Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: Д. С. Пивницкий
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 01.01.2021
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца основанный на построении Z-блоков.
Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые)

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

1 Описание

Требуется написать реализацию алгоритма поиска образца, основанного на использовании Z-функции. Согласно [1], для строки S и позиции $i < 1$ Z-функция $Z_i(\mathrm{S})$ определяется как длина наибольшей подстроки S, которая начинается в i и совпадает с префиксом S.

Для эффективного вычисления Z-функции необходимо ввести следующие понятия:

1. Z-блок – для любой позиции $i > 1$, в которой $Z_i(S) > 0$, это интервал, начинающийся в i и заканчивающийся в позиции $i + Z_i(S) - 1$.

2. r_i – крайняя правая позиция конца Z-блоков, начинающихся не позднее позиции i.

3. l_i – крайняя левая позиция Z-блока, начинающегося не позднее i, с наибольшей крайней правой позицией.

Алгоритм Z-функции вычисляет значения Z_i, r_i , l_i последовательно для каждой позиции, начиная с $i = 2$. Для ускорения этого расчета используются уже вычисленные значения Z. Пусть при работе алгоритма были вычислены Zi для $1 < i \leq k - 1$ и текущие значения l и r. Тогда Z_k и изменения для l и r можно определить следующим образом:

1. Если $k > r$, то значение Z_k вычисляется непосредственным сравнением подстрок, начиная с позиции k и позиции 1. Длина совпадающей части и является Z_k, при $Z_k > 0$ также изменяется значения $l = k$ и $r = k + Z_k - 1$.

2. Если $k \leq r$, то позиция k содержится в текущем Z-блоке и подстрока $S[l..r]$ совпадает с префиксом $S[1..r - l + 1]$, а значит и символ на позиции k совпадает с символом в позиции $k = k - l + 1$. По тем же причинам подстрока $S[k..r]$ (назовём её β) должная совпадать с подстрокой $S[k..Z_l]$, где ранее уже были вычислены значения Z-функции. Здесь возникает два возможных случая: Если $Z_k < |\beta|$, то $Z_k = Z_k$ и r, l не изменяются. Если $Z_k \geq |\beta|$, то вся подстрока $S[k..r]$ должна быть префиксом S, однако не гарантируется то, что эта подстрока наибольшая совпадающая. Необходимо сравнить до несовпадения символы, начиная с позиции $r + 1$, с символами, начиная с позиции $|\beta| + 1$. Пусть несовпадение произошло на символе $q \geq r + 1$. Тогда Z_k полагается равным $q - k$, $r = q - 1$ и $l = k$.

2 Исходный код

В рамках данной задачи работа алгоритма должна происходить не с буквами, а целыми словами текста. Для корректного выполнения Z-функции нужно написать функцию Equals(2), проверяющую два слова на равенства, с учетом регистронезависимости. Также определим метод CalculateZFunc(2), вычисляющий Z-функцию по входной строке и сохраняющий её значения во входной массив. В свою очередь нахождение вхождений в тексте указанного образца будет осуществляться в другом методе – FindOccurs(3), использующем Z-функцию образца и вычисляющем значения Z-функции для строки S = P$T, начиная с начала текста. Поскольку все значения этой Z-функции будут гарантированно меньше длины образца P (т.к. нельзя при сравнении уйти за символ $), то все значения в полученных Z-блоках могут быть найдены с использованием только значений Z-функции образца. Такой подход является довольно эффективным с точки зрения затрат памяти, поскольку не нужно создавать массив с целыми числами Z-функции для целого текста. При нахождении образца в тексте необходимо также вывести номер строки и номер слова, с которого начинается вхождение, для этого можно создать соответствующую структуру TAnswer. Ниже приведён исходный код search.hpp с описанными методами:

Код: main.cpp

```
#include "search.hpp"
int main() {
std::vector<int> zFunc;
std::vector<std::string> sample;
std::vector<int> stringEnd;
std::vector<std::string> text;
std::string word;
word.append(MAX_WORD_LENGTH, 0);
char c = getchar();
bool sampleFinish = false;
bool inputWord = false;
int index = 0;
while (c != EOF) {
if (c == '\n') {
if (inputWord) {
if (!sampleFinish) {
sample.push_back(word);
}
else {
text.push_back(word);
}
for (int i = 0; i < index; ++i) {
word[i] = 0;
}
index = 0;
inputWord = false;
```

```
}
if (!sampleFinish) {
sampleFinish = true;
}
else {
stringEnd.push_back(text.size());
}
}
else if (c == '\t' || c == ' ') {
if (inputWord) {
if (!sampleFinish) {
sample.push_back(word);
}
else {
text.push_back(word);
}
for (int i = 0; i < index; ++i) {
word[i] = 0;
}
index = 0;
inputWord = false;
}
}
else {
word[index] = c;
++index;
inputWord = true;
}
c = getchar();
}
zFunc.assign(sample.size(), 0);
CalculateZFunc(sample, zFunc);
std::vector<TAnswer> ans = FindOccurs(sample, zFunc, text, stringEnd);
for (int j = 0; j < ans.size(); ++j) {
std::cout << ans[j].strPos << ", " << ans[j].wordPos << std::endl;
}
return 0;
}
```

В исходном коде main.cpp описан основной ход работы программы – чтение входного образца и текста с учетом возможного произвольного числа пробелов между словами, вызов функций вычисления Z-функции и поиска образца, а также вывод полученного ответа

