Лабораторная работа №2

Алгоритмы и абстрактные структуры данных

Языки программирования: Rust, Go, C

Сложность: 4/10

Темы: Структуры данных, Алгоритмы, Big O нотация

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 (4 пункта). Эффективность связанного списка (доступ, поиск, вставка, удаление). 4 способа добавления элементов в список. Задача удаления элемента из списка. Вывод элементов списка.

Стек. Эффективность стека на основе связанного списка (добавление, удаление, поиск). Добавление элементов в стек. Задача удаления элемента из стека.

Очередь. Эффективность очереди (добавление, удаление).

Деревья. Определение дерева через АТД и операции. Свойство бинарного дерева. Обозначение элементов дерева. Классификация деревьев. Обходы деревьев в глубину (inorder, postorder, preorder), в ширину. Сравнение обходов деревьев DFS и BFS. Задача

поиска элемента в дереве. Задача поиска предшествующего и последующего элементов в дереве. Вставка элемента. Удаление элемента (виды, схема, псевдокод).

Бинарное дерево. Бинарное дерево поиска. Виды обхода деревьев. Full Binary Tree. Complete Binary Tree. Сбалансированное двоичное дерево (АВЛ-дерево). Эффективность деревьев (через О-нотацию). Красно-черные деревья.

Хеш-таблица. Эффективность хеш-таблицы. Виды коллизий и алгоритмы по их устранению. Формирование хеш функций методом деления, методом умножения, комбинированнм методом. Хеширование цепочками. Хеширование с открытой адресацией.

Ссылки на сайты с визуализациями:

https://clck.ru/35opVK

https://clck.ru/35opWV

https://clck.ru/LS6WD

Варианты распределяются по модулю, указанному в каждом задании.

Пример

17 – порядковый номер студента.

5 – модуль.

Номер варианта в задании 17 mod 5 = 2.

Порядковые номера студентов закреплены в таблице вариантов по распределению деревьев и изменению не подлежат.

Вариант Дерево.xlsx

СОДЕРЖАНИЕ

Задание 1. Стек 3

Задание 2. АТД Множество 6

Задание 3. Множество 7

Задание 4. Массив 10

Задание 5. Двоичное дерево 13

Задание 6. Хеш-таблица 16

Задание 7. LRU/LFU 20

Задание 1. Стек[1]

Вариант 1

Значение арифметического выражения

Задано числовое выражение. Необходимо вычислить его значение или установить, что оно содержит ошибку. В выражении могут встречаться знаки сложения, вычитания, умножения, скобки и пробелы (пробелов внутри чисел быть не должно). Приоритет операций стандартный. Все числа в выражении целые и по модулю не превосходят 2×10^9. Также гарантируется, что все промежуточные вычисления также не превосходят 2×10^9.

Вариант 2

Значение логического выражения

Задано логическое выражение. Необходимо вычислить его значение. В выражении могут встречаться знаки: ! (отрицание), & (логическое «и»), | (логическое «или»), ^ (XOR – «исключающее ИЛИ», «ровно одно из двух – истина») и скобки. Самый высокий приоритет у отрицания, меньше – у &, операции | и ^ имеют самый низкий приоритет (одинаковый) и вычисляются слева направо. Все числа в выражении либо 0, либо 1.

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.

Вариант 4

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.

---

[1] Распределение вариантов по модулю 9

Вариант 5

Проверка порядка выполнения задач

На вход подаётся последовательность задач, где каждая задача должна быть выполнена только после выполнения определённого числа других задач (например, задача зависит от выполнения другой задачи). Необходимо реализовать алгоритм, который проверяет, возможно ли завершить выполнение всех задач, учитывая зависимости (например, задачи A зависит от выполнения задачи B).

