

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

Отчет

Практическая работа №10

Дисциплина Структуры и алгоритмы обработки данных

Тема. Поиск в тексте образца. Алгоритмы. Эффективность алгоритмов.

|  |  |
| --- | --- |
| Выполнил студент | Смольников А.Б. |
|  | Фамилия И.О. |
| Группа | ИКБО-13-21 |
|  | Номер группы |

**Москва 2022**

**Задание 1**

**Вариант №20**

**Разработать и реализовать алгоритм – поиск оператора цикла(поиск Кнута-Морриса-Пратта)**

В соответствии с персональным вариантом необходимо в коде программы на языке С++ определить наличие операторов цикла.

Опишем алгоритм поиска Кнута-Мориса-Пратта:

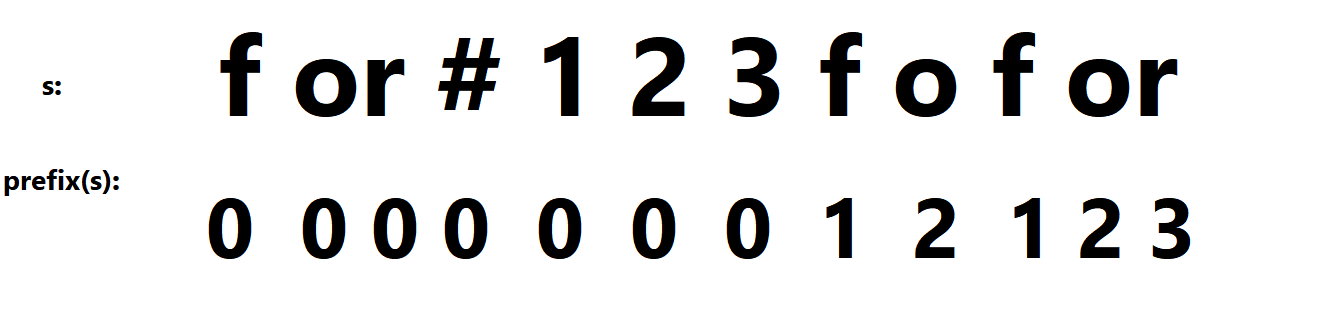
Данный алгоритм выполняет поиск за линейное время и требует наличия префикс-функции. Для начала опишем ее. Префикс функция – функция, возвращающая вектор(или массив) длин наибольших собственных префиксов подстроки, которые одновременно являются и суффиксом этой подстроки. Опишем ее алгоритм. Пусть выходной вектор p размера s.size(), тогда положим первый элемент ноль, циклом от индекса 1 до конца строки выполним действия:

Пока побочный индекс k больше нуля и элементы строки с индексами I и K не равны, то присвоим k значение p[k-1]. Конец цикла

Если эти элементы равны, то инкрементируем k и запишем в p[i]. Префикс функция описана.

Алгоритм сортировки получает на вход шаблон p и строку s. Составляется префикс-функция от строки p+”#”+s, где # - символ, не входящий в алфавит. Далее циклом просматривается префикс функция от # и далее, и если значение префикс-функции в этой ячейке равно длине шаблона, то возвращаем индекс.

Разберем пример, где шаблон - “for”, а строка – “123fofor”



Очевидно, что до индекса 6 строки s значение префикс функции будет равно нулю (префикс не является суффиксом), однако начиная с индекса 7 начнутся вхождения префикса подстроки в суффикс. На индексе 12 значение префикс-функции будет равно 3, что значит, что подстрока найдена. Значение префикс функции не может быть больше, чем длина шаблона, т.к. используется символ вне алфавита.

