Лабораторная работа №5 по курсу Дискретного Анализа: Суффиксные деревья

Выполнил студент группы 08-308 МАИ Жерлыгин Максим Андреевич

Условие

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.

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

Вариант 4: Линеаризовать циклическую строку, то есть найти минимальный в лексикографическом смысле разрез циклической строки.

Формат входных данных

Некий разрез циклической строки.

Формат результата

Минимальный в лексикографическом смысле разрез.

Метод решения

Реализовать алгоритм Укконена, строящий суффиксное дерево за линейное время. Для этого использовать 2 оптимизации.

Оптимизация №1: был листом, листом и останешься. При добавлении символа к листу получаем новый лист. Поэтому пусть, мы создаём лист не только для рассмотренной части строки, а для всей все строки до конца. Для этого правой границе подстроки, соответствующей ребру до листа, сопоставим бесконечность. Таким образом, пропадает необходимость каждый раз продлевать листы. В итоге, если мы для какого-то суффикса создали лист, то этот суффикс мы больше рассматривать не будем. Понятно, что на асимтотику это не влияет, но избавляет нас от ненужных действий.
Оптимизация №2: если просто прошли по ребру, то и в меньших суффиксах мы тоже пройдём по ребру. У нас есть 3 действия с вершинами при добавлении символа:

1) Продление листа (выполняется для листа)
2) Создание развилки (выполняется для мнимой вершины или развилки, из которых

нет ребра с этим символом)
3) Просто проход по ребру (выполняется для мнимой вершины или развилки, из которых есть такое ребро)

Утверждается, что если рассматривать суффиксы по уменьшению длины, то действия с ними будут выполняться в таком порядке 11...12...23..33. (Это следует из того, что из вершины суффикса данной вершины есть хотя бы все рёбра, аналогичные.

Исходный код

Первоначально я создал класс SuffixTreeNode, в котором описал конструктор, построение суффиксного дерева с помощью алгоритма Укконена, а также функцию GetLinear CircleString, которая и является ответом на поставленную задачу. Также у меня имеется массив потомков, от размера алфавита children[ALPHABET_SIZE] типа shared pointer, и suffix_link тогоже типа

```
class SuffixTreeNode {
public:
    SuffixTreeNode(int start, int *end);
    void ExtendSuffixTree(int pos);
    int EdgeLength();
    void GetLinearCircleString();
    void SetSuffixLink();
private:
    std::shared_ptr<SuffixTreeNode> children[ALPHABET_SIZE];
    std::shared_ptr<SuffixTreeNode> suffix_link;
    int start;
    int *end;
    int suffix_index;
};
```

Затем создал структуру ActivePoint, которая состоит из вершины типа shared pointer, символа и длины формата int.

```
struct ActivePoint {
    std::shared_ptr<SuffixTreeNode> node;
    int symbol;
    int length;
};
```

Нужно описать функцию продления дерева ExtendSuffixTree, котрая будет оптимально за линию увеличивать длину нашего дерерва. В этой функции содержится основная часть алгоритма Укконена.

```
void SuffixTreeNode::ExtendSuffixTree(int pos) {
    std::shared_ptr<SuffixTreeNode> cur = root;
    std::shared_ptr<SuffixTreeNode> link = root;
    std::shared_ptr<SuffixTreeNode> next_node, cur_node;

    ++REMINDER;

    while (REMINDER > 0) {

        if (actPoint.length == 0) {
            actPoint.symbol = pos;
        }

        int symbol = text[actPoint.symbol] - 'a';
        if (!actPoint.node->children[symbol]) {
            actPoint.node->children[symbol] = std::make_shared<
               SuffixTreeNode>(pos, &END);

            if (link != root) {
                link->suffix_link = actPoint.node;
                link = root;
            } else link = actPoint.node;
        } else {

            if (actPoint.length >= actPoint.node->children[symbol]->
               EdgeLength()) {
                int length = actPoint.node->children[symbol]->
                   EdgeLength();
                actPoint.length -= length;
                actPoint.node = actPoint.node->children[symbol];
                actPoint.symbol += length;
                continue;
            }

            if (text[actPoint.node->children[symbol]->start + actPoint.
               length] == text[pos]) {
                actPoint.length++;

                if (link != root)
                    link->suffix_link = actPoint.node;
                break;
            }

            cur = actPoint.node->children[symbol];

