Сортировка суффиксов

- Можно к набору суффиксов применить алгоритм поразрядной сортировки
- Если ее выполнить непосредственно, то при длине текста n и соответствующем количестве суффиксов потребуется времени $O(n^2)$
- Такая оценка неудовлетворительна, поэтому алгоритм усложняется – он учитывает структуру суффиксов

Суффиксы строки

- i-й суффикс строки S при $i > 0$ встречается не только как самостоятельный элемент, но и как подстрока других элементов массива суффиксов
- Для любого $h < i$ он является подстрокой суффикса i-h в позиции h
- Пример: суффиксы строки $S = aaba$:

$i = 0$	*aaba*
$i = 1$	*aba*
$i = 2$	*ba*
$i = 3$	*a*

- Каждый суффикс (кроме нулевого) выглядит как сдвиг предыдущего на 1 позицию влево
- На этом наблюдении основан рассматриваемый подход к сортировке суффиксов

Подход к сортировке суффиксов

- Сортируются не суффиксы, а набор циклических сдвигов исходной строки
- Подобно суффиксу, каждый циклический сдвиг определяется своим индексом i – величиной этого сдвига
- Всевозможные циклические сдвиги строки $S = aaba$:

$i = 0$	*aaba*
$i = 1$	*abaa*
$i = 2$	*baaa*
$i = 3$	*aaab*

- Этим же способом нетрудно получить и отсортированные суффиксы
- Перед сортировкой дописать в конец S наименьший терминальный символ (0 для языка *C*), а после сортировки из каждого сдвига исключить все символы начиная с терминального

Сортировка сдвигов – условия

- Дана строка *S* длины *n*. Требуется получить отсортированный массив ее всевозможных левых циклических сдвигов
- Поскольку сортируются циклические сдвиги, то и подстроки рассматриваются как циклические
- Под подстрокой $S[i..j]$ при $i > j$ понимается подстрока $S[i..n-1] + S[0..j]$
- Все индексы в строке берутся по модулю *n*. В целях упрощения записи операция вычисления остатка подразумевается неявно

Сортировка сдвигов – схема

- Алгоритм состоит из стадий количеством порядка lg$n$
- На k-ой стадии (k = 0, …, $\lceil \log n \rceil$) сортируются подстроки длины 2^k, то есть длины последовательно удваиваются
- При сортировке подразумеваются перестановки целых строк, однако сопоставляются эти строки лишь по первым 2^k символам
- На последней стадии сортируются элементы исходного массива целиком

Используемые массивы

- На каждой стадии алгоритм поддерживает перестановку P[0..n-1] индексов исходных циклических подстрок в соответствии с текущей сортировкой
- Вычисляется также массив C[0..n-1] номеров классов эквивалентности
- Для каждой подстроки в S с позиции i и длиной 2^k вычисляется номер C[i] класса эквивалентности, которому эта подстрока принадлежит
- Все равные подстроки относятся к общему классу
- C[i] сохраняют порядок: меньшей подстроке приписывается меньший номер класса

Пример массивов

- Для строки S = *aaba* массивы P и C на каждой стадии k выглядят следующим образом:

k	P	C	Сортируются из S	Результат из P
0	(0, 1, 3, 2)	(0, 0, 1, 0)	*a*, *a*, *b*, *a*	*a*, *a*, *a*, *b*
1	(0, 3, 1, 2)	(0, 1, 2, 0)	*aa*, *ab*, *ba*, *aa*	*aa*, *aa*, *ab*, *ba*
2	(3, 0, 1, 2)	(1, 2, 3, 0)	*aaba*, *abaa*, *baaa*, *aaab*	*aaab*, *aaba*, *abaa*, *baaa*

$i = 0$ *aaba*

$i = 1$ *abaa*

$i = 2$ *baaa*

$i = 3$ *aaab*

Алгоритм – нулевая стадия

- На стадии $k = 0$ сортируются подстроки длины 1 – отдельные символы
- Эту задачу можно выполнить сортировкой подсчётом ($O(n)$)
- Так формируется начальный массив P
- Далее нужно разделить полученный набор символов на классы эквивалентности (равные символы должны относиться к общему классу)
- Просматриваем массив P и сравниваем пары соседних символов: начинаем с класса #0 и для каждого неравного символа класс увеличиваем на 1
- В результате получим массив C

Алгоритм – стадия *k*

- Покажем, как за время $O(n)$ выполнить очередную k-ю стадию
- Заметим, что подстрока длины 2^k сцепляет две подстроки длины 2^{k-1}
- Подстроки половинной длины можно сравнивать за константное время – используется массив C предыдущей стадии (номера классов)
- Для целой подстроки длины 2^k, начинающейся в позиции i исходной строки S, необходимая информация содержится в паре чисел ($C[i]$, $C[i+2^{k-1}]$)

$$\dots \overbrace{\underbrace{s_i \dots s_{i+2^{k-1}-1}}_{\text{length}=2^{k-1},\text{class}=c[i]} \quad \underbrace{s_{i+2^{k-1}} \dots s_{i+2^k-1}}_{\text{length}=2^{k-1},\ \text{class}=c[i+2^{k-1}]}}^{\text{length}=2^k} \dots \overbrace{\underbrace{s_j \dots s_{j+2^{k-1}-1}}_{\text{length}=2^{k-1},\text{class}=c[j]} \quad \underbrace{s_{j+2^{k-1}} \dots s_{j+2^k-1}}_{\text{length}=2^{k-1},\text{class}=c[j+2^{k-1}]}}^{\text{length}=2^k} \dots$$

