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. Т. Бахарев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Кнута-Морриса-Пратта..

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

Формат входных данных: Искомый образец задается 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 на количество слов или числ в них, не накладывается.

Формат выходных данных: В выходной файл нужно вывести информацию о всех вхождениях искомых образцов в обрабатываемый текст: по одному вхождению на строчку.

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

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно. Так же хочется отметить, что никакой элемент ни строки, ни образца не сравниваются больше одного раза!

Основным отличием алгоритма Кнутта - Морриса - Пратта от алгоритма прямого поиска заключается в том, что сдвиг подстроки выполняется не на один символ на каждом шаге алгоритма, а на некоторое переменное количе- ство символов. Следовательно, перед тем, как осуществлять очередной сдвиг, необходимо определить величину сдвига. Для повышения эффек- тивности алгоритма необходимо, чтобы сдвиг на каждом шаге был бы как можно большим.

2 Исходный код

main.cpp

```
#include <iostream>
#include <vector>
#include <string>
std::vector<std::string> PatternParse()
{
    char c;
    std::string buf;
    std::vector<std::string> pattern;
    while (true)
    {
        c = getchar_unlocked();
        if (c == ' ')
        {
            if (!buf.empty())
                pattern.push_back(buf);
            buf.clear();
        }
        else if (c == '\n' || c == EOF)
        {
            if (!buf.empty())
                pattern.push_back(buf);
            break;
        }
        else
            buf.push_back(c);
    }
    return pattern;
}
std::vector<int> Zfunc(std::vector<std::string>& pat)
{
    const unsigned long n = pat.size();
    std::vector<int> zArray(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i)
    {
        if (i <= r)
            zArray[i] = std::min(r - i + 1, zArray[i - l]);
        while (i + zArray[i] < n && pat[zArray[i]] == pat[i + zArray[i]])
            ++zArray[i];
        if (i + zArray[i] - 1 > r)
        {
            l = i;
            r = i + zArray[i] - 1;
        }
    }
    return zArray;
}
```

```
47 std::vector<int> CalcSP(std::vector<int>& zArray, std::vector<std::string>& pat)
48 {
49     const unsigned long n = pat.size();
50     std::vector<int> spArray(n);
51     for (unsigned long j = n - 1; j > 0; --j)
52     {
53         unsigned long i = j + zArray[j] - 1;
54         spArray[i] = zArray[j];
55     }
56     return spArray;
57 }
58 std::vector<int> FailFunction(std::vector<std::string>& pattern)
59 {
60     unsigned long n = pattern.size();
61     std::vector<int> zArray = Zfunc(pattern);
62     std::vector<int> spArray = CalcSP(zArray, pattern);
63     std::vector<int> f(n + 1);
64     f[0] = 0;
65     for (int k = 1; k < n; ++k)
66         f[k] = spArray[k - 1];
67     f[n] = spArray[n - 1];
68     return f;
69 }
70 void Kmp(std::vector<std::string>& pat)
71 {
72     std::vector<std::pair<std::pair<int, int>, std::string>> text;
73     char c = '$'; // default value
74     bool wordAdded = true; // True if last action was to add the word into the vector
75     std::pair<std::pair<int, int>, std::string> temp; // current word
76     const unsigned long n = pat.size(); // Pattern size
77     temp.first.first = 1; // Row counter
78     temp.first.second = 1; // Word counter
79     int p = 0; // Pattern pointer
80     int t = 0; // Text pointer
81     std::vector<int> f = FailFunction(pat);
82     do
83     {
84         while (text.size() < pat.size() && c != EOF)
85         {
86             c = getchar_unlocked();
87             if (c == '\n')
88             {
89                 if (!wordAdded)
90                 {
91                     text.push_back(temp);
92                     temp.second.clear();
93                     wordAdded = true;
94                 }
95                 ++temp.first.second;
```

```
96              temp.first.first = 1;
97          }
98          else if (c == ' ' || c == '\t' || c == EOF)
99          {
100             //skip spaces
101             if (wordAdded)
102                 continue;
103             text.push_back(temp);
104             temp.second.clear();
105             ++temp.first.first;
106             wordAdded = true;
107         }
108         else
109         {
110             if (temp.second.front() == '0') // Deletes unnecessary zeros in the
                    number
111                 temp.second.erase(temp.second.begin());
112             temp.second.push_back(c);
113             wordAdded = false;
114         }
115     }
116     if (text.size() < pat.size())// There are no possible matches
117         return;
118     while (p < n && pat[p] == text[t].second )
119     { // We are checking if pattern suits or not
120         ++p;
121         ++t;
122     }
123     if (p == n)
124     { // We found match
125         printf("%d, %d\n", text[0].first.second, text[0].first.first);
126     }
127     if (p == 0)
128         ++t;
129     p = f[p];
130     text.erase(text.begin(), text.begin() + t - p);
131     t = p;
132 }
133 while (c != EOF);
134 }
135 int main()
136 {
137     std::vector<std::string> pattern = PatternParse();
138     Kmp(pattern);
139     return 0;
140 }
```

