Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: П. А. Мохляков
Преподаватель: Н. С. Капралов
Группа: М8О-308Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из выходных строк, необходимо воспользоваться полученным суффисным деревом для решения своего варианта задания.

Вариант: Найти в заранее известном тексте поступающие на вход образцы.

Алфавит строк: Строчные буквы латинского алфавита (т.е. от a до z).

1 Описание

Суффиксное дерево позволяет производить поиск подстроки в строке за $O(m + k)$, где m это длина подстроки, а k число ее вхождений. Для быстрой работы нам также необходимо быстро строисть само суффиксное дерево. В этом нам помогает алгоритм Укконена, который строит суффиксное дерево за линейное время относительно длины текста.

2 Исходный код

Начнем с построения суффиксного дерева. Суффиксное дерево будет на основе com pact trie, а внесто строк в ребрак будут использоваться номера индексов начала и конца участка в исходном тексте. Trie будет достраиться для каждого символа текста. Добавление символов состоит и нескольких этапов.

Во первых мы добавляем наш символ ко всем ребрам листьям, так как они являлись концами суффиксов строки без нашего нового символа.

Теперь нам нужно добавить все не достающие суффиксы, при этом мы уже на какой-то позиции в trie, мы можем оказаться в одной из ситуаций, символ уже существует (то есть наш суффикс является подстрокой другого), мы между ребрами и ребра, который начинается с нашего символа не существует, мы посреди ребра и следующий символ отличается от нашего. В первом случае мы просто переходим в trie по этому символу и закончить, во втором случае мы должны должны создать новый лист с нужным символом, а в третьем нам придется разорвать ребро на два и уже потом создать новый лист.

Во втором и третьем случае мы должны продолжать цикл до того момента пока мы не окажемся либо в корне после выполнения, либо пока не попадем в первый слечай. При этом мы для повтора каждый раз переходим по суффиксной ссылке или если ребро начинается из корне, переходим по суффиксу ребра.

Для суффксных создания мы при каждом разделении ребра по третьему случаю, записываем указатель на место разделения в отдельюную переменную, а при последующий создании чего-то нового по правилам 2 и 3 проверяем нужно ли нам привязать эту суффиксную ссылку к аналогичному символу.

SuffTree.cpp

```cpp
#include "SuffTree.h"
#include <iostream>

TSuffTree::~TSuffTree(){
    delete Root;
}

TSuffTree::TNode::~TNode(){
    for(auto &i: Edges){
        delete i.second;
    }
}

TSuffTree::TSuffTree(std::string &str){
    Text = str + "$";

    int size = Text.size();
    Root = new TNode(-1,-1,-1);
    CurNode = Root;
    for(int i = 0; i < size; ++i){
        Add(i);
    }
}


void TSuffTree::Add(int inpos){
    LastAdd = nullptr;
    ++CountSuff;
    while(CountSuff){
        if(Pos == 0){
            CurEdge = inpos;
        }
        if(CurNode->Edges.find(Text[CurEdge])==CurNode->Edges.end()){
            CreateList(inpos,CurNode);
            CreateSufflink(CurNode);
        } else {
            if(EdgeFault()){
                continue;
            }
            TNode *Edge = CurNode->Edges[Text[CurEdge]];
            if(Text[Edge->Left+Pos] == Text[inpos]){
                CreateSufflink(CurNode);
                Pos++;
                break;
            } else {
                BreakCreationNode(inpos);
            }
        }
```

```
49              --CountSuff;
50              GoSuffLink();
51          }
52      }
53
54      void TSuffTree::CreateList(int inpos,TNode *Node){
55          CountLists++;
56          TNode *List = new TNode(inpos,Text.size() - 1,CountLists);
57          Node->Edges[Text[inpos]] = List;
58      }
59
60      void TSuffTree::CreateSufflink(TNode *Node){
61          if(LastAdd != nullptr){
62              LastAdd->SuffLink = Node;
63              LastAdd = nullptr;
64          }
65      }
66
67      bool TSuffTree::EdgeFault(){
68          TNode *Edge = CurNode->Edges[Text[CurEdge]];
69          int lenedge = Edge->Right - Edge->Left + 1;
70          if(Pos >= lenedge){
71              CurEdge += lenedge;
72              Pos -= lenedge;
73              CurNode = Edge;
74              return true;
75          }
76          return false;
77      }
78
79      void TSuffTree::BreakCreationNode(int inpos){
80          TNode *Edge = CurNode->Edges[Text[CurEdge]];
81          TNode *SplitNode = new TNode(Edge->Left,Edge->Left + Pos - 1,-1);
82          CurNode->Edges[Text[CurEdge]] = SplitNode;
83          Edge->Left += Pos;
84          SplitNode->Edges[Text[Edge->Left]] = Edge;
85          CreateList(inpos,SplitNode);
86          CreateSufflink(SplitNode);
87          LastAdd = SplitNode;
88      }
89
90      void TSuffTree::GoSuffLink(){
91          if(CurNode == Root){
92              if(Pos > 0){
93                  --Pos;
94                  ++CurEdge;
95              }
96          } else {
97              if(CurNode->SuffLink != nullptr){
```

```
98                CurNode = CurNode->SuffLink;
99            } else {
100               CurNode = Root;
101           }
102       }
103   }
104
105
106
107   void TSuffTree::Find(std::string &pattern, std::vector<int> &ans){
108       ans.clear();
109       int size = pattern.size();
110       TNode *Cur = Root;
111       for(int i = 0;i < size;){
112           if(Cur->Edges.find(pattern[i]) != Cur->Edges.end()){
113               Cur = Cur->Edges[pattern[i]];
114               for(int j = Cur->Left; j <= Cur->Right && i < size; ++i,++j){
115                   if(Text[j] != pattern[i]){
116                       return;
117                   }
118               }
119           } else {
120               return;
121           }
122       }
123       Cur->ListsNums(ans);
124       std::sort(ans.begin(),ans.end());
125   }
126
127
128   void TSuffTree::TNode::ListsNums(std::vector<int> &ans){
129       if(NumList == -1){
130           for(auto &i:Edges){
131               i.second->ListsNums(ans);
132           }
133       } else {
134           ans.push_back(NumList);
135       }
136   }
137
138   void TSuffTree::TNode::Print(int level,std::string &text){
139       for(int i = 0; i < level; ++i){
140           std::cout << "\t";
141       }
142       for(int i = Left; i<=Right;++i){
143           std::cout << text[i];
144       }
145       std::cout << std::endl;
146       for(auto &i:Edges){
```

```
147             i.second->Print(level + 1,text);
148         }
149     }
150 
151     void TSuffTree::Print(){
152         Root->Print(0,Text);
153     }
```

