**Экзаменационная программа по курсу «Дискретный анализ»  
за осенний семестр 2018/19 учебного года**

**I. Сортировки за линейное время**

1. Оценка времени работы сортировок, использующих сравнение элементов.
2. Сортировка подсчетом.   
   Кормен
3. Поразрядная сортировка.   
   Кормен
4. Карманная сортировка.   
   Кормен https://neerc.ifmo.ru/wiki/index.php?title=Карманная\_сортировка
5. Профилирование кода. Бенчмарки. Бенчмарк для сортировок за   
   линейное время.

**II. Сбалансированные и сильноветвящиеся деревья поиска**

1. Бинарные деревья поиска. Основные операции (поиск, вставка, удаление, поворот, поиск следующего/предыдущего).   
   Кормен
2. AVL-деревья. Основные операции.   
   [https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево](https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)

https://habr.com/post/150732/

1. Красно-черные деревья. Основные операции.   
   Кормен + моя реализация
2. Декартовы деревья. https://habr.com/post/101818/ http://e-maxx.ru/algo/treap#7 Основные операции. <https://gist.github.com/sfalexrog/32fc1af1553c53e537af7e077a43d7cd> Неявные декартовы   
   деревья. https://gist.github.com/sfalexrog/5915c176ffb0dff13a1b7152da6918c1
3. 2-3-4 деревья. Основные операции. <https://habr.com/post/273687/> http://ru.knowledgr.com/17372708/234%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%B0 Чтобы вставить значение, мы начинаем с корня дерева 2–3–4:

Если текущий узел является 4-узлом:

Удалите и сохраните среднее значение, чтобы получить 3-узел.

Разделите оставшийся 3-узел на пару из 2-х узлов (теперь пропущенное среднее значение обрабатывается на следующем шаге).

Если это корневой узел (который, таким образом, не имеет родителя):

среднее значение становится новым корневым 2-узлом, а высота дерева увеличивается на 1. Поднимитесь в корень.

В противном случае вставьте среднее значение в родительский узел. Поднимитесь в родительский узел.

Найдите ребенка, чей интервал содержит значение для вставки.

Если этот дочерний элемент является листом, вставьте значение в дочерний узел и завершите.

В противном случае погрузитесь в ребенка и повторите с шага 1 Связь с красно-черными   
деревьями.   
КЧД совмещает в себе простоту алгоритма обхода и поиска, как у произвольных двоичных деревьев, и простоту поддержки сбалансированности, присущей Б-деревьям. И всего-то нужно, двоичный признак (цвет).Ввиду сложности структуры узлов 2-3-4-дерева, алгоритмы операций для него становятся громоздкими и непроизводительными: накладные расходы, связанные с манипулированием сложными структурами узлов, рассмотрением различных сочетаний 2-, 3- и 4-узлов делают алгоритмы медленнее BST–деревьев. Поэтому непосредствен-ная реализация 2-3-4-дерева редко используется. Взамен предла-гается **красно-черное дерево (RB-дерево)**, воспроизводящее структуру и алгоритмы 2-3-4-дерева. В RB-дереве 2-,3-,4- узлы заменяются группами 2-узлов.

Для различения отдельных узлов 2-3-4-дерева в группах принята раскраска узлов. Корневой узел группы имеет черный цвет (*B*), внутренние узла окрашиваются в красный цвет (*R*). Таким образом формируется двоичный эквивалент 2-3-4-дерева – RB-дерево.

1. B-деревья. Основные операции.   
   каждая вершина содержит минимум t - 1 и максимум 2t - 1 ключей. Внутренние вершины имеют не менее t и не более 2t детей.
2. Цифровые деревья поиска. Основные операции.   
   Деревья цифрового (поразрядного) поиска

Методы цифрового поиска достаточно громоздки и плохо иллюстрируются. Рассмотрим бинарное дерево цифрового поиска. Как и в деревьях, рассмотренных выше, в каждой вершине такого дерева хранится полный ключ, но переход по левой или правой ветви происходит не путем сравнения ключа-эталона со значением ключа, хранящегося в вершине, а на основе значения очередного бита аргумента. Реализация цифрового поиска происходит поразрядно (побитово).