- Простое решение: отсортировать подстроки длины 2^k можно по этим парам чисел, что даст требуемый порядок, то есть новый массив P
- Поскольку элементы пар не превосходят n, то можно выполнить поразрядную их сортировку – для 2-х «разрядов»

Сортировка пар чисел - детали

- Сортируем пары сначала по вторым элементам, а затем – по первым, устойчивой сортировкой
- Учитываем, что взятые отдельно вторые элементы пар уже упорядочены как подстроки длины 2^{k-1}, этот порядок задан в массиве *P* предыдущей стадии
- Благодаря цикличности: любая подстрока длины *l* из любой i-ой строки присутствует в строке *i* + *l*, причем эта копия сдвинута влево на *l* позиций
- Остается перейти в парах от индексов правых частей к индексам левых частей, поскольку ими определяются индексы целых пар
- Для этого достаточно из каждого элемента *P*[*i*] вычесть $l = 2^{k-1}$ и сохранить его в *i*-ой позиции
- Таким образом, с помощью лишь вычитаний в количестве *O*(*n*) производится сортировка пар по вторым элементам
- Далее произвести устойчивую сортировку по первым элементам пар (сортировку массива С), которую можно выполнить подсчётом за время *O*(*n*)

Стадия *k* – завершение и оценка

- Для завершения стадии остается пересчитать номера *C* классов эквивалентности.
- Вычисляются просмотром новой перестановки *P* и сравнением соседних элементов
- Вместо строк сравниваются упорядоченные пары чисел предыдущего *C*
- При перерасчете массивов *P* и *C* потребуются их копии
- В итоге сложность каждой стадии алгоритма оценивается как $O(n)$
- Поскольку всего выполняется $O(\lg n)$ стадий, алгоритм сортировки циклических сдвигов работает за время $O(n\lg n)$
- Такую же сложность будет иметь и построение суффиксного массива
- В 2003г. опубликован алгоритм построения суффиксного массива за линейное время

Источники

1. Кормен Т., Лейзерсон Ч., Ривест Р, Штайн К. Алгоритмы: построение и анализ. 3-е издание / Пер. с англ. – М.: Вильямс, 2013. – 1328 с.
2. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. – СПб: Невский диалект, 2003. – 654 с.
3. Смит Б. Методы и алгоритмы вычислений на строках / Пер. с англ. – М: Вильямс, 2006. – 496 с.
4. Karp Richard M., Rabin Michael O. Efficient randomized pattern-matching algorithms // IBM J. Res. Develop. 31–2, 1987. – P. 249–260.
5. Omondi Amos, Premkumar Benjamin, Residue Number Systems: Theory and Implementation // Imperial College Press, 2007. – 296 p.
6. Окулов С.М. Алгоритмы обработки строк. – М.: БИНОМ, 2013. – 255 с.
7. Crochemore, M. and Wojciech, R., Jewels of Stringology, World Scientific, 2002, 310 p.
8. Вяхи Н.И. Алгоритмы в биоинформатике. Курс лекций [Электронный ресурс] / Computer Science клуб. – СПб, 2013.

   URL: http://compsciclub.ru/courses/algorithmsbioinformatics.

Источники

9. Префикс-функция // ИТМО [Электронный ресурс] : сайт wiki-конспектов. Спб, 2014.

   URL: http://neerc.ifmo.ru/wiki/index.php?title=Префикс-функция
10. Лифшиц Ю. Курс "Алгоритмы для Интернета" [Электронный ресурс] / Laboratory of Mathematical Logic at PDMI. – СПб, 2006.

    URL: http://logic.pdmi.ras.ru/~yura/internet.html.
11. Стариковская Т.А. Суффиксные деревья: новые идеи и открытые проблемы. Курс лекций [Электронный ресурс] / Computer Science клуб. – СПб, 2014.

    URL: http://compsciclub.ru/courses/suffixtrees.
12. Швед Д.А. Курс лекций «Алгоритмы: построение и анализ» [Электронный ресурс] / НОУ «ИНТУИТ». – М., 2014.

    URL: http://www.youtube.com/watch?v=Eog0zhbsv_E.
13. Lothaire M. Applied Combinatorics on Words // Encyclopedia of Mathematics and its Applications, 2005. Vol. 90. Cambridge University Press, Cambridge.
14. Ukkonen E. On-line construction of suffix trees // Algorithmica, 1995. Vol. 14, No 3, pp. 249–260.

Источники

15. Биркгоф Г. Теория решеток : пер. с англ. / Г. Биркгоф. – М. : Наука, 1984. – 568 с.
16. Даценко С. Суффиксный массив – удобная замена суффиксного дерева [Электронный ресурс] / IT-сообщество “Хабрахабр”. – 2011.

    URL: http://habrahabr.ru/post/115346/.
17. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск / Пер. с англ. – М.: «Вильямс», 2007. – 824 с.
18. Kasai T., Lee G., Arimura H., Arikawa S., Park K. Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications // Combinatorial Pattern Matching. Lecture Notes in Computer Science, 2001. Vol. 2089, pp. 181–192.
19. Иванов М. Суффиксный массив [Электронный ресурс] / MAXimal. – 2008.

    URL: http://e-maxx.ru/algo/suffix_array.
20. Kärkkäinen J., Sanders P., Burkhardt S. Simple linear work suffix array construction // Automata, Languages and Programming. Lecture Notes in Computer Science, 2003. Vol. 2719, pp. 943–955.

Спасибо за внимание!