ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | В. А. Кузнецов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 4.2 |
| РАЗРАБОТКА АЛГОРИТМА ИСЧЕРПЫВАЮЩЕГО ПОИСКА (ПОЛНОГО ПЕРЕБОРА) |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#__RefHeading___Toc801_316780651)

[2 Схема алгоритма решения 4](#__RefHeading___Toc809_316780651)

[3 Полное описание реализованных функций 5](#__RefHeading___Toc807_316780651)

[3.1 split\_words 5](#__RefHeading___Toc3907_3112333877)

[3.2 find\_indices 5](#__RefHeading___Toc3913_3112333877)

[3.3 transform\_to\_lower 6](#__RefHeading___Toc3911_3112333877)

[3.4 main 6](#__RefHeading___Toc3909_3112333877)

[4 Листинг программы 7](#__RefHeading___Toc805_316780651)

[5 Результаты тестирования программы 8](#__RefHeading___Toc803_316780651)

[6 Результаты измерения времени работы и оценки сложности алгоритма 9](#__RefHeading___Toc803_3167806511)

1. Постановка задачи

Задача: реализовать алгоритм на языке C/С++, выполняющий поставленную задачу. Алгоритм должен быть реализован путем перебора всех возможных вариантов решения. Вариант задания, пример входных и выходных данных представлен в таблице 1. Глобальные параметры использовать запрещено; допустимо использование дополнительных функций.

* Разработанный алгоритм должен быть реализован в виде цельной программной функции (или нескольких функций) так, чтобы мог быть многократно применим с различными исходными данными и при этом не включал команды, не относящиеся к решаемой задаче, например, ввод и вывод исходных данных на консоль или в файл.
* Произвести экспериментальную проверку времени работы разработанного алгоритма, определив его класс сложности для среднего случая. Измерить среднее время для Test\_Count повторений при различных размерностях входных данных.

Таблица 1 – Вариант

|  |  |  |  |
| --- | --- | --- | --- |
| N | Текст задания | Вход | Выход |
| 2 | **Данные:**   * Множество претендентов на вакансию Pi, количество претендентов N. * Каждый претендент обладает множеством навыков {Sj}, j – индекс навыка. * Количество не повторяющихся общих навыков M. * У разных претендентов могут быть пересекающиеся множества навыков. * Любой навык из общего множества навыков присутствует по крайней мере в одном множестве навыков претендента.   **Задача:**  Выбрать как можно меньше претендентов таким образом, чтобы все навыки были охвачены. |  |  |

