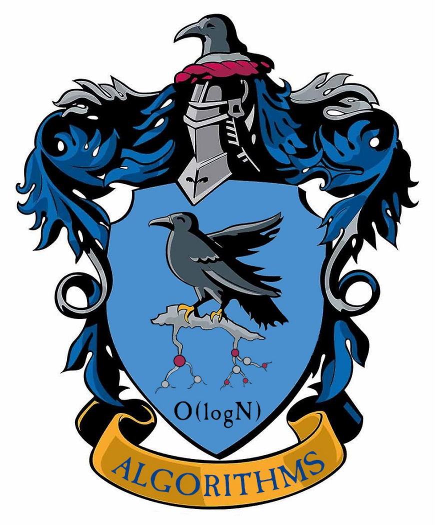
Мини-задача #45 (1 балл)

https://leetcode.com/problems/create-sorted-array-through-instructions/

Решите задачу, используя дерево Фенвика.

Алгоритмы и структуры данных

Дерево Фенвика



Дерево Фенвика (Binary indexed tree)

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

a_0, a_1, a_2, …, a_{n-1}

Необходимо поддержать новые операции следующего вида:

1. update(pos, val) - обновляем значение элемента a_pos
2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

Дерево Фенвика (Binary indexed tree)

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

a_0, a_1, a_2, …, a_{n-1}

Необходимо поддержать новые операции следующего вида:

1. update(pos, val) - обновляем значение элемента a_pos

2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

Но мы уже знаем два варианта решения! (Декартовы деревья и деревья отрезков)

Дерево Фенвика (Binary indexed tree)

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

a_0, a_1, a_2, …, a_{n-1}

Необходимо поддержать новые операции следующего вида:

1. update(pos, val) - обновляем значение элемента a_pos
2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

Но мы уже знаем два варианта решения! (Декартовы деревья и деревья отрезков)

Сегодня посмотрим еще одно решение и обсудим его плюсы.

`a_0, a_1, a_2, …, a_{n-1}`

```
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)
g(x) = x | (x + 1)
```

```
a_0, a_1, a_2, …, a_{n-1}
```

Введем две функции:

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

```
a_0, a_1, a_2, …, a_{n-1}
```

Введем две функции:

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

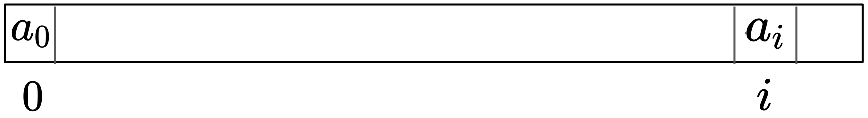
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$



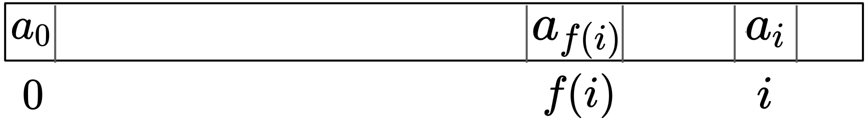
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$



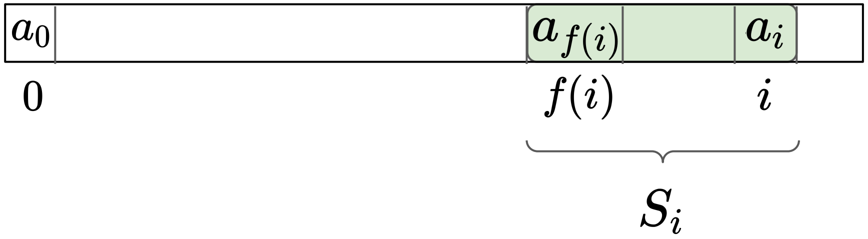
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$



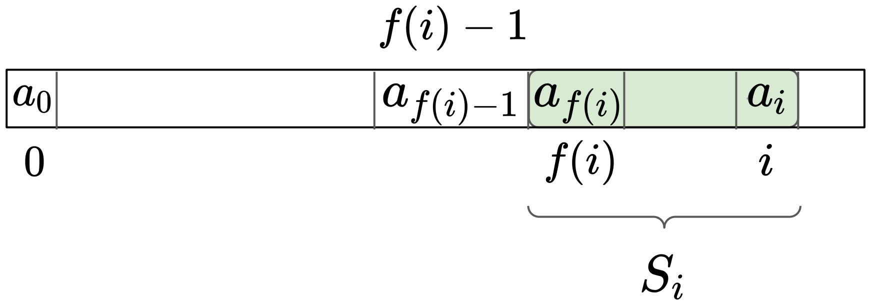
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$



