Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего профессионального образования
Национальный исследовательский институт
«Высшая школа экономики»
Московский институт электроники и математики
Компьютерная безопасность

Отчет
по лабораторной работе №1
по курсу «Методы программирования»

Выполнила:
Кокотова А. А.
СКБ212

Москва, 2024 г.

Задание

Реализовать на высокоуровневом языке программирования сортировки пузырьком, кучей и простыми вставками для массива данных о пассажирах некоторой авиакомпании: номер рейса, дата рейса, ФИО пассажира, номер места в самолете (сравнение по полям – дата рейса, номер рейса, ФИО, номер места).

Перегрузить операторы сравнения (>, <, >=, <=) для сравнения объектов.

Выбрать 7-10 наборов данных для сортировки размерности от 100 и более (но не менее 100000). Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат.

Отчет по коду

Класс `Passenger`

class `Passenger(`*name*`, `*flight_date*`, `*flight_number*`, `*seat_number*`)`

Базовые классы: `object`

Класс для хранения информации о пассажирах авиакомпании.

Параметры:

- `name` (*str*) – ФИО пассажира
- `flight_date` (*datetime*) – Дата рейса
- `flight_number` (*str*) – Номер рейса
- `seat_number` (*str*) – Номер места в самолете

Функция `bubble_sort`

```
bubble_sort(arr: list) -> None
```

Реализация алгоритма сортировки пузырьком.

Параметры:

- `arr` (*list*) – Список элементов для сортировки.

Результат:

Функция сортирует исходный список inplace.

Функция `insert_sort`

```
insert_sort(arr: list) -> None
```

Реализация алгоритма сортировки вставками.

Параметры:

- `arr` (*list*) – Список элементов для сортировки.

Результат:

Функция сортирует исходный список inplace.

Функция `heapify`

```
heapify(arr: list, n: int, root: int) -> None
```

Преобразование поддерева в двоичную кучу.

Параметры:

- `arr` (list) – Список для построения кучи.
- `n` (int) – Размер кучи.
- `root` (int) – Индекс корня поддерева.

Результат:

Функция изменяет исходный список на месте.

Функция `heapsort`

```
heapsort(arr: list) -> None
```

Реализация алгоритма пирамидальной сортировки.

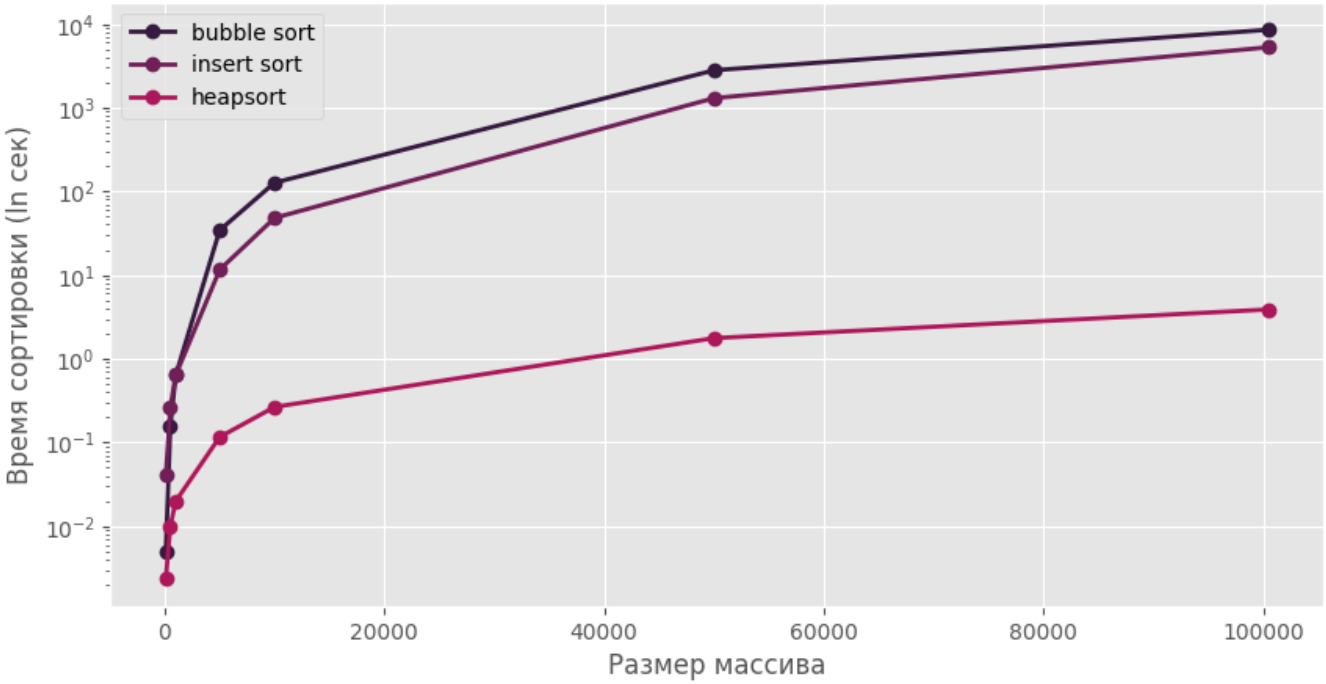
Параметры:

- `arr` (*list*) – Список элементов для сортировки.

Результат:

Функция сортирует исходный список inplace.

Сравнение времени работы сортировок



Ссылка на репозиторий с кодом

https://github.com/ktvlna/hse-programming-techniques/tree/main

Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего профессионального образования
Национальный исследовательский институт
«Высшая школа экономики»
Московский институт электроники и математики
Компьютерная безопасность

Отчет
по лабораторной работе №2
по курсу «Методы программирования»

Выполнила:
Кокотова А. А.
СКБ212

Москва, 2024 г.

Задание

Реализовать поиск заданного элемента в массиве данных о пассажирах некоторой авиакомпании следующими методами:

- с помощью бинарного дерева поиска
- с помощью красно-черного дерева
- с помощью хэш-таблицы

Для хэш таблицы необходимо реализовать хэш функцию и метод разрешения коллизий. Подсчитать число коллизий хэш функции и построить график зависимости от размерности массива.

Выполнить поиск 7-10 раз на массивах разных размерностей от 100 и более (но не менее чем до 100000). Засечь (программно) время поиска для всех способов. По полученным точкам построить сравнительные графики зависимости времени поиска от размерности массива.

Записать входные данные в ассоциативный массив multimap<key, object> и сравнить время поиска по ключу в нем с временем поиска из п.3. Добавить данные по времени поиска в ассоциативном массиве в общее сравнение с остальными способами и построить график зависимости времени поиска от размерности массива.

Отчет по коду

Класс `Node`

```
class Node(key, value=None, left=None, right=None)
```

Класс для представления узла бинарного дерева.

Параметры:

- `key` (*Any*) – Ключ узла
- `value` (*Any*) – Значение узла (по умолчанию None)
- `left` (*Node*) – Левый потомок (по умолчанию None)
- `right` (*Node*) – Правый потомок (по умолчанию None)

Класс `BinaryTree`

```
class BinaryTree(arr=None)
```

Класс для представления бинарного дерева.

Параметры:

- `arr` (*list*) – Список элементов для построения дерева (по умолчанию None)

Методы

```
insert(self, key, value=None)
```

Вставка элемента в дерево.

Параметры:

- `key` (*Any*): Ключ узла.
- `value` (*Any*): Значение узла (по умолчанию None).

`_insert(self, root, key, value)`

Вспомогательная функция для вставки элемента в дерево.

Параметры:

- `root` (*Node*): Корневой узел дерева.
- `key` (*Any*): Ключ узла.
- `value` (*Any*): Значение узла (по умолчанию None).