Ввод: задачи = ['A', 'B', 'C'], зависимости = [('A', 'B'), ('B', 'C')]

Вывод: Возможно

Объяснение: Сначала выполняем задачу C, затем задачу B и, наконец, задачу A

Вариант 6

Обход деревьев и графов

Стек можно использовать для реализации алгоритмов обхода деревьев и графов в глубину (Depth-First Search). При обходе каждый узел добавляется в стек, а затем извлекается, чтобы обойти его соседние узлы.

Вариант 7

XML

Формат XML является распространённым способом обмена данными между различными программами. XML-строка состоит из открывающих и закрывающих тегов. Открывающий тег начинается с открывающей угловой скобки (<), за ней следует имя тега – непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка (>). Примеры открывающих тегов: <a>, <dog>. Закрывающий тег начинается с открывающей угловой скобки, за ней следует прямой слеш (/), затем имя тега – непустая строка из строчных букв латинского алфавита, а затем закрывающая угловая скобка. Примеры закрывающихся тегов: , </dog>. XML-строка называется корректной, если она может быть получена по следующим правилам:

• Пустая строка является корректной XML-строкой.

• Если A и B – корректные XML-строки, то строка AB, получающаяся приписыванием строки B в конец строки A, также является корректной XML-строкой.

• Если A – корректная XML-строка, то строка <X> A </X>, получающаяся приписыванием в начало A открывающегося тега, а в конец – закрывающегося с таким же именем, также является корректной XML-строкой. Здесь X – любая непустая строка из строчных букв латинского алфавита.

Например, представленные ниже строки:

<a> <a> <ab> </ab> <c> </c> <a> <a> <a> – являются корректными XML-строками, а такие строки как:

<a> <a> <a> – не являются корректными XML-строками.

Иванов отправил файл с сохраненной XML-строкой по электронной почте своему коллеге Петрову. Однако, к сожалению, файл повредился в процессе пересылки: ровно один символ в строке заменился на некоторый другой символ. Требуется написать программу, которая по полученной строке, восстановит исходную XML-строку. Длина строки лежит в пределах от 7 до 1000, включительно. Строка содержит только строчные буквы латинского алфавита и символы «<», «>» и «/». Строка заканчивается переводом строки.

Вариант 8

Дан массив asteroids, представляющий собой набор астероидов в одном ряду. Каждый элемент массива содержит пару «число-направление», обозначающие размер и направление движения (влево или вправо) астероида соответственно. Все астероиды движутся с одинаковой скоростью. Если два астероида столкнутся, то меньший из них взорвется, а больший из них станет размера, равный разнице размеров астероидов. Если оба астероида имеют одинаковый размер, то взорвутся оба астероида. Астероиды, летящие в одну сторону, никогда не встретятся. Написать программу, которая возвращает новый массив астероидов после всех возможных столкновений.

Вариант 9

Вы ведете учет очков в бейсбольной игре. На вход вам дан массив operations, в котором содержатся следующие операции:

Целое положительное число X – записать новый счет, равный X;

«+» – записать новый счет, равный сумме двух предыдущих очков;

«D» – записать новый счет, который в два раза больше предыдущего значения очков;

«C» – аннулировать последнее значение очков.

На выход вернуть сумму всех очков после применения всех операций.

Вход: [5, 2, C, D, +]

Выход: 30

Задание 2. АТД Множество

Реализовать основные операции со множеством: добавление элемента, удаление элемента, проверка наличия элемента в множестве.

Запуск задания в консоли:
./<имя вашей программы> --file <путь до файла с данными> --query <запрос к файлу с данными>
Все операции выполняются за $O(1)$
Таблица операций

Тип контейнера	Добавление	Удаление	Проверка, является ли элемент частью множества
Множество	SETADD	SETDEL	SET_AT

Задание 3. Множество[2]

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

Дополнительные операции Множества

Реализовать следующие операции со множеством: объединение множеств, пересечение множеств, разность множеств.

[2] Распределение вариантов по модулю 6

Вариант 4

Максимальное количество пересечений

Даны несколько подмножеств множества натуральных чисел. Необходимо найти два множества, которые имеют максимальное количество общих элементов.