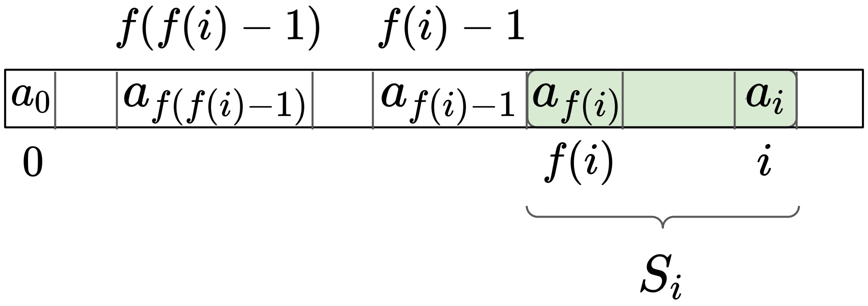
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$



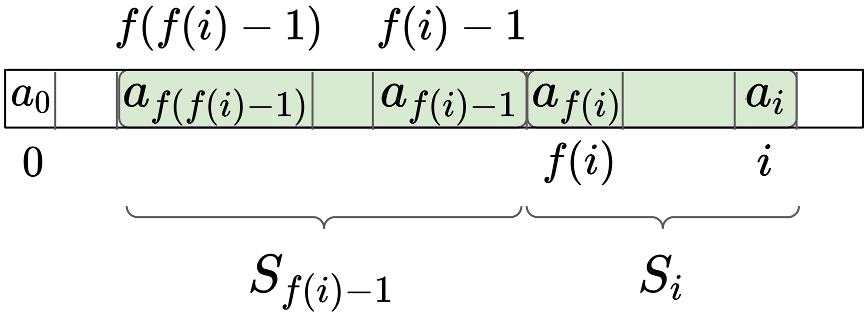
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$



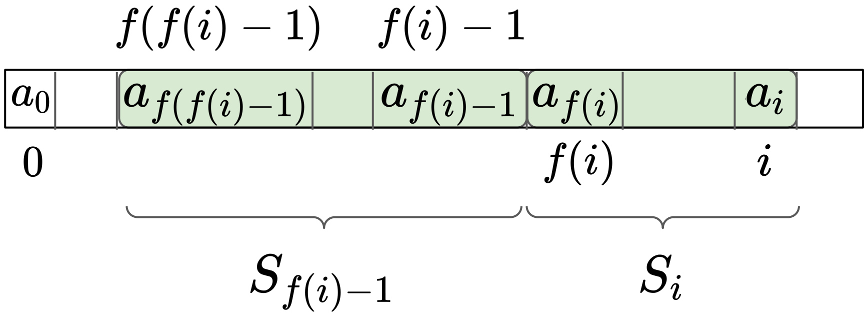
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$



Таким образом можем покрыть весь промежуток до нуля, т.е. получить префиксную сумму!

Prefix sum

```
def buildPrefixes(n: int, arr: int[]) -> int[]:
  prefixes = int[n]

  currentSum = 0
  for i in 0..n-1:
    currentSum += arr[i]
    prefixes[i] = currentSum

  return prefixes
```

Работает за O(N) времени и O(N) памяти

Препроцессинг запустили всего один раз, а запросов может быть много!

```
def sum(l, r: int, prefixes: int[]) -> int:
  return prefixes[r] - prefixes[l - 1]
```

Работает за O(1)

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1) & - побитовое "и"

g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Сложность?

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Сложность?

$a_0, a_1, a_2, \ldots, a_{n-1}$

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

$f(x) = x \,\&\, (x + 1)$ & - побитовое "и"

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"

```
    x = 1…010100111
x + 1 = ?
```

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)

& - побитовое "и"

```
    x = 1…010100111
x + 1 = 1…010101000
```

последняя группа единиц занулилась, первый ноль перед ними стал единицей

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)

& - побитовое "и"

```
    x = 1…010100111
x + 1 = 1…010101000
 f(x) = 1…010100000
```

последняя группа единиц занулилась, первый ноль перед ними стал единицей

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)

& - побитовое "и"

```
    x = 1…010100111
x + 1 = 1…010101000
 f(x) = 1…010100000
```

последняя группа единиц занулилась, первый ноль перед ними стал единицей

f(x) занулит первый блок единиц.

