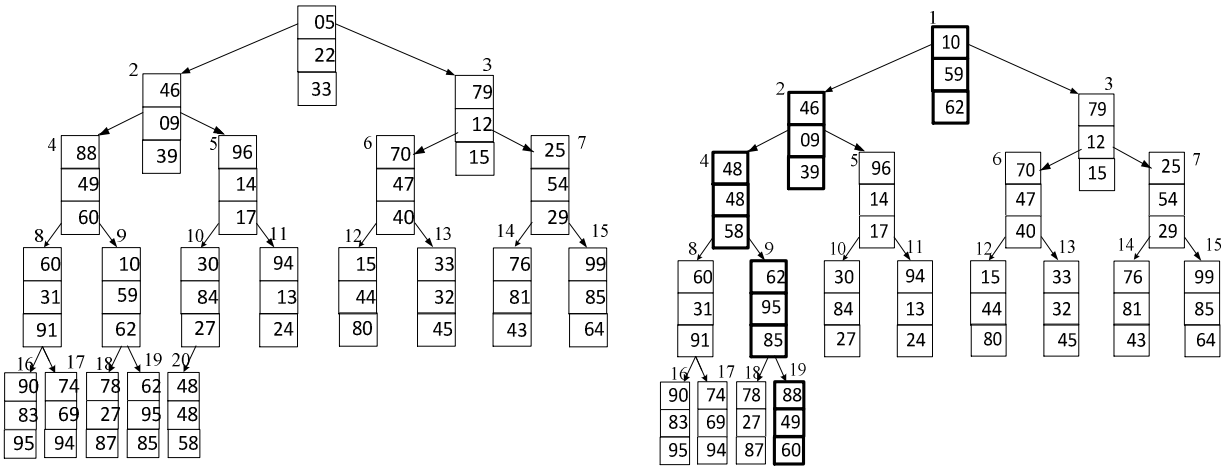


Рис. 6. Удаление элемента с минимальным key_1 в 3-d пирамиде: а - исходная пирамида; б - результирующая пирамида

Базовые возможности k-d пирамиды могут быть расширены для поддержки других операций. Мы кратко представим некоторые из них.

Ранее было показано, что 2-d пирамида может использоваться для представления min-max пирамид. Если каждый элемент имеет n ключей key_1, key_2, ..., key_n, мы можем (концептуально) включить ещё k ключей key_1, key_2, …, key_k, где key_i = - key_i, (знак «минус» представляет общее функциональное отображение). 2k-d пирамида становится двусторонней k-мерной пирамидой. Заметим, что новые ключи не надо хранить. Вместо этого отображение может быть закодировано в операциях, поэтому не требуется дополнительной памяти.

Объединение базовых k-d пирамид является сложной задачей. В особых случаях бинарные пирамиды (1-d пирамиды) требуют $O(\log^2 n)$ времени для объединения, а min-max пирамиды (особый тип 2-d пирамид) - минимум $\Omega(n)$.

С другой стороны, в [5] показано, что введение некоторого числа упрощений в основное свойство min-max пирамиды позволяет получить структуру, похожую на биномиальную очередь, объединение которой можно провести за O(log n). Особенно если пирамида разобрана на блоки, каждый из которых содержит 2^i элементов с уникальными i, в форме полного бинарного дерева плюс единичный элемент. Для нечетных i полное бинарное дерево является упрощённой min-max (max-min) пирамидой (слово «упрощённая» означает, что некоторые узлы не подчиняются основному свойству пирамиды). Уже было показано, как два узла одинакового размера объединяются за постоянное время, поэтому вся пирамида объединяется за логарифмическое время. Эта техника может быть обобщена до k-d пирамид простой декомпозицией, и для полного бинарного дерева высоты i корень будет иметь ключ $key_{mod(i, k)+1}$ в упрощённом порядке. Операции будут более сложными, нежели в случае с упрощёнными min-max пирамидами, но общая сложность будет такой же. Например, объединение двусторонней k-d пирамиды также занимает логарифмическое время.

По определению, пирамида поддерживает эффективный доступ к элементу с наибольшим (и/или наименьшим) ключом. Часто это становится причиной ситуации, когда функция в домене из одного или более приоритетных отношений формирует новое приоритетное отношение и на основе него обращается к элементам. Min-max пирамида является частным случаем. В общем случае все подобные предопределённые, функционально сформированные приоритеты могут быть закодированы в операции, неявное хранение соответст-

