Домашняя работа №4

Шишацкий Михаил, 932

26.04.2020

Задача 6. Пусть без ограничения общности $\pi(r) < \pi(l)$ (если $l = r$, то ответ находится тривиально и равен длине суффикса, т.е. $n - r + 1$). Докажем, что $lcp(s^l, s^r) = min(k_{\pi(r)}, k_{\pi(r)+1}, \cdots, k_{\pi(l)-1})$.

Сначала покажем, что это значение гарантированно достигается. На показанном отрезке минимум k_i означает, что любые две соседние строки имеют общий префикс длины k_{min}, а значит, по транзитивности, строки $\pi(r)$ и $\pi(l)$ имеют общий префикс длины k_{min}. Почему же нельзя получить больший общий префикс? Пусть $(k_{min} + 1)$-ые символы у рассматриваемых двух строк совпадают. Тогда все строки между ними в отсортированной лексикографически последовательности должны также иметь такие же первые $k_{min} + 1$ символов, иначе лексикографическая сортировка отработала некорректно. В таком случае, минимум на отрезке уже $K_{min} + 1$ - противоречие с выбором минимума.

Таким образом, нам необходимо научиться называть минимум на подотрезке массива k. С этой задачей отлично справляется Sparse Table: построим на массиве k таблицу на минимум за $O(n \log(n))$ начиная с высоты 1, нулевой уровень таблицы заполним длинами соответствующих суффиксов (на i-ой позиции - $n - \pi^{-1}(i) + 1$).

Чтобы получить ответ на запрос $lcp(s^l, s^r)$, вызовем запрос минимума на отрезке $[min(\pi(l), \pi(r)); max(\pi(l), \pi(r))]$.

Таким образом, q запросов мы обработаем за $O(q) + O(n \log(n))$ на построение таблицы.

Задача 8. Модифицируем прямое дерево Фенвика: в массиве $tree$ дерева будем хранить значения key - значение максимума на соответствующем ячейке промежутке. Вместе с этим массивом будем хранить исходный массив элементов arr.

Реализуем методы:

```
void FenwickTree::increase(size_t pos, size_t x) {
    arr[pos] += x; // arr[pos] is new candidate for being
                      max in segments including pos
    for (int i = pos; i < size; i |= i + 1) {
        tree[i] = max(tree[i], arr[pos]);
    }
}
```

```
int FenwickTree::prefix_max(size_t idx) {
    int max_ans = -INF;
    for (int i = idx; i >= 0; i = (i & (i + 1)) - 1) {
        max_ans = max(max_ans, tree[i]);
    }
    return max_ans;
}
```

Инициализируем дерево массивом, состоящим из -INF, затем увеличим каждую ячейку соответственно на значение INF+a_i (так сделано, чтобы учесть случай наличия отрицательных элементов массива). Это выполнится за $O(n \log(n))$. Далее обработаем q запросов. Так как каждый запрос обрабатывается за $O(\log(n))$, все запросы мы обработаем за $O(q \log(n))$. Суммарная сложность $O((n + q) \log(n))$