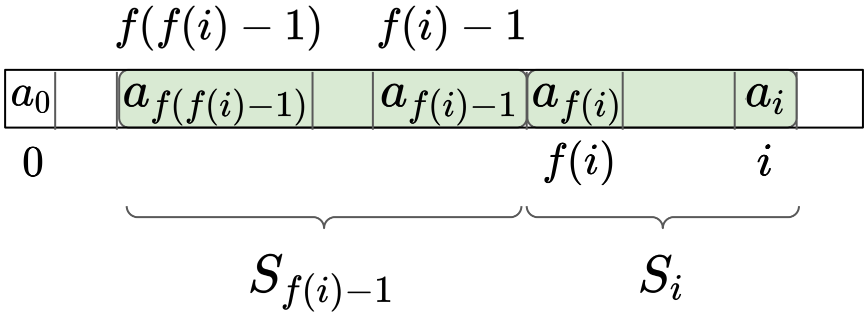
a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$



Таким образом можем покрыть весь промежуток до нуля, т.е. получить префиксную сумму!

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

$f(x) = x \,\&\, (x + 1)$ & - побитовое "и"

$x = 1…010100111$

$x + 1 = 1…010101000$

последняя группа единиц занулилась, первый ноль перед ними стал единицей

$f(x) = 1…010100000$

f(x) занулит первый блок единиц.

$f(x) - 1 = ?$

$a_0, a_1, a_2, \ldots, a_{n-1}$

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

$f(x) = x \& (x + 1)$

& - побитовое "и"

$$
\begin{aligned}
x &= 1\ldots010100111 \\
x + 1 &= 1\ldots010101000 \\
f(x) &= 1\ldots010100000
\end{aligned}
$$

последняя группа единиц занулилась, первый ноль перед ними стал единицей

f(x) занулит первый блок единиц.

$f(x) - 1 = ???????1111$

т.е. f(x) - 1 в записи x превратил младший 0 в 1. Что-то еще поменялось в старших разрядах, но нам не важно!

$a_0, a_1, a_2, \ldots, a_{n-1}$

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

$f(x) = x \& (x + 1)$

& - побитовое "и"

$$x = 1\ldots010100111$$
$$x + 1 = 1\ldots010101000$$

последняя группа единиц занулилась, первый ноль перед ними стал единицей

$$f(x) = 1\ldots010100000$$

f(x) занулит первый блок единиц.

$$f(x) - 1 = ???????1111$$

т.е. f(x) - 1 в записи x превратил младший 0 в 1. Что-то еще поменялось в старших разрядах, но нам не важно!

Сколько раз так можно сделать?

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"

```
    x = 1…010100111
x + 1 = 1…010101000
 f(x) = 1…010100000

f(x) - 1 = ???????1111
```

последняя группа единиц занулилась, первый ноль перед ними стал единицей

f(x) занулит первый блок единиц.

т.е. f(x) - 1 в записи x превратил младший 0 в 1. Что-то еще поменялось в старших разрядах, но нам не важно!

Сколько раз так можно сделать? Не больше, чем бит в числе!

$a_0, a_1, a_2, \ldots, a_{n-1}$

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

$f(x) = x \,\&\, (x + 1)$

& - побитовое "и"

$$x = 1\ldots010100111$$
$$x + 1 = 1\ldots010101000$$

последняя группа единиц занулилась, первый ноль перед ними стал единицей

$$f(x) = 1\ldots010100000$$

f(x) занулит первый блок единиц.

$$f(x) - 1 = ???????1111$$

т.е. f(x) - 1 в записи x превратил младший 0 в 1. Что-то еще поменялось в старших разрядах, но нам не важно!

Сколько раз так можно сделать? Не больше, чем бит в числе!

Если всего элементов в массиве N, то получаем logN.

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Сложность?

a_0, a_1, a_2, …, a_{n-1}

А сколько раз можно звать от числа f(x), пока не дойдем до 0?

