Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №5

Студент: Шипилов К. Ю.

Преподаватель: Макаров Н. К.

Группа: М8О-303Б-22

Дата: 02.11.2024

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Постановка задачи

Реализовать поиск подстрок в тексте с использование суффиксного дерева. Суффиксное дерево можно построить за $O(n^2)$ наивным методом.

Формат ввода

Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Формат вывода

Для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

Алгоритм решения

Сначала в конец текста, поступившего на ввод добавляется символ, который не встречается в тексте - «$».

При построении суффиксного дерева с помощью наивного алгоритма выполняется последовательная всех суффиксов текста, начиная с самого длинного. При вставке каждого суффикса выполняется поиск пути из корня до первого несовпадения. Если поиск остановился в какой-либо вершине и из этой вершины не существует пути, начинающегося на текущий символ, нужно создать из этой вершины путь к новому листу, который будет содержать весь несовпавший остаток суффикса. Если поиск завершился в середине пути к какой-либо вершине, то создается новая внутренняя вершина, которая становится родителем вершины по этому пути. Совпавшая часть пути становится путем к новой вершине, а несовпавшая — к старой. К новой

внутренней вершине добавляется новый лист, путь к которому содержит несовпавший остаток суффикса. Созданным листьям присваиваются номера соответствующие номеру суффикса.

При поиске вхождений образца в текст также необходимо идти из корня по пути совпадения. Если встретилось несовпадение, то возвращается пустой вектор. Если образец закончился, то из текущей вершины необходимо сделать обход всех листьев, добавить все номера в вектор и вернуть его. Номера листьев соответствуют индексам вхождения образца в текст. Для сохранения порядка индексов, каждый номер листа вставляется в вектор как при сортировке вставкой.

Для того чтобы уложиться в ограничения по памяти пути к вершинам содержат не строки, а два числа — индексы начала и конца подстроки, соответствующей пути, в тексте.

Исходный код

main.cpp

```
#include <iostream>
#include <map>
#include <vector>

class SuffixTree {
 private:
  class Node {
   private:
    int start;
    int end;
    std::map<char, Node*> children;
    int leafNumber;
```

```
   public:
    Node(int start, int end = -1, int leafNumber = -1)
        : start(start), end(end), children(),
leafNumber(leafNumber) {}

    ~Node() noexcept {
      for (auto& child : children) {
        delete child.second;
        child.second = nullptr;
      }
    }

    void addChild(int start, int end, char firstLetter, int
leafNumber) {
      children[firstLetter] = new Node(start, end, leafNumber);
    }

    Node* split(int position, Node* parent, std::string_view
text) {
      auto newInternalNode = new Node(start, start + position);
      parent->children[text[start]] = newInternalNode;

      start += position;
      newInternalNode->children[text[start]] = this;

      return newInternalNode;
    }
```

```
int getStart() { return start; }

int getEnd() { return end; }

Node* getChild(char pathFirstLetter) {
  if (children.find(pathFirstLetter) != children.end()) {
    return children[pathFirstLetter];
  } else {
    return nullptr;
  }
}

void checkLeafs(std::vector<int>& leafsNumbers) {
  if (leafNumber < 0) {
    for (auto child : children) {
      child.second->checkLeafs(leafsNumbers);
    }
  } else if (leafsNumbers.empty()) {
    leafsNumbers.push_back(leafNumber);
  } else {
    auto it = leafsNumbers.begin();
    while (it != leafsNumbers.end() && *it < leafNumber) {
      ++it;
    }
    leafsNumbers.insert(it, leafNumber);
  }
}
```

```
  };

  Node* root;
  std::string_view text;

 public:
  SuffixTree(std::string_view text) : root(new Node{-1}),
text(text) {
    for (int i = 0; i < text.size(); ++i) {
      addSuffix(i);
    }
  }

  ~SuffixTree() noexcept { delete root; }

  void addSuffix(int suffixNumber) {
    Node* currentNode = root;
    int currentLetter = suffixNumber;
    while (true) {
      Node* nextNode = currentNode-
>getChild(text[currentLetter]);
      if (!nextNode) {
        currentNode->addChild(currentLetter, text.size(),
text[currentLetter],
                              suffixNumber);
        return;
      }
      int start = nextNode->getStart();
```

```
      int end = nextNode->getEnd();
      for (int i = start; i < end; ++i) {
        if (text[currentLetter] != text[i]) {
          nextNode = nextNode->split(i - start, currentNode,
text);
          nextNode->addChild(currentLetter, text.size(),
text[currentLetter],
                             suffixNumber);
          return;
        }
        ++currentLetter;
      }
      currentNode = nextNode;
    }
  }

  std::vector<int> find(std::string_view pattern) {
    Node* currentNode = root;
    auto currentLetter = pattern.begin();
    while (true) {
      currentNode = currentNode->getChild(*currentLetter);
      if (!currentNode) {
        return std::vector<int>{};
      }
      int start = currentNode->getStart();
      int end = currentNode->getEnd();
      for (int i = start; i < end; ++i) {
        if (*currentLetter != text[i]) {
```

```
          return std::vector<int>{};
        }
        if (++currentLetter == pattern.end()) {
          std::vector<int> leafsNumbers;
          currentNode->checkLeafs(leafsNumbers);
          return leafsNumbers;
        }
      }
    }
  }
};

int main() {
  std::string text;
  std::cin >> text;
  text += "$";
  SuffixTree suffixTree{text};
  std::string pattern;
  int i = 1;
  while (std::cin >> pattern) {
    std::vector<int> patternEntries = suffixTree.find(pattern);
    if (!patternEntries.empty()) {
      std::cout << i << ": ";
      for (int j = 0; j < patternEntries.size(); ++j) {
        if (j == 0) {
          std::cout << patternEntries[j] + 1;
        } else {
          std::cout << ", " << patternEntries[j] + 1;
```

```
        }
      }
      std::cout << "\n";
    }
    ++i;
  }
}
```

Тесты

Test 1

Input	Output
abca ab a ba	1: 1 2: 1, 4

Test 2

Input	Output
abcdabc abcd bcd bc	1: 1 2: 2 3: 2, 6

Test 3

Input	Output

abaaba aba a ba baa	1: 1, 4 2: 1, 3, 4, 6 3: 2, 5 4: 2

Вывод

Выполняя лабораторную работу я изучил наивный алгоритм построения суффиксного дерева для текста и алгоритм поиска вхождений образца в текст по суффиксному дереву. Эти алгоритмы полезны, если заранее известен текст и в нем нужно искать несколько паттернов. Наивный алгоритм построения дерева, работающий за $O(n^2)$, не так эффективен как алгоритм Укконена, работающий за линейное время, но его также можно применять, если заранее известно, что размер текста будет небольшим.