ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ «ВОЛГОГРАДСКИЙ СОЦИАЛЬНО – ПЕДАГОГИЧЕСКИЙ КОЛЛЕДЖ»

Кафедра информационных технологий обучения  
Специальность 09.02.07 Информационные системы и программирование

**Курсовая работа**

**«Внешний поиск с использованием расширяемого хеширования»**

**Студента группы: 21 «Д»**

Поповичев Артем Алексеевич

**Специальность:** 09.02.07 «Информационные системы и

программирование»

**Руководитель:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**Бетиров. А.М.

**Работа допущена к защите:**

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(роспись руководителя)

Волгоград 2023 г.

**Оглавление**

[**Глава 1. Введение** 1](#_Toc134750977)

[**1.1 Вступление** 2](#_Toc134750978)

[**1.2 Цель и задачи исследования:** 4](#_Toc134750979)

[**Г****лава 2.** **Очереди по приоритетам.** 8](#_Toc134750982)

[**2.1****Обзор базовых структур данных: очереди и приоритеты** 9](#_Toc134750983)

[**2.2** **Подробное объяснение принципа работы очередей по приоритетам.** 10](#_Toc134750984)

[**2.3 Рассмотрение различных реализаций очередей по приоритетам (например, на основе массивов или связанных списков)** 11](#_Toc134750985)

[**2.4 Сравнение производительности различных реализаций и анализ их преимуществ и недостатков.** 12](#_Toc134750985)

[**Глава 3. Пирамидальная сортировка** 13](#_Toc134750986)

[**3.1** **Обзор пирамидальной сортировки как эффективного алгоритма сортировки** 14](#_Toc134750987)

[**3.2 Изучение основных принципов пирамидальной сортировки** 15](#_Toc134750988)

[**3.3 Подробное объяснение построения и поддержки пирамиды (кучи)** 16](#_Toc134750989)

[**3.4** **Анализ временной сложности и пространственной сложности пирамидальной сортировки** 17](#_Toc134750989)

[**3.5 Сравнение пирамидальной сортировки с другими алгоритмами сортировки (например, быстрая сортировка или сортировка слиянием)** 18](#_Toc134750989)

[**Глава 4. Примеры и приложения.** 19](#_Toc134750990)

[**4.1 Представление конкретных примеров использования очередей по приоритетам и пирамидальной сортировки.** 20](#_Toc134750991)

[**4.2** **Рассмотрение приложений, где эти структуры данных и алгоритмы могут быть полезными (например, планирование задач, управление ресурсами и т. д.)** 21](#_Toc134750992)

[**4. Заключение.** 22](#_Toc134750994)

[**4. Список литературы.** 23](#_Toc134750995)

**Глава 1. Введение**

**1.1 Вступление**

Очереди по приоритетам и пирамидальная сортировка являются важными концепциями в области структур данных и алгоритмов. Они предоставляют эффективные способы управления и организации данных, а также обеспечивают оптимальные алгоритмы для сортировки элементов. В данной курсовой работе мы сосредоточимся на изучении и анализе этих двух тем.

Очереди по приоритетам представляют собой расширение обычных очередей, где каждому элементу присваивается определенный приоритет. Это позволяет эффективно управлять порядком обработки элементов, обеспечивая предоставление приоритетных элементов раньше остальных. Мы рассмотрим различные реализации очередей по приоритетам и проанализируем их преимущества и недостатки.

Пирамидальная сортировка, с другой стороны, представляет собой один из наиболее эффективных алгоритмов сортировки. Она основана на использовании структуры данных, известной как пирамида или куча. Пирамидальная сортировка обеспечивает сортировку элементов в порядке возрастания или убывания, и при этом обладает линейной временной сложностью, что делает ее очень привлекательным выбором для сортировки больших объемов данных.

Целью данной курсовой работы является более глубокое понимание очередей по приоритетам и пирамидальной сортировки, их принципов работы, основных применений и производительности. Мы также разработаем программную реализацию обоих подходов и проведем тестирование для проверки их корректности и эффективности.