Код: search.hpp

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

```
5  struct TAnswer {
6  int strPos;
7  int wordPos;
8  };
9  const int MAX_WORD_LENGTH = 16;
10 bool Equals(const std::string& first, const std::string& second) {
11 for (int i = 0; i < MAX_WORD_LENGTH; ++i) {
12 char c1 = ('A' <= first[i] && first[i] <= 'Z' ? first[i] - 'A' + 'a' : first[i]);
13 char c2 = ('A' <= second[i] && second[i] <= 'Z' ? second[i] - 'A' + 'a' : second[i]);
14 if (c1 != c2) {
15 return false;
16 }
17 }
18 return true;
19 }
20 void CalculateZFunc(const std::vector<std::string>& sample, std::vector<int>& zFunc) {
21 for (int i = 1, l = 0, r = 0; i < zFunc.size(); ++i) {
22 if (i <= r) {
23 zFunc[i] = std::min(zFunc[i - l], r - i + 1);
24 }
25 while (zFunc[i] + i < zFunc.size() && Equals(sample[zFunc[i]], sample[zFunc[i] + i]))
       {
26 ++zFunc[i];
27 }
28 if (r < i + zFunc[i] - 1) {
29 l = i;
30 r = i + zFunc[i] - 1;
31 }
32 }
33 }
34 std::vector<TAnswer> FindOccurs(const std::vector<std::string>& sample, const std::
       vector<int>& zFunc, const std::vector<std::string>& text, const std::vector<int>&
       stringEnd) {
35 int value;
36 int wordPos = 0;
37 int strPos = 0;
38 std::vector<TAnswer> answer;
39 for (int i = 0, l = 0, r = 0; i < text.size(); ++i) {
40 value = 0;
41 while (i == stringEnd[strPos]) {
42 ++strPos;
43 wordPos = 0;
44 }
45 ++wordPos;
46 if (i <= r) {
47 value = std::min(zFunc[i - l], r - i + 1);
48 }
49 while (value < sample.size() && value + i < text.size() && Equals(sample[value], text[
       value + i])) {
```

```
50 ++value;
51 }
52 if (r < i + value - 1) {
53 l = i;
54 r = i + value - 1;
55 }
56 if (value == sample.size()) {
57 answer.push_back({strPos + 1, wordPos});
58 }
59 }
60 return answer;
61 }
```

3 Консоль

```
(py37) ~ /DA_labs/lab4$ make
g++ -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -o solution main.cpp
(py37) ~ /DA_labs/lab4$ cat test1.txt
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
(py37) ~ /DA_labs/lab4$ ./solution <test1.txt
1,3
1,8
(py37) ~ /DA_labs/lab4$ cat test2.txt
mouse
mo use horse
cat dog cat mo
use mouse mouse
(py37) ~ /DA_labs/lab4$ ./solution <test2.txt
3,2
3,3
```

4 Тест производительности

Тест производительности представляет из себя сравнение времени работы поиска образца в тексте с использованием написанного алгоритма с построением Z-блоков и функции find для класса std::string из стандартной библиотеки C++. В рассматриваемых тестах слова в тексте и образце состоят из не более чем трёх букв, размер алфавита – три символа, количество слов в образце составляет десять, а общее число слов в текстах – сто, десять тысяч, сто тысяч и миллион.

Моя реализация:

```
(py37) ~ /DA_labs/lab4$ ./banchmark <test100.txt
std::find work time = 0.119ms,z-function work time = 0.235ms.
(py37) ~ /DA_labs/lab4$ ./banchmark <test10000.txt
std::find work time = 10.723ms,z-function work time = 33.491ms.
(py37) ~ /DA_labs/lab4$ ./banchmark <test100000.txt
std::find work time = 170.296ms,z-function work time = 327.8ms.
(py37) ~ /DA_labs/lab4$ ./banchmark <test1000000.txt
std::find work time = 1666.659ms,z-function work time = 3118.58ms.
```

Как видно, у обоих алгоритмов наблюдается почти линейный рост времени работы в зависимости от размера входных данных, однако написанная Z-функция значительно уступает функции find из стандартной библиотеки. Возможно, это связано с тем, что используемый алгоритм генерации текстов заполняет их словами алфавита случайным образом, и получается, что в Z-функции используются Z-блоки довольно редко и их длина сравнительно небольшая, поэтому почти всегда возникает необходимость искать новые совпадающие с префиксом подстроки, что почти есть работа функции find, однако в Z-функции выполняется большее число операций из-за проверки на каждой итерации принадлежности позиции Z-блоку, а также условия на изменение границ рабочего Z-блока. По-моему мнению, написанный алгоритм будет работать гораздо эффективнее в сравнении с find при работе с другими текстами, например такими, которые могут состоят только из искомого образца и образца с одним различающимся словом, тогда подстроки будут чаще совпадать с префиксом и Z-блоки будут также использоваться регулярнее

5 Выводы

Выполнив четвёртую лабораторную работу, я научился реализовывать алгоритм поиска образца в тексте, основанный на использовании Z-блоков. Основную трудность в ходе написания программы для меня составило правильное чтение входных данных с учётом произвольного числа разделителей между словами, а также разделения текста на множество строк. Полученный алгоритм можно использовать, например, для поиска информации по ключевым словам или предложениям в различных текстовых файлах.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Поразрядная сортировка — Вики университета ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 01.10.2020).