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: Ю. Ю. Обыденкова
Преподаватель: А. Н. Ридли
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Задача: Требуется разработать программу, осуществляющую ввод образца и поиск его в последующем тексте, используя заданный вариантом алгоритм
Вариант задания: Алгоритм поиска одного образца с использованием Z-блоков.
Вариант алфавита: Алфавит состоит из чисел в диапазоне от 0 до $2^{32} - 1$

1 Описание

Определение Z-функции: Для строки S и некоторой позиции $i > 1$, Z-функция определяется как длина наибольшей подстроки S, которая начинается в i и совпадает с префиксом S. Другими словами, $Z_i(S)$ - это длина наибольшего префикса $S[i..|S|]$, совпадающего с префиксом S.
Z-блоки: Z-блок представляет собой подстроку S максимальной длины, которая совпадает с префиксом S и начинается не с первой позиции $j > 1$, такой что $Z_j > 0$. Его длина равна Z_j.
Алгоритм вычисления Z-функции за линейное время: Z-функция используется во многих алгоритмах поиска подстроки для предварительной обработки образца(это называется препроцессинг). Поэтому крайне важно, чтобы существовал алгоритм, позволяющий быстро её вычислять.
Непосредственно алгоритм:

- Пусть l и r - соответственно левая и правая позиция последнего найденного(наибольшего) Z-блока. Если $k > r$, то найти Z_k непосредственным сравнением до несовпадения подстрок, начинающихся с позиции k и с позиции 1. Длина совпадающей части и дает Z_k. Если $Z_k > 0$, положить $r = k + Z_k - 1$ и $l = k$.

- Если $k >= r$, то позиция k находится в Z-блоке, и следовательно $S(k)$ содержится в подстроке $S[l..r]$(назовём её α), такой, что $l > 1$ и α совпадает с префиксом S. Поэтому символ $S(k)$ стоит и в позиции $k' = k - l + 1$. По тем же причинам подстрока $S[k..r]$ (назовём ее β) должна совпадать с подстрокой $S[k'..Z_l]$. Отсюда следует , что подстрока, начинающаяся с позиции k, должна совпадать по меньшей мере с префиксом S длины $min\{Z_k, |\beta|\} = r - k + 1$. Далее существует два случая.

  - Если $Z_{k'} < |\beta|$, то $Z_k = Z_{k'}$, r и l не изменяются.
  - Если $Z_{k'} >= |\beta|$, то вся подстрока $S[k..r]$ должна быть префиксом S и $Z_k >= |\beta| = r - k + 1$. Однако Z_k может быть больше, чем β, так что нужно сравнить до несовпадения символы, начиная с позиции $r + 1$, с символами, начиная с позиции $|\beta| + 1$. Пусть несовпадение произошло на символе $q >= r + 1$. Тогда Z_k полагается равным $q - k$, $r = q - 1$ и $l = k$.

2 Код программы

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <memory>

struct TPseudoInt{
    explicit TPseudoInt() = default;
    explicit TPseudoInt(unsigned int newNumber) : number(newNumber), isSentinel(false)
        {}

    unsigned int number = 0;
    bool isSentinel = true;
};

bool operator == (TPseudoInt lhs, TPseudoInt rhs) {
    if (lhs.isSentinel || rhs.isSentinel) {
        return lhs.isSentinel && rhs.isSentinel;
    } else {
        return lhs.number == rhs.number;
    }
}

class ConcatenatedString {
public:
    ConcatenatedString(const std::shared_ptr<std::vector<unsigned int>> newSample,
        const std::shared_ptr<std::vector<unsigned int>> newString)
    : sample(newSample), string(newString) {}

    TPseudoInt operator[](size_t index) const {
        if (index < sample->size()) {
            return TPseudoInt((*sample)[index]);
        } else if (index == sample->size()) {
            return TPseudoInt();
        } else {
            return TPseudoInt((*string)[index - sample->size() - 1]);
        }
    }

    size_t Size() const {
        return sample->size() + string->size() + 1;
    }

private:
    const std::shared_ptr<std::vector<unsigned int>> sample;
    const std::shared_ptr<std::vector<unsigned int>> string;
};
```

