Домашняя работа по предмету Информатика

МИИГАиК 2023-ФГиИБ-ПИ-1б

Струков Артемий Викторович

02.11.2023

# Анализ структур данных

[**Анализ структур данных 1**](#_cyu6j3b1dubu)

[**Множество 2**](#_w7wxvgiekjzu)

[Основные операции 2](#_2f1rigpszy7j)

[Применение 2](#_3608qp59kl3w)

[**Массив 3**](#_xm1uvmn3vmue)

[Основные операции 3](#_ke5nckv7zq5w)

[Применение 4](#_i31f1xw986c)

[**Стек 5**](#_s33z68w44v8y)

[Основные операции 5](#_8an01g7wizlk)

[Применение 5](#_w9n5lh14kjw2)

[**Очередь 6**](#_x09vph77jhfj)

[Основные операции 6](#_6leju0k8s4j)

[Применение 6](#_16b9mb518x6t)

[**Связный список 7**](#_um1qg8dtv3uy)

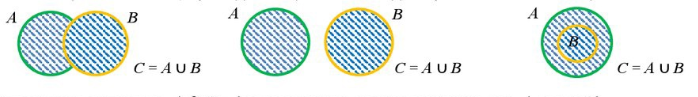
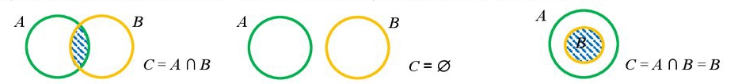
[Основные операции 7](#_h3aiiafgzeov)

[Применение 7](#_i0dnjfdkw6im)

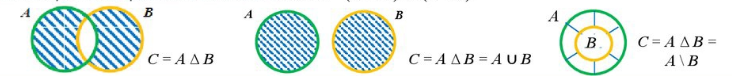
## 

## Множество

### Основные операции

* **Объединение** - составляет новое множество, элементы которого собираются из двух других разных множеств, без дубликатов. Имеет сложность O(N), где N - количество элементов во втором множестве.
* **Пересечение** - составляет новое множество, элементы которого находятся и в первом множестве, и во втором. Имеет сложность O(N)
* **Разность** - составляет новое множество с элементами из одного множества, которых нет в другом множестве. Имеет сложность O(N)



* **Симметричная разность** - составляет новое множество с элементами из двух других множеств, за исключением тех элементов, которые существуют и в первом и во втором множестве. Имеет сложность O(N)

### 

### Применение

Структура данных "множество" ("set") подходит для задач, где нужно хранить набор уникальных элементов без повторений и эффективно выполнять вышеописанные операции. Вот примеры задач, для которых множество будет полезно:

* **Удаление дубликатов**: Если нужно получить уникальные элементы из коллекции, можно использовать множество для удаления повторений.
* **Подсчет уникальных элементов**: Можно использовать множество для быстрого подсчёта количества уникальных элементов в коллекции.
* **Проверка наличия элементов**: Быстрая проверка вхождения, т.е. содержит ли множество определённый элемент. Вместо того чтобы искать элемент в оригинальной огромной коллекции, можно преобразовать ее во множество и искать элемент среди уникальных значений.

## Массив

### Основные операции

* **Индексация (чтение и запись)** - получение и изменение значений элементов в массиве происходит через обращение по индексу (индексация начинается с 0). Имеет сложность O(1)
* **Размерность** - получение размера (кол-во элементов) массива. Имеет сложность O(1)
* **Сравнение** - выявление являются ли два массива равными. Имеет сложность O(N), где N - кол-во элементов в массиве.
* **Копирование / Расширение** - элементы одного массива можно продублировать в другой, либо в изначально пустой (т.е. копирование), либо уже заполненный (т.е. расширение). Имеет сложность O(N)
* **Сортировка** - представление набора элементов в упорядоченном в виде, например от большего к меньшему. В C++ имеется функция std::sort, имеет сложность O(N log N)
* **Поиск элемента** - нахождение индекса определенного элемента в массиве. В C++ имеется функция std::find, имеет сложность O(N)

### 

### Применение

Использование массивов обусловлено необходимостью упорядоченного хранения и обработки данных, и они предоставляют эффективные средства для решения множества задач. Использование массивов имеет несколько важных преимуществ и применений в программировании:

* **Хранение и итеративность**: Массивы позволяют хранить множество элементов одного типа данных в упорядоченной последовательности и обеспечивает итерацию перебор) элементов, что полезно для обработки большого количества данных. Можно использовать циклы для последовательного обхода всех элементов массива.
* **Эффективность**: Поскольку элементы массива хранятся в памяти последовательно, доступ к ним осуществляется быстро и эффективно. Это важно при работе с большими объемами данных.
* **Многомерные массивы**: Массивы могут быть многомерными, что позволяет представлять структуры данных более сложного вида, такие как матрицы и изображения.
* **Работа с алгоритмами**: Массивы часто используются при реализации различных алгоритмов, таких как поиск и сортировка.
* **Фундаментальность**: Массивы являются одной из самых простых и широко используемых структур данных, таких как стеки и очереди.

