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: М. А. Бронников
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена постояния суффиксного дерева за линейное время. Построив таким образом дерево для некотрых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания.

Вариант 3: Поиск образца с использованием статистики совпадений

Найти образец в тексте используя статистику совпадений.

Входные данные: на первой строке располагается образец, на второй — текст.

Выходные данные: последовательность строк содержащихв себе номера позиций, начиная с которых встретился образец. Строки должны быть отсортированны в порядке возрастания номеров.

1 Описание

Как сказано в [1]: «Суффиксное дерево для m-символьной строки S — это ориентированное дерево с корнем, имеющее ровно m листов, занумерованных от 1 до m. Каждая внутренняя вершина, отличная от корня, имеет не меньше 2 детей, а каждая дуга помечена непустой подстрокой строки S (дуговой меткой). Никакие 2 дуги, выходящие из одной и той же вершины, не могут иметь пометок, начинающихся с одного и того же символа. Главная особенность суффиксного дерева заключается в том, что для каждого листа i конкатенация меток дуг на пути от корня к листу i в точности состовляет (произносит) суффикс строки S, которыйначинается в позиции i. То есть этот путь произносит $S[i..m]$».

Вершины дерева будем иногда называть *явными* вершинами, когда как под *мнимой* вершиной будем понимать одно из положений на дугах между символами, которые составляют подстроки, помечающие эти дуги.

Строить суффиксное дерево будем при помощи *алгоритма Укконена.*
Основнй идеей алгоритма является последовательное построение суфиксного дерева для строки $S\$$ длины $n+1$ в режиме online, т.е. в каждой $i+1$-ой фазе из $n+1$ фаз алгоритма преобразуется неявное суффиксное дерево, постоенное в предыдущей фазе алгоритма для префикса строки длины i, для получения неявного дерева префикса строки длины $i+1$ при помощи **3-ёх правил продолжения суффиксов** (при этом неявное суффиксное дерево, построенное в фазе $n+1$ соответствует искомому суффиксному дереву для строки $S\$$).
Правила продолжения суффиксов:

1. Если путь кончается в листе следует добавить символ в конец листа.
2. Если путь кочается во внутренней вершине и не существует дуг, которые начинаются с добавляемого символа:
   (a) Если вершина мнимая, тогда создается новая вершина и в ней развилка к которой добавляется листовая дуга, котороя начинается с добавляемого символа.
   (b) Если вершина явная, тогда к ней просто добавляется новая листовая дуга, котороя начинается с добавляемого символа.
3. Если путь кончается в вершине и существует дуга из вершины, которая начинается с добавляемого символа, ничего делать не надо.

При этом в каждой $i+1$-ой фазе алгоритм осуществляет $i+1$ продолжение суффиксов неявного дерева префикса длины i строки $S\$$.

Данный алгоритм имеет сложность $O(n^3)$, однако применение **4 приёмов реализации**,целесообразность которых вытекает из нетрудных наблюдений за исполнеием описанного алгоритма, обеспечивают $O(n)$ — линейную сложность работы *алгоритма Укконена.*
Приёмы реализации:

1. Переход по *суффиксным связям* с использованием скачка по счетчику для быстрого поиска продолжений суффиксов в дереве вместо пследовательного поиска продолжений от корня.

2. Вместо хранения всей подстроки в каждой дуге хранить только 2 числа, первое из которых обозначает номер первого символа дуги в строке $S\$$, а второе обозначает длину этой строки.

3. Вместо того, чтобы каждый раз применять *правило 1* продолжения ранее созданных листов во всех фазах, сразу при создании устанавливать на дугах длину равную длине всей строки $S\$$. При этом хранить указатель на последнюю вершину в которой было применено правило создания листа (*правило 2*).

4. При первом же использовании *правила 3* прекращать дальнейшее исполнение текущей фазы и переходить к следующей фазе, так как во всех последующих продолжениях не потребуется никаких действий.

