Лабораторная работа №5 по курсу дискретного анализа: Поиск с использованием суффиксного массива

Выполнил студент МАИ группы М8О-201Б *Ефимов Александр.*

Постановка задачи

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

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

Вариант:

Найти в заранее известном тексте поступающие на вход образцы с использованием суффиксного массива.

Метод решения

По условию задачи для реализации суффиксного массива необходимо реализовать суффиксное дерево.

Суффиксное дерево можно реализовать за $O(n*log(n))$: само дерево можно построить за линейное время, а сортировка суффиксов происходит за $O(n*log(n))$ (сортировка суффиксов будет необходима для построения суффиксного массива за один обход).

Построение суффиксного дерева делается за **n** обходов в цикле при помощи алгоритма Укконена, где **n** – длина строки, для которой строится суффиксное дерево. Во время построения будут активно использоваться следующие переменные:

- глобальный указатель на текущий конец суффиксов *globalRef*;
- оставшееся для добавления количество символов *remainder*;
- указатель на активную точку в дереве *activePoint*.

Будет считаться, что индексация начинается с 0 до n - 1, а указатель на конец является past-the-end.

Перед началом цикла *globalRef* и *remainder* должны быть равны нулю, *activePoint* указывает на корень дерева.

Каждый раз при входе в цикл *globalRef* и *remainder* увеличиваются на 1. У каждого листа дерева указатель на конец должен быть одинаковым в каждом листе, поэтому *globalRef* используется, чтобы не обходить все листья дерева. Также, в будущем будет использоваться переменная *prevCreated*, которая указывает на предыдущую созданную вершину дерева, но в начале цикла она указывает на корень.

Дальше начинается еще один цикл, который продолжается до тех пор, пока *remainder* не равен 0 (т.е. пока есть что вставить). В начале этого цикла, если активная точка не

указывает ни на какой символ на гране в дереве (т.е. указывает лишь на вершину в нем), то указать ее на грань этой вершины, начальный символ которой равен символу текущего конца добавляемого суффикса.

Если такая вершина не найдена, то добавить новый лист и попробовать привязать суффиксный указатель прошлой созданной вершины на этот лист.

Если такая грань есть, то сравнить символ, на который указывает активная точка, с текущим концом суффикса: если они равны, то сдвинуть активную точку на один символ вперед, привязать прошлую созданную вершину к текущей активной вершине и выйти из внутреннего цикла. Иначе создать вершину посередине этой грани.

В этом же внутреннем цикле, убавить на один переменную *remainder*. Если текущая активная вершина не является корнем, то перейти на новую вершину, указываемую этой суффиксным указателем. Иначе выбрать грань, начальный символ которого стоит после прошлого добавленного.

После того, как суффиксное дерево было построено (и суффиксы в нем отсортированы в вершинах), обходом в глубину можно найти все суффиксы в отсортированном виде и построить массив.

Для поиска образца в этом массиве, достаточно бинарным поиском найти нижний и верхний границы, в пределах которых образец совпадает с частью суффикса.

Описание программы

1. **SuffTree.h** Содержит в себе объявление классов для вершины дерева *TNode*, для самого дерева *TSuffTree* и для суффиксного массива *TSuffArray*, а также объявление их методов:

   - *TNode*
     - *Конструктор* – принимает на вход ссылку на глобальную строку, ссылку на начальную строку, нижний и верхний пределы, на которые указывает грань перед этой вершиной;
     - *FindChildByChar* – ищет на следующий узел по первому символу соединяющей грани;
     - *AddLeaf* – создает лист у вершины с указанным нижним пределом;
     - *AddNode* – Принимает нижний и верхний пределы как параметры, а также первый символ грани, к которой добавить эту новую внутреннюю вершину;
     - *GetSuffixLink* и *SetSuffixLink* – соответственно получает и меняет суффиксный указатель
     - *GetLower* и *SetLower* – соответственно получает и меняет нижний предел;
     - *GetUpper* – получает верхний предел, причем если вершина - лист, то возвращает значение глобального указателя, иначе значение верхнего предела внутренней вершины;
     - *GetLength* – возвращает длину подстроки, на которую указывает грань;

- *GetChar* – возвращает символ подстроки грани под указанным индексом, причем индексация относительно этой подстроки;
- *GetString* – возвращает всю подстроку грани;
- *FillArray* – принимает массив, в который будут добавляется суффиксы (а именно в виде нижние пределов, т.к. их верхние пределы равны длине строки), а также строку, состоящую из всех подстрок граней до этой вершины.

- *TSuffTree*
  - *Конструкторы* – по умолчанию обнуляет переменные, но если передать строку, то строит по ней дерево;
  - *SetString* – задает новую строку;
  - *Construct* – строит дерево в соответствии с описанным ранее алгоритмом;
  - *IncrementGlobal* – увеличивает значение глобального указателя на верхний предел суффиксов;
  - *TryLink* – попытаться приравнять суффиксный указатель прошлой созданной вершины к параметру, если прошлая созданная переменная не равна корню;
  - *WalkDown* – Если длина текущей активной точки больше длины грани, на которую эта активная переменная указывает, то просто спуститься по этой грани до следующей вершины.
    Переменные:
  - *globalUpper* – глобальное значение верхней грани суффиксов;
  - *remainder* – количество символов, которое осталось добавить;
  - *activePoint* – структура из трех переменных: указатель на активную вершину, первый символ грани из этой вершины и длины (указывает на символ на этой грани).

