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: И. В. Кочкожаров
Группа: М8О-208Б-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.

Формат ввода Искомый образец задаётся на первой строке входного файла. В случае, если в задании требуется найти несколько образцов, они задаются по одному на строку вплоть до пустой строки. Затем следует текст, состоящий из слов или чисел, в котором нужно найти заданные образцы. Никаких ограничений на длину строк, равно как и на количество слов или чисел в них, не накладывается.

Формат вывода В выходной файл нужно вывести информацию о всех вхождениях искомых образцов в обрабатываемый текст: по одному вхождению на строку. Для заданий, в которых требуется найти только один образец, следует вывести два числа через запятую: номер строки и номер слова в строке, с которого начинается найденный образец. В заданиях с большим количеством образцов, на каждое вхождение нужно вывести три числа через запятую: номер строки; номер слова в строке, с которого начинается найденный образец; порядковый номер образца. Нумерация начинается с единицы. Номер строки в тексте должен отсчитываться от его реального начала (то есть, без учёта строк, занятых образцами). Порядок следования вхождений образцов несущественен.

1 Описание

Требуется написать реализацию Z-функции.

Как сказано в [1]: «$Z_i(S)$ – это длина наибольшего префикса $S[i..|S|]$, совпадающего с префиксом S».

2 Исходный код

Будем хранить индексы L и R, обозначающие начало и конец префикса с наибольшим найденным на данный момент значением R. Изначально L = R = 0. Пусть нам известны значения Z-функции для позиций $1..i-1$. Попробуем вычислить значение Z-функции для позиции i. Если $i \in [L..R]$, рассмотрим значение Z-функции для позиции j = i - L. Если $i + Z[j] \leq R$, то $Z[i] = Z[j]$, так как мы находимся в подстроке, совпадающей с префиксом всей строки. Если же i + Z [j] R, то необходимо досчитать значение Z[i] простым циклом, перебирающим символы после R, пока не найдется символ, не совпадающий с соответствующим символом из префикса. После этого изменяем, значение L на i и значение R на номер последнего символа, совпавшего с соответствующим символом из префикса.

Если $i \notin [L..R]$, то считаем значение Z[i] простым циклом, сравнивающим символы подстроки начинающейся с i-го символа и соответствующие символы из префикса. Когда будет найдено несоответствие или будет достигнут конец строки, изменяем значение L на i и значение R на номер последнего символа, совпавшего с соответствующим символом из префикса.

Реализация самого алгоритма z-блоков.

Образуем строку s = pattern + # + text, где # — символ, не встречающийся ни в text, ни в pattern. Вычисляем Z-функцию от этой строки. В полученном массиве, в позициях в которых значение Z-функции равно |pattern|, по определению начинается подстрока, совпадающая с pattern.

Полученные абсолютные индексы начала совпадающей подстроки переводятся в относительные координанты вида номер строки и номер числа в начинающейся используя массив префиксных сумм и бинарный поиск по нему.

3 Асимптотика

Время работы алгоритма, вычисляющего значение Z-функции строки S оценивается в O (|S|). Докажем это. Рассмотрим i-й символ строки. В алгоритме он рассматривается не более двух раз: первый раз, когда попадает в отрезок [L..R], и второй раз при вычислении Z[i]. Таким образом цикл обрабатывает не более 2|S| итераций.

4 Листинги

```
1
2  using ll = long long;
3
4  std::size_t RelativeToAbsolute(const std::pair<size_t, size_t> &relativeIndex,
5                                 const std::vector<std::size_t> &prefixSums) {
6      return prefixSums[relativeIndex.first] + relativeIndex.second;
7  }
8
9  std::pair<size_t, size_t>
10 AbsoluteToRelative(std::size_t absoluteIndex,
11                    const std::vector<std::size_t> &prefixSums) {
12     size_t lineIndex = std::lower_bound(prefixSums.begin(), prefixSums.end(),
13                                         absoluteIndex + 1) -
14                        prefixSums.begin();
15     size_t numberIndex =
16         absoluteIndex - (lineIndex > 0 ? prefixSums[lineIndex - 1] : 0) + 1;
17     return std::make_pair(lineIndex, numberIndex);
18 }
19
20 std::vector<std::size_t> ZFunction(const std::vector<ll> &arr) {
21     std::size_t n = arr.size();
22     std::vector<std::size_t> z(arr.size());
23     for (std::size_t i = 1, l = 0, r = 0; i < arr.size(); ++i) {
24         if (i <= r) {
25             z[i] = std::min(r - i + 1, z[i - l]);
26         }
27         while (i + z[i] < n && arr[z[i]] == arr[i + z[i]]) {
28             ++z[i];
29         }
30         if (i + z[i] - 1 > r) {
31             l = i;
32             r = i + z[i] - 1;
33         }
34     }
35     return z;
36 }
37
38 int main() {
39     std::vector<ll> preparedText;
40     std::string line;
41     std::getline(std::cin, line);
42     uint num;
43     std::istringstream iss{line};
44     std::size_t patternSize = 0;
45     while (iss >> num) {
46         patternSize++;
47         preparedText.push_back(num);
```

```
48      }
49      preparedText.push_back(-1);
50      std::vector<std::size_t> prefixSums(1);
51      while (std::getline(std::cin, line)) {
52          std::istringstream iss{line};
53          std::size_t cnt = 0;
54          while (iss >> num) {
55              preparedText.push_back(num);
56              cnt++;
57          }
58          prefixSums.push_back(prefixSums.back()+cnt);
59      }
60
61      std::vector<std::size_t> zFun = ZFunction(preparedText);
62      for (std::size_t i = patternSize+1; i < zFun.size(); ++i) {
63          if (zFun[i] == patternSize) {
64              std::pair<size_t, size_t> match = AbsoluteToRelative(i-patternSize-1,
                    prefixSums);
65              std::cout << match.first << ", " << match.second;
66              std::cout << '\n';
67          }
68      }
69  }
```

5 Консоль

```
ivan@asus-vivobook ~/c/d/b/lab4 (master)>./lab4
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
1,3
1,8
```

6 Тест производительности

Тест производительности представляет из себя следующее: Сравнение алгоритма для поиска всех вхождений случайного паттерна в случайный текст используя функцию std::search и используя Z-функцию.

```
ivan@asus-vivobook ~/c/d/b/lab4 (master)>./lab4_benchmark 100000
matches: 3
std: 618
z-function: 2689
ivan@asus-vivobook ~/c/d/b/lab4 (master)>./lab4_benchmark 1000000
matches: 11
std: 6345
z-function: 27367
ivan@asus-vivobook ~/c/d/b/lab4 (master)>./lab4_benchmark 10000000
matches: 159
std: 60546
z-function: 260870
```

Как видно, алгоритм из STL обгоняет базовый алгоритм Z-блоков, что говорит о несовершенстве этого алгоритма, даже с учетом эффективного вычисления Z-функции.

7 Выводы

В результате выполнения второй лабораторной работы по курсу «Дискретный анализ», была реализован эффективный алгоритм нахождения Z-функции строки. Были получены знания из такого обширного раздела дискретного анализа, как поиск подстроки в строке, а так же навыки работы с префиксными суммами. Сравнение алгоритма с с функцией из STL показало, что другие алгоритмы, например алгоритм Бойера — Мура, могут работать гораздо быстрее, чем элементарный алгоритм Z-блоков.

Список литературы

[1] Gusfield D. Algorithms on Strings, Trees, and Sequences (англ.): Computer Science and Computational Biology — Cambridge University Press, 1997. — 556 p.