# Алгоритмы и Структуры Данных ДЗ-2

Гарипов Роман M3138

03.05.2020

## Задача №1

#### Описание структуры

Построим дерево отрезков в каждой вершине $v$, отвечающей за отрезок $[tl, tr]$ которого будем хранить матрицу 4 на 4, где $dp[i][j]$ - число способов добраться от $tl + i$ до $tr - j$ прыгая на одну, две или три клетки.

### Объединяем отрезки

Чтобы научиться решать задачу, осталось понять, как посчитать значение в вершине, если уже посчитали в левом и правом ребёнке. Для наглядности объясним это кодом :

```
// v - vertex
// [tl, tr] - cur_segment
// tm = (tl + tr) / 2
// check_border(x, y) - returns true if exists border on segment [x;y],
// we need to check only short segments which length is less or equal than 3,
// so we can count it as constanty works function
  for (int a = 0; a <= 3; a++) {
     for (int b = 0; b <= 3; b++) {  // a, b - offsets for [tl, tm] - segment
         for (int c = 0; c <= 3; c++) {
           for (int d = 0; d <= 3; d++) {  // c, d - offsets for [tm + 1, tr] segment
              int r = tm + 1 + c;
              int l = tm - b;
              if (r - l + 1 <= 3 && !check_border(l, r)) {
                 cur_vert_dp[a][d] += dp[a][b] * dp[c][d]
              }
           }
        }
     }
  }
```

Осталось только понять какие будут начальные значения для вершин соответствующих отрезкам единичной длины. Для них $dp[0][0] = 1$, остальные значения равны 0.

### Отвечаем на запросы

- Когда просят посчитать число способов на отрезке, спустимся по ДО, как обычно делаем это, когда считаем функцию от отрезка, только будем пересчитывать значения, как это указано выше. Получим матрицу $dp$, ответом на запрос будет $dp[0][0]$.

- Когда просят добавить перегородку на позицию $x$, спустимся в ДО до вершины, соответсвующей единичному отрезку $[x, x]$ и скажем, что для него $dp$ - нулевая матрица. Так же пометим, что на этом месте находится перегородка, для того чтобы в дальнейщем работала функция `check_border`.

  Аналогично, когда просят удалить перегородку, ставим в эту вершину матрицу, в которой $dp[0][0] = 1$, остальные - нули. Поднимаемся вверх и пересчитываем значения в вершинах.

**Итого:** Отвечаем на все запросы за $\mathcal{O}(\log n)$ спуском по ДО.

# Задача №3

## Научимся отвечать на такие запросы на префиксе

Выделим первое вхождение каждого возмножного значения массива. Понятно, что ответом на запрос на префиксе - будет количество отмеченных вершин. Это можно считать деревом отрезков на сумму. Давайте поймем как отвечать на запросы на отрезке.

## А теперь на отрезке

Посчитаем массив $next[i]$ - индекс ближайшего справа элемента массива, значение которого равно $a[i]$. Тогда, уберем отметку с первого элемента массива, поставим ее на $next[1]$. Получили массив отметок для всех отрезков $[1; x], x \in [1; n-1]$. Проделаем так, пока не получим массив отметок для всех отрезков. Для того, чтобы это занимало мало памяти, воспользуемся персистентным деревом отрезоков. А именно - $\mathcal{O}(n \log n)$ памяти, так как каждое изменение массива, будет создавать дополнительную ветку персистентного дерева отрезков.

**Итого:**

- Насчитаем массив $next[i]$ и поставим отметки на первое вхождение каждого числа.
- Для всех возможных правых границ запроса пересчитаем значения в персистентном дереве отрезков
- Когда приходит запрос `unique(l, r)` - возьмем версию персистеного дерева отрезков для отрезков начинающихся в `l`. И вернем сумму на отрезке $[l; r]$ в этой версии дерева отрезков.

Получили решение в оффлайне с предпосчётом за $\mathcal{O}(n \log n)$ и ответом на запрос за $\mathcal{O}(\log n)$.

# Задача №4

## Идея решения

Давайте обусловимся, что чтобы посчитать значение в вершине, надо взять сумму значений всех вершин в поддереве. Тогда можно свести запрос прибавления на пути `add-on-path(v, u, x)` - прибавить `x` всем вершинам на пути от `v` до `u`, к следующим операциям :

- `val[v] += x`
- `val[u] += x`
- `val[lca(u, v)] -= x`
- `val[parent(lca(u, v))] -= x`

Теперь давайте поймём почему это действительно то что мы хотим.

(1) Рассмотрим вершину на пути от `u` до `v` кроме их `lca`. Для всех таких вершин сумма в поддереве увеличилась на `x`.

(2) Рассмотрим `lca(v, u)`. Сумма в поддереве изменилась так : $(was + x + x - x) = was + x$. Сумма в поддереве увеличилась на `x`.

(3) Рассмотрим вершину, не принадлежащую пути от `v` до `u`.

Если она находится выше их `lca`, то сумма в поддереве не изменится, так $(was + x + x - x - x) = was$.

Если она находится ниже, т.е либо ниже вершины `v` либо ниже вершины `v`, то для нее в поддереве ничего не изменилось

**Тогда алгоритм действий будет следующий:**

- Выпишем эйлеров обход, но немного упрощенный, запустимся dfs-ом из корня, когда зашли в какую-то вершину, сразу выпишем её в вектор. И для каждой вершины запомним какой отрезок отвечает за ее поддерево.

- Посчитаем двоичные подъёмы за $\mathcal{O}(n\log(n))$, чтобы мы могли вычислять $lca$ за $\mathcal{O}(\log(n))$
- Построим дерево отрезков на сумму на массиве значений в вершинах, выписанных в указанном выше порядке.
- Когда приходит запрос прибавления на пути, делаем прибавление деревом отрезков в 4-х точках :
  $val[v]$`+=` $x$,
  $val[u]$`+=` $x$,
  $val[lca(v,u)]$`-=` $x$,
  $val[parent(lca(v,u))]$`-=` $x$
  $= \mathcal{O}(\log(n))$ операций.
- Когда приходит запрос значения в вершине, берем сумму на отрезке, соответствующему поддереву этой вершины деревом отрезков за $\mathcal{O}(\log(n))$.