main.cpp

```
1  #include <iostream>
2  #include "SuffTree.h"
3  
4  int main(){
5      std::string str;
6      std::cin >> str;
7      TSuffTree tree(str);
8      int i = 1;
9      while(std::cin >> str){
10         std::vector<int> ans;
11         tree.Find(str,ans);
12         if(!ans.empty()){
13             std::cout << i << ": " << ans[0];
14             for(int j = 1;j < ans.size();++j){
15                 std::cout << ", " << ans[j];
16             }
17             std::cout << std::endl;
18         }
19         ++i;
20     }
21     return 0;
22 }
```

main.cpp	
int main()	Считывает текст, вызывает создание суффиксного дерева, считывает паттеры и выводит ответ
SuffTree.hpp	
void Add(int inpos)	Добавить символ в суффиксное дерево
void CreateList(int inpos,TNode *Node)	Создает лист в суффиксном дереве
void CreateSufflink(TNode *Node)	Создает суффиксную ссылку
bool EdgeFault()	Переходит к следующему ребру, если выйти за пределы действующего
void BreakCreationNode(int inpos)	Разрывает ребро и создает новый лист
void GoSuffLink()	Переход по суффиксной ссылке
void Find(std::string &pattern, std::vector<int> &ans)	Поиск паттерна в текте

3 Консоль

```
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB5 cat test
abcdabdaabdc
abcd
bcd
bc
ab
abc
abf
a
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB5 make
rm -rf *.o solution
g++ --std=c++14 -c -O2 main.cpp
g++ --std=c++14 -c -O2 SuffTree.cpp
g++ main.o SuffTree.o -o solution
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB5 cat test| ./solution
1: 1
2: 2
3: 2
4: 1,5,9
5: 1
7: 1,5,8,9
```

4 Тест производительности

```
Size text: 10
Count pattens: 2
Ukkonen generate + Suffix tree find time: 1.567e-05
Naive find time: 4.1e-07
Size text: 100
Count pattens: 18
Ukkonen generate + Suffix tree find time: 0.000124641
Naive find time: 2.8074e-05
Size text: 1000
Count pattens: 998
Ukkonen generate + Suffix tree find time: 0.00493637
Naive find time: 0.0119987
Size text: 1000
Count pattens: 9998
Ukkonen generate + Suffix tree find time: 0.0345626
Naive find time: 0.11349
Size text: 1000
Count pattens: 99998
Ukkonen generate + Suffix tree find time: 0.335328
Naive find time: 1.12146
Size text: 1000
Count pattens: 999998
Ukkonen generate + Suffix tree find time: 3.32867
Naive find time: 11.1543
```

Как видно, создание суффиксного массива и поиск по нему работает быстрее, чем поиск наивным алгоритмом. И разница будет тем больше, чем больше запросов на поиск и ниже энтропия текста и паттернов.

5 Выводы

Выполнив лабораторную работу по курсу «Дискретный анализ», я познакомился с такой структурой данных как суффиксное дерево. Его основое предназначение поиск образцов в тексте. В отличии от расмотренных ранее в предыдущей лабораторвной работе алгоритмов поиска образцов в тексе суффиксное дерево отличается тем, что преобразовывает текст, а не образец.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах.* — БХВ-Петербург, 2003. Перевод с английского: И. В. Романовский — 658 с. (ISBN 5-94157-321-9)