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. В. Постнов
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4

Задача:

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

Вариант алгоритма: Поиск большого количества образцов при помощи алгоритма Ахо-Корасик.

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

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

1 Описание

Алгоритм Ахо-Корасик

Пусть дан набор строк и текст. Нужно найти все вхождения строк в текст.

Решать эту задачу будем следующим образом. Будем обрабатывать символы текста по одному и поддерживать наибольшую строку, являющуюся наибольшим префиксом паттерна, и при этом также суффиксом считанного на данный момент текста. Если эта строка совпадает с каким-то паттерном, то отметим текущий символ — в нём заканчивается какая-то строка.

Для этой задачи нам нужно как-то эффективно хранить и работать со всеми префиксами паттернами — для этого нам и понадобится префиксное дерево.

Добавим все слова в префиксное дерево и пометим соответствующие им вершины как терминальные. Теперь наша задача состоит в том, чтобы при добавлении очередного символа быстро находить вершину в префиксном дереве, которая соответствует max входящему в бор суффиксу нового выписанного префикса. Для этого нам понадобятся несколько вспомогательных понятий.

1) связи неудач (они ведут ровно в те вершины, которые соответствуют самому длинному «сматченному» суффиксу)

2) связи выхода (они были в моей первой реализации алгоритма). Связи выхода необходимы, так как некоторые паттерны могут быть подстроками других паттернов. Поэтому они просто ведут в терминальную вершину.

Алгоритм поиска:

Обрабатываем текст посимвольно. Пытаемся проходить по ветке бора, если вершина терминальная, то выписываем ответ, а также обрабатываем рекурсивно связи выхода. Если пройти по вершине не получается, то проходим по связи неудач.

Алгоритм построения связей неудач:

для всех паттернов длины 1 - ведет в корень. Для всех паттернов длины n - идем по связи неудач длины $n-1$(она точно есть), пытаемся пройти из этой вершины дальше, если не получается, то идем дальше по связи неудач. Повторяем, пока не получится пройти дальше или не дойдем до корня.

Реализация с помощью BFS

Алгоритм построения связи выхода:

во время построения связей неудач проверяем:

1) если для текущей вершины связь неудачи ведет в терминальную вершину, то она является связью выхода.

2) если для текущей вершины связь неудачи ведет в нетерминальную вершину со связью выхода. То она так же является связью выхода для текущей вершины.

2 Исходный код

Структура программы состоит из 2 файлов:

1. *aho.hpp* Реализация алгоритма Ахо-Корасик.
2. *main_cpp* Обработка паттернов и текста, вывод ответа.
3. *CmakeLists.txt* Описание сборки программы

```
#ifndef AHO_HPP
#define AHO_HPP

#include <queue>
#include <unordered_map>
#include <vector>
#include <iostream>
#include <sstream>

class TAho {

private:
    struct TNode {
        unsigned long long value;
        std::vector<unsigned long long> id;
        TNode* parent;
        TNode* fail;
        std::vector<size_t> size;
        std::unordered_map<unsigned long long, TNode*> go;
        bool isLeaf;
    };
    TNode* root;

    static void Destroy(TNode* node) {
        if (node != nullptr) {
            for (auto sons : node->go) {
                Destroy(sons.second);
            }
            delete node;
        }
    }

    TNode* GetFailPointerFromLastFail(TNode* node, unsigned long long num) {
        if (node->go[num] == nullptr) {
            if (node != root) {
                return GetFailPointerFromLastFail(node->fail, num);
            }
            return root;
        }
        return node->go[num];
    }
```

```
42
43          TNode* GetFail(TNode* node, unsigned long long num) {
44              if (node == root) {
45                  return root;
46              }
47              if (node->parent == root) {
48                  return root;
49              }
50              return GetFailPointerFromLastFail(node->parent->fail, num);
51          }
52
53          TNode* Next(TNode* node, unsigned long long num) {
54              if (node == root) {
55                  if (node->go[num] != nullptr) {
56                      return node->go[num];
57                  }
58                  return node;
59              }
60              if (node->go[num] != nullptr) {
61                  return node->go[num];
62              }
63              return Next(node->fail, num);
64          }
65
66      public:
67
68          TAho() {
69              root = new TNode;
70          }
71
72          ~TAho() {
73              Destroy(root);
74          }
75
76          void AddVectorOfNums(std::vector<unsigned long long>& nums, unsigned long long
                pid) {
77              auto *temp = root;
78              for (auto num : nums) {
79                  if (temp->go[num] == nullptr) {
80                      temp->go[num] = new TNode;
81                      temp->go[num]->parent = temp;
82                      temp = temp->go[num];
83                      temp->value = num;
84                  } else {
85                      temp = temp->go[num];
86                  }
87              }
88              temp->isLeaf = true;
89              temp->id.push_back(pid);
```

```
90              temp->size.push_back(nums.size());
91
92          }
93
94
95          void Init() {
96              std::queue <TNode*> queue;
97              queue.push(root);
98              while (!queue.empty()) {
99                  auto* node = queue.front();
100                 queue.pop();
101                 node->fail = GetFail(node, node->value);
102                 if (node != root && node->fail->isLeaf) {
103                     node->isLeaf = true;
104                     for (size_t i = 0; i < node->fail->size.size(); ++i) {
105                         node->size.push_back(node->fail->size[i]);
106                         node->id.push_back(node->fail->id[i]);
107                     }
108                 } else if (node != root && node->fail->fail != nullptr && node->fail->
                        fail->isLeaf) {
109                     node->isLeaf = true;
110                     for (size_t i = 0; i < node->fail->fail->size.size(); ++i) {
111                         node->size.push_back(node->fail->fail->size[i]);
112                         node->id.push_back(node->fail->fail->id[i]);
113                     }
114                 }
115                 for (auto elem : node->go) {
116                     if (elem.second != nullptr) {
117                         queue.push(elem.second);
118                     }
119                 }
120             }
121         }
122
123         void AddToAnswer(std::vector <std::pair<unsigned long long, unsigned long long
                >>& answer, TNode* temp, unsigned long long pos) {
124             if (temp->isLeaf && temp != root) {
125                 for (size_t i = 0; i < temp->size.size(); ++i) {
126                     answer.emplace_back(pos - temp->size[i], temp->id[i]);
127                 }
128             }
129
130         }
131
132         std::vector <std::pair<unsigned long long, unsigned long long>> FindPosPatterns
                (const std::vector<unsigned long long >& nums) {
133             std::vector <std::pair<unsigned long long, unsigned long long>> answer;
134             auto* temp = root;
135             unsigned long long pos = 0;
```

```
            for (const auto& num : nums) {
                AddToAnswer(answer, temp, pos);
                temp = Next(temp, num);
                pos++;
            }
            AddToAnswer(answer, temp, pos);
            return answer;
        }
    };

    #endif
```

```
    #include <iostream>
    #include <string>
    #include <sstream>
    #include "aho.hpp"


    int main() {
        TAho aho;
        std::string line;
        std::getline(std::cin, line);
        int pid = 0;
        while (!line.empty()) {
            pid++;
            int num;
            auto stream = std::istringstream(line);
            std::vector<unsigned long long int> nums;
            while (stream >> num) {
                nums.push_back(num);
            }
            aho.AddVectorOfNums(nums, pid);
            std::getline(std::cin, line);
        }
        aho.Init();
        std::vector<std::pair<int, int>> info;
        std::vector<unsigned long long int> text;
        int page = 0;
        int num = 0;
        line = "";
        std::getline(std::cin, line);
        while (!line.empty()) {
            page++;
            auto stream = std::istringstream(line);
            num = 0;
            int elem;
            while (stream >> elem) {
                num++;
                text.push_back(elem);
                info.emplace_back(page, num);
```

```
            }
            std::getline(std::cin, line);
        }
        auto answers = aho.FindPosPatterns(text);;
        for (auto answer : answers) {
            std::cout << info[answer.first].first << ", " << info[answer.first].second
                << ", " << answer.second << "\n";
        }
    }
```

```
add_executable(lab4_aho main.cpp include/aho.hpp)
target_include_directories(lab4_aho PRIVATE include)
```