```
46
47  std::vector<int> zFunction(const ConcatenatedString& string) {
48      std::vector<int> zFunc(string.Size(), 0);
49      int leftBound = 0, rightBound = 0;
50      for (int i = 1; i < string.Size(); ++i) {
51          zFunc[i] = std::max(0, std::min(rightBound - i, zFunc[i - leftBound]));
52          while (i + zFunc[i] < string.Size() && string[zFunc[i]] == string[i + zFunc[i
                ]]) {
53              zFunc[i]++;
54          }
55          if (i + zFunc[i] > rightBound) {
56              leftBound = i;
57              rightBound = i + zFunc[i];
58          }
59      }
60      return zFunc;
61  }
62
63  template <typename T>
64  std::ostream& operator << (std::ostream& os, const std::vector<T>& container) {
65      bool first = true;
66      for (auto elem : container) {
67          if (!first) {
68              os << " ";
69          } else {
70              first = false;
71          }
72          os << elem;
73      }
74      return os;
75  }
76
77  int main() {
78      std::ios::sync_with_stdio(false);
79      std::cin.tie(nullptr);
80
81      std::string line;
82      std::shared_ptr<std::vector<unsigned int>> sample = std::make_shared<std::vector<
            unsigned int>>();
83      std::shared_ptr<std::vector<unsigned int>> string = std::make_shared<std::vector<
            unsigned int>>();
84      bool first = true;
85      std::vector<unsigned int> linesPosition = {0};
86      std::shared_ptr<std::vector<unsigned int>> vectorToWork = sample;
87      while (getline(std::cin, line)) {
88          std::istringstream is(line);
89          unsigned int num;
90          while (is >> num) {
91              vectorToWork->push_back(num);
```

```
92          }
93
94          if (first) {
95              first = false;
96              vectorToWork = string;
97          } else {
98              linesPosition.push_back(vectorToWork->size());
99          }
100
101     }
102
103     ConcatenatedString str(sample, string);
104     std::vector<int> zFunc = zFunction(str);
105
106     int lineIndex = 1;
107     for (size_t i = sample->size() + 1; i < zFunc.size(); ++i) {
108         if (zFunc[i] == sample->size()) {
109              size_t stringIndex = i - sample->size() - 1;
110              while (stringIndex >= linesPosition[lineIndex]) {
111                  lineIndex++;
112              }
113              std::cout << lineIndex << ", " << stringIndex - linesPosition[lineIndex -
                     1] + 1 << "\n";
114         }
115     }
116
117
118     return 0;
119 }
```

3 Тест производительности

Напишем простую программу для генерации случайных произвольных тестов, чтобы сравнить реализованный алгоритм поиска с использованием Z-функции и, допустим, наивный алгоритм поиска. Также добавим в программы с реализациями алгоритмов поиска специальные вставки, выводящие время выполнения в стандартный поток ошибок.

```
1  #include <ctime>
2  #include <random>
3  #include <map>
4  #include <limits>
5  #include <tuple>
6  #include <string>
7  #include <vector>
8  #include <cstdlib>
9  #include <iostream>
10 #include <chrono>
```

```cpp
#include <iomanip>

using namespace std;

default_random_engine rng;

uint64_t get_number(uint64_t min = 0,uint64_t max = numeric_limits<unsigned long long
    >::max()) {
    uniform_int_distribution<unsigned long long> dist_ab(min, max);
    return dist_ab(rng);
}

int main(int argc, char* argv[]) {
    rng.seed(std::chrono::system_clock::now().time_since_epoch().count());
    size_t alphabet_size, pattern_size, count;
    if (argc == 4) {
        alphabet_size = std::stoll(argv[1]);
        pattern_size = std::stoll(argv[2]);
        count = std::stoll(argv[3]);
    } else {
        cin >> alphabet_size >> pattern_size >> count;
    }

    alphabet_size--;

    for (size_t i = 0; i < pattern_size; ++i) {
        std::cout << get_number(0,alphabet_size) << " ";
    }
    for (int i = 0; i < count; ++i) {
        if (i % 100 == 0) {
            std::cout << "\n";
        }
        std::cout << get_number(0,alphabet_size) << " ";
    }
    std::cout << "\n";
    return 0;
}
```

1 Протокол тестирования производительности

```
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 4 10
10000 >test_10k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 4 10
100000 >test_100k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 4 10
1000000 >test_1000k
```

```
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 4 10
10000000 >test_10000k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_10k
Z-algorithm: 6 ms
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_100k
Z-algorithm: 25 ms
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_1000k
Z-algorithm: 242 ms
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_10000k
Z-algorithm: 2406 ms
122,91
6982,91
8162,48
21447,57
24590,59
34195,95
34638,62
64146,21
64703,35
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 3 10
10000 >test_10k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 3 10
100000 >test_100k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 3 10
1000000 >test_1000k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./generator 3 10
10000000 >test_10000k
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_10k
Z-algorithm: 2 ms
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_100k
Z-algorithm: 27 ms
477,28
532,39
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_1000k
Z-algorithm: 248 ms
49,10
1188,5
1449,51
1924,85
2295,28
2485,51
```

