Лабораторная работа № 4 по курсу дискретного анализа: поиск образца в строке

Выполнил студент группы 08-308 МАИ *Шарапов Леонид*.

Условие

1. Общая постановка задачи.

   Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

2. Вариант задания.

   Реализовать поиск одного образца-маски, в котором могут встречаться «джокеры», равные любому другому элементу алфавита , состоящий из слов не более 16 регистронезависимых латинских знаков. При работе алгоритма нужно выделить образцы, не содержащие «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.

Метод решения

1. Разделим образец с джокерами на шаблоны без них.
2. Найдем их стартовые позиции в образце с масками.
3. Посчитаем количество безмасочных шаблонов.
4. Для их хранения создадим дерево ключей (бор). Они соответствуют его листам.
5. Далее строим связи неудач в этом дереве.
6. Создаем массив C.
7. Зануляем его.
8. Выполняем поиск образцов в тексте.
9. При их обнаружении прибавляем единицу к C[позиция начала образца в тексте + начальная позиция образца в шаблоне с джокерами - 1]
10. Просматриваем массив C, если значение какого-либо элемента равно количеству безмасочных образцов в шаблоне, то номер этого элемента является начальной позицией образца с джокерами в тексте

Использовались книга Гасфилда и aho-corasick.narod.ru

Описание программы

Весь код находится в main.cpp

Используемые типы данных

1. const int
2. int
3. int*
4. char
5. char*
6. _Bool - переменная, хранящая 0 или 1
7. struct stackstate - хранит указатель на состояние автомата и следующее для него состояние
8. struct stack Occur - хранит вхождения образцов
9. struct keywordTree - хранит указатели на элемент алфавита, следующее состояние, связь неудачи и очередь вхождений

Используемые функции

1. addS - добавляет в очередь элемент
2. addSO - добавляет в очередь элемент
3. deleteS - удаляет элемент очереди
4. addTree - добавляет в состояние в автомат
5. deleteTree - удаляет дерево с заданного состояния
6. g - выполняет переход в другое состояние для текущего элемента алфавита
7. add - считывает элементы алфавита со стандартного ввода, добавляет их в дерево и строит связи неудач
8. search - считывает текст со стандартного ввода и заполняет массив P для обнаружения образца с масками
9. check - обнаруживает образцы в тексте и выводит их в соответствии с заданием ЛР
10. main - выделение и очистка памяти и запуск функций работы с деревом

Дневник отладки

1. runtime test01. Причиной является потребление пустой программой на C++ 72 кб (выделение). Решением был переход на язык C

2. time limit test01.t. Исправлено бесконечное считывание символов в конструкции while(scanf), то есть добавлена проверка на равенство -1

3. runtime test03.t. Исправлена работа с джокерами: учитываются последние подряд идущие в образце джокеры.

4. wrong answer test05.t. Исправлено учитывание пустых строк текста

5. runtime error test10.t signal 11. Заменен стек на очередь в алгоритме построения связей неудач.

6. wrong answer test11.t. Исправлено добавление в непустой список элемента: новое вхождение записывалось в head, а не в tmp. Также обнаружена неполная работа алгоритма. Он не помечал вхождения подстрок в текст, которые являются подстроками других. Причиной этого является отстутствие в состояниях учета вхождений.

Тест производительности

Сложность поиска шаблона длины n с масками в количестве z в тексте длины m занимает O(n+m+z) времени

В тестировании n,m и z линейно увеличивались

Тестирование

1. 136 кБайт - 2,934 секунд

2. 1360 кБайт - 24,830 секунд

3. 2720 кБайт - 49,078 секунд

По результатам тестирования сложность алгоритма соответствует заявленной

Выводы

Основными сложностями программирования были изучение большого количества литературы, составление аналитических решений на бумаге, нахождение ошибки в работе алгоритма и настройка правильного вывода. В первых версиях алгоритма было слишком большое потребление памяти из-за использования языка C++, потому что пустая программа на нем потребляет 72 килобайта, поэтому чекер отказывался принимать программу, а ЛР была переписана на C. Бесконечное считывание символов было из-за отстутствия проверки равенства возвращаемого значения функцией scanf на 1 (кол-во переменных).