Лабораторная работа №2
Алгоритмы и абстрактные структуры данных

Цели и задачи работы: изучение алгоритмов работы с абстрактными структурами данных.

Задание к работе: Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

Методика выполнения работы:

1. Разработать алгоритм решения задачи по индивидуальному заданию.

2. Написать и отладить программу решения задачи (C, Go или Rust).

3. Протестировать работу программы на различных исходных данных.

4. По запросу преподавателя быть готовым модифицировать алгоритм и добавить операцию работы с данными.

Перечень понятий к защите лабораторной работы 2.

Понятие алгоритма. Свойства алгоритма. Алгоритм Евклида. Математический анализ алгоритма. Временная сложность алгоритма. Асимптотическая нотация. Оценки скорости роста функции. Базовые приемы при анализе программы. Основные классы алгоритмов по временной сложности. Графическое преставление скорости роста алгоритмов. Исполнители алгоритмов.

Обоснование выбора алгоритма. Тип данных. Скалярные типы данных. Структурированные типы данных. Фундаментальные структуры. Динамические структуры. Концепция абстрактного типа данных. Понятие структуры данных. Инкапсуляция. Интерфейс. Внутренняя реализация. Интерфейс абстракции. Классификация структур хранения данных. Понятие линейного списка. Последовательность формирования АТД в виде линейного списка. Классификация линейных списков (достоинства и недостатки разных видов линейных списков).

Соглашения при организации связного списка. Эффективность связанного списка (доступ, поиск, вставка, удаление).

Стек. Эффективность стека на основе связанного списка (добавление, удаление, поиск).

Очередь. Эффективность очереди (добавление, удаление).

Хеш-таблица. Эффективность хеш-таблицы (добавление, удаление).

Графы в структурах данных. Бинарное дерево. Бинарное дерево поиска. Full Binary Tree. Complete Binary Tree. Сбалансированное двоичное дерево (АВЛ-дерево). Эффективность деревьев. B-дерево.

АТД таблица, ассимптотические оценки времени поиска, поиск в линейных таблицах. Последовательный поиск. Усовершенствованный поиск с барьерным элементом. Бинарный поиск (рекурсивная реализация). Бинарный поиск (реализация итерациями). Сортировки.

Ссылки на сайты с визуализациями:

https://clck.ru/35opVK
https://clck.ru/35opWV
https://clck.ru/LS6WD

Варианты распределяются по модулю 3. Варианты 4 и далее могут заменить задачи варианта в своей категории.

Задание 1. Стек

Вариант 1

Проверка правильности скобочной последовательности.

На вход передаётся строка, в которой содержатся различные символы, включая символы скобок "({[". Необходимо реализовать алгоритм, который определяет, правильно ли вложены скобки, используя стек. Например, последовательность "({})" - верная, последовательности "({)}" и "(][)" - неверные.

Вариант 2

Обратная польская запись.

Постфиксная (или обратная польская) запись арифметического выражения — это способ записи выражений, в котором знак операции записывается после операндов. Так, операторы больше не являются неоднозначными по отношению к своим операндам. Например, запишем выражение 3+1*4 в постфиксную форму и произведем вычисление. В постфиксной форме получим выражение 314*+. Мы умножаем числа 4 и 1, и результат складываем с 3. Необходимо реализовать алгоритм, который вычисляет выражения в обратной польской записи с использованием стека.

Вариант 3

Преобразование инфиксной нотации в постфиксную.

Инфиксная запись - это способ записи выражений, в котором знак операции записывается между двух операндов, с которыми он работает. То есть, инфиксная запись является для нас привычной. Постфиксная (или обратная польская) запись арифметического выражения — это способ записи выражений, в котором знак операции записывается после операндов. Например, выражение A+B*C будет записано как ABC*+, а выражение (A+B)*C - AB+C*. Так, операторы больше не являются неоднозначными по отношению к своим операндам. Необходимо реализовать алгоритм преобразования инфиксной нотации в постфиксную.

Задание 2. Множество

Вариант 1. Разбиение множества.

Необходимо реализовать алгоритм, который должен разбить множество натуральных чисел на непересекающиеся подмножества, все из которых имеют равную сумму. Если множество не может быть поделено таким образом, вывести сообщение об ошибке.

Пример:

множество S = {4, 10, 5, 1, 3, 7}.

Необходимо в каждом подмножестве расположить элементы, сумма которых равна 15.

Получим два подмножества {4, 10, 1} и {5, 3, 7}.

Необходимо в каждом подмножестве расположить элементы, сумма которых равна 10.