В ходе работы будут рассмотрены различные аспекты, связанные с очередями по приоритетам и пирамидальной сортировкой, включая сравнение различных реализаций, анализ временной и пространственной сложности, а также примеры применения в различных областях.

Далее мы перейдем к изучению очередей по приоритетам и пирамидальной сортировки, а также проведем практическую реализацию и тестирование для более полного понимания этих важных структур данных и алгоритмов.

**1.2 Цель и задачи исследования:**

Целью данной курсовой работы является более глубокое понимание очередей по приоритетам и пирамидальной сортировки, их принципов работы, основных применений и производительности. В рамках этой цели ставятся следующие задачи:

1. Изучение основных концепций и определений, связанных с очередями по приоритетам и пирамидальной сортировкой. Это включает ознакомление с базовыми понятиями очередей, приоритетами, пирамидой и сортировкой.
2. Рассмотрение различных реализаций очередей по приоритетам. Исследование различных подходов к реализации очередей по приоритетам, таких как основанные на массивах или связанных списках, с целью понять их особенности, преимущества и недостатки.
3. Анализ производительности различных реализаций очередей по приоритетам. Сравнение временной и пространственной сложности различных реализаций для оценки их эффективности и применимости в различных сценариях использования.
4. Изучение пирамидальной сортировки. Подробное изучение принципов работы пирамидальной сортировки, включая построение и поддержку пирамиды (кучи), а также анализ ее временной сложности и пространственной сложности.
5. Сравнение пирамидальной сортировки с другими алгоритмами сортировки. Осуществление сравнительного анализа пирамидальной сортировки с другими популярными алгоритмами сортировки, такими как быстрая сортировка или сортировка слиянием, с целью определения ее преимуществ и областей применения.
6. Представление примеров и приложений. Исследование и представление конкретных примеров использования очередей по приоритетам и пирамидальной сортировки в различных областях, таких как планирование задач, управление ресурсами и другие.
7. Разработка программной реализации и тестирование. Разработка программного кода для реализации очередей по приоритетам и пирамидальной сортировки, а также проведение тестирования для проверки корректности и эффективности реализованных структур данных и алгоритмов.

Путем выполнения данных задач исследование позволит получить более глубокое понимание очередей по приоритетам и пирамидальной сортировки, их применения и производительности, а также разработать практическую реализацию этих структур данных и алгоритмов.

.**Глава 2. Очереди по приоритетам**

## **2.1 Обзор базовых структур данных: очереди и приоритеты**

Очередь (Queue) - это структура данных, которая работает по принципу "первым пришёл, первым вышел" (FIFO - First-In-First-Out). Это означает, что элементы добавляются в конец очереди и извлекаются из начала очереди. Элементы в очереди могут быть различных типов данных и добавляются в порядке поступления. Очереди широко используются во многих приложениях, включая планирование задач, управление ресурсами и обработку событий.

Приоритет (Priority) - это значение, присвоенное элементу в структуре данных, чтобы определить его важность или порядок обработки. Он используется для определения приоритета элемента в контексте очередей по приоритетам и других алгоритмов, где элементы должны быть обработаны в определенном порядке. Приоритеты могут быть представлены числами, строками или другими сущностями, в зависимости от конкретной реализации и требований задачи.

Очереди по приоритетам (Priority Queues) - это структуры данных, которые комбинируют принцип работы очереди с присвоением приоритетов элементам. В очереди по приоритетам элементы извлекаются не в порядке их добавления, а на основе их приоритета. Элементы с более высоким приоритетом имеют приоритет при извлечении. Очереди по приоритетам широко используются в задачах, где необходимо обрабатывать элементы в порядке их важности или срочности. Примерами могут служить планирование задач, обработка событий в реальном времени или управление приоритетами в операционных системах.

Реализации очередей по приоритетам могут быть различными и зависят от конкретных требований и контекста приложения. Одна из распространенных реализаций - это использование двоичной кучи (Binary Heap), которая представляет собой бинарное дерево с определенными свойствами. Другие реализации могут включать использование связанных списков, массивов или более сложных структур данных.