Результат:

- `Node`: Обновленное поддерево.

`build(self, arr: list)`

Построение бинарного дерева из списка элементов.

Параметры:

- `arr` (*list*): Список элементов для построения дерева.

`search(self, key)`

Поиск элемента в дереве по ключу.

Параметры:

- `key` (*Any*): Ключ для поиска.

Результат:

- `Any`: Значение узла, если найден, иначе None.

Класс `RBNode`

`class RBNode(key, value, parent, left, right, color)`

Класс для представления узла красно-черного дерева.

Параметры:

- `key` (*Any*) – Ключ узла
- `value` (*Any*) – Значение узла
- `parent` (*RBNode*) – Родительский узел
- `left` (*RBNode*) – Левый потомок
- `right` (*RBNode*) – Правый потомок
- `color` (*str*) – Цвет узла ('red' или 'black')

Класс `RedBlackTree`

`class RedBlackTree()`

Класс для представления красно-черного дерева.

Методы

`insert(self, key, value)`

Вставка элемента в красно-черное дерево.

Параметры:

- `key` (*Any*): Ключ узла.
- `value` (*Any*): Значение узла.

`search(self, key)`

Поиск элемента в дереве по ключу.

Параметры:

- `key` (*Any*): Ключ для поиска.

Результат:

- `Any`: Значение узла, если найден, иначе None.

`fix_insert(self, node)`

Восстановление свойств красно-черного дерева после вставки.

Параметры:

- `node` (*RBNode*): Вставленный узел.

`left_rotate(self, node)`

Левый поворот вокруг узла.

Параметры:

- `node` (*RBNode*): Узел для поворота.

`right_rotate(self, node)`

Правый поворот вокруг узла.

Параметры:

- `node` (*RBNode*): Узел для поворота.

Класс `HashTable`

```
class HashTable(size=924907)
```

Класс для представления хеш-таблицы.

Параметры:

- `size` (*int*) – Размер хеш-таблицы (по умолчанию 924907)

Методы

```
insert(self, key, value)
```

Вставка элемента в хеш-таблицу.

Параметры:

- `key` (*Any*): Ключ узла.
- `value` (*Any*): Значение узла.

```
search(self, key)
```

Поиск элемента в хеш-таблице по ключу.

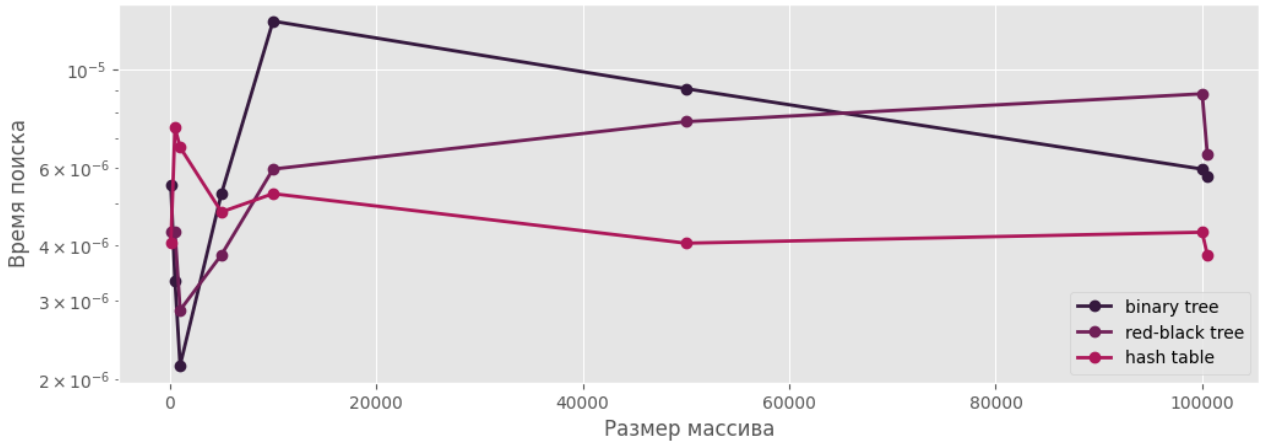
Параметры:

- `key` (*Any*): Ключ для поиска.