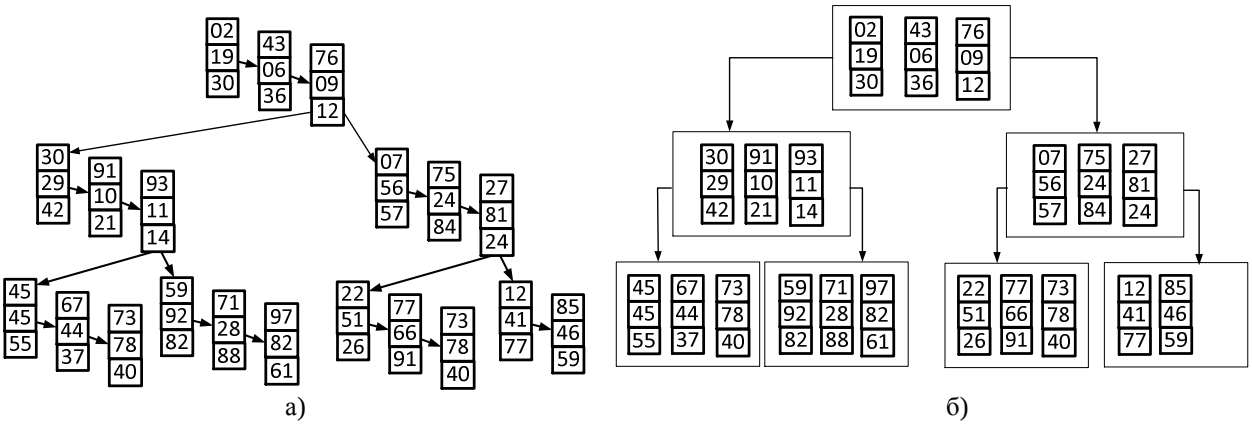
вующих ключей не является необходимостью. В этом случае k-d пирамиды могут быть использованы для представления обобщенных приоритетных очередей. Двусторонние пирамиды являются частным случаем обобщения.

Пирамиды, которые мы недавно обсуждали, имеют очень малый порядок отношений, поэтому в любой части пирамиды при нахождении минимального ключа в корне второй наименьший ключ может находиться где угодно среди первых k уровней потомков. Результатом этого является требование экспоненциального времени для удаления элемента. Когда k большое, в случае с двусторонними представлениями, этот недочет должен быть исправлен, для того чтобы такие структуры данных можно было применять на практике. Кратко опишем способ решения этой проблемы.

Вместо построения полного бинарного дерева для пирамиды мы позволим каждому узлу на уровне i (mod(i, k) ≠0) иметь не более одного потомка, в то время как узлы на уровнях i (mod(i, k) =0) всё ещё могут иметь двух. Рис. 7а показывает модифицированную структуру для 3-d пирамиды, приведенной на рис. 6а. В случае возможности эффективного неявного представления каждая цепочка из k узлов может быть упакована в один узел, поэтому основное бинарное дерево всё ещё остается полным (рис. 7б).

Очевидно, что операции погружения теперь требуется проверить последние 2k потомков для выбора минимума. С другой стороны, высота пирамиды почти в два раза меньше, поэтому время, затрачиваемое на операцию удаления, снизилось до $O(k^2)$. Сложность операции вставки, однако, увеличилась до k. Также можно заметить небольшое увеличение времени для операции создания пирамиды (но всё ещё $O(k^2n)$).

Хотя рис. 7 показывает модификацию для k=3, такая структура предпочтительна только при больших k. На практике малые значения k, например 2 или 3, встречаются чаще, а в таком случае оригинальная структура более эффективна.



а) б)

Рис. 7. Улучшенная структура для больших значений k и её представление

Заключение

Мы рассмотрели k-d пирамиду - структуру данных, эффективно представляющую многомерную приоритетную очередь без использования дополнительной памяти. Одна форма поддерживает вставку, удаление любого минимума, создание пирамиды за O(log n), $O(2^k \log n)$ и $O(k^2n)$ соответственно (с особой простотой операций для k=2), в то время как другие формы дают времена O(k*log n), $O(k^2 \log n)$, $O(k^2n)$. (Заметим, что для типичных значений k разница между 2^k и k^2 невелика.) Более того, k-d пирамида может быть расширена для поддержки двусторонних операций и операций слияния.

Представление k-d пирамиды является очень простым. Мы закодировали основные операции на C, что заняло около 120 строк кода. Поэтому данная структура очень практична. Полное представление со сравнением производительности приведено в [6].

Остаются открытыми несколько связанных со структурой проблем. Можно

улучшить ограничения для вставки и удаления минимального элемента до O(k*log n), используя только O(n) дополнительного пространства. Мы не знаем, как этого можно достичь, используя небольшое дополнительное пространство.

Другой интересной проблемой является проблема использования межприоритетных отношений для улучшения эффективности операций над k-d пирамидой. В случае двусторонней приоритетной очереди мы увидели, что такие отношения дают значительное снижение сложности. Воздействие на другие представленные отношения также достойно изучения.