Очереди по приоритетам и приоритеты важны для многих алгоритмических проблем, таких как планирование задач, оптимальное распределение ресурсов, управление сетевым трафиком и другие. Понимание этих структур данных и их применений помогает разработчикам создавать эффективные и оптимальные решения для широкого спектра задач.

## **2.2 Подробное объяснение принципа работы очередей по приоритетам**

## Очереди по приоритетам - это структуры данных, в которых каждому элементу присваивается определенный приоритет, и при извлечении элементы выбираются согласно их приоритету. Принцип работы очередей по приоритетам заключается в том, что элементы с более высоким приоритетом имеют приоритет при извлечении из очереди. Это означает, что элементы с более высоким приоритетом будут извлекаться раньше, чем элементы с более низким приоритетом.

## Существует несколько способов реализации очередей по приоритетам, но одной из наиболее популярных является использование структуры данных, называемой двоичной кучей (Binary Heap). Двоичная куча представляет собой полное двоичное дерево, где каждый узел имеет значение, связанное с ним, и выполняются определенные свойства кучи.

## В основе двоичной кучи лежит бинарное дерево, в котором для каждого узла выполняются два основных свойства:

## 1. Свойство упорядоченности: Значение каждого узла больше (или меньше) значений его потомков в соответствии с заданной политикой приоритета. Например, в мин-куче значение каждого узла меньше или равно значений его потомков, а в макс-куче - больше или равно.

## 2. Свойство заполненности: Двоичная куча является полным двоичным деревом, то есть все уровни дерева заполнены элементами, кроме, возможно, последнего уровня, который заполняется слева направо.

## Операции над очередями по приоритетам включают добавление элемента (вставку) и извлечение элемента с наивысшим приоритетом.

## При добавлении элемента в очередь по приоритетам, новый элемент вставляется в конец очереди, а затем происходит перестройка кучи (heapify), чтобы восстановить свойства кучи. В результате новый элемент займет свое правильное место в куче с учетом его приоритета.

## При извлечении элемента из очереди по приоритетам, элемент с наивысшим приоритетом (вершина кучи) извлекается и удаляется из очереди. Затем происходит перестройка кучи для восстановления свойств кучи. Это обеспечивает, что следующий элемент с наивысшим приоритетом становится новой вершиной кучи.

## Преимуществом очередей по приоритетам является эффективное извлечение элемента с наивысшим приоритетом, что делает их полезными для решения задач, где необходимо обрабатывать элементы в порядке их важности или срочности.

## **2.3 Рассмотрение различных реализаций очередей по приоритетам (например, на основе массивов или связанных списков)**

Очереди по приоритетам могут быть реализованы различными способами, включая использование массивов или связанных списков. Каждая реализация имеет свои особенности, преимущества и недостатки. Рассмотрим две распространенные реализации:

1. Очередь по приоритетам на основе массивов:

- В этой реализации элементы с приоритетами хранятся в массиве, где индекс элемента отражает его приоритет.

- При добавлении элемента в очередь, его приоритет используется как индекс для определения позиции в массиве, куда он будет помещен.

- При извлечении элемента с наивысшим приоритетом, просматривается массив, начиная с наивысшего приоритета, и выбирается элемент с наивысшим приоритетом.

- Преимущества: Простая реализация, быстрое извлечение элемента с наивысшим приоритетом.

- Недостатки: Вставка нового элемента требует перестроения массива, если место для нового приоритета не было выделено заранее. Это может привести к перераспределению всего массива и занимать больше памяти.

2. Очередь по приоритетам на основе связанных списков:

- В этой реализации элементы с приоритетами хранятся в связанных списках, где каждый список соответствует определенному приоритету.

- При добавлении элемента в очередь, он добавляется в список с соответствующим приоритетом. Если список для данного приоритета еще не существует, он создается.

- При извлечении элемента с наивысшим приоритетом, происходит поиск списка с наивысшим приоритетом, а затем извлекается элемент из этого списка.

- Преимущества: Гибкость в добавлении новых приоритетов и элементов, более эффективное использование памяти при разреженных приоритетах.

- Недостатки: Извлечение элемента с наивысшим приоритетом может потребовать просмотра всех списков, что может занимать больше времени.

