

**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: А. А. Боглаев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

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: Числа в диапазоне от 0 до $2^{32} - 1$.

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

1 Описание

Требуется реализовать поиск одного образца основанный на построении z-блоков. Учитывая, что алфавит состоит из чисел, нужно преобразовать "слово" в число. При работе с числами мы не можем использовать привычные нам разделители: #, \$.

Согласно [1]: «Z-функция - это длина наибольшей подстроки S , которая начинается в i и совпадает с пре- префиксом S .»

Есть два вида алгоритма Z-функции: наивный за $O(n^2)$ и эффективный за $O(|S|)$, где S - строка. Для выполнения лабораторной работы нужно использовать эффективную версию алгоритма, иначе можем превысить ограничения для программы по памяти и по времени.

В основе эффективного алгоритма лежат Z-блоки. Согласно [1] Z-блок - это подстрока с началом в позиции i и длиной $Z[i]$. Для работы алгоритма нужно завести две переменные: l и r , левая и правая границы Z-блока. Изначально эти переменные равны 0.

Пусть нам известны значения Z-функции от 0 до $i - 1$. Найдём $Z[i]$.

Рассмотрим два случая:

1. $i > r$

Просто пробегаемся по строке S и сравниваем символы на позициях $S[i+j]$ и $S[j]$. Пусть j первая позиция в строке S для которой не выполняется равенство $S[i+j] = S[j]$, тогда j это и Z-функция для позиции i . Тогда $l = i$, $r = i + j - 1$. В данном случае будет определено корректное значение $Z[i]$ в силу того, что оно определяется наивно, путем сравнения с начальными символами строки.

2. $i \leq r$

Сравним $Z[i - l] + i$ и r . Если r меньше, то надо просто наивно пробежаться по строке начиная с позиции r и вычислить значение $Z[i]$. Корректность в таком случае также гарантирована. Иначе мы уже знаем верное значение $Z[i]$, так как оно равно значению $Z[i - l]$.

Данный алгоритм эффективнее предыдущего, так как мы не вычисляем Z-функцию для каждого элемента, только для некоторых символов (чисел). А остальные Z-функции можем получить на основе предыдущих Z-функций.

Алгоритм Z-функции посещает каждую букву не более двух раз. Его сложность $O(n + m)$, где m - длина текста, а n - длина паттерна.

2 Исходный код

Для выполнения работы нам потребуется: написать Z-функцию, реализовать ввод и вывод данных, поиск подстроки в строке. А также провести тесты производительности.

Входной файл содержит числа, которые нужно рассматривать, как слова в тексте. Для вывода результата, необходимо знать номер строки, в которой найдено совпадение и номер слова в строке, с которого начинается совпадение.

Для этого определим вектор, элемент которого это пара, состоящая из числа и пары (номер слова и номер строки). Объявим псевдоним *TDigitSRH*, чтобы упростить работу с программой и сделать ее более разборчивой.

Так как на вход подаются строки, то их надо преобразовать в вектор чисел.

Также создадим переменную *SEP*, эта переменная нужна для реализации поиска подстроки в строке. Чтобы найти паттерн в тексте, соединим паттерн с текстом в один вектор, а между ними поместим разделитель. Исходный алфавит состоит из числе от 0 до $2^{32} - 1$, значит нужно взять в качестве разделителя такое число, которое не встретится ни в паттерне, ни в тексте.

Таким числом является любой отрицательное, например -1 , подходящее под наш тип данных.

Так как мы используем отрицательные числа, то нам уже не подойдет тип *uint32_t*, нужно брать *int64_t*.

Далее для нового вектора подсчитаем Z-функцию. Если $Z[i] == n$, (n - длина паттерна), то мы нашли вхождение паттерна в текст.

Главное учесть, что цикл, для прохода по Z-функции, нужно начинать не с 0, а с $n + 1$, так как в начале нашего вектора находится паттерн и разделитель.

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <sstream>
4 |
5 |
6 | const int SEP = -1;
7 | using TDigitSRH = std::pair<int64_t, std::pair<size_t, size_t>>;
8 |
9 |
10 | std::vector<int> Z_func(const std::vector<TDigitSRH>& S) {
11 |     int n = S.size();
12 |     int l = 0, r = 0;
13 |     std::vector<int> Z(n);
14 |
15 |     for (int i = 1; i < n; i++) {
16 |
17 |         Z[i] = std::max(0, std::min(r - i, Z[i - l]));
```

```

18
19     while (i + Z[i] < n && S[Z[i]].first == S[i + Z[i]].first) {
20         Z[i]++;
21     }
22
23     if (i + Z[i] > r) {
24         l = i;
25         r = i + Z[i];
26     }
27 }
28
29 return Z;
30 }
31
32
33 int main() {
34     std::string P;
35     getline(std::cin, P);
36     std::vector<TDigitSRH> vec_p;
37
38
39     std::istringstream ss(P);
40     int64_t number = 0;
41     size_t cnt_n = 0;
42     while (ss >> number) {
43         vec_p.push_back(std::make_pair(number, std::make_pair(cnt_n, 1)));
44         cnt_n++;
45     }
46
47     int p_sz = vec_p.size();
48     TDigitSRH sep_elem = std::make_pair(SEP, std::make_pair(vec_p.size(), 1));
49
50     vec_p.push_back(sep_elem);
51
52
53     std::string T;
54     size_t cnt_line = 1;
55
56     while (getline(std::cin, T)) {
57
58         std::istringstream ss(T);
59         int64_t num = 0;
60         size_t cnt_num = 0;
61         while (ss >> num) {
62             vec_p.push_back(std::make_pair(num, std::make_pair(cnt_num, cnt_line)));
63             cnt_num++;
64         }
65         cnt_line++;
66     }

```

```

67 ||
68 ||
69 || std::vector<int> Z = Z_func(vec_p);
70 ||
71 || for (size_t i = p_sz + 1; i < Z.size(); i++) {
72 ||     if (Z[i] == p_sz) {
73 ||         std::cout << vec_p[i].second.second << ", " << vec_p[i].second.first + 1 <<
74 ||             std::endl;
75 ||     }
76 || }

```

3 Консоль

```
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_04$ cat tst.txt
51 89 51 89 122
0051 89 051 0089 51 89 122      51
89 51 89 122
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_04$ g++ -std=c++17 -pedantic
-Wall -Wextra -Werror main.cpp
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_04$ ./a.out <tst.txt
1,3
1,8
```

4 Тест производительности

Реализованный алгоритм Z-функции сравним с наивным алгоритмом Z-функции.
Будем тестировать на последовательности из: 10^3 , 10^4 , 10^5 .

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я ознакомился со структурой данных В-дерево и реализовал ее. Также я узнал, что данная структура используется внутри многих баз данных для работы с индексами. Так как с ее помощью можно быстрее работать с информацией, которая лежит во внешней памяти.

В ходе работы столкнулся с трудностями при работе с индексами массива, справился с этим мне помог отладчик. Также возникли трудности при работе с файлами. Нужно было аккуратно написать функции для чтения из файла и записи в файл, чтобы не потерять информацию.

Список литературы

[1] Гасфилд Дэн

Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. Пер. с англ. И. В. Романовского. — СПб.: Невский Диалект; БХВ-Петербург, 2003. — 654 с: ил. ISBN 5-7940-0103-8 ("Невский Диалект"), ISBN 5-94157-321-9 ("БХВ-Петербург")

[2] *Z-функция* — Викиконспекты.

URL: <https://neerc.ifmo.ru/wiki/index.php?title=Z-функция> (дата обращения: 29.05.2024).