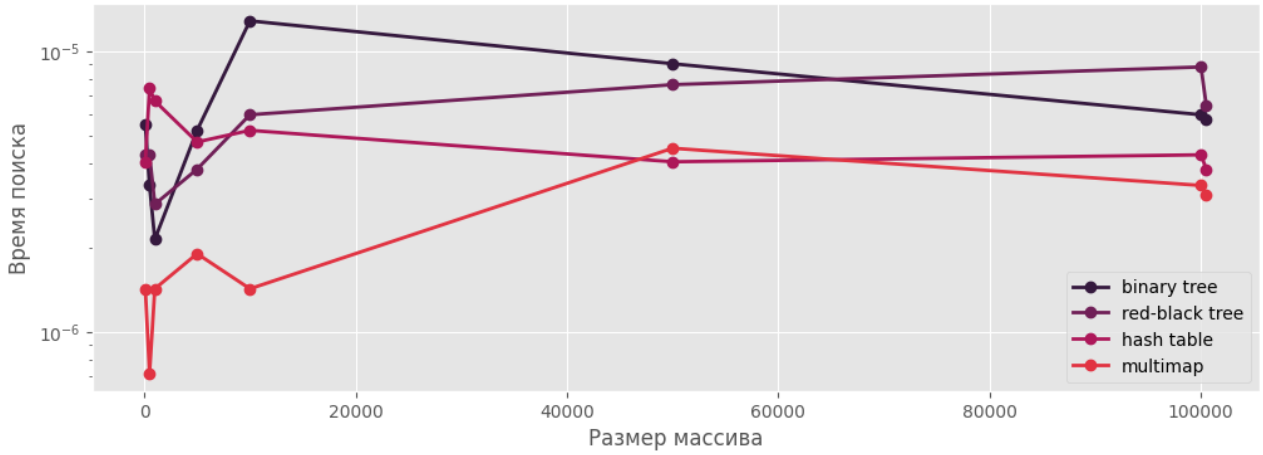
Результат:

- `Any`: Значение узла, если найден, иначе строка с сообщением о ненахождении.

Сравнение времени работы поиска



Сравнение времени работы с встроенным словарем



Ссылка на репозиторий с кодом

https://github.com/ktvlna/hse-programming-techniques/tree/main

Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего профессионального образования
Национальный исследовательский институт
«Высшая школа экономики»
Московский институт электроники и математики
Компьютерная безопасность

Отчет
по лабораторной работе №3
по курсу «Методы программирования»

Выполнила:
Кокотова А. А.
СКБ212

Москва, 2024 г.

Задание

1) Модифицировать (предложить собственные) два метода генерации псевдослучайных чисел.
2) Получить не менее 20 выборок каждым методом (диапазон чисел в каждой выборке не менее 5000) объемом не менее 100 элементов каждая.
3) Для каждой выборки посчитать среднее, отклонение и коэффициент вариации.
4) Каждую выборку проверить на равномерность распределения и случайность, используя критерий Хи-квадрат.
5) Для каждого алгоритма осуществить проверку с помощью не менее 3-х тестов NIST и/или Diehard. Сделать выводы и сравнить их с п.4.
6) Засечь время генерации чисел от тысячи до миллиона элементов обоими предложенными методами и любым стандартным методом используемого языка программирования. Построить графики сравнения скоростей в зависимости от объема выборки.

Отчет по коду

Функция `modified_lcg`

```
def modified_lcg(seed=None, m=2**31 - 1, a0=1220703125, c0=7, size=1)
```

Генератор псевдослучайных чисел, использующий модифицированный линейный конгруэнтный метод.

