Условие

Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания.
Алфавит строк: строчные буквы латинского алфавита (т.е., от a до z).

Вариант 4: Линеаризация циклической строки.

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

Входные данные
Некий разрез циклической строки.

Выходные данные
Минимальный в лексикографическом смысле разрез.

Описание алгоритма:

В алгоритме используется такое понятие как суффиксное дерево. Суффиксное дерево строки S - это бор, содержащий все суффиксы данной строки. Однако оно немного отличается от бора тем, что вершины имеющие только одного ребенка соединены вместе. Такие вершины называются неявными (но корень это всегда явная вершина). Таким образом ребра суффиксного дерева помечены не одним символом, а строками.

Для описания алгоритма сначала будет описан алгоритм работающий за время $O(n^3)$ а затем ускорим его до $O(n)$.

Алгоритм за $O(n^3)$

Алгоритм очень прост. Необходимо последовательно добавить в дерево все суффиксы префиксов строки S. Добавление происходит следующим образом: сначала проходим в дереве $k-1$ символ, где k - длина добавляемого суффикса. Затем нужно рассмотреть несколько случаев (для определенности будем считать что мы добавляем символ 's'):

1. Находимся в явной вершине и нет ребра начинающегося с 's': тогда нужно создать лист и пустить в него ребро помеченное символом 's'.

2. Находимся в неявной вершине и следующий символ не 's': нужно разбить это ребро на 2 части новой вершиной и выполнить действия в пункте 1.

3. Находимся в листе: продлить лист символом 's'.

4. Из текущей вершины ведет ребро начинающееся с символа 's': ничего делать не надо.

Добавление одного суффикса происходит за $O(n)$ так как это просто прохождение по ребрам дерева и возможно создание новых вершин, а количество таких операций $O(n^2)$. Из этого следует указанная выше оценка времени работы.

Алгоритм за $O(n^2)$

Для ускорения до $O(n^2)$ вводится такое понятие как суффиксная ссылка. Пусть $x\alpha$ обозначает произвольную строку, где x — ее первый символ, а α — оставшаяся подстрока (возможно пустая). Если для внутренней вершины v с путевой меткой $x\alpha$ существует другая вершина $s(v)$ с путевой меткой α, то ссылка из v в $s(v)$ называется суффиксной ссылкой. Проще говоря, переход по суффиксной ссылке это "отрезание первого символа текущей подстроки". Суффиксные ссылки определяются только для внутренних вершин так как суффиксная ссылка ведущая из листа может указывать на неявную вершину, а для внутренних вершин это невозможно. Очевидно что для каждой внутренней вершины всегда существует суффиксная ссылка (по построению). Однако для корня суффиксной ссылки нет. Для того чтобы сократить количество проверяемых условий и сделать алгоритм единым для всех вершин введем фиктивную вершину $dummy$ в которую будет вести суффиксная ссылка из корня. Из $dummy$ в корень будет вести столько ребер сколько символов в алфавите и каждое ребро будет помечено одним символом.

Наконец опишем ускоренный алгоритм: вместо того чтобы на очередной итерации проходить по дереву для каждого суффикса текущего префикса, мы один раз проходим по предыдущему префиксу и добавляем следующий символ, затем поднимаемся к ближайшей внутренней вершине, переходим по суффиксной ссылке и снова спускаемся вниз, проходя тот же путь который прошли поднимаясь наверх после чего добавляем следующий символ и повторяем описанный процесс до тех пока не доберемся до корня. При создании новой вершины выпускаем в нее суфиксную ссылку из предыдущей созданной вершины на этом шаге. Таким образом сокращается количество прохождений по дереву и итерация превращается в добавление очередного символа к дереву. Первое прохождение займет времени $O(k)$, затем будет совершено $O(k)$ переходов по суффиксным ссылкам (где k - длина текущего префикса). Амортизационно получим что добавление одного суффикса занимает $O(1)$. Итого сложность построения дерева $O(n^2)$.

