Изображение выглядит как символ, эмблема, герб, нашивка

Контент, сгенерированный ИИ, может содержать ошибки.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Отчет по выполнению практического задания №6.2

**Тема: «**Поиск образца в тексте»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Жижикин Л.С.

группа ИКБО-50-24

**Москва 2025**

Содержание

[Ход работы 3](#_Toc210298486)

[Цель 3](#_Toc210298487)

[Формулировки задач 3](#_Toc210298488)

[Задача №1 3](#_Toc210298489)

[Алгоритм Ахо-Корасик 3](#_Toc210298490)

[Структура узла автомата 3](#_Toc210298491)

[Структура автомата и методы для его построения 4](#_Toc210298492)

[Метод поиска 5](#_Toc210298493)

[Решение задания 6](#_Toc210298494)

[Обработка входной строки 6](#_Toc210298495)

[«Подсветка» совпадений 6](#_Toc210298496)

[Функция задания 1 7](#_Toc210298497)

[Тестирование 8](#_Toc210298498)

[Содержимое файла ‘task1.txt’ 8](#_Toc210298499)

[Результат отработки программы 8](#_Toc210298500)

[Задача №2 8](#_Toc210298501)

[Алгоритм Кнута-Морриса-Пратта 8](#_Toc210298502)

[Создание LPS-массива 8](#_Toc210298503)

[Поиск 8](#_Toc210298504)

[Поиск только первого вхождения 9](#_Toc210298505)

[Решение задания 10](#_Toc210298506)

[Функция задания №2 10](#_Toc210298507)

[Вывод 11](#_Toc210298508)

[Список литературы 11](#_Toc210298509)

# Ход работы

## Цель

Освоить приёмы реализации алгоритмов поиска образца в тексте.

## Формулировки задач

1. Дано предложение, слова в котором разделены пробелами и запятыми. Распечатать те слова, которые являются обращениями других слов в этом предложении.
2. Даны две строки a и b. Требуется найти максимальную длину префикса строки a, который входит как подстрока в строку b. При этом считать, что пустая строка является подстрокой любой строки. Реализация алгоритмом Кнута-Мориса-Пратта.

## Задача №1

### Алгоритм Ахо-Корасик

#### Структура узла автомата

|  |
| --- |
| struct Node {      std::unordered\_map<char, Node\*> children;      Node\* failure\_link;      std::unordered\_set<string> dict\_links;      Node() {          this->children = std::unordered\_map<char, Node\*>();          this->failure\_link = nullptr;          this->dict\_links = std::unordered\_set<string>();      }      //  Children manipulation      bool has\_child(char key) {          return (this->children.find(key) != this->children.end());      }      Node\* get\_child(char key) {          return this->has\_child(key) ? this->children[key] : nullptr;      }      void set\_child(char key, Node\* node) {          this->children[key] = node;      }      //  Dictionary links      void add\_dict\_link(string dict\_link) {          this->dict\_links.insert(dict\_link);      }      void copy\_dict\_links(Node\* target) {          for (string dict\_link: target->dict\_links)              this->dict\_links.insert(dict\_link);      }  }; |

#### Структура автомата и методы для его построения

|  |
| --- |
| class Trie {  protected:      Node\* Root;      //  Add new pattern into the trie      void add\_pattern(string pattern) {          Node \*cur\_node = Root;          for (char key : pattern) {              if (!(cur\_node->has\_child(key)))                  cur\_node->set\_child(key, new Node());              cur\_node = cur\_node->get\_child(key);          }          cur\_node->add\_dict\_link(pattern);      }      //  Create failure and dictionary links      void InitializeLinks() {          Node \*cur\_node;          Root->failure\_link = Root;          std::queue<Node\*> queue;          //  '2nd layer' to the queue          for (const auto& [\_, c] : Root->children) {              c->failure\_link = Root;              queue.push(c);          }          while(!queue.empty()) {              cur\_node = queue.front();              queue.pop();              //  add next layer to the queue, while handling the parents              for (const auto& [key, child] : cur\_node->children) {                  queue.push(child);              /\*              Trace back the failure link by searching until we:                  a) Find the Node with (key) as a child                  b) Get into a root self-loop              \*/                  Node\* Traceback = cur\_node->failure\_link;                  while (!Traceback->has\_child(key) && (Traceback != Root))                      Traceback = Traceback->failure\_link;                  child->failure\_link =                      Traceback->get\_child(key) == nullptr                      ? Root                      : Traceback->get\_child(key);                  child->copy\_dict\_links(child->failure\_link);              }          }      }  public:      Trie(Collection patterns) {          this->Root = new Node();          for (string pattern: patterns)              this->add\_pattern(pattern);          InitializeLinks();      }  }; |