Выбор конкретной реализации очереди по приоритетам зависит от требований задачи, ожидаемой производительности, доступной памяти и других факторов. Каждая реализация имеет свои особенности и, может быть наилучшей в определенных сценариях использования.

## **2.4 Сравнение производительности различных реализаций и анализ их преимуществ и недостатков.**

Сравнение производительности различных реализаций очередей по приоритетам и анализ их преимуществ и недостатков поможет нам выбрать наиболее подходящую реализацию в конкретной ситуации. Вот обзор преимуществ и недостатков двух распространенных реализаций:

Очередь по приоритетам на основе массивов:

Преимущества:

Быстрое извлечение элемента с наивысшим приоритетом: извлечение элемента выполняется за константное время O(1), поскольку наивысший приоритет всегда находится в определенной позиции (например, начало массива).

Простая реализация: использование массива позволяет легко управлять элементами и их приоритетами.

Недостатки:

Медленная вставка: добавление нового элемента может потребовать перестроения массива, если место для нового приоритета не было выделено заранее. В худшем случае вставка может потребовать O(n) операций, где n - размер очереди.

Затраты на память: массив должен быть предварительно выделен с достаточным размером для хранения ожидаемого числа приоритетов. В случае разреженных приоритетов это может привести к неэффективному использованию памяти.

Очередь по приоритетам на основе связанных списков:

Преимущества:

Гибкость в добавлении новых приоритетов и элементов: связанные списки позволяют легко добавлять новые элементы и приоритеты без необходимости перестройки всей структуры данных.

Эффективное использование памяти: при разреженных приоритетах связанные списки могут использовать меньше памяти, поскольку создаются только списки с фактически присутствующими приоритетами.

Недостатки:

Медленное извлечение: извлечение элемента с наивысшим приоритетом может потребовать просмотра всех списков, что занимает O(k), где k - количество различных приоритетов.

Дополнительная память для хранения указателей: каждый узел связанного списка требует дополнительной памяти для хранения указателей на следующий элемент.

Выбор между этими реализациями зависит от конкретных требований задачи. Если вы ожидаете частые операции извлечения элементов с наивысшим приоритетом и размер очереди заранее известен, то массивы могут быть предпочтительными из-за их быстрого доступа к элементам. Однако, если вы ожидаете частые операции вставки и удаления элементов с произвольными приоритетами, и если размер очереди может динамически меняться, то связанные списки могут быть более эффективными.

Важно учитывать требования по времени выполнения операций, использование памяти и ожидаемый размер очереди при выборе реализации очереди по приоритетам. Также стоит рассмотреть другие возможные реализации, такие как двоичная куча или сбалансированные деревья, которые могут предоставить еще большую эффективность для определенных сценариев использования.

# **Глава 2. Пирамидальная сортировка**

## **3.1 Обзор пирамидальной сортировки как эффективного алгоритма сортировки**

Пирамидальная сортировка (Heap Sort) является одним из эффективных алгоритмов сортировки, который основан на использовании структуры данных, известной как пирамида (куча).

Принцип работы пирамидальной сортировки следующий:

1. Построение пирамиды (Heapify):
   * Исходный массив рассматривается как пирамида, где каждый элемент рассматривается как узел.
   * Построение пирамиды начинается с последнего уровня дерева и продолжается к корневому узлу.
   * На каждом шаге узлы сравниваются с их потомками, и если не выполняется условие пирамиды (свойство упорядоченности), элементы меняются местами.
   * Этот процесс повторяется для каждого узла до корневого узла, что приводит к построению пирамиды.
2. Извлечение элементов:
   * Корень пирамиды (наивысший приоритет) содержит наибольший (или наименьший) элемент массива.
   * Корень извлекается из пирамиды и помещается в конец отсортированного массива.
   * После извлечения корня, пирамида перестраивается (Heapify) для оставшихся элементов, чтобы восстановить свойства пирамиды.
   * Этот процесс повторяется до тех пор, пока все элементы не будут извлечены и помещены в отсортированный массив.