Поиск начинается от корня дерева. Если содержащийся в корневой вершине ключ не совпадает с аргументом поиска, то анализируется самый левый бит аргумента. Если он равен 0, происходит переход по левой ветви, если 1 – по правой. Если не обнаруживается совпадение ключа с аргументом поиска, то анализируется следующий бит аргумента и т.д. Поиск завершается, когда будут проверены все биты аргумента либо встретится вершина с отсутствующей левой или правой ссылкой.

1. Trie. Основные операции.

Книга индусов.

https://habr.com/post/111874/

1. PATRICIA trie. Основные операции.   
   Структура данных Patricia (Практический Алгоритм Извлечения Информации, Кодированной Буквенно-Цифровой) - это сжатый двоичный файл, в котором узлы ветви и элемента были объединены в один тип узла. Рассмотрим сжатый двоичный файл на рисунке 28.18. Круговые узлы являются узлами ветвления, а прямоугольные узлы являются узлами элементов. Число внутри узла ветви является его полем номера бита; левый потомок узла ветви соответствует случаю, когда соответствующий бит ключа равен 0, а правый потомок случаю, когда этот бит равен 1. Соединение узлов ветви и элемента выполняется путем перемещения каждого элемента из его узла элемента в родительский ветвь. Поскольку количество узлов ветвления на единицу меньше количества узлов элементов, мы вводим узел заголовка и делаем сжатый двоичный файл вторым левым поддеревом заголовка. Указатели, которые первоначально шли от узла ветвления к узлу элемента, теперь переходят от этого узла ветвления к узлу ветвления, в который был добавлен соответствующий элемент. На рисунке 28.19 показан возможный результат слияния узлов на рисунке 28.18. Число вне узла - это значения битового числа. Толстые указатели - это обратные указатели, которые заменяют указатели ветвления на узел элемента на рисунке 28.18. Обратный указатель обладает тем свойством, что значение номера бита в начале указателя равно ≥ значения номера бита в его конце. Для исходных указателей от узла ветвления к узлу ветвления (также называемых нисходящими указателями) значение номера бита в конце указателя всегда больше значения номера бита в начале указателя.

28.10.1 Поиск

Чтобы найти элемент с ключом theKey, мы используем биты ключа слева направо для перемещения вниз по экземпляру Patricia так же, как мы ищем сжатый двоичный файл. Когда следует обратный указатель, мы сравниваем ключ с ключом в достигнутом узле Патриции. Например, чтобы найти ключ = 1101, мы начнем с перемещения в левое поддерево узла заголовка. Указатель, за которым следует указатель, является нисходящим указателем (номер начального бита равен 0, а номер конечного бита равен 1). Мы разветвляемся, используя бит 1 ключа. Поскольку этот бит равен 1, мы следуем за указателем правого дочернего элемента. Номер начального бита для этого указателя равен 1, а номер конечного бита равен 2. Итак, опять же, следовал нисходящий указатель. Число битов для достигнутого узла равно 2. Таким образом, мы используем бит 2 клавиши, чтобы двигаться дальше. Бит 2 равен 1. Итак, мы используем правый дочерний указатель, чтобы добраться до узла, который содержит 1100. Опять же, был использован нисходящий указатель. Из этого узла перемещение выполняется с использованием бита 4 клавиши. Это возвращает нас к узлу, который содержит 1101. На этот раз следовал обратный указатель (начальный бит указателя равен 4, а конечный бит равен 1). Когда следует обратный указатель, мы сравниваем ключ с ключом в достигнутом узле. В этом случае ключи совпадают, и мы нашли нужный элемент. Обратите внимание, что тот же путь поиска выбирается, когда theKey = 1111. В этом случае окончательное сравнение завершается неудачно, и мы заключаем, что у нас нет элемента, ключ которого равен 1111.