Пример:

Ввод: {{1, 2, 3}, {2, 3, 4}, {5, 6}, {3, 4, 5}}

Вывод: {{2, 3, 4}, {3, 4, 5}}, количество общих элементов: 2 (элементы 3 и 4)

Вариант 5

Инопланетный геном

Геном жителей системы Тау Кита содержит 26 видов оснований, для обозначения которых будем использовать буквы латинского алфавита от A до Z, а сам геном записывается строкой из латинских букв. Важную роль в геноме играют пары соседних оснований, например, в геноме «ABBACAB» можно выделить следующие пары оснований: AB, BB, BA, AC, CA, AB. Степенью близости одного генома другому геному называется количество пар соседних оснований первого генома, которые встречаются во втором геноме. Вам даны два генома, определите степень близости первого генома второму геному. Программа получает на вход две строки, состоящие из заглавных латинских букв. Каждая строка непустая, и её длина не превосходит 10^5. Программа должна вывести одно целое число – степень близости генома, записанного в первой строке, геному, записанному во второй строке.

Пример:

Ввод: ABBACAB

BCABB

Вывод: 4

Вариант 6

Черепахи

Три черепахи ползут по дороге. Одна черепаха говорит: «Впереди меня две черепахи». Другая черепаха говорит: «Позади меня две черепахи». Третья черепаха говорит: «Впереди меня две черепахи и позади меня две черепахи». Как такое может быть? Ответ: третья черепаха врет!

По дороге одна за другой движутся N черепах. Каждая черепаха говорит фразу вида: «Впереди меня a_i черепах, а позади меня b_i черепах». Ваша задача определить, какое максимальное количество черепах могут говорить правду.

Пример:

Ввод: 5
0 4
1 3
2 2
3 1
4 0

Вывод: 5

Ввод: 3
2 0
0 2
2 2

Вывод: 2

Задание 4. Массив[3]

Вариант 1

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 2

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]

Вариант 3

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".

[3] Распределение вариантов по модулю 6

Вариант 4

Пирамида

Для строительства двумерной пирамиды используются прямоугольные блоки, каждый из которых характеризуется шириной и высотой. Можно поставить один блок на другой, только если ширина верхнего блока строго меньше ширины нижнего (блоки нельзя поворачивать). Самым нижним в пирамиде может быть блок любой ширины. По заданному набору блоков определите, пирамиду какой наибольшей высоты можно из них построить. В первой строке входных данных задается число N – количество блоков (1≤N≤100000). В следующих N строках задаются пары натуральных чисел w_i и h_i ($1 \le w_i, h_i \le 10^9$) – ширина и высота блока соответственно.

Пример:

Ввод: 3

3 1

2 2

3 3

Вывод: 5

Вариант 5

Контрольная по ударениям

Дано N слов в словаре с правильными ударениями (заглавная буква указывает на ударение) и строка текста, в которой Петя расставил ударения. Вася проверяет этот текст: если слово есть в словаре, оно должно соответствовать хотя бы одному из вариантов правильного ударения. Если слово отсутствует в словаре, оно считается правильным, если в нём ровно одно ударение. Слово считается ошибочным, если ударений нет или их больше одного. Необходимо определить количество ошибок в тексте, найденных Васей.

Пример:

Ввод: 4

cAnnot

cannOt

fOund

pAge

thE pAge cAnnot be found

Вывод: 2

Пояснение: в слове cannot, согласно словарю, возможно два варианта расстановки ударения. Две ошибки, совершенные Петей – это слова be (ударение вообще не поставлено) и fouNd (ударение поставлено неверно). Слово thE отсутствует в словаре, но поскольку в нем Петя поставил ровно одно ударение, признается верным.

Ввод: 4

cAnnot

cannOt

fOund

pAge

The PAGE cannot be found

Вывод: 4

Пояснение: неверно расставлены ударения во всех словах, кроме The (оно отсутствует в словаре, в нем поставлено ровно одно ударение). В остальных словах либо ударные все буквы, либо не поставлено ни одного ударения.