Параметры:

- `seed` (*int*) – Начальное значение генератора (по умолчанию текущее время).
- `m` (*int*) – Модуль (по умолчанию 2^31 - 1).
- `a0` (*int*) – Множитель (по умолчанию 1220703125).
- `c0` (*int*) – Аддитивная константа (по умолчанию 7).
- `size` (*int*) – Количество генерируемых чисел (по умолчанию 1).

Результат:

Сгенерированные псевдослучайные числа.

Тип результата:

int или *list of int*

Функция `blum_blum_shub`

```
def blum_blum_shub(seed=None, bits=13, size=1)
```

Генератор псевдослучайных чисел, использующий алгоритм Блюм-Блюм-Шуба.

Параметры:

- `seed` (*int*) – Начальное значение генератора (по умолчанию текущее время).
- `bits` (*int*) – Количество бит для генерации каждого числа (по умолчанию 13).

- `size` (*int*) – Количество генерируемых чисел (по умолчанию 1).

Результат:

Сгенерированные псевдослучайные числа.

Тип результата:

int или *list of int*

Функция `test_uniformity_chi_square`

```
def test_uniformity_chi_square(data, num_bins=10)
```

Проверка равномерности распределения с использованием критерия хи-квадрат.

Параметры:

- `data` (*list*) – выборка.
- `num_bins` (*int*) – Количество бинов для гистограммы (по умолчанию 10).

Результат:

Статистика хи-квадрат и p-значение.

Тип результата:

tuple (chi_stat, p_value)

Функция `poisson_distribution_check`

```
def poisson_distribution_check(sample, lamda=16)
```

Проверка на принадлежность выборки к распределению Пуассона с использованием критерия хи-квадрат для теста Birthday spacings.

Параметры:

- `sample` (*list*) – выборка
- `lamda` (*int*) – Параметр λ для распределения Пуассона (по умолчанию 16).

Результат:

Проверка на соответствие распределению Пуассона.

Тип результата:

bool

Функция `block_frequency`

```
def block_frequency(sample, M=10, alpha=0.01)
```

Проверка частоты блоков с использованием критерия хи-квадрат.

Параметры:

- `sample` (*list*) – Список данных для тестирования.

- `M` (*int*) – Количество блоков (по умолчанию 10).
- `alpha` (*float*) – Уровень значимости (по умолчанию 0.01).

Результат:

Проверка на равномерность распределения частот в блоках.

Тип результата:

bool

`def runs_test(bits)`

Проверка последовательности на наличие чередований с использованием теста на серии.

Параметры:

- `bits` (*list of str*) – Последовательность битов для тестирования.