3 Консоль

```
[alex@fedora build]$ cmake ../
--The C compiler identification is GNU 12.2.1
--The CXX compiler identification is GNU 12.2.1
--Detecting C compiler ABI info
--Detecting C compiler ABI info -done
--Check for working C compiler: /usr/bin/cc -skipped
--Detecting C compile features
--Detecting C compile features -done
--Detecting CXX compiler ABI info
--Detecting CXX compiler ABI info -done
--Check for working CXX compiler: /usr/bin/c++ -skipped
--Detecting CXX compile features
--Detecting CXX compile features -done
--Configuring done (6.3s)
--Generating done (0.0s)
--Build files have been written to: /home/alex/mai-da-labs/build
[alex@fedora build]$ cmake --build .
[23/23] Linking CXX executable tests/avl_test
[alex@fedora build]$ cd lab4_aho/
[alex@fedora lab4_aho]$ cat test.txt
1 0002 001 2
1 2 1
2 02 2 2

01 2 1 2   1 2 1 3
2 2 2 2 02
[alex@fedora lab4_aho]$ ./lab4_aho <test.txt
1,1,2
1,1,1
1,3,2
1,3,1
1,5,2
2,1,3
2,2,3
```

4 Тесты

Сравнивать правильность и производительность алгоритма буду с наивным алгоритмом (подстановка каждого паттерна на каждую позицию текста)

```
1   #include <iostream>
2   #include <gtest/gtest.h>
3   #include <random>
4   #include <chrono>
5
6   #include "aho.hpp"
7
8   std::mt19937 randomKey(std::chrono::steady_clock::now().time_since_epoch().count())
        ;
9
10  std::vector<std::pair<unsigned long long, unsigned long long >> WeakAlgo(std::
        vector<std::vector<unsigned long long>>& patterns, std::vector<unsigned long
        long>& text) {
11      std::vector<std::pair<unsigned long long, unsigned long long>> answer;
12      for (size_t i = 0; i < text.size(); ++i) {
13          for (size_t j = 0; j < patterns.size(); ++j) {
14              auto count = patterns[j].size();
15              if (i + count > text.size()) {
16                  continue;
17              }
18              int flag = 0;
19              for (size_t k = i, m = 0; m < count; k++, m++) {
20                  if (text[k] != patterns[j][m]) {
21                      flag = 1;
22                      break;
23                  }
24              }
25              if (flag == 0) {
26                  answer.emplace_back(i, j + 1);
27              }
28          }
29      }
30      return answer;
31  }
32
33
34  TEST(Lab4Test, CommonTest) {
35      std::vector <unsigned long long> text;
36      const int countWords = 1000;
37      const int countPatterns = 1000;
38      std::vector <std::vector<unsigned long long >> patterns(countPatterns);
39      for (size_t i = 0; i < countWords; ++i) {
40          text.push_back(randomKey() % 2 + 1);
41      }
```

```
42          for (size_t i = 0; i < countPatterns; ++i) {
43              auto sizePattern = randomKey() % 100 + 1;
44              std::vector <unsigned long long> pattern;
45              for (size_t j = 0; j < sizePattern; ++j) {
46                  pattern.push_back(randomKey() % 2 + 1);
47              }
48              patterns[i] = pattern;
49          }
50          TAho aho;
51          for (size_t i = 0; i < patterns.size(); ++i) {
52              aho.AddVectorOfNums(patterns[i], i + 1);
53          }
54          aho.Init();
55          auto ahoAnswer = aho.FindPosPatterns(text);
56          auto answer = WeakAlgo(patterns, text);
57
58          std::sort(answer.begin(), answer.end());
59          std::sort(ahoAnswer.begin(), ahoAnswer.end());
60          ASSERT_EQ(answer.size(), ahoAnswer.size());
61          for (size_t i = 0; i < answer.size(); ++i) {
62              ASSERT_EQ(answer[i].first, ahoAnswer[i].first);
63              ASSERT_EQ(answer[i].second, ahoAnswer[i].second);
64          }
65          std::cout << ahoAnswer.size() << "\n";
66      }
67
68      TEST(Lab4Test, Benchmark) {
69          std::vector<unsigned long long> text;
70          const int countWords = 100000;