## Стек

### Основные операции

* **Push (Вставка)** - Операция добавления элемента в верхнюю часть стека. Имеет сложность O(1)
* **Pop (Удаление)** - Операция удаления верхнего элемента из стека. Имеет сложность O(1)
* **Peek (Просмотр)** - Операция просмотра верхнего элемента стека без его удаления. Это позволяет узнать, какой элемент будет удален при следующей операции pop. Имеет сложность O(1)
* **isEmpty (Пуст ли стек)**: Проверка, пуст ли стек. Если стек пуст, то операция pop вызовет ошибку (попытку удаления из пустого стека). Имеет сложность O(1)

### 

### Применение

Стек используются в различных областях и задачах из-за своей специфической структуры и свойств. Стеки - это мощный инструмент, который может быть использован в различных областях, где важен порядок выполнения операций и временное хранение данных. Вот несколько примеров использования:

* **Инвертирование текста и скобок:** Стеки помогают при решении задач, связанных с обработкой текста и проверкой правильной вложенности скобок (например, проверка синтаксиса в выражениях).
* **Отмена и возврат действий в текстовых редакторах и графических приложениях:** Стеки используются для хранения и отслеживания истории действий, позволяя пользователям отменять и восстанавливать изменения.
* **История посещенных веб-страниц**: Браузеры используют стек для хранения истории посещенных страниц, что позволяет пользователям возвращаться к предыдущим страницам.

## Очередь

### Основные операции

* **Enqueue (добавление элемента)**: Добавление нового элемента в конец очереди. Сложность зависит от реализации очереди, если реализация очереди поддерживает быструю вставку в конец, то O(1), иначе O(N), где N - кол-во элементов в очереди.
* **Dequeue (извлечение элемента)**: Извлечение элемента из начала очереди, элементы сдвигаются к началу. Сложность зависит от реализации, если очередь поддерживает быстрое извлечение, то O(1), иначе, в том числе и во случаях расширение памяти (но это редкое событие), O(N)
* **Peek (просмотр элемента):** Просмотр элемента, находящегося в начале очереди, без его извлечения. Это позволяет узнать, какой элемент будет следующим для извлечения. Имеет сложность O(1)
* **isEmpty (проверка на пустоту)**: Проверка, является ли очередь пустой. Если очередь пуста, то она не содержит элементов. Имеет сложность O(1)
* **Size (размер очереди)**: Получение количества элементов, находящихся в очереди. Имеет сложность O(1)

### Применение

Очереди используются во множестве приложений и областей, где важен принцип "первым пришёл, первым обслужен" (FIFO). Вот несколько примеров:

* **Многопоточные приложения**. В многопоточных приложениях очереди используются для синхронизации и обмена данными между потоками.
* **Задачи**. В системах обработки задач, таких как системы управления задачами, очереди используются для упорядочения и выполнения задач.
* **Обработка данных в потоках**. Очереди позволяют упорядочить и обработать данные в потоках, гарантируя правильный порядок обработки.
* **Событийное моделирование**. Очереди используются для управления событиями и их выполнением в правильной последовательности.

## Связный список

### Основные операции

* **Вставка (Insertion)** - вставка элемента в начало (O(1)), конец (O(N) ), на определенную позицию (O(n) ).
* **Удаление (Deletion)** - удаление первого (O(1)), последнего (O(N) ), по индексу (O(N) ), по значению элемента (O(N) )
* **Итеративность** - получение элемента по его индексу в связном списке не доступно, точно также как и получение размерности и поиска по значению, поэтому используют обход всех элементов от начала пока не достигается нужный элемент, в случае с определением размерности - конечный (O(N) ).
* **Копирование (Copying)** - создание копии связанного списка (O(N) ).

### 

### Применение

Связные списки обладают уникальными характеристик и преимуществ, однако стоит отметить, что они могут иметь более высокий накладный расход на память и могут быть менее эффективными для операций случайного доступа по индексу, по сравнению с массивами. Вот некоторые примеры применения:

* **Эффективная реализация других структур данных**. Связные списки могут использоваться для создания других структур данных, таких как стеки, очереди или графы.
* **Необходимость во вставке/удалении в середине**. В случаях, когда требуется часто вставлять или удалять элементы в середине списка, связный список может быть более эффективным, чем массив, где эти операции могут потребовать сдвига большого количества элементов.
* **Циклические структуры данных**. Связные списки могут быть использованы для создания циклических структур данных, таких как циклические очереди или кольцевые списки.
* **Рекурсивные структуры**. В некоторых алгоритмах и структурах данных, например, в рекурсивных абстракциях типа бинарных деревьев, связные списки могут использоваться для реализации структур данных.