#### Метод поиска

|  |
| --- |
| std::vector<entry> search(string text) {      std::vector<entry> found{};      Node\* state = Root;      size\_t i = 0;      char c;      while (i < text.size()) {          c = text[i];          if (state->has\_child(c)) {              state = state->get\_child(c);              ++i;              if (!state->dict\_links.empty())                  for (string dict\_link : state->dict\_links)                      found.push\_back(entry{ dict\_link, i - dict\_link.size(), i });          }          else if (state == Root)              ++i;          else              state = state->failure\_link;      }      return found;  } |

### Решение задания

#### Обработка входной строки

|  |
| --- |
| const string PUNCTUATION\_SEPARATORS = ",\n\t.-:;@#~\"{}[]()!?";  const string PUNCTUATION\_CONNECTORS = "-:\_`\'";  string preprocess(string text) {      for (auto s : PUNCTUATION\_SEPARATORS)          std::replace( text.begin(), text.end(), s, ' ' );      for (auto s : PUNCTUATION\_CONNECTORS)          text.erase(std::remove(text.begin(), text.end(), s), text.end());        std::transform( text.begin(), text.end(), text.begin(),                      [](unsigned char c){ return std::tolower(c); } );      return " " + text + " ";  } |

#### «Подсветка» совпадений

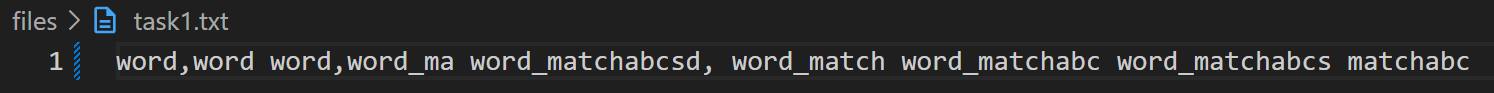
|  |
| --- |
| string highlight\_matches(string text, std::vector<std::tuple<string, size\_t, size\_t>> matches, bool highlight\_all\_prefixes=true) {      string text\_highlighted = text;      int count\_inserts = 0;      for(auto match : matches) {          string word;          size\_t start, end;            std::tie(word, start, end) = match;          /\*              Если добавить еще 1 char в рамки вхождения, то этот символ -              пробел тогда и только тогда, когда этим вхождением является              искомое слово, а не сокращение          \*/          if (text.substr(start, end - start + 1) != word + " ") {              //  word has already been higlighted if it has the right '|'              auto pos\_h = text\_highlighted.substr(start + count\_inserts, end - start + 1).find("|");              if (pos\_h != string::npos) {                  if (highlight\_all\_prefixes)                      text\_highlighted.insert(end + count\_inserts++, "\\");                  continue;              }              text\_highlighted.insert(end + count\_inserts++, "|");              text\_highlighted.insert(start + count\_inserts++, "|");          }      }      return text\_highlighted;  } |

#### Функция задания 1

|  |
| --- |
| void task\_1 () {      std::unordered\_set<string> patterns{};      std::ifstream file("./files/task1.txt");      string text, cur\_string;      getline(file, text);      //  preprocess the text      text = preprocess(text);      std::cout << "\tSearching in:\n" << text << "\n\n";      //  create set of searched prefixes      for (char c : text) {          if (c == ' ') {              if (!cur\_string.empty())                  patterns.insert(" " + cur\_string);              cur\_string = "";              continue;          }          cur\_string.push\_back(c);      } if (!cur\_string.empty())          patterns.insert(" " + cur\_string);      //  search      Trie trie = Trie(patterns);      auto matches = trie.search(text);      const bool HIGHLIGHT\_ALL = true;      std::cout << "\tHighlighted prefixes:\n" << highlight\_matches(text, matches, HIGHLIGHT\_ALL) << "\n";  } |

### Тестирование

#### Содержимое файла ‘task1.txt’



#### Результат отработки программы

A screen shot of a computer program

AI-generated content may be incorrect.