СПИСОК ЛИТЕРАТУРЫ

1. Гулаков, В.К. Многомерные структуры данных: монография / В.К. Гулаков, А.О. Трубаков. - Брянск: БГТУ, 2010. - 387 с.
2. Гулаков, В.К. Сокращение размерности данных методом сингулярного разложения / В.К. Гулаков, В.Н. Матюшин // Информационные технологии. Радиоэлектроника. Телекоммуникации (ITRT-2012): сб. ст. II междунар. заоч. науч.-техн. конф. / Поволж. гос. ун-т сервиса. - Тольятти: Изд-во ПВГУС, 2012. - Ч. 1. - С. 415-422.
3. Кормен, Т.Х. Алгоритмы: построение и анализ: [пер. с англ.] / Т.Х. Кормен, Ч.И. Лейзерсон, Р. Ривест, К. Штайн. - 2-е изд. - М.: Вильямс, 2005. - 1296 с.
4. Atkinson, M. Min-Max Heaps and Generalized Priority Queues / M. Atkinson, J. Sack, N. Santoro, T. Strothotte // ACM. - 1986. - Vol. 29. - C. 996-1000.
5. Ding, Y. The Relaxed Min-Max Heap: A Mergeable Double-Ended Priority Queue / Y. Ding, M.A. Weiss //Acta Informatica. - 1993. - Vol. 30.
6. Ding, Y. Efficient Implementations of Multidimensional Priority Queues / Y. Ding, M.A. Weiss // School of Computer Science. - Florida International University, 1993.
7. Ding, Y. The K-D heap: an efficient multidimensional priority queue / Y. Ding, M.A. Weiss // Proc. 3rd Workshop on Algorithms and Data Structures. Lecture Notes in Computer Science. - 1993. - Vol. 709. - C. 302-313.

1. Gulakov, V.K. *Data Multidimensional Structures*: monograph / V.K. Gulakov, A.O. Trubakov. - Bryansk: BSTU, 2010. – pp. 387.
2. Gulakov, V.K. Data dimensionality reduction by method of singular decomposition / V.K. Gulakov, V.N. Matyushin // Information Technologies. Radio Electronics. Telecommunications (ITRT-2012): *Proceedings of the II-d Inter. Extra-Mural Scientific Technical Conf.* / Volga State University Service. – Togliatti: Publishing House of VVSUS, 2012. - Part. 1. - pp. 415-422.
3. Kormen, T.H. *Algorithms: Formation and Analysis*: [transl. from Engl.] / T.H. Kormen, Ch.I. Leizerson, R. Rivest, K. Stein. - 2-d Ed. - M.: Williams, 2005. – pp. 1296.
4. Atkinson, M. Min-Max Heaps and Generalized Priority Queues / M. Atkinson, J. Sack, N. Santoro, T. Strothotte // ACM. - 1986. - Vol. 29. - C. 996-1000.
5. Ding, Y. The Relaxed Min-Max Heap: A Mergeable Double-Ended Priority Queue / Y. Ding, M.A. Weiss //Acta Informatica. - 1993. - Vol. 30.
6. Ding, Y. Efficient Implementations of Multidimensional Priority Queues / Y. Ding, M.A. Weiss // School of Computer Science. - Florida International University, 1993.
7. Ding, Y. The K-D heap: an efficient multidimensional priority queue / Y. Ding, M.A. Weiss // Proc. 3rd Workshop on Algorithms and Data Structures. Lecture Notes in Computer Science. - 1993. - Vol. 709. - C. 302-313.

Статья поступила в редколлегию 24.01.17
Рецензент: к.т.н., доцент Брянского государственного технического университета Подвесовский А.Г.

Сведения об авторах:

Гулаков Василий Константинович, к.т.н., профессор кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: gvk10@yandex.ru.
Гулаков Константин Васильевич, к.т.н., доцент кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: gulakov32@yandex.ru.
Савостин Игорь Анатольевич, студент кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: iansav7@gmail.com.

Трубаков Андрей Олегович, к.т.н., доцент кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: trubakovao@gmail.com.

Трубаков Евгений Олегович, к.т.н., доцент кафедры «Информатика и программное обеспечение» Брянского государственного технического университета, e-mail: trubakoveo@gmail.com.

Gulakov Vasily Konstantinovich, Can. Eng., Prof. of the Dep. "Informatics and Software", Bryansk State Technical University, e-mail: gvk10@yandex.ru.
Gulakov Konstantin Vasilievich, Can. Eng., Assistant Prof. of the Dep. "Informatics and Software", Bryansk State Technical University, e-mail: gulakov32@yandex.ru.
Savostin Igor Anatolievich, Student of the Dep. "Informatics and Software", Bryansk State Technical University, e-mail: iansav7@gmail.com.

Trubakov Andrey Olegovich, Can. Eng., Assistant Prof. of the Dep. "Informatics and Software", Bryansk State Technical University, e-mail: trubakovao@gmail.com.
Trubakov Eugene Olegovich, Can. Eng., Assistant Prof. of the Dep. "Informatics and Software", Bryansk State Technical University, e-mail: trubakoveo@gmail.com.