МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасика

Студент гр. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы:

Изучить и использовать на практике алгоритм Ахо-Корасика.

Индивидуализация.

Вариант 5. Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вы́резать из строки поиска все найденные образцы и вывести остаток строки поиска.

Первое задание.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст ($T, 1 \leq |T| \leq 100000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1, \dots, p_n\}$ $1 \leq |p_i| \leq 75$

Все строки содержат символы из алфавита *{A,C,G,T,N}*

Выход:

Все вхождения образцов из P в T.

Каждое вхождение образца в текст представить в виде двух чисел - i p

Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1).

Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Sample Input:

NTAG

3

TAGT

TAG

T

Sample Output:

2 2

2 3

Второе задание.

Используя реализацию точного множественного поиска, решить задачу точного поиска для одного образца с джокером.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* необходимо найти все вхождения P в текст T.

Например, образец *ab??c?* с джокером ? встречается дважды в тексте *xabvccbababcax*.

Символ джокер не входит в алфавит, символы которого используются в *T*. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {*A,C,G,T,N*}

Вход:

Текст ($T, 1 \leq |T| \leq 100000$)

Шаблон ($P, 1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

Описание алгоритма:

- *Построение бора из образцов*

Бор представляет собой дерево с символами на ребрах. Строки получаются путем прохождения дерева от корня до терминальной вершины.

Создается пустой корень, затем каждый из шаблонов добавляется посимвольно. Пока это возможно, алгоритм следует по ребрам, соответствующим символам шаблона. Если следующий переход невозможен, то для текущей вершины создается новое ребро с соответствующим символом на конце. Если образец заканчивается на текущей вершине, то она помечается терминальной.

- *Преобразование бора в автомат*

Для создания автомата необходимо обработать ситуацию, при которой переход по ребру бора невозможен, но автомат должен перейти в какое-либо состояние. Для этого вводятся суффиксные ссылки. Суффиксная ссылка для каждой вершины u — это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине u. Единственный особый случай — корень бора: для удобства суффиксную ссылку из него проведём в себя же. Заметим, что если мы хотим узнать суффиксную ссылку для некоторой вершины *v*, то мы можем перейти в предка p текущей вершины (пусть *c* — буква, по которой из *p* есть переход в *v*), затем перейти по его суффиксной ссылке, а затем из неё выполнить переход в автомате по букве *c*. Если при переходе по суффиксной ссылке алгоритм попадает в терминированную вершину, то предыдущая вершина тоже помечается терминированной. Все суффиксные ссылки можно найти обходом дерева в ширину.

- *Поиск всех слов в тексте*

Изначальное состояние автомата – корень дерева. Текст рассматривается посимвольно. Каждый следующий символ передается в автомат, где с помощью него осуществляется переход в следующее состояние по ребру бора либо, если это не возможно, по суффиксным ссылкам. При переходе к терминальной вершине, из нее возвращается информация о словах, которые оканчиваются по данному символу.

- *Поиск образца с джокером*

Входной образец разбивается на подслова, состоящие только из символов, которые не являются джокером, для которых строится бор. Затем идет посимвольный разбор текста по алгоритму Ахо-Корасика. Дополнительно для искомого слова хранятся индексы, в которых начинается каждое подслово, также создается дополнительный массив predictions размера текста, заполненный 0. Далее, при нахождении подслова в тексте в массиве predictions инкрементируется значение соответствующее началу основного слова относительно найденного подслова. Таким образом, если в массиве predictions встретится число равное количеству подслов в слове, то на этой позиции начинается искомое слово

- *Индивидуализация*

Максимальное количество дуг в боре находится во время обхода в ширину для создания суффиксных ссылок. Вырезание слов из текста осуществляется на основании информации о найденных словах.

Сложность алгоритмов

При построении бора рассматривается каждый символ каждого слова, поэтому сложность по времени будет $O(n)$, где n – сумма длин паттернов. Обход в ширину для построения суффиксных ссылок: $O(v+e) = O(n)$. Посимвольный разбор строки – $O(l)$, где l – длина строки. В итоге получаем $O(n + l)$.