## Задача №2

### Алгоритм Кнута-Морриса-Пратта

#### Создание LPS-массива

|  |
| --- |
| std::vector<size\_t> computeLPSArray(string pattern) {      int saved\_len = 0;      std::vector<size\_t> LPS(pattern.length());      int i = 1;      while (i < pattern.length()) {          if (pattern[i] == pattern[saved\_len])              LPS[i++] = ++saved\_len;          else {              if (saved\_len)                  saved\_len = LPS[saved\_len - 1];              else                  LPS[i++] = 0;          }      }      return LPS;  } |

#### Поиск

|  |
| --- |
| std::vector<std::tuple<string, size\_t, size\_t>> KMP(string pattern, string text) {      int m = pattern.length();      int n = text.length();      std::vector<std::tuple<string, size\_t, size\_t>> ans {std::tuple{"", 0, 0}};      std::vector<size\_t> LPS = computeLPSArray(pattern);        size\_t i = 0, j = 0;      string cur\_found = "";      while (i < n) {          if (pattern[j] == text[i]) {              cur\_found.push\_back(pattern[j]);              i++;              j++;          }          if (j == m) {              ans.push\_back(std::tuple{cur\_found, i - j, i});              j = LPS[j - 1];              cur\_found = "";          }          else if (i < n && pattern[j] != text[i]) {              if (j != 0) {                  j = LPS[j - 1];                  cur\_found = "";              }              else                  i++;          }      }      return ans;  } |

#### Поиск только первого вхождения

|  |
| --- |
| std::tuple<string, size\_t, size\_t> KMP\_first(string pattern, string text) {      int m = pattern.length();      int n = text.length();      std::vector<size\_t> LPS = computeLPSArray(pattern);      size\_t i = 0, j = 0;      string cur\_found = "";      while (i < n) {          if (pattern[j] == text[i]) {              cur\_found.push\_back(pattern[j]);              i++;              j++;          }          if (j == m)              return (std::tuple{cur\_found, i - j, i});          else if (i < n && pattern[j] != text[i]) {              if (j != 0) {                  j = LPS[j - 1];                  cur\_found = "";              }              else                  i++;          }      }      return std::tuple{ "", 0, 0 };  } |

### Решение задания

#### Функция задания №2

|  |
| --- |
| void task\_2() {      const int CASES = 3;      string a\_c[CASES] = {"isddkhuf", "scvbrressdg", "aaaaa"};      string b\_c[CASES] = {"dfposgofdgisddkjm", "scvhfjsscvbrrj21", "babaabaaaabaaabaaaaab"};      for (int i = 0; i < CASES; ++i) {          string a = a\_c[i], b = b\_c[i];          string word;          size\_t start, end;          for (int i = a.length(); i >= 0; --i) {              std::tie(word, start, end) = KMP\_first(a.substr(0, i), b);              if (word != "") {                  std::cout << "found: '" << word << "' with length " << i << "\n";                  b.insert(end, "|");                  b.insert(start, "|");                  std::cout << b << "\n";                  break;              }              if (i == 0) {                  std::cout << "found: '" << word << "' with length " << i << "\n";                  b.insert(end, "|");                  b.insert(start, "|");                  std::cout << b << "\n";              }          }          std::cout << std::endl;      }  } |

### Тестирование

#### Результат отработки программы

A screen shot of a computer

AI-generated content may be incorrect.

# Вывод

В ходе данной работы я освоил приёмы хеширования и эффективного поиска элементов множества.

# Список литературы

1. Документация по языку С++ [Электронный ресурс]. URL: https://cppreference. com/ (дата обращения: 11.09.2025).

2. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения: 11.09.2025).

3. Керниган Б., Ритчи Д. Язык программирования Си / Керниган Б., Ритчи Д., 3- е изд., 1988.

4. Страуструп Б. Программирование. Принципы и практика с использованием C+ + / Страуструп Б., 2-е изд., 2016.