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Сложность? O(logN)

Дерево Фенвика

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_0, a_1, a_2, \dots, a_{n-1}$$

Необходимо поддержать новые операции следующего вида:

1. update(pos, val) - обновляем значение элемента a_pos
2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

Дерево Фенвика

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_0, a_1, a_2, \ldots, a_{n-1}$$

Необходимо поддержать новые операции следующего вида:

????? 1. update(pos, val) - обновляем значение элемента a_pos

O(logN) 2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1) & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

Тогда update кроме элемента массива должен будет обновлять насчитанные суммы S_i

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

```
    i = 1…010100111
i + 1 = 1…010101000
 f(i) = 1…010100000
```

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

i = **1…010100** 111

Раз у i и у f(i) общий индекс (до младшего блока из единиц в i), такой же индекс должен быть и у pos!

f(i) = **1…010100** 000

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

i = **1…010100** 111

pos = **1…010100** ...

f(i) = **1…010100** 000

Раз у i и у f(i) общий индекс (до младшего блока из единиц в i), такой же индекс должен быть и у pos!

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

```
   i = 1…010100 111
 pos = 1…010100 ...
f(i) = 1…010100 000
```

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos = 1…01010011101110110

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos = 1…0101001110111011 1

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

pos/i_1 = **1…0101001110111011 0** 1

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

pos/i_1 = 1…01010011101110110 1

f(i_1) = 1…01010011101110110 0

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
pos/i_1 = 1…0101001110111011 0 1
pos     = 1…0101001110111011 0 1
f(i_1)  = 1…0101001110111011 0 0
```

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos/i_1 = 1…010100111011110110 1

i_2 = 1…0101001110111 0 1111

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
i_2    = 1…01010011101110 1111
pos    = 1…01010011101110 1101
f(i_2) = 1…01010011101110 0000
```

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

pos/i_1 = 1…010100111011101101 1

i_2 = 1…01010011101110 1111

i_3 = 1…0101001110 11111111

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

i_3 = 1…0101001110 11111111

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

f(i_3) = 1…0101001110 00000000

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
i_3    = 1…0101001110 11111111
pos    = 1…01010011101110 1101
f(i_3) = 1…0101001110 00000000
```

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
i_? = 1…0101001110111 10101
pos = 1…0101001110111 01101
```

Почему бы не взять префикс, сбросив пару единиц из суффикса?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
i_?    = 1…0101001110111 10101
pos    = 1…0101001110111 01101
f(i_?) = 1…0101001110111 10100
```

Почему бы не взять префикс, сбросив пару единиц из суффикса?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
i_?    = 1…0101001110111 10101
pos    = 1…0101001110111 01101
f(i_?) = 1…0101001110111 10100
```

Почему бы не взять префикс, сбросив пару единиц из суффикса? Тогда f(i_?) будет больше, чем pos, что нарушает условие!

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos/i_1 = 1…0101001110111011 0 1

i_2 = 1…01010011101110 1111

i_3 = 1…0101001110 11111111

Таким образом, нам точно подойдет S_{pos}, его точно обновлять. А как найти остальные такие i?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos/i_1 = 1…0101001110111011 0 1

i_2 = 1…0101001110111 0 1111

i_3 = 1…0101001110 11111111

Т.е. чтобы перебирать подходящие нам i, выставляем младший 0 в 1 => получаем более широкий блок последних единиц => новый общий с pos префикс => новый i

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

pos/i_1 = **1…0101001110111011 0** 1   И как же это сделать?

i_2 = **1…01010011101110** 1111

i_3 = **1…0101001110** 11111111

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos/i_1 = 1…01010011101110110 1

i_2 = 1…01010011101110 1111

i_3 = 1…0101001110 11111111

И как же это сделать?
Так это же функция g(x)!

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
pos/i_1 = 1…01010011101110110 1
i_1 + 1 = 1…010100111011101110
```

И как же это сделать?
Так это же функция g(x)!

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
pos/i_1 = 1…01010011101110110 1
i_1 + 1 = 1…010100111011101110
 g(i_1) = 1…01010011101111011 11
```

И как же это сделать?
Так это же функция g(x)!