Рассчитаем количество сравнений для успешного поиска построки в начале строки и для безуспешного поиска(при помощи статик счетчика):

Для безуспешного поиска:

Для вхождения шаблона в начале строки):

Тесты проведем на двух наборах данных: в первом случае искомый шаблон будет вставлен в середину строки при помощи команды insert, второй же поиск будет безуспешным

***Таблица 1. Сводная таблица результатов успешного поиска***

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время,mks** | **Тэт=f(C+M)-функция, mks** | **Т**эп**=Cф+Mф- количество** |
| 100 | 495 | 9 | 910 |
| 1000 | 993 | 95 | 9530 |
| 10000 | 1710 | 952 | 95210 |
| 100000 | 2481 | 9465 | 946543 |
| 1000000 | 16336 | 94529 | 9452948 |

***Таблица 2. Сводная таблица результатов безуспешного поиска***

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время,mks** | **Тэт=f(C+M)-функция, mks** | **Т**эп**=Cф+Mф- количество** |
| 100 | 496 | 10 | 1026 |
| 1000 | 992 | 100 | 10086 |
| 10000 | 1487 | 1006 | 100646 |
| 100000 | 1986 | 8513 | 851308 |
| 1000000 | 14383 | 88576 | 8857688 |

Из таблиц 1 и 2 отчетливо видна линейная зависимость времени поиска от количества входных данных, это и делает поиск Кнута-Мориса-Пратта теоретически самым быстрым с асимптотикой O(n), однако и зависимость памяти тоже линейная.

*Функция prefix:*

Предусловие: string s – входная строка

Постусловие: vector<int> - результат работы префикс функции

1. vector<int> prefix(string s){
2. vector<int> p(s.size());
4. p[0] = 0;
5. int k = 0;
6. for(int i = 1; i < s.size(); i++){
7. while(k > 0 && s[i] != s[k]){
8. k = p[k-1];
9. }
10. if(s[i] == s[k]) k++;
11. p[i] = k;
12. }
13. return p;
14. }

*Функция kmp2:*

Предусловие: string p - шаблон поиска string t - строка

Постусловие: int – индекс первого вхождения

1. int kmp2(string p, string t){
2. int pl = p.size();
3. int tl = t.size();
4. vector<int> pr = prefix(p+"#"+t);
5. for(int i = pl+1; i < tl+pl+1; i++){
6. if(pr[i] == pl){
7. return i-2\*pl;
8. }
9. }
11. return -1;
12. }

**Задание 2. Разработать программу проверки IPv4 адреса на корректность.**

IPv4 адрес – адрес, содержащий 4 числа в диапазоне 0-255 и разделенных точками. Разработаем регулярное выражение:

Число в диапазоне 0-255 можно записать как

(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)

Вертикальная черта – аналог ИЛИ в регулярных выражениях, значит написано либо число 250-255, либо число 200-249, либо число 0-199. Описание последнего отличается, знак вопроса обозначает число повторений: используется(или не используется) 0 или 1, число в диапазоне 0-9 и число в диапазоне 0-9. Теперь запишем полный IPv4 адрес:

^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$

Где ^ обозначает начало регулярного выражения, а $ - конец. Конструкция (X).{3} означает, что литерал или выражение X повторяется ровно три раза. В первых трех случаях к числу в конце добавлена точка – разделитель, а после «цикла» в регулярном выражении еще раз записано число в диапазоне 0-255.

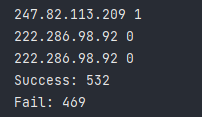
Для проверки на корректность будем использовать встроенную библиотеку С++ regex. Функция regex\_match(string,regex) возвращает булевую переменную – признак сравнения, а само регулярное выражение создается собственным конструктором, принимающим на выход строку – выражение. Таким образом полученная функция:

Функция isValidIPv4:

Предусловие: строка s

Постусловие: bool – результат сравнения

1. bool isValidIpV4(string s){
2. const regex r("^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$");
3. return regex\_match(s, r);
4. }



**Выводы**

В ходе работы был изучен алгоритм Кнута-Мориса-Пратта и написан линейный поиск подстроки, а также исследован синтаксис регулярных выражений.

