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: М. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-208Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №4

Задача: Вариант №5-1 Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.
Вариант алгоритма: Поиск большого количества образцов при помощи алгоритма Ахо-Корасик.
Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые). Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.
Формат входных данных: 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 результата 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.

Примеры:

Входные данные:
cat dog cat dog
CAT dog CaT
Dog doG dog dOg

Cat doG cat dog cat dog cat Parrot
doG dog DOG DOG dog

Результат работы:
1, 1, 1
1, 1, 2
1, 3, 1
1, 3, 2
1, 5, 2
2, 1, 3
2, 2, 3

1 Описание

Алгоритм Ахо — Корасик — алгоритм поиска подстроки, разработанный Альфредом Ахо и Маргарет Корасик в 1975 году, реализует поиск множества подстрок из словаря в данной строке. Широко применяется в системном программном обеспечении, например, используется в утилите поиска grep.
Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

Несколько строк поиска можно объединить в дерево поиска, так называемый бор (префиксное дерево). Бор является конечным автоматом, распознающим одну строку из m — но при условии, что начало строки известно.

Первая задача в алгоритме — научить автомат «самовосстанавливаться», если подстрока не совпала. При этом перевод автомата в начальное состояние при любой неподходящей букве не подходит, так как это может привести к пропуску подстроки (например, при поиске строки aabab, попадается aabaabab, после считывания пятого символа перевод автомата в исходное состояние приведёт к пропуску подстроки — верно было бы перейти в состояние a, а потом снова обработать пятый символ). Чтобы автомат самовосстанавливался, к нему добавляются суффиксные ссылки, на-

груженные пустым символом (так что детерминированный автомат превращается в недетерминированный). Например, если разобрана строка aaba, то бору предлагаются суффиксы aba, ba, a. Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик (в данном случае a).

Для корневого узла суффиксная ссылка — петля. Для остальных правило таково: если последний распознанный символ — c, то осуществляется обход по суффиксной ссылке родителя, если оттуда есть дуга, нагруженная символом c, суффиксная ссылка направляется в тот узел, куда эта дуга ведёт. Иначе — алгоритм проходит по суффиксной ссылке ещё и ещё раз, пока либо не пройдёт по корневой ссылке-петле, либо не найдёт дугу, нагруженную символом c.