```

```
42              int *position = new int;
43              *position = cur->start + actPoint.length - 1;
44              int next_pos = cur->start + actPoint.length;
45              int next_symbol = text[next_pos] - 'a';
46              int cur_symbol = text[pos] - 'a';
47
48              cur_node = std::make_shared<SuffixTreeNode>(pos, &END);
49              next_node = std::make_shared<SuffixTreeNode>(cur->start,
                    position);
50
51              actPoint.node->children[symbol] = next_node;
52              cur->start = next_pos;
53              next_node->children[next_symbol] = cur;
54              next_node->suffix_index = -1;
55              next_node->children[cur_symbol] = cur_node;
56
57              if (link != root)
58                  link->suffix_link = next_node;
59              link = next_node;
60          }
61
62          REMINDER--;
63
64          if (actPoint.node == root && actPoint.length > 0) {
65              actPoint.length--;
66              actPoint.symbol = pos - REMINDER + 1;
67          }
68          else {
69              actPoint.node = actPoint.node->suffix_link;
70          }
71
72      }
73  }
```

И, теперь нужно описать функцию ответа - GetLinearCircleString. Чтобы выполнить поставленную задачу оптимально и за линейное время мне нужно входную строку p длины n увеличить вдвое таким образом, что за исходной идет она же, без разделителей. Таким образом, я имеею строку pp, длины 2n, и для этой строки строю суффиксное дерево за O(2n). Затем, осталось только произвести лексикографический поиск по этому дерево, чтобы найти наименьшую строку, которая и будет ответом.

```
1
2  void SuffixTreeNode::GetLinearCircleString() {
3      SuffixTreeNode *cur = root.get();
4      size_t size = 0;
```

```
5       while (true) {
6           for (size_t i = 0; i < ALPHABET_SIZE; ++i) {
7               if (cur->children[i]) {
8                   for (int k = cur->children[i]->start; k <= *(cur->
                        children[i]->end); ++k) {
9                       if (size < SIZE)
10                          std::cout << text[k];
11                      else {
12                          std::cout << "\n";
13                          return;
14                      }
15                      ++size;
16                  }
17
18                  cur = cur->children[i].get();
19                  break;
20              }
21          }
22      }
23  }
```

Пример работы

```
1  mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
       lab5$ make clean
2  rm -f *.o solution
3  mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
       lab5$ make
4  g++ -std=c++17 -pedantic -Wall -O2 -c main.cpp -o main.o
5  g++ -std=c++17 -pedantic -Wall -O2 main.o -o solution
6  mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
       lab5$ ./solution
7  xabcd
8  abcdx
9  mmaxim2710@DESKTOP-RDPBU3D:/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/
       lab5$ ./solution
10 cdab
11 abcd
```

Вывод

Для меня эта лабораторная работа оказалась одной из самых сложных, как в плане понимания, так и в плане реализации. Не достаточно было только понять как работа-

ет этот алгоритм, но нужно было еще и придумать как это написать. Однако все же удалось с помощью различных источников реализовать поставленную задачу. Дерево корректно строится за $O(n)$, доказательство этого лежит в доказательстве корректности алгоритма Укконена.

Т.к. переход по прямой ссылке работает за $O(1)$, а их не более N, то достаточно только рассмотреть переход вниз по рёбрам. Для этого рассмотрим длину смещения по ребру, по которому мы поднялись к предку. Пусть она равна L. Тогда при каждом прохождении во рёбрам вниз эта длина уменьшается хотя бы на 1. А увеличивается она только в основоной процедуре при проходе по ребру по символу C, причём не более чем на 1 за раз. А таких проходов N. Конечное значение смещения лежит в диапазоне $[0, N]$, количество удлинений N, то и суммарное количество сокращений может быть не более $2N$. Значит, суммарное количество продвижений вниз при переходе по суффикной ссылке $O(3N)$. В итоге, количество каждого вида действий $O(N)$, а значит и суммарное количество всех действий $O(N)$.

Первая, и возможно самая распространённая, практическая задача, связанная с суффиксными деревьями, это нахождение включения одной строки в текст. Для этой задачи есть и другие алгоритмы, но если мы её немного модифицируем, например, нужно будет искать много раз, но мы не может обработать все запросы заранее, как в алгоритме Ахо-корасика, то тогда просто проходя по дереву по символам из поискового запроса, мы либо попытаемся выйти из дерева, тогда ответ отрицательный, либо, если запрос закончился, ответ положительный.