Лабораторная работа № 4 по курсу дискретного анализа: Поиск образца в строке

Выполнил студент группы 08-208 МАИ *Левштанов Денис*.

Условие

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

2. Поиск одного образца-маски: в образце может встречаться «джокер», равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

Метод решения

Считывается первая строка входных данных - образец, делится по джокерам на подобразцы, и составляется из них бор. В каждой вершине бора содержатся: ссылки перехода в формате map<Значение, ссылка на следующую вершину>, суфиксная ссылка, ссылка выхода, является ли вершина концом слова, позиции в главном образце, оканчивающихся в этой вершине подобразцов.

После разбора всего образца, происходит «прошивка» бора, сначала «прошиваются» корень и следующие за ним вершины, затем следующие уровни по очереди.

Потом считывается весь текст в вектор <символ, <строка, позиция» и отправляется на поиск. В котором создается вектор равный размеру текста и заполняется нулями, и при нахождении какого-либо образца в этом веторе элемент под индексом [текущая позиция в тексте - позиция подобразца в образце] увеличивался на единицу. В итоге образец считается найденным в позиции i, если в масиве на позиции i, число равно количеству образцов.

Описание программы

1. TNode.h(Объявление узла и функций):

   map<uint32_t, TNode*> child; - переходы

   TNode* suffLink; - суфиксная ссылка

   TNode* exitLink; - ссылка выхода

   bool leaf; - является ли вершина концом образца

   vector<size_t> patSize; - позиции в главном образце, оканчивающихся в этой вершине подобразцов

2. TNode.cpp(Описание функций объявленных в TNode.h)

   void addPattern(TNode* root, vector<uint32_t>& pat, size_t patSize); - добавление подобразца в бор

void processTrie(TNode* root, size_t maxLen); - "прошивка"бора

void Search(TNode* root, vector< pair<int32_t, pair<size_t, size_t> > > text, size_t patLen, size_t patCount); - поиск в тесте всех вхождений образца

3. main.cpp(Считывание данных, управление программой)

Дневник отладки

Программа не проходила тест №11 из-за ошибки в алгоритме прошивки и тесты №12, 17 из-за ошибки в алгоритме поиска, решилось всё правкой алгоритмов.

Тест производительности

1. Образец длинной 1000 символов 1 подобразцов, 100 000 символов в тексте:
   Time of working: 1.05

2. Образец длинной 1000 символов 1 подобразцов, 500 000 символов в тексте:
   Time of working: 9.991

3. Образец длинной 500 символов 50 подобразцов, 500 000 символов в тексте:
   Time of working: 2.595

4. Образец длинной 500 символов 50 подобразцов, 5 000 000 символов в тексте:
   Time of working: 34.081

Выводы

Алгоритм Ахо-Корасика позволяет искать сразу несколько образцов в строке за линейную сложность, из-за чего этот алгоритм часто используется: от утилиты grep до антивирусов. Время работы также зависит от организации данных. Если таблицу переходов бора хранить как индексный массив — расход памяти O(na), вычислительная сложность O(na + H + k), где H — длина текста, n — общая длина всех слов в словаре, a — размер алфавита, k — общая длина всех совпадений. Если таблицу переходов автомата хранить как красно-чёрное дерево — расход памяти снижается до O(n), однако вычислительная сложность поднимается до O((H + n) log a + k).