Преимущества пирамидальной сортировки:

1. Гарантированная сложность O(n log n): Пирамидальная сортировка имеет одинаковую временную сложность для наихудшего, среднего и лучшего случаев, что делает ее предсказуемой и эффективной для больших наборов данных.
2. In-place сортировка: Пирамидальная сортировка выполняется непосредственно в исходном массиве, не требуя дополнительной памяти для хранения промежуточных результатов.
3. Устойчивость: Пирамидальная сортировка является устойчивой сортировкой, то есть она сохраняет относительный порядок равных элементов.

Недостатки пирамидальной сортировки:

1. Неэффективна для небольших наборов данных: Пирамидальная сортировка имеет некоторые дополнительные затраты на построение пирамиды, что делает ее неоптимальной для небольших массивов.
2. Не подходит для часто изменяющихся данных: Если данные часто изменяются, требуется повторное построение пирамиды для каждого изменения, что может быть неэффективным.
3. Не сохраняет исходный порядок равных элементов: Пирамидальная сортировка не сохраняет исходный порядок равных элементов, что может быть важным для некоторых приложений.

Несмотря на некоторые ограничения, пирамидальная сортировка является эффективным алгоритмом сортировки для больших наборов данных, где стабильность сортировки или изменяемость данных не являются критическими факторами.

## **3.2 Изучение основных принципов пирамидальной сортировки**

Пирамидальная сортировка (Heap Sort) основана на использовании структуры данных, называемой пирамида (куча), и включает несколько ключевых принципов. Вот основные принципы пирамидальной сортировки:

1. Структура данных "пирамида":

Пирамида представляет собой бинарное дерево, где каждый узел имеет значение (элемент) и удовлетворяет свойству пирамиды. В пирамиде максимальное значение находится в корне, а каждый родительский узел имеет значение, большее или равное значению его потомков (для максимальной пирамиды) или меньшее или равное значению его потомков (для минимальной пирамиды).

2. Построение пирамиды (Heapify):

- Исходный массив рассматривается как пирамида, где каждый элемент массива является узлом пирамиды.

- Построение пирамиды начинается с последнего уровня дерева и движется вверх к корню.

- Для каждого узла пирамиды проверяется свойство пирамиды: если узел имеет значение, меньшее (для максимальной пирамиды) или большее (для минимальной пирамиды) значения его потомков, то значения узла и его потомка меняются местами.

- Этот процесс продолжается до достижения корневого узла, что приводит к построению пирамиды.

3. Извлечение элементов из пирамиды:

- Корень пирамиды содержит наибольший (для максимальной пирамиды) или наименьший (для минимальной пирамиды) элемент массива.

- Извлечение элемента происходит следующим образом: корень пирамиды меняется местами с последним элементом пирамиды, затем размер пирамиды уменьшается на один элемент.

- Для восстановления свойств пирамиды после извлечения элемента применяется операция "просеивания вниз" (sift down). Это означает, что корень пирамиды сравнивается со своими потомками, и если не выполняется свойство пирамиды, значения узла и наибольшего (для максимальной пирамиды) или наименьшего (для минимальной пирамиды) потомка меняются местами.

- Этот процесс повторяется для кажд

ого извлеченного элемента до тех пор, пока не будут извлечены все элементы и получен отсортированный массив.

Основной принцип пирамидальной сортировки заключается в использовании свойств пирамиды для обеспечения правильного порядка элементов и постепенного извлечения наибольших (для максимальной пирамиды) или наименьших (для минимальной пирамиды) элементов для формирования отсортированного массива.

## **3.3 Подробное объяснение построения и поддержки пирамиды (кучи)**

Построение и поддержка пирамиды (кучи) являются важными шагами в пирамидальной сортировке и других алгоритмах, использующих структуру данных пирамида. Вот подробное объяснение построения и поддержки пирамиды:

1. Построение пирамиды (Heapify):
   * Построение пирамиды начинается с последнего уровня дерева и движется вверх к корню.
   * Для каждого узла пирамиды проверяется свойство пирамиды: если узел имеет значение, меньшее (для максимальной пирамиды) или большее (для минимальной пирамиды) значения его потомков, то значения узла и его потомка меняются местами.
   * Процесс построения пирамиды продолжается для каждого узла до достижения корневого узла.