Вариант 6

Гистограмма

Вовочке нужно построить гистограмму частоты символов в зашифрованном сообщении. Гистограмма – это набор вертикальных столбиков, где высота каждого столбика пропорциональна количеству раз, которое символ встречается в тексте. Символы выводятся под соответствующими столбиками. Дан текст сообщения, состоящий из латинских букв (прописных и строчных), цифр, знаков препинания и пробелов (Пробелы и переводы строк в гистограмме не учитываются). Выведите гистограмму: для каждого символа, кроме пробелов и переводов строк, выведите столбик из символов «#» высотой, равной количеству вхождений символа в тексте. Символы должны быть отсортированы по возрастанию их кодов.

Пример:

Ввод: Hello, world!

Вывод:

```
     #
     ##
#########
!,Hdelorw
```

Задание 5. Двоичное дерево[4]

Вариант 1

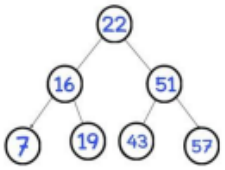
Реализовать центрированный метод обхода дерева в глубину.

Вариант 2

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 3

АВЛ–сбалансированность

Дерево называется АВЛ–сбалансированным, если для любой его вершины высота левого и правого поддерева для этой вершины различаются не более чем на 1. Вводится последовательность целых чисел, оканчивающаяся нулем. Сам ноль в последовательность не входит. Постройте дерево, соответствующее данной последовательности. Определите, является ли дерево сбалансированным, выведите слово YES или NO.

Пример:

Ввод: 7 3 2 1 9 5 4 6 8 0

Вывод: YES

Вариант 4

Вывод развилок

Для бинарного дерева поиска, построенного на заданных элементах, выведите список всех вершин, имеющих по два ребёнка, в порядке возрастания. Вводится последовательность целых чисел, оканчивающаяся нулем. Сам ноль в последовательность не входит. Постройте по этой последовательности дерево.

Пример:

[4] Распределение вариантов по модулю 8. **Вариант * для студентов с рейтингом 4-5 прошлого семестра**

Ввод: 7 3 2 1 9 5 4 6 8 0

Вывод: 3 5 7

Вариант 5

Проверка дерева

Дано бинарное дерево поиска, в котором ровно два узла были поменяны местами по ошибке. Восстановите правильное дерево, при этом, не меняя его структуру.

Вариант 6

Вывод листьев

Для бинарного дерева поиска, построенного на заданных числах, выведите список всех листьев (вершин, не имеющих потомков) в порядке возрастания.

Пример:

Ввод: 7 3 2 1 9 5 4 6 8 0

Вывод: 1 4 6 8

Вариант 7

Высота дерева

Реализуйте бинарное дерево поиска для целых чисел. Программа должна последовательно обрабатывать вводимые числа. Если очередное число есть в дереве, ничего делать не нужно. Если числа в дереве нет, нужно добавить его в соответствующее место дерева. Балансировку дерева производить не нужно. Найдите высоту получившегося дерева.

Пример:

Ввод: 7 3 2 1 9 5 4 6 8 0

Вывод: 4

Вариант 8

Глубина добавляемых элементов

В бинарное дерево поиска добавляются элементы. Выведите глубину для каждого добавленного элемента в том порядке, как они добавлялись. Если элемент уже есть в дереве, то ничего добавлять и выводить не нужно. Глубиной называется расстояние от корня дерева до элемента включительно.

Пример:

Ввод: 7 3 2 1 9 5 4 6 8 0

Вывод: 1 2 3 4 2 3 4 4 3

Вариант *

Реализовать алгоритм Хаффмана. Программа должна уметь преобразовать исходную строку в бинарный код и, наоборот, восстановить исходную строку из кода.