Получим три подмножества {10}, {3, 7} и {4, 5, 1}.

Вариант 2. Схожие подмножества.

Необходимо реализовать алгоритм, который должен разбить множество натуральных чисел на непересекающиеся подмножества, разница между суммами которых была бы минимальна. Вывести получившиеся подмножества и разницу их сумм.

Пример:

множество S = {5, 8, 1, 14, 7}.

Получим два подмножества {5, 14} и {8, 1, 7}. Разница между их суммами - 3.

Вариант 3. Множество.

Реализовать основные операции со множеством: добавление элемента, удаление элемента, проверка наличия элемента в множестве, объединение множеств, пересечение множеств, разность множеств.

Задание 3. Массив

Вариант 1. Последовательность с максимальной суммой.

Необходимо реализовать алгоритм, который находит в круговом массиве подмассив с наибольшей суммой.

Пример:

Массив [4, -7, 1, 5, -4, 0, -3, 2, 4, 1]

Подмассив с наибольшей суммой - [2, 4, 1, 4]

Вариант 2. Сопоставление с паттерном.

Необходимо реализовать алгоритм, который сравнивает последовательность символов с шаблоном и выводит сообщение о соответствии. Учтите, что символ "?" может заменять один символ, а "*" - любую последовательность, в том числе нулевую.

Пример:

последовательность "meow@stud.nstu.ru" соответствует шаблону "*@stud.nstu.ru".

последовательность "hello" не соответствует шаблону "h?lo".

Вариант 3. Различные подмассивы.

Необходимо реализовать алгоритм, который выводит все различные подмассивы массива.

Например, S = {x, y, z}.

Результат работы алгоритма: [{}, {x}, {y}, {z}, {x, y}, {x, z}, {y, z}, {x, y, z}]

Вариант 4. Поиск суммы.

Необходимо реализовать алгоритм, который находит подмассив, сумма элементов которого равна заданному числу.

Пример:

Массив [4, -7, 1, 5, -4, 0, -3, 2, 4, 1], цель 5.

Подмассивы:

[5]

[4, 1]

```
[5, -4, 0, -3, 2, 4, 1]
[1, 5, -4, 0, -3, 2, 4]
```

Задание 4. Двоичное дерево:

Вариант 1. Проверка дерева

Необходимо реализовать алгоритм, который будет проверять, является ли дерево BST[1].

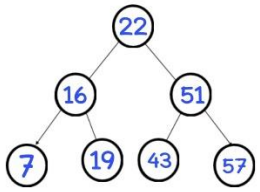
Вариант 2. Удаление узла.

Дано бинарное дерево поиска, в котором хранятся уникальные целые числа. Найдите вершину с заданным значением и удалите её из дерева так, чтобы дерево осталось корректным бинарным деревом поиска.

Вариант 3. Чтение змейкой

Реализовать алгоритм, который будет выводить содержимое дерева "змейкой". Использовать очередь для достижения O(n).

Пример:



Результат:

22 – 51 – 16 – 7 – 19 – 43 – 57

или

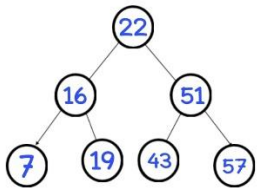
22 – 16 – 51 – 57 – 43 – 19 – 7

Вариант 4. Сумма разностей

Задано двоичное дерево. Необходимо найти сумму разностей для всех неконечных узлов.

Пример:

[1] Двоичное дерево поиска (англ. binary search tree, BST)



1 узел = 22-(16+51) = -45

2 узел = 16-(7+19) = -10

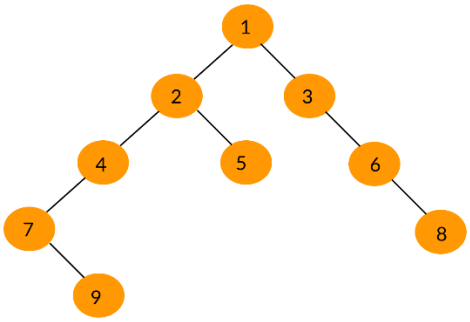
3 узел = 51-(43+57) = -49

Результат: -104

Вариант 5. K-тая максимальная сумма

Задано двоичное дерево и целочисленное число k. Необходимо найти сумму каждого уровня дерева, и вывести k-тое из них.

