Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: Т. Д. Голубев
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №5

Задача: Найти образец в тексте используя статистику совпадений.

1 Описание

Требуется реализовать алгоритм Укконена для построения суффиксного дерева и статистики совпадений.

В [1] сказано: «Алгоритм Укконена для построения суффиксных деревьев за линейное время». Там же и описан этот алгоритм.

Статистика совпадений определена в [1] как: «Определим $ms(i)$ как длину наибольшей подстроки T, начинающейся с позиции i, которая совпадает где-то (но мы не знаем, где) с подстрокой P. Эти значения называются статистикой совпадений.»

Соответственно, алгоритм поиска подстроки в строке:

1. Построить суффиксное дерево из паттерна с помощью алгоритма Укконена.

2. Составить статистику совпадений.

3. Найти индексы, где статистика совпадений равна длине паттерна.

Сложность такого алгоритма – $O(n+m)$, где n – длина паттерна, m – длина текста.

2 Исходный код

main.cpp	
void CreateTree()	Метод создания дерева из строки.
void AddSuffix(int position)	Добавить в дерево i-тый символ текста.
void DestroyTree(TNode* node)	Удаление дерева.
static int CurveLength(TNode* node)	Длина текста на ребре.
void SplitCurve(TNode* node, int position)	разделение ребра (создание внутреннего узла)
void MatchingStatistic(std::vector<int>& ms, const std::string& str)	Составление статистики совпадений

```
1  const char SENTINEL = '$';
2
3  class TSuffixTree {
4  private:
5    class TNode {
6      public:
7        int begin;
8        int* end;
9        TNode* suffixLink;
10       bool isLeaf;
11       std::unordered_map<char, TNode *> children;
12
13       TNode(int start, int* finish, TNode* suffixLink, bool leaf);
14       ~TNode() = default;
15   };
16
17   struct TreeData {
18     TNode* currentNode;
19     int currentIndex;
20     int jumpCounter;
21     int plannedSuffixes;
22     TNode* lastInnerNode;
23   };
24
25
26   TNode* root;
27   std::string str;
28   int suffixTreeEnd;
29   TreeData params;
30
31   void CreateTree();
32   void AddSuffix(int position);
33   void DestroyTree(TNode* node);
34
```

```
35      static int CurveLength(TNode* node);
36      void SplitCurve(TNode* node, int position);
37
38    public:
39      TSuffixTree(std::string& input_str);
40      ~TSuffixTree();
41      void MatchingStatistic(std::vector<int>& ms, const std::string& str);
42  };
43
44  };
```

3 Консоль

```
cat-mood@nuclear-box:~/programming/mai-da-labs/lab05/build$ ./lab05_exe
abcab
abcabcabcababcabcabcababc
1
4
7
12
15
18
```

4 Тест производительности

Тест производительности представляет из себя следующее: Решение с использованием алгоритма Укконена и статистики совпадений сравнивается с наивным построением суффиксного дерева и поиска в нём. В первом тесте 5 символов в паттерне и 25 в тексте. Во втором – 100 в паттерне, 100 000 в тексте.

```
cat-mood@nuclear-box:~/programming/mai-da-labs/lab05$ ./a.out <./build/test.txt
Naive: 19 ms
Ukkonen + Matching Statistics: 9 ms

cat-mood@nuclear-box:~/programming/mai-da-labs/lab05$ ./a.out <./build/large_test.txt
Naive: 2639 ms
Ukkonen + Matching Statistics: 21 ms
```

Как видно, решение с использованием алгоритма Укконена и статистики совпадений работает быстрее.

5 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я реализовал алгоритм Укконена и статистику совпадений. В ходе работы столкнулся с массой проблем. Среди них множественные seg faults и time limits. После множественного рефакторинга кода мне удалось сделать эту лабораторную работу.

Список литературы

[1] Гасфилд Дэн Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. – СПб.: Невский Диалект; БХВ-Петербург, 2003. – 654 с.: ил.