28.10.2 Вставка элемента

Мы используем пример для иллюстрации алгоритма вставки. Начнем с пустого экземпляра. Такой экземпляр не имеет узла; даже не заголовок. Для нашего примера мы рассмотрим ключи, которые имеют 7 бит. Для первой вставки мы используем ключ 0000101. При вставке в пустой экземпляр мы создаем узел заголовка, левый указатель которого указывает на узел заголовка; новый элемент вставляется в заголовок; и поле номера бита заголовка установлено в 0.

Конфигурация рисунка 28.20 (а) приводит к результатам. Обратите внимание, что правое дочернее поле узла заголовка не используется.

Ключ для следующей вставки - 0000000. Поиск этого ключа заканчивается в заголовке. Мы сравниваем ключ вставки и ключ заголовка и определяем, что первый бит, в котором они отличаются, это бит 5. Итак, мы создаем новый узел с полем номера бита 5 и вставляем новый элемент в этот узел. Поскольку бит 5 ключа вставки равен 0, левый дочерний указатель нового узла указывает на новый узел, а правый дочерний указатель - на узел заголовка (рисунок 28.20 (b)).

Для следующей вставки предположим, что ключ - 0000010. Поиск этого ключа заканчивается на узле с 0000000. Сравнивая два ключа, мы определяем, что первый бит, в котором два ключа отличаются, это бит 6. Итак, мы создаем новый узел с полем номера бита 6 и поместите новый элемент в этот новый узел. Новый узел вставляется в путь поиска таким образом, что поля битового числа увеличиваются вдоль этого пути. В нашем примере новый узел вставляется как левый потомок узла с 0000000. Поскольку бит 6 ключа вставки равен 1, указатель правого потомка нового узла является самостоятельным указателем, а указатель левого потомка указывает на узел с 0000000. Рисунок 28.20 (с) показывает результат.

Общая стратегия вставки элемента, отличного от первого, приведена ниже. Ключ вставляемого элемента - theKey.

1. Поиск ключа. Пусть достигаемый будет ключом в узле endNode, где поиск заканчивается.

2. Определите крайнюю левую позицию бита lBitPos, в которой отличаются ключ и достигнут ключ. lBitPos четко определен, если один из двух ключей не является префиксом другого.

3. Создайте новый узел с полем номера бита lBitPos. Вставьте этот узел в путь поиска от заголовка до endNode, чтобы битовые числа увеличивались вдоль этого пути. Эта вставка разбивает указатель от узла p к узлу q. Указатель теперь идет от p до нового узла.

4. Если бит lBitPos theKey равен 1, правый дочерний указатель нового узла становится самостоятельным указателем (то есть он указывает на новый узел); в противном случае левый дочерний указатель нового узла становится самостоятельным указателем. Оставшийся дочерний указатель нового узла указывает на q.

Для нашей следующей вставки ключ вставки - 0001000. Поиск этого ключа завершается на узле с 0000000. Мы определяем, что первый бит, на котором отличаются ключ вставки и достигли ключа 0000000, - это бит 4. Мы создаем новый узел с битом. номер 4 и поместите новый элемент в этот узел. Новый узел вставляется в путь поиска, чтобы обеспечить увеличение полей битовых номеров вдоль этого пути. Таким образом, новый узел вставляется как левый потомок узла заголовка (рисунок 28.18 (d)). Это разрывает указатель от заголовка на узел q с 0000000. Поскольку бит 4 ключа вставки равен 1, правый дочерний указатель нового узла является самостоятельным указателем, а левый дочерний указатель переходит на узел q.

Мы рассмотрим еще две вставки. Попробуйте вставить элемент, ключ которого - 0000100. Достигнут ключ: 0000101 (в заголовке). Мы видим, что первый бит, в котором отличаются клавиши вставки и достигнутого ключа, это бит 7. Итак, мы создаем новый узел с битом 7; новый элемент помещается в новый узел; новый узел вставляется в путь поиска, чтобы обеспечить увеличение битовых чисел вдоль этого пути (для этого необходимо, чтобы новый узел стал правым дочерним узлом с 0000000, разбив дочерний указатель с 0000000 на заголовок); для сломанного указателя p - узел с 0000000, а q - заголовок; левый дочерний указатель нового узла является самостоятельным указателем (потому что бит 7 ключа вставки равен 0); оставшийся дочерний указатель (в данном случае правый дочерний) нового узла указывает на q (см. рис. 28.21 (а)).