```
2574,4
2609,28
2750,42
3096,11
3569,37
4771,25
6897,17
7061,75
7141,38
7277,22
7925,83
8002,45
8167,21
9303,6
julia@julia21novo:~/CLionProjects/da_04/cmake-build-debug$ ./da_04 <test_10000k
Z-algorithm: 2683 ms
1644,54
2377,35
2447,74
3101,91
3411,51
3938,9
3947,23
4269,42
4280,15
6265,63
6705,95
7977,74
8146,84
8195,48
8266,71
8912,54
8925,19
10005,83
10796,29
11966,77
13090,64
13216,90
13676,37
14159,61
15122,27
```

16112,29
16349,4
16676,39
16722,74
17335,71
17721,19
18515,32
18627,23
22297,32
22336,11
22465,12
22786,24
24054,29
24650,22
25711,33
26372,26
26430,69
27103,64
27990,37
28017,81
28653,70
28819,8
29087,21
31138,51
31624,2
31690,27
31851,25
32034,58
32323,78
33244,22
34030,84
34300,20
34379,79
34939,2
35269,86
35700,82
36341,63
36647,31
36778,90
37992,15
38185,36

38658,9
38849,64
40325,95
40422,67
40454,80
40475,75
40649,28
40978,15
41495,49
42323,67
42650,36
42697,28
43586,61
43673,74
44309,28
44660,63
44953,20
46922,21
47313,76
47482,58
47503,35
47977,46
49024,83
50492,15
50747,63
50878,49
51311,70
52294,5
53492,86
53548,35
53683,15
54660,88
54939,2
55322,27
55444,66
57336,73
57645,37
58281,19
58947,53
59525,62
59581,69

59742,74
59953,7
60449,73
60599,30
61419,10
61960,8
62373,8
62391,55
63060,28
63246,17
63849,99
63961,24
64466,58
64564,94
64596,10
65030,80
65073,24
65499,22
65818,91
65901,81
66328,38
66447,87
66815,68
66901,18
67729,81
68136,88
68563,61
68583,48
69098,93
70402,45
70440,32
71782,79
72009,52
72162,26
72804,100
73374,34
74681,19
75081,48
75579,63
77618,80
77888,83

78362,56
79167,71
79367,86
79630,11
79912,79
81608,52
81608,99
81849,75
81985,83
82139,39
82258,48
82642,60
83521,42
84139,85
85684,26
86381,81
86546,67
86587,29
86650,66
87199,21
87406,21
87546,28
87847,79
88677,99
88910,45
89653,75
89964,19
89989,66
90157,66
90620,84
91141,87
91422,94
92275,13
93211,72
93710,56
94024,33
94502,22
94516,43
94982,88
95293,61
96196,1

```
96816,47
97183,95
98730,10
99706,81
```

Были сгенерированы случайные тексты и образцы для алфавитов состоящих из 3 и из 4 символов. Из полученного времени работы можно сделать вывод, что алгоритм действительно работает за линейное время.

4 Выводы

Выполнив четвертую лабораторную работу, я научилась реализовывать и отлаживать алгоритмы поиска подстроки в строке. Я узнала об основном препроцессинге образца, используемом во многих других алгоритмах и реализовала с его помощью программу, выводящую все пощиции совпадения образца с текстом. Алгоритмы, использующие препроцессинг - одни из самых быстрых. Z-функция - это один из видов препроцессинга. Таким образом, весь алгоритм представляет из себя два случая, которые фактически различаются только начальным значением z[i]: в первом случае оно полагается равным нулю, а во втором — определяется по предыдущим значениям по формуле. После этого обе ветки алгоритма сводятся к выполнению тривиального алгоритма, стартующего сразу с указанного начального значения. Алгоритм получился весьма простым. Несмотря на то, что при каждом i в нём так или иначе выполняется тривиальный алгоритм — мы достигли существенного прогресса, получив алгоритм, работающий за линейное время

Список литературы

[1] Дэн Гасфилд *Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология* — Издательство«СПб.: Невский диалект», 2003. — 656 с. (ISBN 5-7940-0103-8)