**Полный код программы**

1. #include <iostream>
2. #include <fstream>
3. #include <string>
4. #include <vector>
5. #include <regex>
6. #include <chrono>
8. using namespace std;
10. void generate\_file() {
11. const string valid = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789\_-+";
12. ofstream file("../input.txt");
13. for (int i = 0; i < 1000000; i++) {
14. file << valid[rand()%valid.size()];
15. }
16. file.close();
17. }
19. void printvector(vector<int> v){
20. for(int i = 0; i < v.size(); i++){
21. cout << v[i] << " ";
22. }
23. cout << endl;
24. }

27. vector<int> prefix(string s){
28. vector<int> p(s.size());
30. p[0] = 0;
31. int k = 0;
32. for(int i = 1; i < s.size(); i++){
33. while(k > 0 && s[i] != s[k]){
34. k = p[k-1];
35. }
36. if(s[i] == s[k]) k++;
37. p[i] = k;
38. }
39. return p;
40. }



45. int kmp2(string p, string t){
46. int pl = p.size();
47. int tl = t.size();
48. vector<int> pr = prefix(p+"#"+t);
49. for(int i = pl+1; i < tl+pl+1; i++){
50. if(pr[i] == pl){
51. return i-2\*pl;
52. }
53. }
55. return -1;
56. }
57. //check string with regex
58. void generate\_ip\_file(){
59. ofstream file("../ip\_input.txt");
60. for(int i = 0; i < 1000; i++){
61. string temp = "";
62. vector<int> v(4);
63. for(int j = 0; j < 4; j++){
64. temp+= to\_string(rand()%300)+".";
65. }
66. file << temp.substr(0,temp.size()-1) << endl;
67. }
68. file.close();
69. }
70. bool isValidIpV4(string s){
71. const regex r("^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$");
72. return regex\_match(s, r);
73. }
75. int main() {
76. srand(time(NULL));
77. //=========================CHECK CYCLES=========================
78. ifstream fin\_check("../input.txt");
79. string s;
80. string cycleTemplate[3] = {"while", "for", "do"};
81. if(!fin\_check.is\_open()){
82. cout << "File not found!" << endl;
83. return 0;
84. }
85. fin\_check.close();
86. /\*for(int i = 0; i < 3; i++){
87. ifstream fin("../input.txt");
89. int pos = -1;
90. int max\_i = -1;
91. while(!fin.eof()){
92. fin>>s;
93. //pos = kmp2(cycleTemplate[i],s);
94. pos=0;
95. max\_i = max(max\_i, pos);
96. }
97. if(max\_i != -1) cout << "Found - " << cycleTemplate[i] <<endl;
98. else cout << "Not found - " << cycleTemplate[i] <<endl;
99. fin.close();
100. }\*/
101. // generate\_file();
102. // return 0;
103. for(int i=100;i<=1000000;i\*=10){
104. ifstream fin("../input.txt");
105. const string substr = "for";
106. string s;
107. fin>>s;
108. s = s.substr(0,i);
109. //s.insert(i/2,substr);
110. int index;
111. auto start = chrono::high\_resolution\_clock::now();
112. index = kmp2(substr,s);
113. auto finish = chrono::high\_resolution\_clock::now();
114. auto duration = chrono::duration\_cast<chrono::microseconds>(finish-start);
115. cout<<"i: "<<i<<" position: "<<index<<" time: "<<duration.count()<<" mks"<<endl;
116. fin.close();
117. }
119. //=========================CHECK IP=========================
120. cout<<isValidIpV4("192.168.1.1")<<endl;
121. generate\_ip\_file();
122. ifstream fin("../ip\_input.txt");
123. string st;
124. if(!fin.is\_open()){
125. cout << "File not found!" << endl;
126. return 0;
127. }
128. int suscess = 0;
129. int fail = 0;
130. while(!fin.eof()){
131. fin>>st;
132. bool state = isValidIpV4(st);
133. cout << st << " " << state<< endl;
134. if(state) suscess++;
135. else fail++;
136. }
137. cout << "Success: " << suscess << endl;
138. cout << "Fail: " << fail << endl;
139. }