2 Исходный код

Для хранения вершин опишем структуру $TNode$.
Массив $child$ будет хранить указатели на вершины, из которых есть дуги от текущей вершины.
$list$ содержит в себе 0 для всех развилок или номер листа, которым является дуга ведущая в эту вершину, если она является листовой.
Число $lenght$ — длина дуги, ведущей в эту вершину.
Так как дуга определяется не только длиной, но и номером первого символа дуги в строке, для которой строится дерево, то число $first$ будет хранить в себе этот номер для дуги, ведущей в рассматриваемую вершину.
Наконец суффиксная ссылка для явной вершины задается указателем $suffix$ перехода к следующей вершине

Для хранения суффиксного дерева создадим класс $SufTree$, который будет считывать строку str и строить для нее дерево.
Как и для любых других деревьев для реализации суффиксного дерева нам понадобится иметь доступ к корню $head$ дерева.
Для оптимизации алгоритма и уменьшеня количества сравнений введем вспомогательную вершину $dime$, на которую будет указывать суффиксная ссылка корня. Все дуги из $dime$ для каждого символа алфавита будут указывать на корень и будут иметь длину 1.
Не самым удачным решением с точки зрения оптимизации памяти было объявления вектора $mass$, необходимого для быстого освобождения памяти в деструкторе (Более оптимальным решением было бы создание вектора вершин, а в реализации вершины хранить не указатели, а индексы в этом векторе).

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cstring>
5
6  using namespace std;
7
8  #define ALFASIZE 27 // 26 + terminal
9
10 typedef struct TNode{
11   TNode* child[ALFASIZE];
12   unsigned list;
13   unsigned lenght;
14   unsigned first;
15   TNode* suffix;
16 }TNode;
17
```

```
18 class SufTree{
19 public:
20   SufTree(istream& is);
21   string& GetStr();
22   ~SufTree();
23   void StatFind(istream& is);
24
25 private:
26   TNode dime;
27   TNode head;
28   string str;
29   std::vector<TNode*> mass;
30 };
```

Основные функции и процедуры	
SufTree::SufTree(istream& is)	Конструктор суффиксного дерева по алгоритму Укконена для образца из потока
SufTree:: SufTree()	Деструктор суффиксного дерева
string& SufTree::GetStr()	Функция, возвращающая строку, для которой построенно суффиксное дерево
void SufTree::StatFind(istream& is)	Процедура, выполняющая поиск статистики совпадений для текста из потока и выводящая позицию вхождения образца в текст, если статистика совпадает с длиной строки S

3 Консоль

Тесты для этой лабораторной я генерировал вручную, так как каждый тест имел целью проверить определенные особенности работы программы, которые исследовать программно сгенерированными тестовыми файлами довольно трудно.

```
max@max-X550CC:~/DA/lab5$ cat test1
abcab
abcaabcabcabaebcabdsklkkmabcabcab
abeabcbrye
ababebsfslkjlksjdhflkjhfkljhdszflzhflzlsd
aaaaaaaa
aaaaaaaaaaaaa
r
abckahfkfhrkhfahealeijfrlkjvhrkljvsarlkjdrlkjsadr
babcabr
ababcabrbrbabcabr
max@max-X550CC:~/DA/lab5$ g++ -Wall -pedantic main.cpp -o run
max@max-X550CC:~/DA/lab5$ ls
main.cpp  run  test1
max@max-X550CC:~/DA/lab5$ ./run <test1
5
8
26
29
1
2
3
4
5
6
11
24
30
37
42
49
2
11
```

4 Выводы

Выполнив пятую лабораторную работу по курсу «Дискретный анализ», я получил знания о таком понятии, как суффиксные деревья. Мне пришлось узнать и разобраться в том, как их строить за линейное время при помощи алгоритма Укконена на языке $C++$. Мне было интесно познакомится с множеством приложений суффиксных деревьях в самых различных предметных областях, ведь ранее я и не подозревал о том насколько они могут быть полезны и эффективны.

Эта лабораторная оказалась самой трудной из тех, что я делал до этого, так как все алгоритмы, применяемые мною ранее имели, на мой взгляд, более оптимальное для программной реализации описание. На всем протяжении работы мне казалось, что выбранная мной идея описания функций и структур для построения дерева является неверной или неудачной, что заставляло меня тратить огромное количество драгоценного времени на размышления о том как сделать код более «элегантным» и эффективным.

Однако несмотря на все сложности и неприятности я не пожалел о том, что мне пришлось столкнуться с этой темой, ведь она является необходимой для любого, кто планирует стать хорошим программистом, а трудности встречают любого на пути к успеху.

Список литературы

[1] Дэн Гасфилд *Строки, деревья, и последвательности в алгоритмах: Информатика и вычислительная биология* — Издательский дом «Невский диалект», 2003. Перевод с английского: И. В. Романовского — 654 с. (ISBN 5-7940-0103-8 (рус.))

[2] *Построение суффиксного дерева: алгоритм Укконена*
URL: `https://habr.com/ru/post/111675` (дата обращения: 30.05.2019).