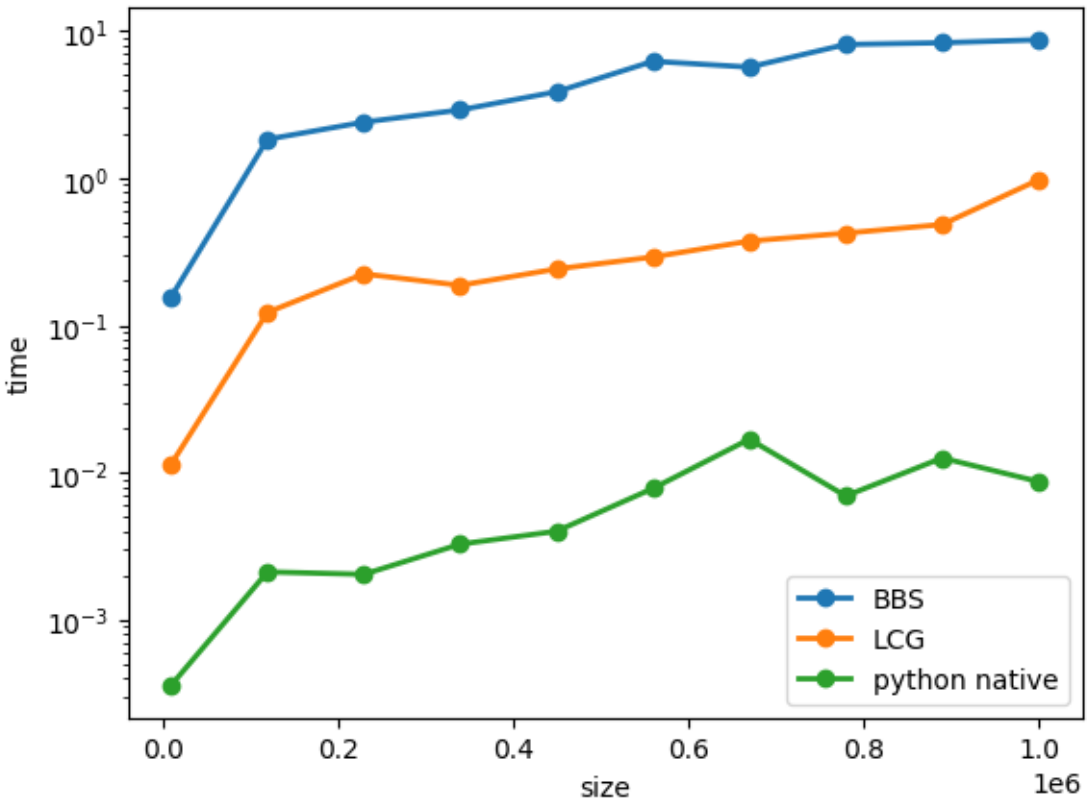
Результат:

Проверка на случайность чередований в последовательности.

Тип результата:

bool

Сравнение времени генераторов



Ссылка на репозиторий с кодом

https://github.com/ktvlna/hse-programming-techniques/tree/main

Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего профессионального образования
Национальный исследовательский институт
«Высшая школа экономики»
Московский институт электроники и математики
Компьютерная безопасность

Отчет
по лабораторной работе №4
по курсу «Методы программирования»

Выполнила:
Кокотова А. А.
СКБ212

Москва, 2024 г.

Задание

Используя паттерн Adapter (адаптер), реализовать собственную структуру типа стек неограниченного размера, основанную на типе list из стандартной библиотеки. Обязательные методы: push, pop. Также необходимо реализовать собственный класс исключений для случая извлечения элемента из пустого стека.

Отчет по коду

Класс `StackEmptyError`

```
class StackEmptyError(Exception)
```

Класс исключения для ошибки пустого стека.

Методы:

```
__str__(self)
```

- **Описание**: Метод для возврата строки сообщения об ошибке.
- **Параметры**:
  - `self`: Экземпляр класса.
- **Возвращаемое значение:**
  - Строка с сообщением об ошибке: "Stack is empty".

Класс `Stack`

```
class Stack
```

Класс, реализующий структуру данных "стек".

Методы:

- `__init__(self)`
  - **Описание**: Метод для инициализации пустого стека.
  - **Параметры**:
    - `self`: Экземпляр класса.
  - **Возвращаемое значение:**
    - None.
- `push(self, item)`
  - **Описание:** Метод для добавления элемента в стек.
  - **Параметры**:
    - `self`: Экземпляр класса.
    - `item`: Элемент, добавляемый в стек.
  - **Возвращаемое значение:**
    - None.
- `pop(self)`
  - **Описание**: Метод для удаления и возврата последнего добавленного элемента из стека. Если стек пуст, вызывается исключение `StackEmptyError`.
  - **Параметры**:
    - `self`: Экземпляр класса.
  - **Возвращаемое значение:**
    - Элемент, удаленный из стека.
  - **Исключения:**
    - `StackEmptyError`: Если стек пуст.

Ссылка на репозиторий с кодом

https://github.com/ktvlna/hse-programming-techniques/tree/main