Для нашей последней вставки ключ вставки - 0001010. Поиск этого ключа заканчивается на узле с 0000100. Первый бит, на котором отличаются ключи вставки и достигнутые, это бит 6. Итак, мы создаем новый узел с битом 6; новый элемент помещается в новый узел; новый узел вставляется в путь поиска, чтобы обеспечить увеличение битовых чисел вдоль этого пути (для этого необходимо, чтобы новый узел стал правым потомком узла с 0001000; поэтому p = q = узел с 001000); правый дочерний указатель нового узла является самостоятельным указателем (потому что бит 6 ключа вставки равен 1); и оставшийся дочерний элемент (в данном случае левый дочерний элемент) нового узла равен q (см. рис. 28.21 (b)).

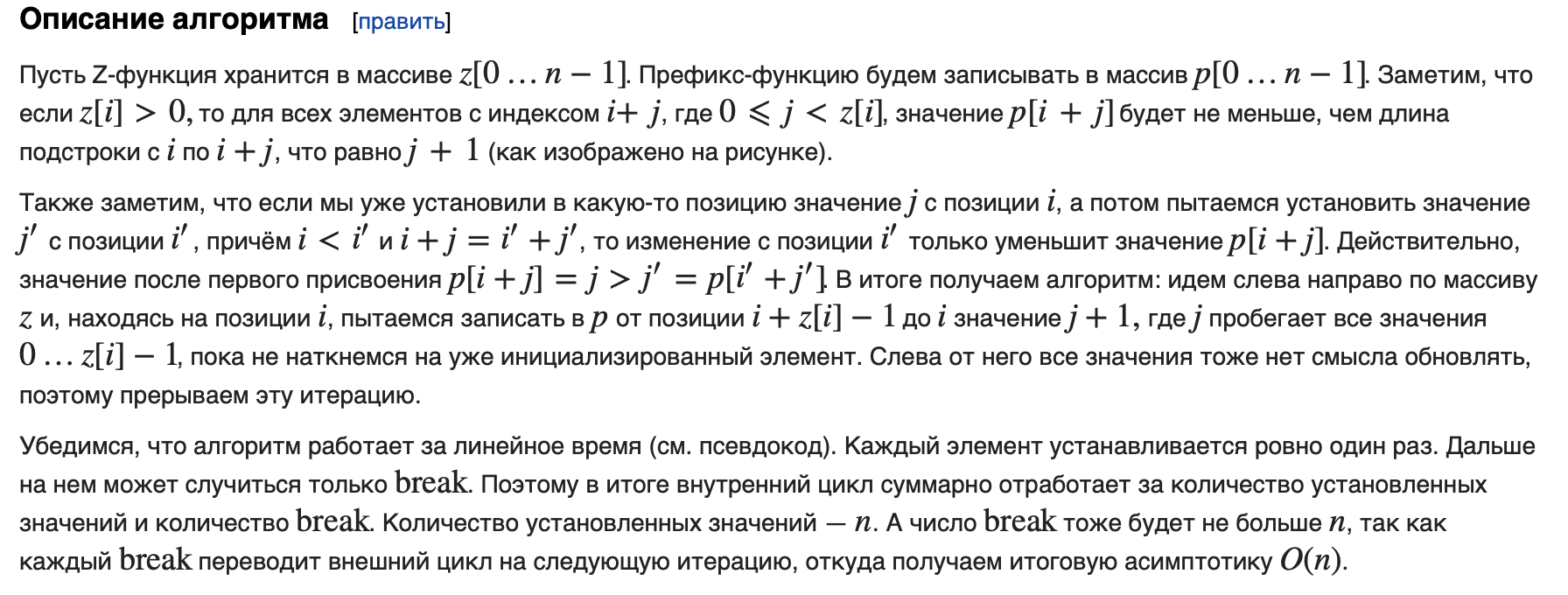
28.10.3 Удаление элемента

Пусть p будет узлом, который содержит элемент, который должен быть удален. Мы рассмотрим два случая, когда p- (a) p имеет собственный указатель и (b) p не имеет собственного указателя. Когда p имеет собственный указатель, а p - заголовок, экземпляр Patricia становится пустым после удаления элемента (рисунок 28.20 (а)). В этом случае мы просто избавляемся от заголовка. Когда p имеет собственный указатель, а p не является заголовком, мы устанавливаем указатель от родителя p на значение не-указателя p. После этого изменения указателя узел p располагается. Например, чтобы удалить элемент с ключом 0000010 из рисунка 28.21 (a), мы меняем левый дочерний указатель в узле с 0000000, чтобы он указывал на узел, на который указывает не собственный указатель p (то есть узел с 0000000). Это заставляет левый дочерний указатель 0000000 стать самостоятельным указателем. Узел с 0000010 затем удаляется.