- *TSuffArray*
  - *Конструктор* – принимает множество нижних пределов, из которых можно построить суффиксы строки, а также саму строку;
  - *Find* – принимает образец и ищет пределы, в которых этот образец равен началу суффикса или всему ему, после чего возвращает индексы всех позиций в начальной строке.

2. **SuffTree.cpp** – сама реализация объявленных функций.

3. **main.cpp** – сама программа, последовательно вызывающая методы объектов для получения результата.

Дневник отладки

Номер	Ошибка	Обнаруженная причина	Исправление
1	Не полная привязка суффиксных ссылок	Отсутствовала привязка в случае, если новый символ уже был в дереве	Добавить привязку
2	Ошибка компиляции	Передача списка инициализации туда, где ожидалось std::pair	Добавлять новые элементы в std::map с помощью std::make_pair

Тест производительности

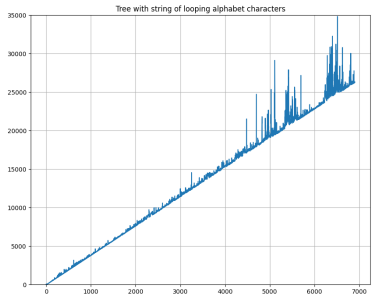

Рис. 1: Дерево, построенное на строке из повторяющихся символов в порядке алфавита

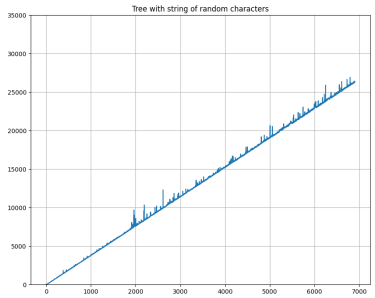

Рис. 2: Дерево, построенное на строке из случайных символов

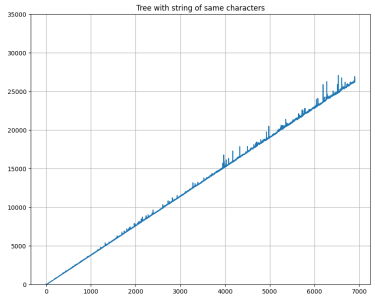

Рис. 3: Дерево, построенное на строке из одного повторяющегося символа

На рисунках 1-3 показаны время построения дерево из строк разных форматов. Как можно заметить, построение дерева имеет линейную сложность, как и заявлял алгоритм Укконена, независимо от формата строки.

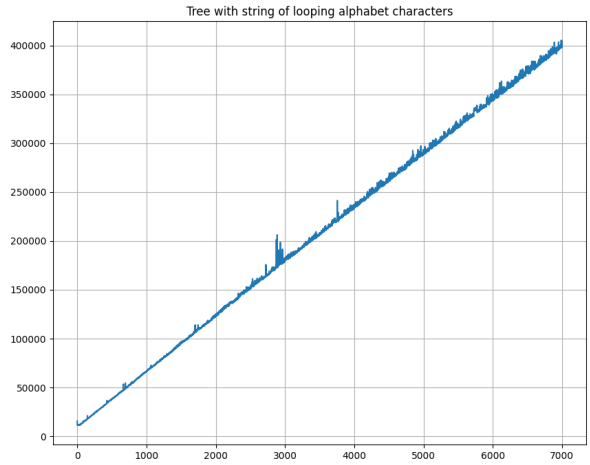

Рис. 4: Поиск с помощью массива суффиксов

Рисунок 4 показывает время поиска для строки длины 5000. Абсцисса показывает количество образцов длиной в 2 символа.

Выводы

Суффиксные деревья являются сжатыми *trie*, в который кроме самой строки добавлены все возможные его суффиксы. Они являются очень гибкими в применении и могут появляться в самых неожиданных местах, но к известным относятся:

- Поиск множества вхождений на единой строке за $O(n)$;
- Нахождений наибольшей повторяющейся подстроки;
- Наибольшая общая подстрока для двух (и более) строк.

Наивно, суффиксное дерево строится за $O(n^2)$ просто добавлением в trie каждого суффикса, но этот процесс может быть улучшен учитывая тот факт, что добавляются именно суффиксы, а не различные строки, что и делает алгоритм Укконена.

Но само дерево может занимать очень много места: хоть его пространственная сложность и есть $O(n)$, скрытая константа может быть большой за счет внутреннего хранения списков указателей. Во избежание этого был придуман суффиксный массив. Он является сжатой версией суффиксного дерева, которую можно построить либо за $O(n)$ из суффиксного дерева или за $O(nlog(n))$ алгоритмом Манбера Муерса, неиспользованного здесь. Сжатие происходит за счет того, что вместо узлов дерева в массиве

хранятся всего лишь указатели на начала суффиксов в строке так, что, пройдясь по массиву, можно построить суффиксы в лексически упорядоченном виде.

Вместе с суффиксным массивом строится и массив Longest Common Prefix – массив, в котором на i-ой позиции стоит число, указывающее наибольший общий префикс между i-ым и $i+1$-ым суффиксами (можно и наоборот, зависит от реализации). У этого массива есть свойство, что общий наибольший общий префикс между i-ой и j-ой суффиксами есть наименьшее значение массива в индексах $k \in [i, j-1]$.

Пара "суффиксный массив"и "Longest Common Prefix"способна выполнить возможно все задачи суффиксного дерева за счет усложнения алгоритмов.