2 Исходный код

main.cpp:

```
#ifndef TTRIE
#define TTRIE

#include <memory>
#include <string>
#include <iostream>
#include <queue>
#include <new>
#include <vector>
#include <cstring>
#include <unordered_map>
#include <ctime>

const unsigned short MAX_WORD = 17;
#define T unsigned long long
#define S std::string
#define Pair std::pair

const S TERML = "$";

class TNode
{
private:
    std::unordered_map<S, TNode *> children;
    TNode *fail;
    TNode *exit;
    Pair<T, T> *lengt;

    void Destroy();
    TNode *CreateChild(const S sim);

public:
    TNode()
    {
        lengt = nullptr;
        exit = nullptr;
        fail = nullptr;
    }
    void notTerml(T numStr, T len)
    {
        if (this->children.find(TERML) == this->children.end())
        {
            Pair<S, TNode *> tmpPair = std::make_pair(TERML, new TNode);
            Pair<T, T> *tmpPairInt = new Pair<T, T>;
            *tmpPairInt = std::make_pair(numStr, len);
            tmpPair.second->lengt = tmpPairInt;
```

```
47              this->children.insert(tmpPair);
48          }
49      }
50      ~TNode(){};
51      friend class TTrie;
52  };
53
54  void TNode::Destroy()
55  {
56      if (!this->children.empty())
57      {
58          for (std::unordered_map<S, TNode *>::iterator it = this->children.begin(); it
                != this->children.end(); ++it)
59          {
60              if (it->second != nullptr)
61              {
62                  it->second->Destroy();
63                  delete it->second;
64              }
65          }
66          this->children.clear();
67      }
68      delete this->lengt;
69      this->lengt = nullptr;
70      this->fail = nullptr;
71      this->exit = nullptr;
72  }
73
74  TNode *TNode::CreateChild(const S sim)
75  {
76      auto tmpPair = std::make_pair(sim, new TNode);
77      this->children.insert(tmpPair);
78      return tmpPair.second;
79  }
80
81  class TTrie
82  {
83  public:
84      TTrie()
85      {
86          this->root = new TNode;
87          it = this->root;
88      }
89      void Create(const S &symb);
90      void Itfind(const std::vector<Pair<T int, T int>> &vec, const S &symb);
91      void Linking();
92      void ItReset();
93      ~TTrie()
94      {
```

```
95          this->root->Destroy();
96          this->it = nullptr;
97          delete this->root;
98      };
99
100     TNode *ItGet()
101     {
102         return this->it;
103     }
104
105 private:
106     TNode *root;
107     TNode *it;
108     bool Do(const S &symb);
109     void GetRes(const std::vector<Pair<T int, T int>> &vec, std::unordered_map<S, TNode
            *>::iterator &it);
110 };
111
112 void TTrie::ItReset()
113 {
114     it = this->root;
115 }
116
117 void TTrie::Create(const S &symb)
118 {
119
120     std::unordered_map<S, TNode *>::iterator itt;
121
122     if (this->it->children.empty())
123     { //
124         this->it = it->CreateChild(symb);
125     }
126     else
127     {
128         itt = this->it->children.find(symb);
129         if (itt != this->it->children.end())
130         {
131             it = itt->second;
132             //
133         }
134         else
135         {
136             this->it = it->CreateChild(symb); //
137         }
138     }
139 }
140 bool TTrie::Do(const S &symb)
141 {
142     std::unordered_map<S, TNode *>::iterator itt;
```

```
143     if (!this->it->children.empty())
144     { //
145         itt = this->it->children.find(symb); //
146         if (itt != this->it->children.end())
147         { //
148             it = itt->second;
149         }
150         else
151         { //   ,     fail
152             while (itt == this->it->children.end() && it != this->root)
153             { //    fail   2
154                 it = it->fail;
155                 itt = this->it->children.find(symb);
156             }
157             if (itt != this->it->children.end())
158             { // 1)
159                 it = itt->second; //
160             }
161             else if (it == this->root)
162             { // 2)
163                 itt = this->it->children.find(symb); //
164                 if (itt != this->it->children.end())
165                 { //
166                     it = itt->second; //
167                 }
168             }
169         }
170     }
171     else
172     {
173         if (it->fail)
174         {
175             it = it->fail;
176         } //  fail ,
177     }
178     if (it == this->root)
179     {
180         return false;
181     }
182     else
183     {
184         return true;
185     }
186 }
187
188 void TTrie::GetRes(const std::vector<Pair<T int, T int>> &vec, std::unordered_map<S,
        TNode *>::iterator &it)
189 {
190     auto it1 = this->it;
```

```
191     while (it1)
192     {
193         it = it1->children.find(TERML);
194         if (it == it1->children.end())
195         { //
196             it1 = it1->exit;
197             continue;
198         }
199         auto ptrVec = vec.end();
200         --ptrVec;
201         T patternlenght = it->second->lengt->second;
202         while (ptrVec->second < patternlenght)
203         { //
204             patternlenght -= ptrVec->second;
205             ptrVec--; //
206         }
207         auto WordNumb = ptrVec->second;
208         WordNumb -= patternlenght - 1;
209         printf("%llu,%llu,%llu\n", ptrVec->first, WordNumb, it->second->lengt->first);
210         it1 = it1->exit;
211     }
212 }
213
214 void TTrie::Itfind(const std::vector<Pair<T int, T int>> &vec, const S &symb)
215 {
216     std::unordered_map<S, TNode *>::iterator it;
217     if (Do(symb))
218     { //  Do true  it != root
219         this->GetRes(vec, it);
220     }
221 }
222
223 void TTrie::Linking()
224 {
225     TNode *tmpNode = root;
226     std::queue<TNode *> queue;
227     for (auto it = this->root->children.begin(); it != this->root->children.end(); it
