Часть 5

5.3

Задача 3

Решение:
Алгоритм:
0. Даны два слова

1. Проверить длину слов:
   Если длины двух слов разные, они не могут быть анаграммами → вернуть False.
2. Привести слова к единому регистру (например, к нижнему):
   Это нужно, чтобы регистр букв не влиял на проверку (например, "Апельсин" и "апельсин" считаются анаграммами, если буквы совпадают).
3. Отсортировать буквы в каждом слове:
   Если после сортировки последовательности букв совпадают → слова являются анаграммами → вернуть True. Иначе → False.

Задача 4

Предложить алгоритм поиска наилучшего маршрута для пассажира метрополитена. Записать критерии, которые могут быть использованы для сравнения качества двух маршрутов
(Для машинной компьютерной программы, ток технические характеристики)

Критерии:

1. Количество станций
2. Количество пересадок
3. Время(отрезки пути, пересадка)

Алгоритм:

1. Определяем начальную и конечную станции.
2. Найти все возможные пути
3. Для каждого пути посчитать время и количество станций

4. Выбрать пути с наименьшим количеством станций при наименьшем времени. Если можно проехать за меньшее время, но количество станций будет больше, то скорее всего дело в пересадках, и нужно учесть их.
5. Вывести несколько вариантов путей.

Алгоритмом Дейкстры:

1. Определяем начальную и конечную станцию.
2. Для каждой станции записываем {Лучшее время, Количество пересадок, количество станций}. Для начальной станции все 0, для остальных ∞
3. Расчитываем параметры для соседних станций {лучшее время = нынешнее + время пути, пересадки+= пересадки, число станций +=1} при условии что из данной станции можно попасть в нужную.
4. Повторяем 3 пункт пока не дойдем до конечной с минимальными параметрами.
5. Восстанавливаем путь от конца к началу.

Задача 5

Напишите алгоритм, который находит координаты общей точки двух отрезков, если она существует.

2-мерное пространство

1. Обозначаются два отрезка A и B:
   A задан точками $A_1(x_1, y_1)$ и $A_2(x_2, y_2)$,
   B задан точками $B_1(x_3, y_3)$ и $B_2(x_4, y_4)$.
2. Вычисляются коэффициенты уравнений прямых, содержащих отрезки:

- $a_1 = y_2 - y_1$
- $b_1 = x_1 - x_2$
- $c_1 = a_1 \cdot x_1 + b_1 \cdot y_1$
- $a_2 = y_4 - y_3$
- $b_2 = x_3 - x_4$
- $c_2 = a_2 \cdot x_3 + b_2 \cdot y_3$

3. Вычисляется определитель:

- $D = a_1 \cdot b_2 - a_2 \cdot b_1$

4. Если $D = 0$, то прямые параллельны или совпадают — переход к шагу 9.
5. В противном случае находятся координаты точки пересечения прямых:

- $x = \frac{b_2 \cdot c_1 - b_1 \cdot c_2}{D}$
- $y = \frac{a_1 \cdot c_2 - a_2 \cdot c_1}{D}$

6. Проверяется, принадлежит ли точка (x, y) отрезку A:

- $\min(x_1, x_2) \leq x \leq \max(x_1, x_2)$
- $\min(y_1, y_2) \leq y \leq \max(y_1, y_2)$

7. Аналогично проверяется принадлежность точки отрезку B:

- $\min(x_3, x_4) \leq x \leq \max(x_3, x_4)$
- $\min(y_3, y_4) \leq y \leq \max(y_3, y_4)$

8. Если обе проверки выполняются, то (x, y) — точка пересечения отрезков.
9. Если $D = 0$, проверяется, лежат ли отрезки на одной прямой.
   Для этого используется условие коллинеарности векторов:

- $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) = 0$

Если равенство не выполняется — отрезки параллельны и не пересекаются.

10. Если отрезки лежат на одной прямой, проверяется пересечение их проекций:

Для оси X:

- $\max(\min(x_1, x_2), \min(x_3, x_4)) \leq \min(\max(x_1, x_2), \max(x_3, x_4))$

Для оси Y:

- $\max(\min(y_1, y_2), \min(y_3, y_4)) \leq \min(\max(y_1, y_2), \max(y_3, y_4))$

11. Если проекции пересекаются, отрезки имеют хотя бы одну общую точку.
    В качестве результата может быть выбрана начальная точка общей части.
12. Если проекции не пересекаются, общей точки нет.

5.5

1

а) Замена i-ого элемента
1сп : $a[i] := b$
2сп : $i = j \cap (N - 1)$
3сп: $i = \frac{j}{N}$ (целочисленное)
б) Удаление i-ого элемента

Один раз	Многократно
- $a[i] = a[i+1]$ до момента пока не запишем последний элемент $(i+1 < length)$ - уменьшить размер массива на 1	

в) Добавление в конце последовательности

- Увеличить на 1 размер массива
- Записать значение в $a[n+1]$ ($a[n]$)
  г) Добавление в начало последовательности
- Всем элементам меняем индекс по формуле $i = i + 1$
- В "освободившийся" 1-ый слот записываем значение
  д) Добавление после указанного
- Начиная с $i + 1$: $i = i + 1$
- В $i + 1$ записываем значение

2

Команда	Итог
Начало	
push(a)	a
push(b)	b
pop	a
push(c)	ac
push(d)	acd
pop	ac

3

Команда	Итог
Начало	
en(a)	a
en(b)	ab

Команда	Итог
de	b
en(c)	bc
en(d)	bcd
de	cd

4

Выбранная структура: Отсортированный массив с бинарным поиском

Причины выбора:

- Количество элементов фиксировано и невелико
- Все ключи уникальны и могут быть отсортированы
- Доступ к элементам за логарифмическое время O(log n)
- Минимальные накладные расходы памяти
- Простота реализации

Описание реализации:

1. **Хранение данных:**
   - Пары "ключ-значение" хранятся в виде массива структур
   - Массив поддерживается в отсортированном состоянии по ключу
   - Пример структуры:

```
states = [
    ("Alabama", "Montgomery"),
    ("Alaska", "Juneau"),
    ("Arizona", "Phoenix"),
    ...
    ("Wyoming", "Cheyenne")
]
```

2. **Операции:**

а) Поиск :

3. Выполняется бинарный поиск по отсортированному массиву
4. Сравнение ключей происходит лексикографически

5. При нахождении совпадения возвращается значение
6. Время выполнения: O(log n)

b) Вставка :
7. Находится позиция для вставки бинарным поиском
8. Если ключ уже существует - ошибка (по условию все элементы уникальны)
9. Новый элемент вставляется в массив с сохранением порядка
10. Время выполнения: O(n) из-за необходимости сдвига элементов

c) Удаление :
11. Находится позиция элемента бинарным поиском
12. Элемент удаляется из массива
13. Оставшиеся элементы сдвигаются
14. Время выполнения: O(n)

Особенности реализации:

- Массив создается один раз и остается отсортированным
- Для поиска используется стандартный алгоритм бинарного поиска
- Вставка и удаление требуют сдвига элементов, но для 50 элементов это не критично
- Минимальный расход памяти - только на хранение самих данных
- Отлично подходит для статических или редко изменяемых данных

Преимущества:

- Быстрый поиск
- Минимальные затраты памяти
- Простота реализации и отладки
- Предсказуемое время выполнения операций
- Не требует сложных структур данных