Лабораторная работа № 4 по курсу дискретного анализа: Строковые алгоритмы

Выполнил студент группы 08-215 МАИ *Тараскаев Давид*.

Условие

Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

- Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура.
- Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые).

Метод решения

Программа реализует поиск заданного паттерна в текстовом файле с использованием алгоритма Бойера-Мура. Сначала паттерн считывается из первой строки входного файла. Затем весь текст считывается построчно. Каждое слово в паттерне и тексте преобразуется в нижний регистр и ему присваивается уникальный целочисленный идентификатор. Это позволяет ускорить операции сравнения. Для паттерна, представленного в виде последовательности идентификаторов, строятся две эвристики, используемые в алгоритме Бойера-Мура: таблица "плохого символа"и таблица "хорошего суффикса". Затем программа сканирует текст, используя эти таблицы для определения максимально возможного сдвига паттерна при несовпадении или после нахождения очередного вхождения.

Описание программы

- `toLower(string &s)`: Функция, которая преобразует все символы в переданной строке `s` к нижнему регистру.
- `splitWords(string &line)`: Функция, которая разделяет строку `line` на отдельные слова, преобразует каждое слово в нижний регистр и возвращает вектор полученных слов.
- `buildBadCharInt(const vector<int>& pat)`: Функция, которая строит таблицу сдвигов "плохого символа"для паттерна `pat`, представленного вектором целочисленных идентификаторов. Таблица хранит для каждого идентификатора из паттерна позицию его последнего вхождения.

- `buildGoodSuffixInt(const vector<int> &pat)`: Функция, которая строит таблицу сдвигов "хорошего суффикса"для паттерна `pat`. Эта таблица помогает определить, на какое расстояние можно сдвинуть паттерн, если часть его суффикса совпала с текстом, а затем произошло несовпадение.
- `unordered_map<string, int> wtid`: Ассоциативный массив для отображения строковых представлений слов в их уникальные целочисленные идентификаторы.
- `vector<int> patId`: Вектор, хранящий паттерн в виде последовательности целочисленных идентификаторов слов.
- `vector<int> textId`: Вектор, хранящий весь текст (последовательность всех слов из файла) в виде целочисленных идентификаторов.
- `vector<pair<int,int» pos`: Вектор пар, где каждый элемент хранит номер строки и номер слова в строке для соответствующего слова в `textId`.
- `unordered_map<int,int> badChar`: Таблица, используемая эвристикой "плохого символа".
- `vector<int> goodSuffix`: Таблица, используемая эвристикой "хорошего суффикса".

Дневник отладки

В ходе работы над программой поиска шаблона по алгоритму Бойера-Мура изначально была обнаружена ошибка в построении таблицы хороших суффиксов, после исправления которой программа получила вердикт OK. Однако производительность оставалась неудовлетворительной из-за высоких затрат на аллокацию векторов и сравнение строк. Для оптимизации были применены следующие меры: резервирование памяти для векторов , что сократило долю времени на реаллокацию с 33% до 24%, а также переход с посимвольного сравнения строк на сравнение их id. В результате этих изменений скорость работы программы значительно возросла, а потребление памяти стало более эффективным.

Тест производительности

Было произведено 2 теста:

- Тестирующая система на яндекс контесте:
  - Алгоритм Бойера-Мура - 2.15 секунд.
  - Наивный алгоритм - 2.531 секунд.
- Поиск шаблона из 13 слов в "Война и мир":

– Алгоритм Бойера-Мура - 0.0173 секунд.

– Наивный алгоритм - 0.0558 секунд.

Выводы

Так как средняя временная сложность алгоритма Бойера-Мура O(N), он исключительно эффективен для задач поиска последовательностей в больших объемах текстовых данных. Это делает его предпочтительным выбором для таких приложений, как функции поиска в текстовых редакторах, анализ исходного кода, поиск специфических последовательностей в биоинформатике, обнаружение известных сигнатур в системах компьютерной безопасности и быстрая фильтрация данных в объемных лог-файлах