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

pos/i_1 = 1…0101001110111011 1

g(pos) = 1…0101001110111 1111

g(g(pos))=1…0101001110 11111111

И как же это сделать?
Так это же функция g(x)!

```
f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
pos/i_1 = 1…0101001110111011 1
 g(pos) = 1…0101001110111 1111
g(g(pos))=1…0101001110 11111111
```

И как же это сделать?
Так это же функция g(x)!

Сколько раз так можно сделать?

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

```python
def increment(pos: int, val: int) -> int:
```

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

```python
def increment(pos: int, val: int) -> int:
    i = pos
    while i < n:
```

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить $\Leftrightarrow$ $f(i) \leq pos \leq i$

```
def increment(pos: int, val: int) -> int:
    i = pos
    while i < n:
     S_i += val
     i = g(i)
```

```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

Пришел запрос increment(pos, val). Какие S_i нужно обновить?

S_i нужно обновить ⇔ $f(i) \leq pos \leq i$

```
def increment(pos: int, val: int) -> int:
    i = pos
    while i < n:
     S_i += val
     i = g(i)
```

Цикл идет не больше, чем количество битов в n => сложность O(logN)

Дерево Фенвика

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_0, a_1, a_2, \ldots, a_{n-1}$$

Необходимо поддержать новые операции следующего вида:

1. update(pos, val) - обновляем значение элемента a_pos
2. sum(l, r) - найти сумму элементов: a_l + … _ a_r

Дерево Фенвика

Решение:

```python
def f(x: int) -> int:
    return x & (x + 1)
```

```python
def g(x: int) -> int:
    return x | (x + 1)
```

Дерево Фенвика

Решение:

```
def f(x: int) -> int:
    return x & (x + 1)
```

```
def g(x: int) -> int:
    return x | (x + 1)
```

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Дерево Фенвика

Решение:

```
def f(x: int) -> int:
    return x & (x + 1)
```

```
def g(x: int) -> int:
    return x | (x + 1)
```

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

```
def increment(pos, val: int) -> int:
    i = pos
    while i < n:
     S_i += val
     i = g(i)
```

Дерево Фенвика

Решение:

```
def f(x: int) -> int:
    return x & (x + 1)
```

```
def g(x: int) -> int:
    return x | (x + 1)
```

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

```
def increment(pos, val: int) -> int:
    i = pos
    while i < n:
     S_i += val
     i = g(i)
```

```
def update(pos, val: int) -> int:
    increment(pos, val - a[pos])
```

Дерево Фенвика

Решение:

```
def f(x: int) -> int:
    return x & (x + 1)
```

```
def g(x: int) -> int:
    return x | (x + 1)
```

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

```
def increment(pos, val: int) -> int:
    i = pos
    while i < n:
     S_i += val
     i = g(i)
```