Во втором случае мы сначала находим узел q, который имеет обратный указатель на узел p. Например, когда мы должны удалить 0001000 из рисунка 28.21 (b), узел q является узлом с 0001010. Элемент qElement в q (в нашем примере 0001010) перемещается в узел p, и мы вместо этого приступаем к удалению узла q узла р. Обратите внимание, что узел q - это узел, с которого мы достигли узла p при поиске элемента, который должен быть удален. Чтобы удалить узел q, мы сначала находим узел r, который имеет обратный указатель на q (для нашего примера r = q). Узел r находится с помощью ключа qElement. Как только r найдено, обратный указатель на q, находящийся в r, изменяется с q на p, чтобы должным образом учесть тот факт, что мы переместили qElement в p. Наконец, нисходящий указатель от родителя q на q заменяется дочерним указателем q, который использовался для определения местоположения r. В нашем примере p является родителем q, а правый дочерний указатель p изменяется с q на правый дочерний элемент q, который сам по себе был изменен на p.

Мы видим, что время для каждой из операций поиска, вставки и удаления в Patricia равно O (h), где h - высота экземпляра Patricia. Для получения дополнительной информации о Патрисии и общих попытках см. [2, 3].

**III. Поиск образца в тексте**



1. Z-блоки, построение за линейное время. Поиск подстроки в строке за линейное время с использованием Z-блоков. <https://habr.com/post/113266/>

*index: 0 1 2 3 4 5 6 7*

*pattern: a b a b c a b a*

*sp: 0 0 1 2 0 1 2 3*

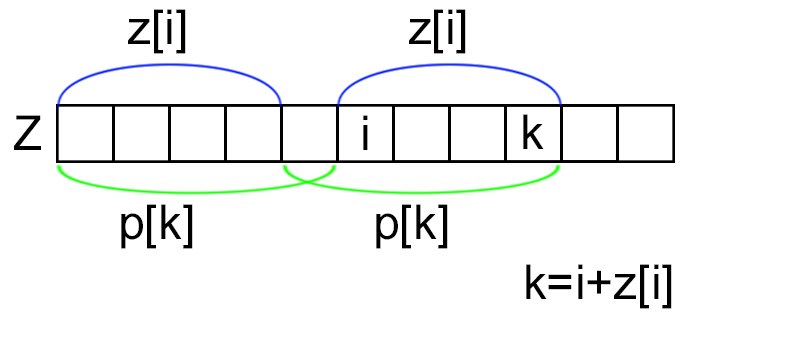
*sp’: 0 0 1 2 0 1 2 3*

Построение префикс-функции по Z-функции

**Постановка задачи**

Дан массив с корректной [Z-функцией](https://neerc.ifmo.ru/wiki/index.php?title=Z-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) для строки s, получить за O(n)