Теоретическая информация. Алгоритм Хаффмана – это алгоритм сжатия информации, основанный на частотах появления символов в сообщении и построении бинарного дерева поиска. Работает по следующему принципу:

1) Вычисляются частоты появления каждого символа в сообщении. Частоты сортируются по убыванию.

2) Строится список свободных узлов. Каждый узел содержит в себе символ и вес, равный частоте появления этого символа.

3) Выбираются два узла с наименьшим весом.

4) Создается родитель этих двух узлов, представляющий собой объединение их символов, с весом, равный сумме их весов.

5) Сформированный узел добавляется в список свободных узлов вместо двух его узлов-потомков.

6) Левой дуге, выходящей из родителя, присваивается бит 0, а правой – бит 1. Присвоение битов не зависит от весов потомков.

7) Повторить шаги 3-6, пока в списке не останется один узел.

Таким образом, строится таблица символов, каждый из которых обладает уникальным бинарным кодом. Расшифрование кода Хаффмана происходит путем обхода дерева от корня до листьев, выбирая соответствующие битам дуги.

Вход: abracadabra

Выход: 01110101000010010111010

Таблица:

a	b	r	c	d
0	11	101	1000	1001

Задание 6. Хеш-таблица[5]

Общее задание по вариантам

Реализовать хеш-таблицу.
Вариант 1. Открытая адресация и Метод цепочек.
Вариант 3. Двойное хеширование и Хеширование кукушки

Проведите эмпирический анализ хеширования двумя представленными методами, определите время выполнения M поисков при заданной последовательности N элементов.

Вариант 2

Метод свертки

Напишите хеш-функцию, реализующую метод свертки, который заключается в разбиении ключа на несколько частей, которые затем комбинируются так, чтобы получилось меньшее число. Полученное число используется в виде значения хеш-функции или уменьшается еще раз.

Пример:

Ввод: 523456795

Вывод: 523+456+795 = 1774.

Вариант 2

Реструктуризация

Реализуйте функцию реструктуризации хеш-таблицы в случае ее заполнения, а именно – когда таблица заполнена на 90%, создайте новую хеш-таблицу размером в два раза большим и передайте в нее все ключи. При этом для всех ключей выполняется повторное хеширование и повторная вставка в таблицу. Рекомендуется использовать библиотеку работы с большими числами.

Вариант 3

Вставка

Напишите программу, которая с использованием хеширования с цепочками вставляет N случайных чисел в таблицу размером N/100, а затем определяет длину самого короткого и самого длинного списка (N = 5000, 10000, 20000). Рекомендуется использовать библиотеку работы с большими числами.

[5] Распределение вариантов по модулю 2 и 9 соответственно

Вариант 4

Преобразование целого числа в римскую цифру

Дано целое число, необходимо преобразовать его в строку, представляющую это число в римской системе счисления. Римские цифры формируются на основе набора символов с фиксированными значениями: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000. Цифры записываются путем сложения соответствующих значений, начиная с наибольшего символа. В некоторых случаях используется форма вычитания, когда меньший символ предшествует большему. Такие случаи включают числа, оканчивающиеся на 4 или 9 (например, IV для 4 и IX для 9). Последовательные символы, представляющие степени десяти (I, X, C, M), могут повторяться не более трех раз подряд. Задача заключается в том, чтобы на основе этих правил, преобразовать данное число в римскую цифру.

Пример:

Ввод: 3749

Вывод: “MMMDCCXLIX”

Вариант 5

Оптимизация маршрутов с минимальной задержкой

В сети серверов даны маршруты между серверами с известной задержкой. Используя хеш-таблицу для хранения маршрутов и задержек, необходимо найти маршрут с минимальной задержкой между двумя заданными серверами. Алгоритм должен поддерживать динамическое добавление маршрутов и обновление задержек.

Пример:

Ввод: маршруты = {("A", "B"): 1, ("B", "C"): 2, ("A", "C"): 4}, начальный сервер = "A", конечный сервер = "C"

Вывод: минимальная задержка = 3 (маршрут "A" -> "B" -> "C")

Вариант 6

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 7

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 8

Самая длинная подстрока

Дана строка `s`. Требуется определить максимальную длину подстроки, в которой все символы различны. Длина строки может варьироваться от 0 до 50,000 символов и содержать любые символы.