71          const int countPatterns = 1000;
72          std::vector<std::vector<unsigned long long >> patterns(countPatterns);
73          for (size_t i = 0; i < countWords; ++i) {
74              text.push_back(randomKey() % 2 + 1);
75          }
76          for (size_t i = 0; i < countPatterns; ++i) {
77              auto sizePattern = randomKey() % 100 + 1;
78              std::vector<unsigned long long> pattern;
79              for (size_t j = 0; j < sizePattern; ++j) {
80                  pattern.push_back(randomKey() % 2 + 1);
81              }
82              patterns[i] = pattern;
83          }
84          TAho aho;
85          for (size_t i = 0; i < patterns.size(); ++i) {
86              aho.AddVectorOfNums(patterns[i], i + 1);
87          }
88          auto begin = std::chrono::high_resolution_clock::now();
89          aho.Init();
90          auto ahoAnswer = aho.FindPosPatterns(text);
```

```
91          auto end = std::chrono::high_resolution_clock::now();
92          auto ahoTime = std::chrono::duration_cast<std::chrono::milliseconds>(end -
                begin).count();
93
94          begin = std::chrono::high_resolution_clock::now();
95          auto answer = WeakAlgo(patterns, text);
96          end = std::chrono::high_resolution_clock::now();
97          auto weakTime = std::chrono::duration_cast<std::chrono::milliseconds>(end -
                begin).count();
98
99          std::cout << "aho time: " << ahoTime << " ms\n";
100         std::cout << "weak time " << weakTime << " ms\n";
101     }
```

```
/var/lib/snapd/snap/clion/237/bin/cmake/linux/x64/bin/ctest --extra-verbose
-I 6,6,,6
Testing started at 0:41 ...
UpdateCTestConfiguration  from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
UpdateCTestConfiguration  from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
Test project /home/alex/mai-da-labs/build
Constructing a list of tests
Done constructing a list of tests
Updating test list for fixtures
Added 0 tests to meet fixture requirements
Checking test dependency graph...
Checking test dependency graph end

6: Test command: /home/alex/mai-da-labs/build/tests/lab4_test
6: Working Directory: /home/alex/mai-da-labs/build/tests
6: Test timeout computed to be: 10000000
6: Running main() from /home/alex/mai-da-labs/build/_deps/googletest-src/googletest/sr
6: [==========] Running 2 tests from 1 test suite.
6: [----------] Global test environment set-up.
6: [----------] 2 tests from Lab4Test
6: [ RUN      ] Lab4Test.CommonTest
6: 8871
6: [       OK ] Lab4Test.CommonTest (166 ms)
6: [ RUN      ] Lab4Test.Benchmark
6: aho time: 142 ms
6: weak time 2715 ms
6: [       OK ] Lab4Test.Benchmark (2906 ms)
6: [----------] 2 tests from Lab4Test (3072 ms total)
6:
```

```
6: [----------] Global test environment tear-down
6: [==========] 2 tests from 1 test suite ran. (3072 ms total)
6: [  PASSED  ] 2 tests.

100% tests passed,0 tests failed out of 1

Total Test time (real) =   3.08 sec
Process finished with exit code 0
```

Как видно, алгоритм Ахо-Корасик значительно быстрее наивного алгоритма

Сложность алгоритма Ахо-Корасик $O(n+m+k)$, где n - длина текста, m - суммарная длина паттернов, k - количество вхождений паттернов в текст.

Сложность наивного алгоритма $O(n*m)$, где n - длина текста, m - суммарная длина паттернов.

5 Выводы

В ходе выполнения лабораторной работы я изучил алгоритм Ахо-Корасик, смог реализовать его на C++. Также изучил другие алгоритмы на строках. Столкнулся с ошибкой RE(runtime-error). Ошибка была вызвана с переполнением стека рекурсии из-за сбора ответов с помощью связей выхода. Поэтому в узле бора я сразу собирал всю информацию при инициализации, что помогло избежать рекурсии.

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] *Ахо-Корасик — Вики-конспекты ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Ахо-Корасик` (дата обращения: 18.05.2023).