массив с префикс-функцией для строки s



**Псевдокод**[[править](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F&action=edit&section=11)]

**int**[] buildPrefixFunctionFromZFunction(**int**[] z):

**int**[] p = **int**[z.length]

fill(p, 0)

**for** i = 1 **to** z.length - 1

**for** j = z[i] - 1 **downto** 0

**if** p[i + j] > 0

**break**

**else**

p[i + j] = j + 1

**return** p

1. Алгоритм Кнута-Морриса-Пратта. Построение префикс-функции на основе Z-блоков. [https://neerc.ifmo.ru/wiki/index.php?title=Префикс-функция](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) Псевдокод

**int**[] buildPrefixFunctionFromZFunction(**int**[] z):

**int**[] p = **int**[z.length]

fill(p, 0)

**for** i = 1 **to** z.length - 1

**for** j = z[i] - 1 **downto** 0

**if** p[i + j] > 0

**break**

**else**

p[i + j] = j + 1

**return** p

Алгоритмреальноговремени**.**

1. Классический алгоритм Кнута-Морриса-Пратта.
2. Алгоритм Бойера-Мура. Правило плохого символа, правило   
   хорошего суффикса.
3. Алгоритм Апостолико-Джанкарло.
4. Алгоритм Ахо-Корасик. Приложения алгоритма: задача о поиске с   
   джокером, задача о двумерном точном совпадении.

**IV. Суффиксные деревья**

1. Суффиксные деревья. Неявные суффиксные деревья. Простейший алгоритм построения суффиксных деревьев.
2. Алгоритм Укконена. Суффиксные связи и прыжки по счетчику. Доказательство линейности алгоритма.   
   <https://habr.com/post/111675/>

Наблюдение 1

Когда будет найдено, что последний суффикс, который нам нужно вставить, уже существует в дереве, само дерево вообще не изменяется (мы только обновляем активную точку и остаток).

Наблюдение 2

Если в какой-то момент active\_length больше или равен длине текущего ребра (edge\_length), мы перемещаем нашу активную точку вниз до тех пор, пока edge\_length не станет строго больше, чем active\_length.

Теперь давайте переопределим правила:

Правило 1

Если после вставки из активного узла = root активная длина больше 0, то:

* активный узел не изменен
* активная длина уменьшается
* активное ребро смещено вправо (к первому символу следующего суффикса мы должны вставить)

Правило 2

Если мы создадим новый внутренний узел ИЛИ создадим вставку из внутреннего узла, и это не первый внутренний узел на текущем шаге, то мы свяжем предыдущий узел с ЭТИМ через суффиксную ссылку.

Это определение правила 2 отличается от jogojapan ', поскольку здесь мы учитываем не только вновь созданные внутренние узлы, но и внутренние узлы, из которых мы делаем вставку.

Правило 3

После вставки из активного узла, который не является корневым узлом, мы должны перейти по ссылке суффикса и установить активный узел на узел, на который он указывает. Если нет ссылки на суффикс, установите активный узел в качестве корневого узла. В любом случае, активный край и активная длина остаются неизменными.

В этом определении правила 3 ​​мы также рассматриваем вставки листовых узлов (не только расщепленных узлов).

И наконец, Наблюдение 3:

Когда символ, который мы хотим добавить к дереву, уже находится на краю, мы, согласно Наблюдению 1, обновляем только активную точку и остаток, оставляя дерево без изменений. НО, если есть внутренний узел, помеченный как нуждающийся в суффиксной ссылке, мы должны соединить этот узел с нашим текущим активным узлом через суффиксную ссылку.

1. Обобщенные суффиксные деревья.
2. Приложения суффиксных деревьев: поиск подстроки в строке,   
   множественный поиск, подстрока для базы образцов, линеаризация циклической строки, наибольшая общая подстрока (для двух и более строк за O(Kn)), статистика совпадений.
3. Суффиксные массивы. Построение через суффиксное дерево.
4. Бинарный поиск образца в суффиксном массиве. Ускорение поиска   
   до O(n + log m).
5. Построение суффиксного массива с помощью цифровой сортировки   
   циклических строк.