        ++)
228     {
229         queue.push(it->second);
230         it->second->fail = root;
231     }
232     while (!queue.empty())
233     {
234         tmpNode = queue.front();
235         queue.pop();
236         std::unordered_map<S, TNode *>::iterator it; //   unordered_map
237         for (it = tmpNode->children.begin(); it != tmpNode->children.end(); ++it)
238         {
```

```
239             TNode *child = it->second;
240             TNode *parentFail = tmpNode->fail;
241             S childsymbols = it->first;
242             queue.push(child);
243             while (true)
244             {
245                 if (childsymbols != TERML)
246                 {
247                     std::unordered_map<S, TNode *>::iterator existingNode = parentFail->
                        children.find(childsymbols); //      fail
248                     if (existingNode != parentFail->children.end())
249                     { //
250                         if (existingNode->second != child)
251                         { //
252                             child->fail = existingNode->second; // fail
253                             if (existingNode->second->children.find(TERML) !=
                                existingNode->second->children.end())
254                             { //
255                                 child->exit = existingNode->second; // ,
256                             }
257                             else
258                             {
259                                 if (existingNode->second->exit)
260                                 { //
261                                     child->exit = existingNode->second->exit; //
262                                 }
263                             }
264                         }
265                         else
266                         { //
267                             child->fail = root;
268                         }
269                         break;
270                     }
271                 }
272                 else
273                 {
274                     break;
275                 }
276                 if (parentFail == root)
277                 { // fail
278                     child->fail = root; // fail
279                     break;
280                 }
281                 else
282                 { //   fail
283                     parentFail = parentFail->fail;
284                 }
285             }
```

```
286         }
287     }
288 }
289 #endif
290 
291 int main()
292 {
293     char buffer[MAX_WORD];
294     bool sp = false;
295     T l = 1;
296     T numStr = 0;
297     TTrie trie;
298     unsigned short i = 0;
299 
300     char c = tolower(getchar());
301     memset(buffer, 0, MAX_WORD);
302     while (c > 0)
303     {
304         while (c == '\t' || c == ' ')
305         {
306             sp = true;
307             c = tolower(getchar());
308         }
309         i = 0;
310         while (c >= 'a' && c <= 'z')
311         { //
312             sp = false;
313             buffer[i] = c;
314             ++i;
315             c = tolower(getchar());
316         }
317         if (c == '\t' || c == ' ')
318         {
319             sp = true;
320             if (buffer[0] != '\0')
321             {
322                 trie.Create(buffer);
323             }
324             ++l;
325             memset(buffer, 0, MAX_WORD);
326         }
327         else if (c == '\n')
328         {
329             if (buffer[0] != '\0')
330             {
331                 trie.Create(buffer);
332             }
333             ++numStr;
334             if (sp)
```

```
335             {
336                 --l;
337             }
338             trie.ItGet()->notTerml(numStr, l);
339             memset(buffer, 0, MAX_WORD);
340             trie.ItReset();
341             l = 1;
342             c = tolower(getchar());
343             if (c == '\n')
344             {
345                 trie.ItReset();
346                 break;
347             }
348             else
349             {
350                 continue;
351             }
352         }
353         else
354         {
355             std::cerr << "ERROR: incorrect puttern.\n";
356             return 0;
357         }
358         c = tolower(getchar());
359     }
360
361     trie.Linking();
362
363     std::vector<Pair<T int, T int>> vec;
364     Pair<T int, T int> PairStr;
365     PairStr = std::make_pair<T int, T int>(1, 0);
366     vec.push_back(PairStr);
367     c = tolower(getchar());
368
369     while (c > 0)
370     {
371         while (c == '\t' || c == ' ')
372         {
373             sp = true;
374             c = tolower(getchar());
375         }
376         i = 0;
377         while (c >= 'a' && c <= 'z')
378         {
379             sp = false;
380             buffer[i] = c;
381             ++i;
382             c = tolower(getchar());
383         }
```

```
384         if (c == '\t' || c == ' ')
385         {
386             sp = true;
387             if (buffer[0] != '\0')
388             {
389                 ++vec.back().second;
390                 trie.Itfind(vec, buffer); //
391                 memset(buffer, 0, MAX_WORD);
392             }
393         }
394         else if (c == '\n')
395         {
396             if (buffer[0] != '\0')
397             {
398                 if (!sp)
399                 {
400                     ++vec.back().second;
401                 }
402                 trie.Itfind(vec, buffer);
403                 memset(buffer, 0, MAX_WORD);
404             }
405             ++PairStr.first;
406             vec.push_back(PairStr);
407         }
408         else
409         {
410             std::cerr << "ERROR: incorrect input.\n";
411             return 0;
412         }
413         c = tolower(getchar());
414     }
415     return 0;
416 }
```