1. Схема алгоритма решения

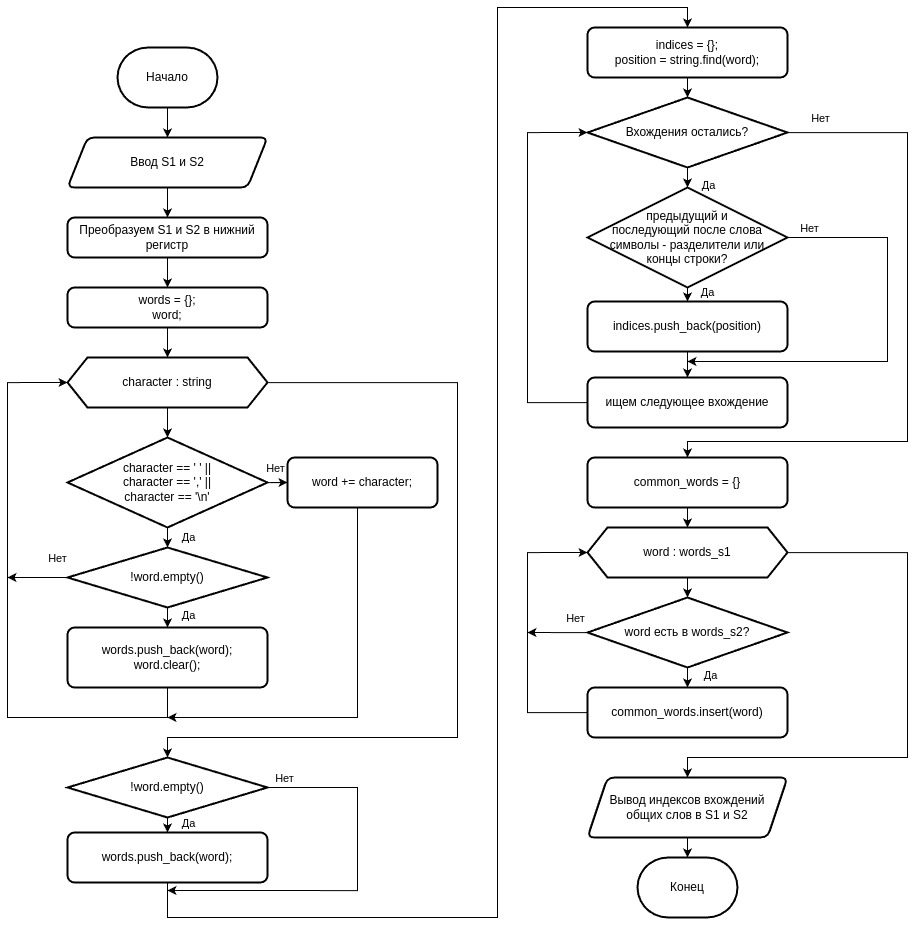


Рисунок 1 – Блок-схема алгоритма

1. Полное описание реализованных функций
   1. split\_words

Функция split\_words разделяет строку на слова, используя пробелы, запятые и переносы строк в качестве разделителей. Принимает следующие аргументы:

1. const std::string &string — строка, которую нужно разделить на слова.

Возвращает std::vector<std::string> — вектор строк, каждая из которых является словом из исходной строки. Работа функции происходит следующим образом:

1. Создаётся пустой вектор words и временная строка word.
2. Для каждого символа в строке проверяется, является ли он разделителем (пробел, запятая, перенос строки).
3. Если символ является разделителем и word не пуст, добавляем word в words и очищаем word.
4. Если символ не является разделителем, добавляем его к word.
5. В конце добавляем последнее слово в words, если оно не пустое.
   1. find\_indices

Функция find\_indices находит все индексы вхождения слова в строку. Принимает следующие аргументы:

1. const std::string &string — строка, в которой ищем вхождения слова.
2. const std::string &word — слово, индексы вхождения которого ищем.

Возвращает std::vector<int> — вектор индексов, где слово встречается в строке. Работа функции происходит следующим образом

1. Создаётся пустой вектор indices.
2. Используем string.find для поиска первого вхождения слова в строке.
3. Пока находятся вхождения, проверяем, является ли предыдущий и последующий после слова символы разделителями или концом строки.
4. Если условие выполняется, добавляем индекс начала слова в indices.
5. Ищем следующее вхождение, начиная с позиции position + 1.
   1. transform\_to\_lower

Функция transform\_to\_lower преобразует все символы строки в нижний регистр. Принимает следующие аргументы:

1. const std::string &string — строка, которую нужно преобразовать.

Возвращает std::string — строка, преобразованная в нижний регистр. Работа функции происходит следующим образом:

1. Копируем исходную строку в новую строку lower.
2. Используем std::transform для преобразования каждого символа строки в нижний регистр с помощью функции ::tolower.
   1. main
3. Программа запрашивает ввод строк S1 и S2 от пользователя.
4. Преобразуем S1 и S2 в нижний регистр с помощью transform\_to\_lower.
5. Разделяем S1 и S2 на слова с помощью функции split\_words.
6. Для каждого слова в S1 и S2 находим индексы вхождений в исходные строки, используя функцию find\_indices.
7. Находим слова, которые есть в обеих строках, используя unordered\_set.
8. Для каждого общего слова выводим индексы их вхождений в S1 и S2.
9. Листинг программы

Листинг 1

1. Результаты тестирования программы

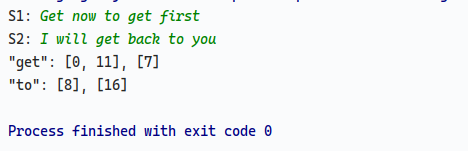


Рисунок 2

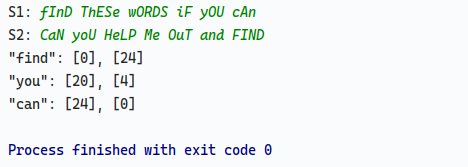


Рисунок 3

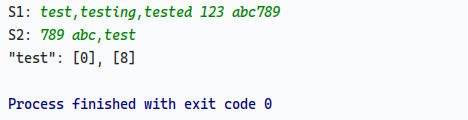


Рисунок 4

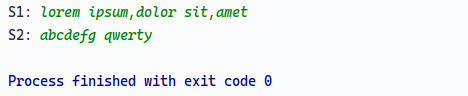


Рисунок 5

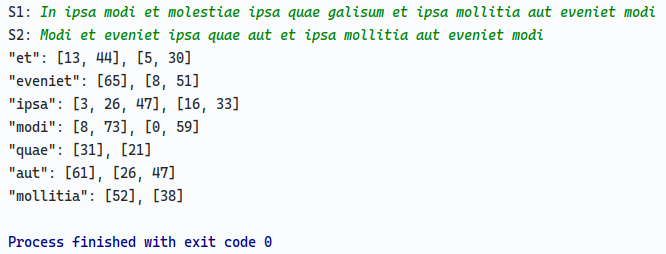


Рисунок 6

1. Результаты измерения времени работы и оценки сложности алгоритма