Пример:

Ввод: "abcabcbb"

Вывод: 3 (abc)

Пояснение: Самая длинная подстрока без повторяющихся символов – это "abc", её длина равна 3.

Ввод: "bbbbb"

Вывод: 1 (b)

Пояснение: Самая длинная подстрока без повторяющихся символов – это "b", её длина равна 1.

Вариант 9

Фильтр Блума

Реализовать фильтр Блума.

Вариант 7

Родословная: число потомков

В генеалогическом древе у каждого человека, кроме родоначальника, есть ровно один родитель. Для каждого элемента дерева определите число всех его потомков (не считая его самого). Программа получает на вход число элементов в генеалогическом древе N. Далее следует N−1 строка, задающие родителя для каждого элемента древа, кроме родоначальника. Каждая строка имеет вид имя_потомка имя родителя. Выведите список всех элементов в лексикографическом порядке, для каждого элемента выводите количество всех его потомков и само древо.

Пример:

Ввод: 9

Alexei Peter_I

Anna Peter_I

Elizabeth Peter_I

Peter_II Alexei

Peter_III Anna

Paul_I Peter_III

Alexander_I Paul_I

Nicholaus_I Paul_I

Вывод: Alexander_I 0

Alexei 1

Anna 4

Elizabeth 0

Nicholaus_I 0

Paul_I 2

Peter_I 8

Peter_II 0

Peter_III 3

Вариант 8

Задание 7. LRU/LFU[6]

Вариант 1

LRU кэш

Разработайте структуру данных, которая работает как кэш LeastRecentlyUsed. На вход подается запрос, который может быть двух типов:

SET x y: устанавливает значение ключа x со значением y. Если кэш достигает своей емкости, должен быть аннулирован элемент, который дольше всего не запрашивался.

GET x: возвращает ключ x, если он присутствует, иначе возвращает −1.

В конструкторе класса должна быть инициализирована емкость кэша.

Пример:

cap обозначает емкость кэша, а Q − количество запросов.

cap = 2, Q = 2

Queries = SET 1 2 GET 1

Вывод: 2

cap = 2, Q = 8

Queries = SET 1 2 SET 2 3 SET 1 5 SET 4 5 SET 6 7 GET 4 SET 1 2 GET 3

Вывод: 5 −1

Cache Size = 2

SET 1 2 : 1 -> 2

SET 2 3 : 1 -> 2, 2 -> 3

SET 1 5 : 2 -> 3, 1 -> 5

SET 4 5 : 1 -> 5, 4 -> 5 (размер кэша − 2, удаление пары)

SET 6 7 : 4 -> 5, 6 -> 7

GET 4 : 5 (6 -> 7, 4->5)

SET 1 2 : 4 -> 5, 1 -> 2

GET 3 : −1

[6] Распределение вариантов по модулю 2

Вариант 2

LFU кэш

Разработайте структуру данных, которая работает как кэш LeastFrequentlyUsed. На вход подается запрос, который может быть двух типов:

SET x y: устанавливает значение ключа x со значением y. Если кэш достигает своей емкости, должен быть аннулирован элемент, к которому обращались реже всего.

GET x: возвращает ключ x, если он присутствует, иначе возвращает −1.

В конструкторе класса должна быть инициализирована емкость кэша.

Пример:

cap обозначает емкость кэша, а Q − количество запросов.

cap = 2, Q = 2

Queries = SET(5,7) SET (4,6) SET (3,5) SET (2,4) SET (1,3) GET(1) GET (2) GET (3) GET (4) GET (5)

Вывод: 3 4 5 −1 −1.

При выполнении SET(2,4), (5,7) становится недействительным. Аналогично, когда выполняется SET(1,3), (4,6) становится недействительным.