Для хранения автомата требуется O(n) памяти. Дополнительно для второй задачи требуется $O(n + l)$ памяти для хранения подслов, их индексов в главном слове и предполагаемых вхождениях слова.

Функции и структуры данных

Реализованные функции:

Инициализация автомата

Сигнатура: `void init(const std::vector<std::string> &words)`

Аргументы:

- words – слова, которые нужно будет найти в тексте

Алгоритм:

- Пройтись посимвольно для каждого слова по алгоритму построения бора, создавая новые узлы или переходя по старым
- Добавить суффиксные ссылки с помощью обхода в ширину, добавляя информацию о терминальных вершинах по ссылкам

Обработка символа в автомате:

Сигнатура: `std::vector<int> process(char letter)`

Аргументы:

- letter – символ

Возвращаемое значение:

(std::vector<int>) – массив индексов слов, для которых текущий узел оказался терминальным

Алгоритм:

- Найти данный символ в узлах текущего узла, либо переходя вверх по суффиксным ссылкам
- Перейти в соответствующий узел
- Вернуть индексы слов, для которых этот узел терминальный

Поиск слов в тексте:

Сигнатура: `std::string find_all_words(const std::string &text)`

Аргументы:

- text – текст, в котором нужно найти слова

Возвращаемое значение:

(std::string) – строка, в которой вырезаны слова

Алгоритм:

- Построить автомат по словам
- Последовательно передавать каждый символ в автомат, сохраняя найденные слова
- Вырезать найденные слова и вернуть результат

Поиск слов с джокером:

Сигнатура: `std::string find_with_joker(const std::string &text)`

Аргументы:

- text – текст, в котором нужно найти слова

Возвращаемое значение:

(std::string) – строка, в которой вырезаны слова

Алгоритм:

- Выделить подслова, состоящие из символов не джокеров
- Построить автомат по подсловам
- Последовательно передавать каждый символ в автомат, учитывая найденные слова в predictions для позиции, где должно начинаться слово относительно подслова
- Вырезать найденные слова и вернуть результат

Тестирование:

Задание 1:

№	Входные данные	Вывод
1	NTAG 3 TAGT TAG T	2 2 2 3 String without words: N Max edges: 1
2	abcacdabdca 3 ac abc ca	1 2 3 3 4 1 10 3 String without words: dabd Max edges: 2
3	abababa 2 aba ab	1 1 1 2 3 1 3 2 5 1 5 2

		String without words: Max edges: 1

Задание 2:

№	Входные данные	Вывод
1	ACTANCA A$$A$ $	1 String without words: CA Max edges: 1
2	AXAXAXAXA A$A $	1 3 5 7 String without words: Max edges: 1
3	bobbibbab b** *	1 3 4 6 String without words: b Max edges: 1

Вывод:

В результате выполнения работы был изучен, реализован и применен на практике алгоритм Ахо-Корасика.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <vector>
#include <unordered_map>
#include <string>
#include <queue>
#include <iostream>
#include <algorithm>


//#define TASK1
#define TASK2
#define DEBUG

typedef std::unordered_map<char, int> node;

class AhoKorasik {
 public:
  std::vector<node> bor;
  std::vector<std::vector<int>> term;
  std::vector<int> suff_link;
  int curPos, maxEdges = 0;

  AhoKorasik() {}

  // Initialize machine
  void init(const std::vector<std::string> &words) {
    // Build bor
#ifdef DEBUG
    std::cout << "Building tree for words:";
    for (auto &word : words)
      std::cout << "  " << word << "\n";
    std::cout << "\n";
#endif
    int total = 0, word_cnt = 0;
    std::unordered_map<char, int> root;
    this->bor.push_back(root);  // root of tree
    this->term.push_back(std::vector<int>());
    this->suff_link.push_back(0);

    // reading patterns by chars
    for (auto &word : words) {
#ifdef DEBUG
      std::cout << "Current word: " << word << "\n\n";
#endif
      int cur = 0;  // start from root
      for (char c : word) {
#ifdef DEBUG
        std::cout << "Current symbol: " << c << "\n";
#endif
        if (this->bor[cur].find(c) != this->bor[cur].end()) {  // if edge for 'c'
found, go there
#ifdef DEBUG
          std::cout << "  This symbol is child of current node, go there\n";
#endif
          cur = this->bor[cur][c];
        } else {  // else add new node
#ifdef DEBUG
          std::cout << "  No child node with this symbol, creating new and go
there\n";
#endif
          node newNode;
          this->bor.push_back(newNode);
```