```
def update(pos, val: int) -> int:
    increment(pos, val - a[pos])
```

Дерево Фенвика (Binary indexed tree)

Дерево Фенвика (Binary indexed tree)

Почему оно дерево?

Дерево Фенвика (Binary indexed tree)

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)

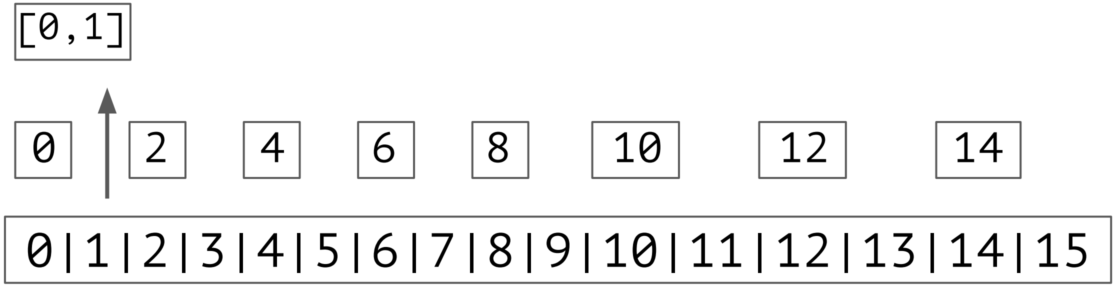
```
для четных f(x) = x => поэтому
покрываемый регион - самое это число
```

0	2	4	6	8	10	12	14

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)

f(1) = 0 => получаем сумму от 0 до 1



индексы:

Дерево Фенвика (Binary indexed tree)

f(3) = 0 => получаем сумму от 0 до 3

[0,3]

[0,1]

0 2 4 6 8 10 12 14

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

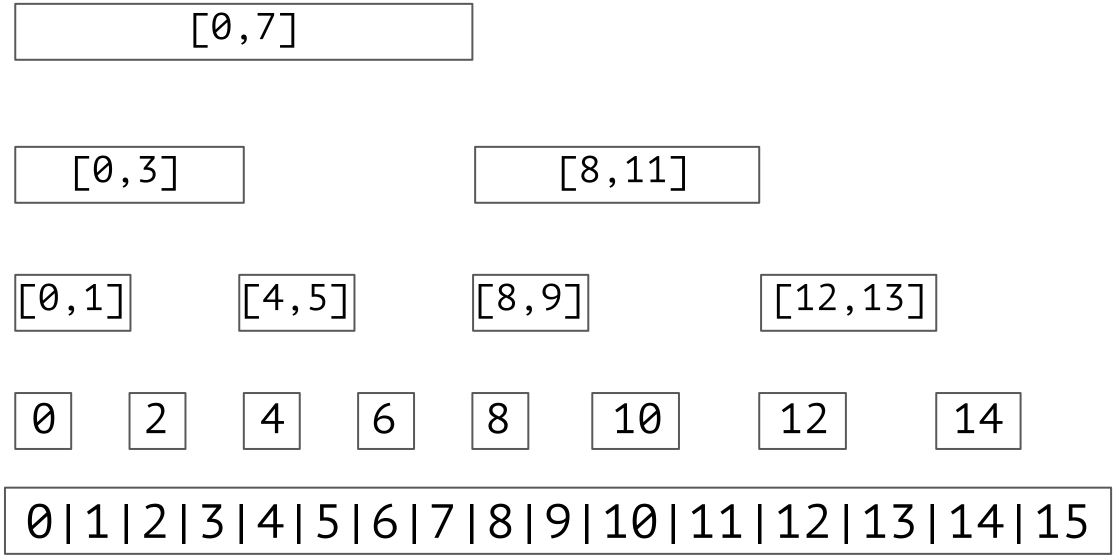
Дерево Фенвика (Binary indexed tree)

```
f(5) = 4 => получаем сумму от 4 до 5
```

```
             [0,3]

         [0,1]      [4,5]