Псевдокод для построения максимальной пирамиды (для минимальной пирамиды изменяется условие сравнения):

1.heapify(array):

2. n = размер массива

3. start = (n / 2) - 1 # Начинаем с последнего узла, имеющего потомков

4. # Проходим по каждому узлу, начиная с последнего и до корня

5. for i from start до 0:

6. siftDown(array, i, n)

Операция "просеивания вниз" (sift down):

* После извлечения элемента из пирамиды или вставки нового элемента в пирамиду может быть нарушено свойство пирамиды.
* Операция "просеивания вниз" выполняется для восстановления свойств пирамиды, где узел сравнивается со своими потомками и, если не выполняется свойство пирамиды, значения узла и наибольшего (для максимальной пирамиды) или наименьшего (для минимальной пирамиды) потомка меняются местами.
* Процесс "просеивания вниз" повторяется для каждого нарушенного узла, пока свойство пирамиды не будет восстановлено.

Псевдокод для операции "просеивания вниз":

1. siftDown(array, i, n):

2. largest = i # Инициализация наибольшего значения как текущего узла

3. left = 2 \* i + 1 # Индекс левого потомка

4. right = 2 \* i + 2 # Индекс правого потомка

5. # Если левый потомок существует и его значение больше значения текущего узла

6. if left < n and array[left] > array[largest]:

7. largest = left

8. # Если правый потомок существует и его значение больше значения текущего узла

9. if right < n and array[right] > array[largest]:

10. largest = right

11. # Если нарушается свойство пирамиды, меняем значения узла и наибольшего потомка

12. if largest != i:

13. swap(array[i], array[largest])

14. siftDown(array, largest, n)

## **3.4 Анализ временной сложности и пространственной сложности пирамидальной сортировки**

Пирамидальная сортировка имеет следующую временную и пространственную сложности:

Временная сложность:

- В худшем, среднем и лучшем случаях пирамидальная сортировка имеет временную сложность O(n log n), где n - количество элементов в массиве.

- Построение пирамиды (Heapify) занимает O(n) времени.

- Извлечение каждого элемента из пирамиды и восстановление свойств пирамиды (просеивание вниз) занимает O(log n) времени.

- Всего требуется выполнить O(n) операций извлечения элементов, поэтому общая временная сложность составляет O(n log n).

Пространственная сложность:

- Пространственная сложность пирамидальной сортировки составляет O(1) (константная сложность), так как сортировка выполняется на месте, без использования дополнительной памяти, кроме исходного массива.

Пирамидальная сортировка обладает хорошей производительностью для больших наборов данных, так как ее временная сложность имеет порядок O(n log n), что является оптимальным для сортировки на сравнении основанных алгоритмов. Однако, у нее есть некоторые недостатки, такие как относительно большая константа и неустойчивость сортировки (при наличии равных элементов порядок их следования может изменяться).

## **3.5 Сравнение пирамидальной сортировки с другими алгоритмами сортировки (например, быстрая сортировка или сортировка слиянием)**

Пирамидальная сортировка, быстрая сортировка и сортировка слиянием являются популярными алгоритмами сортировки. Вот сравнение пирамидальной сортировки с этими алгоритмами:

1. Пирамидальная сортировка и быстрая сортировка:

- Пирамидальная сортировка и быстрая сортировка оба имеют временную сложность O(n log n) в среднем и лучшем случаях.

- Пирамидальная сортировка требует O(1) дополнительной памяти, в то время как быстрая сортировка требует O(log n) дополнительной памяти для рекурсивных вызовов стека.

- Быстрая сортировка может быть более эффективной на практике для средних и больших наборов данных, благодаря своей лучшей константе временной сложности и лучшей локальности доступа к данным.

- Однако, в худшем случае быстрая сортировка имеет временную сложность O(n^2), что является недостатком, в то время как пирамидальная сортировка всегда имеет временную сложность O(n log n).

2. Пирамидальная сортировка и сортировка слиянием:

- Оба алгоритма имеют временную сложность O(n log n) в худшем, среднем и лучшем случаях.

