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

1. struct Node {

2.     std::unordered\_map<char, Node\*> children;

3.     Node\* failure\_link;

4.     std::unordered\_set<string> dict\_links;

5.     Node() {

6.         this->children = std::unordered\_map<char, Node\*>();

7.         this->failure\_link = nullptr;

8.         this->dict\_links = std::unordered\_set<string>();

9.     }

10.

11.     //  Children manipulation

12.     bool has\_child(char key) {

13.         return (this->children.find(key) != this->children.end());

14.     }

15.     Node\* get\_child(char key) {

16.         return this->has\_child(key) ? this->children[key] : nullptr;

17.     }

18.     void set\_child(char key, Node\* node) {

19.         this->children[key] = node;

20.     }

21.

22.     //  Dictionary links

23.     void add\_dict\_link(string dict\_link) {

24.         this->dict\_links.insert(dict\_link);

25.     }

26.     void copy\_dict\_links(Node\* target) {

27.         for (string dict\_link: target->dict\_links)

28.             this->dict\_links.insert(dict\_link);

29.     }

30. };

#### Структура автомата и методы для его построения

1. class Trie {

2. protected:

3.     Node\* Root;

4.     //  Add new pattern into the trie

5.     void add\_pattern(string pattern) {

6.         Node \*cur\_node = Root;

7.         for (char key : pattern) {

8.             if (!(cur\_node->has\_child(key)))

9.                 cur\_node->set\_child(key, new Node());

10.             cur\_node = cur\_node->get\_child(key);

11.         }

12.         cur\_node->add\_dict\_link(pattern);

13.     }

14.     //  Create failure and dictionary links

15.     void InitializeLinks() {

16.         Node \*cur\_node;

17.         Root->failure\_link = Root;

18.         std::queue<Node\*> queue;

19.         //  '2nd layer' to the queue

20.         for (const auto& [\_, c] : Root->children) {

21.             c->failure\_link = Root;

22.             queue.push(c);

23.         }

24.

25.         while(!queue.empty()) {

26.             cur\_node = queue.front();

27.             queue.pop();

28.             //  add next layer to the queue, while handling the parents

29.             for (const auto& [key, child] : cur\_node->children) {

30.                 queue.push(child);

31.

32.             /\*

33.             Trace back the failure link by searching until we:

34.                 a) Find the Node with (key) as a child

35.                 b) Get into a root self-loop

36.             \*/

37.                 Node\* Traceback = cur\_node->failure\_link;

38.                 while (!Traceback->has\_child(key) && (Traceback != Root))

39.                     Traceback = Traceback->failure\_link;

40.

41.                 child->failure\_link =

42.                     Traceback->get\_child(key) == nullptr

43.                     ? Root

44.                     : Traceback->get\_child(key);

45.                 child->copy\_dict\_links(child->failure\_link);

46.             }

47.         }

48.     }

49.

50. public:

51.     Trie(Collection patterns) {

52.         this->Root = new Node();

53.         for (string pattern: patterns)

54.             this->add\_pattern(pattern);

55.         InitializeLinks();

56.     }

57. };

#### Метод поиска

1. std::vector<entry> search(string text) {

2.     std::vector<entry> found{};

3.     Node\* state = Root;

4.     size\_t i = 0;

5.     char c;

6.     while (i < text.size()) {

7.         c = text[i];

8.         if (state->has\_child(c)) {

9.             state = state->get\_child(c);

10.             ++i;

11.

12.             if (!state->dict\_links.empty())

13.                 for (string dict\_link : state->dict\_links)

14.                     found.push\_back(entry{ dict\_link, i - dict\_link.size(), i });

15.         }

16.         else if (state == Root)

17.             ++i;

18.         else

19.             state = state->failure\_link;

20.     }

21.     return found;

22. }

### Решение задания

#### Обработка входной строки

1. const string PUNCTUATION\_SEPARATORS = ",\n\t.-:;@#~\"{}[]()!?";

2. const string PUNCTUATION\_CONNECTORS = "-:\_`\'";

3. string preprocess(string text) {

4.     for (auto s : PUNCTUATION\_SEPARATORS)

5.         std::replace( text.begin(), text.end(), s, ' ' );

6.

7.     for (auto s : PUNCTUATION\_CONNECTORS)

8.         text.erase(std::remove(text.begin(), text.end(), s), text.end());

9.

10.     std::transform( text.begin(), text.end(), text.begin(),

11.                     [](unsigned char c){ return std::tolower(c); } );

12.     return " " + text + " ";

13. }

#### «Подсветка» совпадений

1. string highlight\_matches(string text, std::vector<std::tuple<string, size\_t, size\_t>> matches, bool highlight\_all\_prefixes=true) {

2.     string text\_highlighted = text;

3.     int count\_inserts = 0;

4.     for(auto match : matches) {

5.         string word;

6.         size\_t start, end;

7.

8.         std::tie(word, start, end) = match;

9.         /\*

10.             Если добавить еще 1 char в рамки вхождения, то этот символ -

11.             пробел тогда и только тогда, когда этим вхождением является

12.             искомое слово, а не сокращение

13.         \*/

14.         if (text.substr(start, end - start + 1) != word + " ") {

15.             //  word has already been higlighted if it has the right '|'

16.             auto pos\_h = text\_highlighted.substr(start + count\_inserts, end - start + 1).find("|");

17.             if (pos\_h != string::npos) {

18.                 if (highlight\_all\_prefixes)

19.                     text\_highlighted.insert(end + count\_inserts++, "\\");

20.                 continue;

21.             }

22.             text\_highlighted.insert(end + count\_inserts++, "|");

23.             text\_highlighted.insert(start + count\_inserts++, "|");

24.         }

25.     }

26.     return text\_highlighted;

27. }

28.