```
          this->term.push_back(std::vector<int>());
          total++;
          this->bor[cur][c] = total;
          cur = total;
        }
      }
#ifdef DEBUG
      std::cout << "End of word. Mark last node as terminal\n\n";
#endif
      this->term[cur].push_back(word_cnt++);  // add teminal node for current word
    }

    // All suffix links points to root by default
    for (int i = 0; i < this->bor.size(); i++) {
      this->suff_link.push_back(0);
    }

#ifdef DEBUG
    std::cout << "Creating suffix links...\n";
#endif
    // BFS to find suffix links
    std::queue<std::pair<int, node>> nodes;  // queue for bfs
    maxEdges = this->bor[0].size();
    // filling queue from root
    for (auto kv : this->bor[0]) {
      auto o = make_pair(kv.second, this->bor[kv.second]);
      nodes.push(o);
    }
#ifdef DEBUG
    std::cout << "Current queue: " << std::endl;
    auto printQueue = nodes;
    while (!printQueue.empty()) {
      std::cout << printQueue.front().first << " ";
      printQueue.pop();
    }
    std::cout << "\n";
#endif
    while (!nodes.empty()) {
      // get node from queue
      auto p = nodes.front();
      nodes.pop();
      int i = p.first;
      node curnode = p.second;

      // update maximum edges
      if (maxEdges < curnode.size())
        maxEdges = curnode.size();

#ifdef DEBUG
      std::cout << "Current node: " << i << "\n";
      std::cout << "Suffix link: " << this->suff_link[i] << "\n";
#endif

      // add next nodes to queue
      for (auto kv : curnode) {
        auto pp = make_pair(kv.second, this->bor[kv.second]);
        nodes.push(pp);
      }
      // build suffix links for children
      for (auto kv : curnode) {
        char child = kv.first;
        int pos = kv.second;
        int f = this->suff_link[i];
#ifdef DEBUG
        std::cout << "Current child: " << pos << ", " << child << "\n";
#endif
        // Find nearest suffix link of parents that has edge to child's symbol
```

```
        while (f != 0 && this->bor[f].find(child) == bor[f].end()) {
          f = this->suff_link[f];
#ifdef DEBUG
          std::cout << "No such symbol in children of current link " << f << " go
futher\n";
          std::cout << "Next link: " << f << "\n";
#endif
        }
        // create suffix link if node was found
        if (this->bor[f].find(child) != bor[f].end()) {
          f = this->bor[f][child];
        }
#ifdef DEBUG
          std::cout << "Create new suffix link: " << pos << " -> " << f << "\n";
#endif
        this->suff_link[pos] = f;
        // Add termination flag for all nodes which suffix chain leads to
        // termination node
        if (!this->term[this->suff_link[pos]].empty()) {
#ifdef DEBUG
        std::cout << "Suffix link was terminal node, add same mark to cuurent
child\n";
#endif
          for (auto item : this->term[this->suff_link[pos]]) {
            this->term[pos].push_back(item);
          }
        }
      }
    }
    this->curPos = 0;  // back to root

#ifdef DEBUG
    std::cout << "\n Built automation: \n";
    for (int i = 0; i < bor.size(); i++) {
      std::cout << "Index: " << i << "\n";
      if (!term[i].empty()) {
        std::cout << "Terminal node, indices of words: " << "\n";
        for (auto item : term[i]) {
          std::cout << item << " ";
        }
        std::cout << "\n";
      }
      std::cout << "Suffix link: " << suff_link[i] << "\n";
      if (bor[i].empty()) {
        std::cout << "No children";
      } else {
        std::cout << "Children: ";
        for (auto item : bor[i]) {
          std::cout << "{" << item.first << ", " << item.second << "} ";
        }
      }
      std::cout << "\n\n";
    }
#endif
  }