- Сортировка слиянием требует дополнительной памяти объемом O(n), так как использует дополнительный массив для слияния отсортированных подмассивов, в то время как пирамидальная сортировка требует O(1) дополнительной памяти.

- Сортировка слиянием является стабильной, то есть она сохраняет относительный порядок равных элементов, в то время как пирамидальная сортировка не является стабильной.

- Сортировка слиянием обычно более эффективна на практике для сортировки связных списков или внешней сортировки, тогда как пирамидальная сортировка может быть эффективнее для массивов, особенно если память ограничена.

Итак, выбор между пирамидальной сортировкой, быстрой сортировкой и сортировкой слиянием зависит от конкретных требований и характеристик данных. Быстрая сортиров

ка обычно является предпочтительным выбором для общего использования, если нет особых ограничений, в то время как пирамидальная сортировка может быть полезной, если требуется минимальное использование дополнительной памяти и гарантированное время выполнения. Сортировка слиянием может быть предпочтительной, если нужна стабильность сортировки или работа с большими объемами данных и доступ к памяти не является узким местом.

# **Глава 4. Примеры и приложения**

## **4.1 Представление конкретных примеров использования очередей по приоритетам и пирамидальной сортировки.**

Очереди по приоритетам и пирамидальная сортировка имеют множество применений в различных областях. Вот некоторые конкретные примеры использования обеих структур данных:

Примеры использования очередей по приоритетам:

1. Распределение ресурсов: Очереди по приоритетам могут использоваться для управления и распределения ресурсов с учетом их приоритета. Например, в операционных системах они могут использоваться для планирования задач и определения приоритетов выполнения процессов.

2. Алгоритмы планирования: Очереди по приоритетам могут использоваться в алгоритмах планирования, таких как планирование задач в компьютерных сетях или планирование задач в многопроцессорных системах. Задачи с более высоким приоритетом будут обрабатываться раньше.

3. Системы обработки событий: Очереди по приоритетам могут использоваться для обработки событий различных типов, таких как сетевые события, события пользовательского ввода или события системы. События с более высоким приоритетом будут обрабатываться в первую очередь.

Примеры использования пирамидальной сортировки:

1. Сортировка массивов: Пирамидальная сортировка может использоваться для сортировки массивов любого типа данных. Благодаря своей эффективности и временной сложности O(n log n), она широко применяется в практике для сортировки больших объемов данных.

2. Приоритезация задач: Пирамидальная сортировка может использоваться для приоритезации задач в системах управления задачами. Каждой задаче присваивается приоритет, и пирамидальная сортировка позволяет эффективно упорядочить задачи в порядке их приоритетов.

3. Статистические вычисления: Пирамидальная сортировка может быть полезна при выполнении статистических вычислений, таких как нахождение медианы или квартилей в больших наборах данных. Сортировка массива позволяет быстро и эффективно найти нужные статистические показатели.

В целом, очереди по приоритетам и пирамидальная сортировка являются мощными инструментами с различными применениями в области управления ресурсами, планирования, обработки событий и сортировки данных.

## **4.2 Рассмотрение приложений, где эти структуры данных и алгоритмы могут быть полезными (например, планирование задач, управление ресурсами и т. д.)**

Очереди по приоритетам и пирамидальная сортировка могут быть полезными во множестве приложений и областей. Вот несколько примеров, где эти структуры данных и алгоритмы могут быть применены:

1. Планирование задач и управление ресурсами: Очереди по приоритетам используются для планирования и управления выполнением задач в системах, где ресурсы ограничены. Например, в операционных системах они могут использоваться для планирования выполнения процессов на процессоре с учетом их приоритетов.

2. Алгоритмы маршрутизации и обработки сетевых пакетов: В сетевых приложениях очереди по приоритетам могут использоваться для маршрутизации пакетов и обработки сетевых запросов. Более важные или приоритетные пакеты могут быть обработаны в первую очередь, обеспечивая более эффективную доставку данных.

3. Системы обработки событий: Очереди по приоритетам используются в системах обработки событий, где различные типы событий должны быть обработаны в определенном порядке. Например, в графических приложениях они могут использоваться для обработки пользовательских событий или анимаций с учетом их приоритетов.