Алгоритм за $O(n)$

Из-за того что продолжать ускорение с текущим потреблением памяти невозможно, необходимо модифицировать структуру дерева: теперь ребра будут помечены не строками а парой чисел (левая и правая граница). Одно из ускорений состоит в том что добавление символа завершается после того как при добавлении сработало правило 4. Рассмотрим общую схему добавления символа в дерево: Сначала используется правило 3. Затем встречается первая вершина не являющаяся листом. Обозначим ее как $ActivePoint(AP)$ - первая вершина на пути не являющаяся листом. Далее работают правила 1 и 2 так как понятно что количество детей не может уменьшаться следовательно листьев мы больше не встретим. И, наконец, в какой-то момент срабатывает правило 4. Обозначим вершину

в которой это произошло как $EndPoint(EP)$. Далее работа завершается. Теперь немного об ускорениях:

1. В начале мы занимаемся довольно бесполезной работой а именно продляем листья на один символ. Но это эквивалентно тому что мы просто прибавим единицу к правой границе на ребре. Так же ясно что для всех листьев эта граница будет одинаковой. Для сокращения бесполезной работы введем счетчик cur который будет являться правой границей для всех листьев. Таким образом мы не смотрим на листья совсем и начинаем работу с AP. Однако возникает вопрос а как же найти AP для очередного шага? Для этого опишем следующий пункт.

2. EP предыдущего шага - это родитель AP текущего шага (для нахождения AP нужно пройти по только что добавленному символу). Это так потому что предыдущие вершины в которые ведут ребра с только что добавленным сиволом являются листьями. А ребенок EP не лист потому что даже если он был листом то предыдущем шаге к нему был приписан еще один символ.

Итак сам алгоритм:

1. Создаем корень и фиктивную вершину $dummy$. Выпускаем суффиксную ссылку из корня в $dummy$ и соединяем $dummy$ с корнем ребрами которые помечены всеми символами алфавита. Устанавливаем AP в корне.

2. Увеличиваем счетчик cur.

3. При добавлении символа начиная с AP пользуемся правилами 1-3 для вставки и из предыдущей созданной вершины на этом шаге выпускаем суффиксную ссылку в только что созданную(если такой нет то для уменьшения количества проверок выпускаем ссылку из $dummy$). Затем проходим по суффиксной ссылке выполняя действия описанные выше и перемещаем указатель AP на вершину в которую пришли только что. Повторяем до тех пор пока не сработало правило 4.

4. Выпускаем из предыдущей вершины суффиксную ссылку в текущую, и проходим по текущему добавляемому символу. Перемещаем указатель AP туда.

5. Продолжаем 2-4 для каждого следующего символа

Нетрудно показать что количество внутренних вершин меньше чем количество листьев а листьев в явном суффиксном дереве n. Так как алгоритм только создает новые вершины следовательно и время работы будет $O(n)$.

Расход памяти:

Для работы алгоритма необходимо для каждой вершины хранить таблицу переходов по символам. Можно реализовать ее с помощью массива и тогда занимаемая память $O(|\Sigma|n)$, где $|\Sigma|$ -мощность алфавита, но поиск переходов будет немного быстрее. Либо можно использовать деревья для экономии памяти, но поиск будет медленне. В данной реализации был использован массив из 26 элементов.

Решение исходной задачи:

Для нахождения минимального разреза циклической строки необходимо к заданному разрезу присать его же и построить для полученной строки суффиксное дерево. Далее начиная с корня проходим по дереву выбирая ребра начинающиеся на наименьший символ и выводим символы по которым прошли. Продолжаем до тех пор пока не выведено ровно n символов. И задача решена.

Дневник отладки

Возникли небольшие трудности с созданием суффиксных ссылок. В частности при достижении EP суффиксная ссылка не выпускалась из предыдущей вершины. Поэтому программа получила $WA\#4$. После исправления этой ошибки решение было принято чекером.

Выводы

Суффиксное дерево это довольно интересная и полезная структура. Она может быть полезна например для решения задачи поиска подстроки в строке в случае когда задан текст а искомые подстроки неизвестны или их суммарная длинна гораздо больше чем размер текста. Для построения суффиксного дерева использован алгоритм Укконена так как он проще чем другие (например алгоритм Мак-Крейта). Однако у него есть несколько недостатков: во-первых дерево потребляет слишком много памяти и во-вторых суффиксное дерево должно быть полностью загружено в ОП что затрудняет его использование для больших объемов данных.