  // Process one symbol in machine
  std::vector<int> process(char letter) {
#ifdef DEBUG
    std::cout << "Searching symbol " << letter << "\n";
#endif
    // Find the matching position in machine by suffix links.
    while (this->curPos > 0 && this->bor[this->curPos].find(letter) == this-
>bor[this->curPos].end()) {
      this->curPos = this->suff_link[this->curPos];
#ifdef DEBUG
      std::cout << "No matches for index " << curPos << "\n";
```

```
      std::cout << "Go to suffix link " << curPos << "\n";
#endif
    }
    // go to node if it was found
    if (this->bor[this->curPos].find(letter) != this->bor[this->curPos].end()) {
      this->curPos = this->bor[this->curPos][letter];
#ifdef DEBUG
      std::cout << "Symbol found on index " << curPos << "\n";
#endif
    }

    // return all words that end on this node
#ifdef DEBUG
    std::cout << "Return all words which terminates on this node: " << "\n";
    for (auto item : this->term[this->curPos]) {
      std::cout << item << " ";
    }
    std::cout << "\n\n";
#endif
    return this->term[this->curPos];
  }

  //find all patterns in text + cut text
  std::string find_all_words(const std::string &text) {
    int n = 0;
    std::cin >> n;

    std::vector<std::string> words;
    for (int i = 0; i < n; i++) {
      std::string word;
      std::cin >> word;
      words.push_back(word);
    }

    init(words);

    std::vector<std::pair<int, int>> res;
    for (int pos = 0; pos < text.length(); pos++) {
      std::vector<int> pats = process(text[pos]);
      if (!pats.empty()) {
        for (auto point : pats) {
          int patternLength = words[point].length();
          res.emplace_back(std::make_pair(pos - patternLength + 2, point + 1));
        }
      }
    }

    std::string crop = text;

    std::sort(res.begin(), res.end());
    for (auto item : res) {
      for (int i = item.first - 1; i < item.first + words[item.second - 1].size() -
1; i++)
        crop[i] = '-';
      std::cout << item.first << " " << item.second << std::endl;
    }
    for (int i = 0; i < crop.size(); i++) {
      if (crop[i] == '-') {
        crop.erase(crop.begin() + i);
        i--;
      }
    }
    return crop;
  }

  std::string find_with_joker(const std::string &text) {
    std::string word;
```

```
    std::cin >> word;
    char joker;
    std::cin >> joker;

    std::vector<int> predictions(text.length(), 0);
    std::vector<std::string> patterns;
    std::vector<int> index;
    int j = 0;
    while (j < word.length()) {
      std::string small;
      int i = j;
      while (j < word.length() && word[j] != joker) {
        small += word[j];
        j++;
      }
      if (!small.empty()) {
        patterns.push_back(small);
        index.push_back(i);
      }
      j++;
    }

    init(patterns);

    for (int pos = 0; pos < text.length(); pos++) {
      std::vector<int> pats = process(text[pos]);
      if (!pats.empty()) {
        for (auto point : pats) {
          int firstLetter = pos - patterns[point].length() + 1;
          int idx = firstLetter - index[point] + 1;
          if (idx >= 0 && idx < predictions.size()) {
            predictions[firstLetter - index[point] + 1]++;
          }
        }
      }
    }

    std::string crop = text;
    for (int i = 0; i < predictions.size() - word.size() + 1; i++) {
      if (predictions[i] == patterns.size()) {
        for (int j = i - 1; j < word.size() + i - 1; j++)
          crop[j] = '-';
        std::cout << i << std::endl;
      }
    }

    for (int i = 0; i < crop.size(); i++)
      if (crop[i] == '-') {
        crop.erase(crop.begin() + i);
        i--;
      }
    return crop;
  }
};

int main() {
  std::string seq;
  std::cin >> seq;

  AhoKorasik instance;
  std::string entr;
#ifdef TASK1
  entr = instance.find_all_words(seq);
#else
  entr = instance.find_with_joker(seq);
#endif
```

```
#ifdef DEBUG
  std::cout << "\nString without words: " << entr << "\n";
  std::cout << "Max edges: " << instance.maxEdges << "\n";
#endif

  return 0;
}
```