3 Консоль

```
alex$make
g++ -o da_4 -pg -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long main.cpp -lm
alex$cat 01.t
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
alex$./*4 <01.t
1,3
1,8
alex$cat 02.t
80

80
00080  80

90
alex$./*4 <02.t
2,1
3,1
3,2
alex$
```

4 Тест производительности

Для оценки производительности алгоритма КМП я решил сравнить его с наивным алгоритмом поиска подстроки в строке. Его суть заключается в сравнении всех элементов шаблона с текстом. При несовпадении мы производим сдвиг шаблона на еденицу и повторяем все шаги заново. Ниже приведен листинг бенчмарка:
benchmark.cpp

```
#include <iostream>
#include <string>
void SloySort(const std::string& pattern, const std::string& text)
{
    for(int i = 0; i < text.size() - pattern.size(); ++i)
    {
        int k = i;
        for(int j = 0; j < pattern.size(); ++j)
        {
            if(text[k] == pattern[j])
                ++k;
            else break;
        }
        if(k == i + pattern.size())
            std::cout<<"Match\n";
    }
}
int main()
{
    std::string pattern;
    std::string text;
    long val;
    while(std::cin.get() != '\n')
    {
        std::cin.unget();
        std::cin>>val;
        pattern += std::to_string(val) + " ";
    }
    while(pattern.back() == ' ')
        pattern.pop_back();
    while(std::cin>>val)
    {
        text += std::to_string(val);
        int ch;
        ch = std::cin.get();
        if(ch == ' ') text += " ";
        else text += "\n";
    }
    SloySort(pattern, text);
    return 0;
}
```

```
alex$head 01.t
77 71 69 93
55 82 56 69100 82 95 8868 73 81 5654 77 69 9755 85 69 5382 67 79 6388 78 84
7165 55 54 9077 65 57 5390 82 94 8093 71 92 6066 58 96 5451 66 64 8476 54 58
8060 67 80 6256 85 100 5579 61 83 5152 93 94 7774 79 98 5454 82 55 6970 51
58 9089 99 95 69100 92 76 5069 55 93 6552 74 51 9890 79 88 7862 96 75 6562
88 58 7780 86 94 8065 75 60 9381 80 67 9851 94 56 8487 55 66 9371 80 61 6497
92 83 9190 81 92 69100 91 65 7182 62 90 6979 97 87 7196 73 61 5582 74 86 7461
82 54 6750 71 52 7983 75 52 7261 58 71 10055 58 54 93100 78 52 6581 96 66 9957
63 56 5761 98 53 8757 100 91 9872 61 86 8488 90 58 6087 87 63 8970 85 78 5364
60 81 10057 88 85 6173 55 71 5776 72 91 7886 86 55 8852 52 79 9272 76 92 6961
71 85 5454 84 56 9195 77 79 5360 74 95 7684 71 86 9282 66 77 5156 83 75 8883
62 70 5259 82 60 7476 91 90
alex$make
g++ -o da_4 -pg -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
main.cpp -lm
alex$time ./*4 <01.t >res*

real    0m20.619s
user    0m20.568s
sys     0m0.040s
alex$g++ -std=c++11 bench*
alex$time ./a.out <01.t >res*

real    0m23.504s
user    0m23.208s
sys     0m0.280s
alex$
```

Из теста производительности видно, что КМП работает быстрее, чем наивный алгоритм. Однако в нашем случае разница не значительна. Дело в том, что КМП выигрывает у самого просто алгоритма, когда несовпадение в образце происходит ближе к его концу. Время работы будет $O(m*n)$, тогда как КМП алгоритм будет линейным – $O(m+n)$. В данном тесте я использовал "хорошие"данные – то есть несовпадение в тексте и образце происходило не в самом конце. При "плохих"входных данных алгоритм КМП будет значительно быстрее наивного. Однако и в нашем случае мы видим, что алгоритм Кнутта-Морриса-Пратта производительней.

5 Выводы

Данная лабораторная работа познакомила меня с крайне полезным и эффективным алгоритмом поиска образца в строке – Кнута-Морриса-Пратта. Во время написания этой работы возникали некоторые трудности в работе с данными – входной поток надо было обрабатывать по довольно сложным критериям, а так же было необходимо обрабатывать текст моментально, а не хранить его в строке. Так же, я познакомился с различными контейнерами из STL, что будет для меня очень полезно.

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] *Это маленькое чудо — алгоритм Кнута-Морриса-Пратта (КМП).* URL: `https://habr.com/post/307220/` (дата обращения: 21.09.2018).

[3] *Префикс-функция. Алгоритм Кнута-Морриса-Пратта* `http://e-maxx.ru/algo/prefix-function` (дата обращения: 21.09.2018).