Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: В. С. Епанешников
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача:

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.
3. Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

1 Описание

Алгоритм Ахо — Корасик — алгоритм поиска подстроки, разработанный Альфредом Ахо и Маргарет Корасик в 1975 году, реализует поиск множества подстрок из словаря в данной строке. Широко применяется в системном программном обеспечении, например, используется в утилите поиска grep.

Описание кода: Считывается первая строка входных данных - образец, делится по джокерам на подобразцы, и составляется из них trie. В каждой вершине структуры содержатся: ссылки перехода в формате unordered_map<Значение, ссылка на следующую вершину>, суфиксная ссылка, ссылка выхода, является ли вершина концом слова, позиции в главном образце, оканчивающихся в этой вершине подобразцов, длина подобразца. После разбора всего образца, происходит «прошивка» структуры, сначала «прошиваются» корень и следующие за ним вершины, затем следующие уровни по очереди. Далее 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.

2 Исходный код

trie.cpp	
TTrie::TTrie()	Конструктор по умол.
void TTrie::add(std::pair<std::vector<std::string>, size_t>& pat)	Добавление шаблона в trie
void TTrie::process(size_t max)	Прошивка дерева
void TTrie::deleteTrie(TNode *node)	Очистка памяти
std::vector<std::pair<size_t, size_t» TTrie::search(std::vector<std::pair<std::string, std::pair<size_t, size_t»>& text, size_t patCount, size_t patLen)	Алгоритм поиска подстроки
TTrie:: TTrie()	Деструктор

```
1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include <unordered_map>
5  #include <utility>
6  #include <vector>
7  class TTrie;
8  class TNode {
9  public:
10   friend TTrie;
11   TNode() : failLink(nullptr), exitLink(nullptr), leaf(false) {}
12 private:
13   std::unordered_map<std::string, TNode*> childs;
14   TNode *failLink, *exitLink;
15   bool leaf;
16   size_t patSize;
17   std::vector<size_t> pos;
18 };
19 class TTrie {
20 public:
21   TTrie();
22   ~TTrie();
23   void add(std::pair<std::vector<std::string>, size_t> &pat);
24   void process(size_t max);
25   std::vector<std::pair<size_t, size_t>> search(std::vector<std::pair<std::string,
         std::pair<size_t, size_t>>>& text, size_t patCount, size_t patLen);
26 private:
27   void deleteTrie(TNode* node);
28   void processLvl(TNode *node, size_t max);
29   void processNode(TNode *parent, TNode *node, std::string nodeSym);
30   TNode* root;
31 };
```

3 Консоль

```
MacBook:solution vladislove$ ./solution
cat dog cat dog bird
CAT dog CaT Dog Cat
DOG bird CAT
dog cat dog bird
1,3
2,3

MacBook:solution vladislove$ ./solution
a b ? c d e ? ? a b
a b c d e
a b c c d
e
a
b a b
2,1

MacBook:solution vladislove$ ./solution
? ? ? ? ?
MAI eto ya
MAI eto mi
MAI eto lucshie ludi strani
1,1
1,2
1,3
2,1
2,2
2,3
3,1
```

4 Тест производительности

Протестировал программу на разных количествах символов в тексте и в образце, с шаблоном и без.

Результаты:

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

5 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. Я реализовал автомат в структуре trie. Общая сложность алгоритма O(n + m + k), где n - количество символов в тексте, m - количество символов в образце, k - сумма длин всех подобразцов.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах.*

[2] *Алгоритм Ахо — Корасик Wiki.*