Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №1**

**ПО КУРСУ «КОМПЬЮТЕРНАЯ ГРАФИКА»**

Студент: Куценко Максим Дмитриевич

Группа: М8О–312Б-22

Преподаватель: Филиппов Г.С.

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2024.

**Тема:** Основы 2D-графики и трансформаций

**Задача:** Научиться работать с графическим API для отрисовки 2D-примитивов, освоить основные 2D-трансформации (перемещение, масштабирование, поворот) и изучить алгоритмы построения 2D-кривых. Для этого использовать на выбор С++ (OpenGL + SFML) или C# (OpenTK). Программа должна работать в реальном времени, обновляя изображение в цикле. Визуальный результат необходимо продемонстрировать на экране с возможностью управления через интерфейс.

**Вариант №12:** Построение ломаной кривой (Poly-Line) - реализовать отрисовку ломаной кривой через несколько контрольных точек, которые можно перемещать. Обеспечить динамическое обновление кривой при перемещении контрольных точек. Добавить возможность создания и удаления точек через интерфейс. Дополнительно: реализовать возможность изменения формы кривой со временем, где точки автоматически перемещаются.

**РЕШЕНИЕ**

Для выполнения данной лабораторной работы я использовал библиотеку SFML, в которой уже есть готовый инструментарий для отрисовки простых 2Д-фигур и взаимодействия с пользователем.

Программа состоит из двух циклов while, которые называют game loop и event loop. В начале game loop происходит проверка на то, не случился ли какой-нибудь прописанный пользователем event — в данной лабораторной работе это исключительно действия со стороны пользователя. Пока такие event’ы будут, программа будет выполнять действия в event loop. Далее в game loop происходит обновление экрана и отрисовка изображения по-новой.

Ломаная хранится в двух векторах — один используется для отрисовки точек, другой — для отрисовки линий.

Список ивентов: нажатие ПКМ, удерживание ЛКМ, нажатие СКМ, удерживание пробела.

При нажатии ПКМ по позиции курсора мыши создаётся точка и заносится координата для отрисовки линии.

При нажатии СКМ удаляется точка

**Метод решения**

Для выполнения задания создадим суффиксное дерево, - compact TRIE, в каждой вершине которого, являющейся рёбёнком корня, будем хранить суффиксы для данного текста из первой строки.

Дерево будем создавать по методу Укконена. Метод Укконена строит суффиксное дерево при помощи использования суффиксных ссылок - так, для вершины с входящей в неё дугой xS суффиксная ссылка ведёт в вершину с входящей в неё дугой S. По построению для каждой вершины необходимо перейти не более чем на одну вершину вверх, чтобы попасть в вершину с суффиксной ссылкой или корень.

Создание дерева Укконена делится на фазы i и продолжения j, причём каждая фаза имеет i продолжений. В каждой фазе мы берём префикс текста увеличивающейся длины, в каждом продолжении берём суффиксы уменьшающейся длины для данного префикса и заносим в дерево.

Вместо строк в дугах дерева содержатся индекс начала и конца строки в тексте. Переход вниз по дереву при построении происходит за счёт сравнения длин строк.

В каждом листе хранится общий глобальный индекс конца текущего префикса, который увеличивается на единицу, позволить мы это можем себе т.к. вершины-листы навсегда ими остаются. Благодаря глобальному индексу мы можем пропустить j\_i первых продолжений для каждой фазы, где j\_i – число продолжений в прошлой фазе, в которых не оказалось, что добавляемая строка уже оказалась в дереве.

**Описание программы**

Каждая вершина представляет из себя объект класса Node, в ней хранятся ссылки на дочерние вершины в std::map, индекс на котором добавили эту вершину (0 для не-листов), суффиксная связь, ссылка на родителя, индексы входящей в вершину дуги. Имеются функции get\_length для получения длины входящей дуги, is\_leaf для определения, лист ли это.

Дерево представляет из себя объект класса STree, в нём хранится ссылка на корень, данные по прошлому и текущему продолжению, j\_i, ссылка на последнее продолжение.

Функции get\_char, get\_child, add\_child, remove\_child имеют применение, очевидное из их названий.

Функция ends\_in\_node определяет, заканчивается ли продолжение в вершине или в середине дуги. Функция continues\_with\_char используется для определения, есть ли после данного места дальнейший путь с данным символом.

SPA – функция одной фазы, вызывает функцию SEA. Текущая фаза прерывается при применении 3-го правила продолжения (строка уже в дереве) в одном из продолжений.

Функция walk\_up используется в SEA для (возможного) перехода в вершину с имеющейся суффиксной связью, функция skip\_count позволяет найти место, где кончается текущий суффикс. Далее для фазы i смотрим, что мы не в листе и что continues\_with\_char не выполняется, тогда выполняем правило 2 – создание новой вершины. Иначе имеем правило 3 и прерываем фазу.

В правиле 2 либо создаём новую вершину, либо “разбиваем дугу”, т.е. вставляем ещё одну вершину между двумя другими.

Поиск в дереве осуществляется следующим образом: вызываем find, в find\_match убеждаемся, что строка полностью есть в дереве. Затем через поиск в глубину get\_leafs берём индексы листов, сортируем их и возвращаем вектором.

**Дневник отладки**

1) Неправильно хранили и ссылались на прошлое явное продолжение. Добавил last\_leaf.

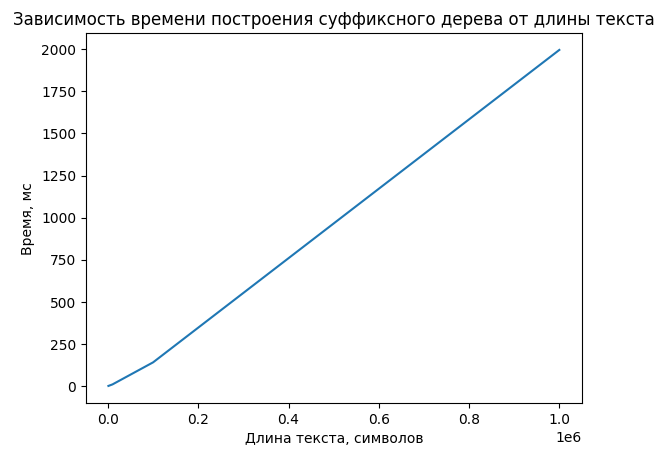
2) Неправильно осуществлялась вставка. Исправил индексы.

3) Неправильно осуществлялся поиск и переход по суффиксным ссылкам. Изменил функцию.

4) Неправильно производился вывод. Изменил его так, что для пустых строк не выводится ничего, для строк без найденных результатов не выводится ничего, но при этом оба перечисленных выше случая учитываются в индексации.

**Тест производительности**

График времени создания дерева



Тесты проводились для случайно сгенерированных величин.

Сложность создания дерева – O(m), где m – длина текста.

Сложность поиска в дереве — O(n + k), где n – длина образца, k – число вхождений образца в тексты.

**Выводы**

Изучил структуру суффиксных деревьев, построил суффиксное дерево по алгоритму Укконена.