solution.c	
class TNode	Структура узла
TNode()	Конструктор узла
void notTerml(T numStr, T len)	создаёт терминальный узел и вставляет его следующим за тем на котором мы нааходимся
$\sim TNode()$	деструктор узла
void TNode::Destroy()	унечтожение узла и освобождение его памяти
TNode *TNode::CreateChild(const S sim)	создание дочернего узла
class TTrie	класс для хранения трая
TTrie()	конструктор трая
$\sim TTrie()$	деструктор трая
TNode *ItGet()	возвращает указатель на итератор
void TTrie::ItReset()	переставляет итератор на корень
void TTrie::Create(const S &symb)	создание узла трая с учётом состояния трая
bool TTrie::Do(const S &symb)	поиск строки в трае, если не находит, то возвращается в корень иначе остаётся на искомой строке
void TTrie::GetRes(const std::vector<Pair<T int, T int» &vec, std::unordered$_map < S, TNode* >:: iterator\&it$)	поиск паттернов в трае
void TTrie::Itfind(const std::vector<Pair<T int, T int» &vec, const S &symb)	поиск в трае
void TTrie::Linking()	создание хороших и нет суффиксных ссылок
int main()	главная функция программы

3 Консоль

```
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ g++ -o p main.cpp
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ ./p <test
1,4,3
2,1,3
3,3,2
3,9,3
7,7,2
9,4,2
11,3,1
11,9,3
```

4 Тест производительности

Тест производительности представляет из себя следующее: программа запускается на трёх разных тестах и ищет данные шаблоны в файлах один из которых 100000 строк, другой 15 и 100.

```
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ time ./p <test
real    0m0.022s
user    0m0.006s
sys     0m0.000s
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ time ./p <test >out
real    0m0.021s
user    0m0.000s
sys     0m0.006s
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ time ./p <test1 >out
real    0m0.027s
user    0m0.007s
sys     0m0.000s
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ mv prof pro.txt
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab4$ time ./p <pro.txt
>out
real    0m0.022s
user    0m0.000s
sys     0m0.007s
```

5 Выводы

В результате выполнения лабораторной работы по курсу «Дискретный анализ», были получены навыки использования префиксных деревьев, закреплена работа с утилитой valgrind и навыки использования итераторов и стандартных контейнеров языка C++.

Список литературы

[1] *Префиксное дерево*
URL:https://ru.wikipedia.org/wiki/Префиксное_дерево
(дата обращения: 27.12.2020).

[2] *Trie, или нагруженное дерево*
URL: https://habr.com/ru/post/111874/ (дата обращения: 27.12.2020).

[3] *Алгоритм Ахо-Корасик*
URL: https://habr.com/ru/post/330644/(дата обращения: 27.12.2020).

[4] *Алгоритм Ахо — Корасик*
URL: https://ru.wikipedia.org/wiki/Алгоритм_Ахо_-_Корасик
(дата обращения:27.12.2020).