#### Функция задания 1

1. void task\_1 () {

2. std::unordered\_set<string> patterns{};

3. std::ifstream file("./files/task1.txt");

4. string text, cur\_string;

5. getline(file, text);

6. // preprocess the text

7. text = preprocess(text);

8. std::cout << "\tSearching in:\n" << text << "\n\n";

9. // create set of searched prefixes

10. for (char c : text) {

11. if (c == ' ') {

12. if (!cur\_string.empty())

13. patterns.insert(" " + cur\_string);

14. cur\_string = "";

15. continue;

16. }

17. cur\_string.push\_back(c);

18. } if (!cur\_string.empty())

19. patterns.insert(" " + cur\_string);

20. // search

21. Trie trie = Trie(patterns);

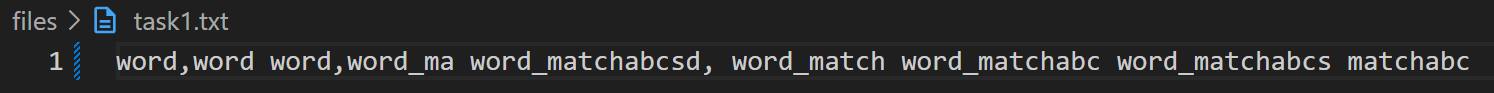
22. auto matches = trie.search(text);

23. const bool HIGHLIGHT\_ALL = true;

24. 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

1. std::vector<size\_t> computeLPSArray(string pattern) {

2.     int saved\_len = 0;

3.     std::vector<size\_t> LPS(pattern.length());

4.

5.     int i = 1;

6.     while (i < pattern.length()) {

7.         if (pattern[i] == pattern[saved\_len])

8.             LPS[i++] = ++saved\_len;

9.         else {

10.             if (saved\_len)

11.                 saved\_len = LPS[saved\_len - 1];

12.             else

13.                 LPS[i++] = 0;

14.         }

15.     }

16.     return LPS;

17. }

#### Поиск

1. std::vector<std::tuple<string, size\_t, size\_t>> KMP(string pattern, string text){

2.     int m = pattern.length();

3.     int n = text.length();

4.

5.     std::vector<std::tuple<string, size\_t, size\_t>> ans {std::tuple{"", 0, 0}};

6.

7.     std::vector<size\_t> LPS = computeLPSArray(pattern);

8.

9.     size\_t i = 0, j = 0;

10.     string cur\_found = "";

11.     while (i < n) {

12.         if (pattern[j] == text[i]) {

13.             cur\_found.push\_back(pattern[j]);

14.             i++;

15.             j++;

16.         }

17.

18.         if (j == m) {

19.             ans.push\_back(std::tuple{cur\_found, i - j, i});

20.             j = LPS[j - 1];

21.             cur\_found = "";

22.         }

23.

24.         else if (i < n && pattern[j] != text[i]) {

25.             if (j != 0) {

26.                 j = LPS[j - 1];

27.                 cur\_found = "";

28.             }

29.             else

30.                 i++;

31.         }

32.     }

33.     return ans;}

#### Поиск только первого вхождения

1. std::tuple<string, size\_t, size\_t> KMP\_first(string pattern, string text) {

2.     int m = pattern.length();

3.     int n = text.length();

4.     std::vector<size\_t> LPS = computeLPSArray(pattern);

5.

6.     size\_t i = 0, j = 0;

7.     string cur\_found = "";

8.     while (i < n) {

9.         if (pattern[j] == text[i]) {

10.             cur\_found.push\_back(pattern[j]);

11.             i++;

12.             j++;

13.         }

14.

15.         if (j == m)

16.             return (std::tuple{cur\_found, i - j, i});

17.

18.         else if (i < n && pattern[j] != text[i]) {

19.             if (j != 0) {

20.                 j = LPS[j - 1];

21.                 cur\_found = "";

22.             }

23.             else

24.                 i++;

25.         }

26.     }

27.     return std::tuple{ "", 0, 0 };

28. }

### Решение задания

#### Функция задания №2

1. void task\_2() {

2. const int CASES = 3;

3. string a\_c[CASES] = {"isddkhuf", "scvbrressdg", "aaaaa"};

4. string b\_c[CASES] = {"dfposgofdgisddkjm", "scvhfjsscvbrrj21", "babaabaaaabaaabaaaaab"};

5. for (int i = 0; i < CASES; ++i) {

6. string word, a = a\_c[i], b = b\_c[i];

7. size\_t start, end;

8. for (int i = a.length(); i >= 0; --i) {

9. std::tie(word, start, end) = KMP\_first(a.substr(0, i), b);

10. if (word != "") {

11. std::cout << "found: '" << word << "' with length " << i << "\n";

12. b.insert(end, "|");

13. b.insert(start, "|");

14. std::cout << b << "\n";

15. break;

16. }

17. if (i == 0) {

18. std::cout << "found: '" << word << "' with length " << i << "\n";

19. b.insert(end, "|"); b.insert(start, "|");

20. std::cout << b << "\n";

21. }

22. } std::cout << std::endl;

23. } }

### 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.