4. Реализация планировщиков задач: Очереди по приоритетам могут быть полезными при разработке планировщиков задач, которые определяют порядок выполнения задач в многопоточных или параллельных системах. Приоритетные задачи могут быть выполнены в первую очередь, обеспечивая более эффективное использование ресурсов.

5. Сортировка данных: Пирамидальная сортировка может быть применена во многих приложениях, где требуется сортировка больших объемов данных. Например, в базах данных она может использоваться для сортировки результатов запросов или индексации данных.

6. Приоритизация задач и обработка очередей: Очереди по приоритетам могут быть использованы для приоритизации задач в системах управления задачами. Например, в планировщиках задач они могут использоваться для определения порядка выполнения задач с учетом их важности или временных ограничений.

7. Оптимизация

алгоритмов поиска: Пирамидальная сортировка может быть использована для оптимизации алгоритмов поиска, таких как поиск наилучшего совпадения или нахождение минимального/максимального элемента в наборе данных. Отсортированные данные позволяют эффективно применять бинарный поиск или другие алгоритмы поиска.

Это лишь некоторые примеры применения очередей по приоритетам и пирамидальной сортировки. В зависимости от конкретной области применения, эти структуры данных и алгоритмы могут быть адаптированы и использованы для решения различных задач эффективно и эффективно.

# **4. Заключение.**

В заключение, очереди по приоритетам и пирамидальная сортировка представляют собой важные инструменты в области структур данных и алгоритмов. Очереди по приоритетам позволяют эффективно управлять ресурсами, планировать задачи и обрабатывать события с учетом их приоритетов. Пирамидальная сортировка, в свою очередь, обеспечивает эффективную сортировку больших объемов данных.

Очереди по приоритетам предоставляют гибкую структуру для управления данными с учетом приоритетов, что позволяет решать различные задачи планирования и управления ресурсами. Они широко применяются в операционных системах, сетевых приложениях, системах обработки событий и других областях.

Пирамидальная сортировка, в свою очередь, обеспечивает эффективную сортировку данных и может быть использована в различных приложениях, где требуется упорядочение больших объемов информации. Она обладает временной сложностью O(n log n) и может быть применена для сортировки массивов любого типа данных.

Каждая из этих структур данных и алгоритмов имеет свои преимущества и недостатки, а выбор между ними зависит от конкретных требований и характеристик приложения. Очереди по приоритетам обеспечивают более гибкое управление ресурсами, в то время как пирамидальная сортировка обеспечивает эффективную сортировку данных.

В целом, понимание и применение очередей по приоритетам и пирамидальной сортировки являются важными навыками для разработчиков и инженеров, и эти инструменты могут быть применены во многих областях для улучшения производительности и эффективности работы с данными.

# **4. Список литературы.**

1. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Практическое руководство с примерами на C++ / Д. Гасфилд. — М.: Триумф, 2018. — 544 с. — ISBN 978-5-94836-301-6.
2. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — 3-е изд. — М.: Вильямс, 2013. — 1328 с. — ISBN 978-5-8459-1794-4.
3. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / Д. Кнут. — 2-е изд. — М.: Вильямс, 2007. — 832 с. — ISBN 978-5-8459-0082-3.
4. Литвинов Ю. В. Современные информационные технологии в задачах моделирования: учебное пособие / Ю. В. Литвинов. — М.: Лань, 2016. — 400 с. — ISBN 978-5-8114-1418-7.
5. Литвинов Ю. В. Алгоритмы и структуры данных: учебное пособие / Ю. В. Литвинов. — М.: ФИЗМАТЛИТ, 2019. — 592 с. — ISBN 978-5-9221-1943-4.
6. Литвинов Ю. В. Теория и практика алгоритмов: учебное пособие / Ю. В. Литвинов. — М.: ФИЗМАТЛИТ, 2020. — 704 с. — ISBN 978-5-9221-2046-1.
7. Седжвик Р. Алгоритмы на Java. Том 1. Фундаментальные алгоритмы / Р. Седжвик, К. Уэйн. — М.: ДМК Пресс, 2018. — 688 с. — ISBN 978-5-97060-715-8.