Пример:



, k = 4

1 уровень – 1

2 уровень – 5

3 уровень – 15

4 уровень – 15

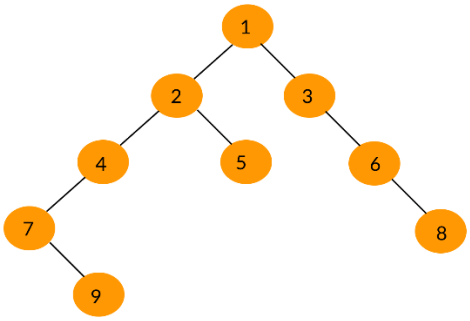
5 уровень – 9

Результат: 5.

Вариант 6. Расстояние до листьев

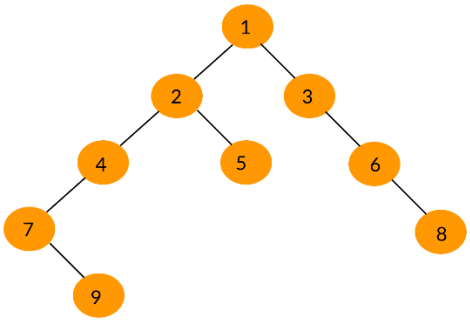
Задано двоичное дерево и целочисленное число k, которое является значение узла. Необходимо найти, какое количество уровней находится между узлом со значением k и последним уровнем его поддерева.

Пример:



, k = 2

Результат: 3.



, k = 6

Результат: 1.

Задание 5. Очередь

Вариант 1. Ход конем.

Дана шахматная доска размером NxN и позиция коня. Задается новая позиция. Необходимо вычислить и вывести кратчайший путь до этой позиции, используя очередь.

Пример:

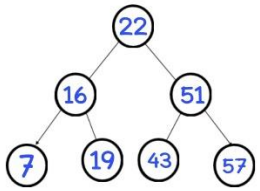
Доска 4x4, позиция (1, 4), цель (3, 1).

Результат: (1, 4) -> (3, 3) -> (1, 2) -> (3, 1).

Вариант 2. Поиск правого соседа BST.

Задано двоичное дерево. Необходимо для каждого узла вывести узел, который находится "справа" от него, используя очередь. Если соседа нет, вывести null.

Пример:

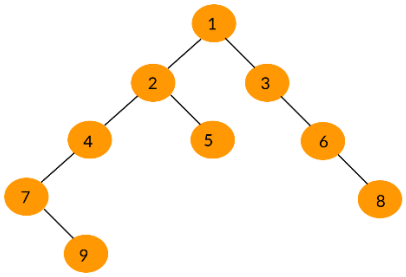


22 – null, 16 – 51, 51 – null, 7 – 19, 19 – 43, 43 – 57, 57 – null

Вариант 3. Уровни BST.

Задано двоичное дерево и два числа - уровни дерева. Необходимо вывести все узлы, которые находятся между этими уровнями, используя очередь.

Пример:



, a = 3, b = 4

Результат:

4 5 6

7 8

Задание 6. Хеш-таблица

Вариант 1. Изоморфные строки.

Необходимо реализовать алгоритм, который определяет, являются ли строки изоморфными.

Две строки a и b считаются изоморфными, если символы в a можно заменить на b.

Символ сопоставляется только одному другому, порядок символов должен сохраняться.

Пример:

"fall" и "redd" изоморфны.

"mad" и "odd" не изоморфны.

Вариант 2. Метод цепочек

Реализовать хеш-таблицу, в которой коллизии будут обрабатываться при помощи метода цепочек.

Вариант 3. Метод открытой адресации

Реализовать хеш-таблицу, в которой коллизии будут обрабатываться при помощи метода открытой адресации.

Вариант 4. Количество сотрудников

Дана таблица, содержащая сопоставление сотрудника и его менеджера в виде ряда пар (сотрудник, менеджер). Необходимо реализовать алгоритм, получающий список сотрудников, подчиненных каждому менеджеру в иерархии.

Пример:

Таблица {«A», «B»}, {«C», «B»}, {«D», «E»}, {«B», «E»}, {«E», «E»}.

Результат: A-0, B-2, C-0, D-0, E-4.

Вариант 5. Сравнение хеш-таблиц

Сравнить хеш-таблицы с различными способами разрешения коллизий: метод цепочек и метод открытой адресации.

Вариант 6. Сравнение хеш-таблиц

Сравнить хеш-таблицы с различными способами разрешения коллизий: метод цепочек и метод двойного хеширования.

Вариант 7. Сравнение хеш-таблиц

Сравнить хеш-таблицы с различными способами разрешения коллизий: метод открытой адресации и метод двойного хеширования.

Вариант 8. Фильтр Блума

Реализовать фильтр Блума.