         0   2   4   6   8   10   12   14

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15
```

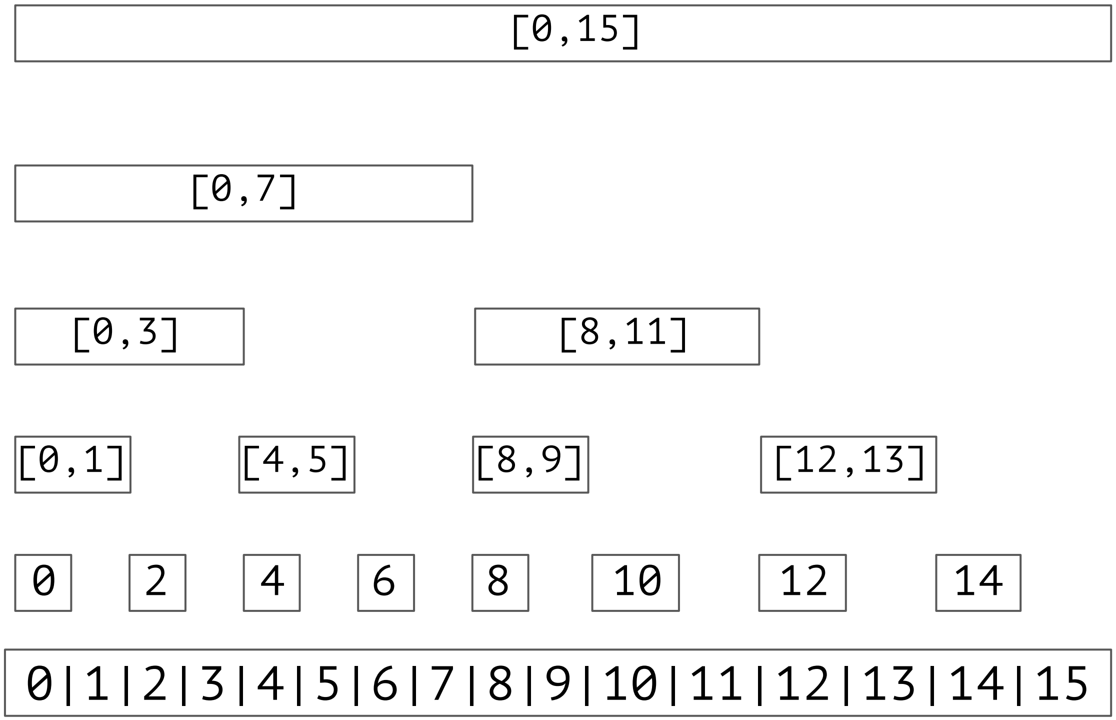
Дерево Фенвика (Binary indexed tree)

f(5) = 4 => получаем сумму от 4 до 5



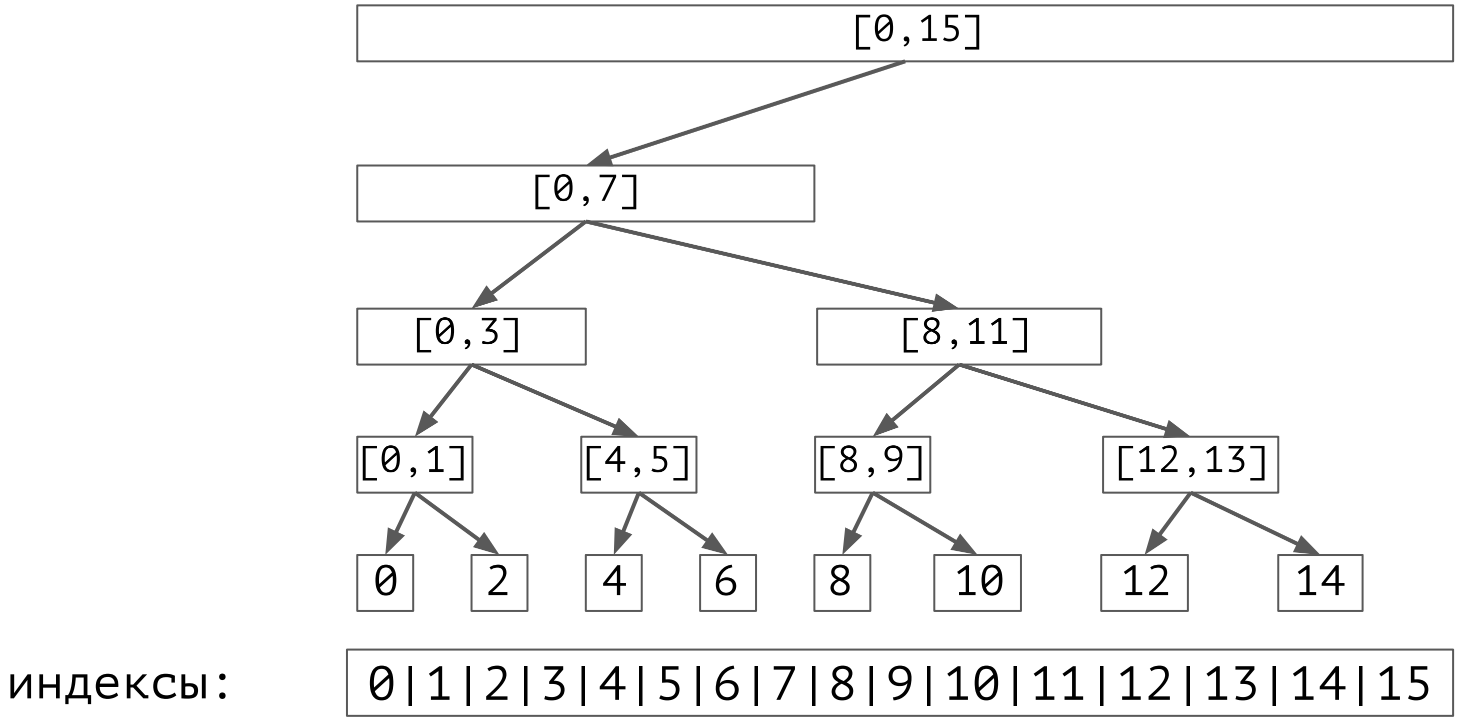
индексы:

Дерево Фенвика (Binary indexed tree)

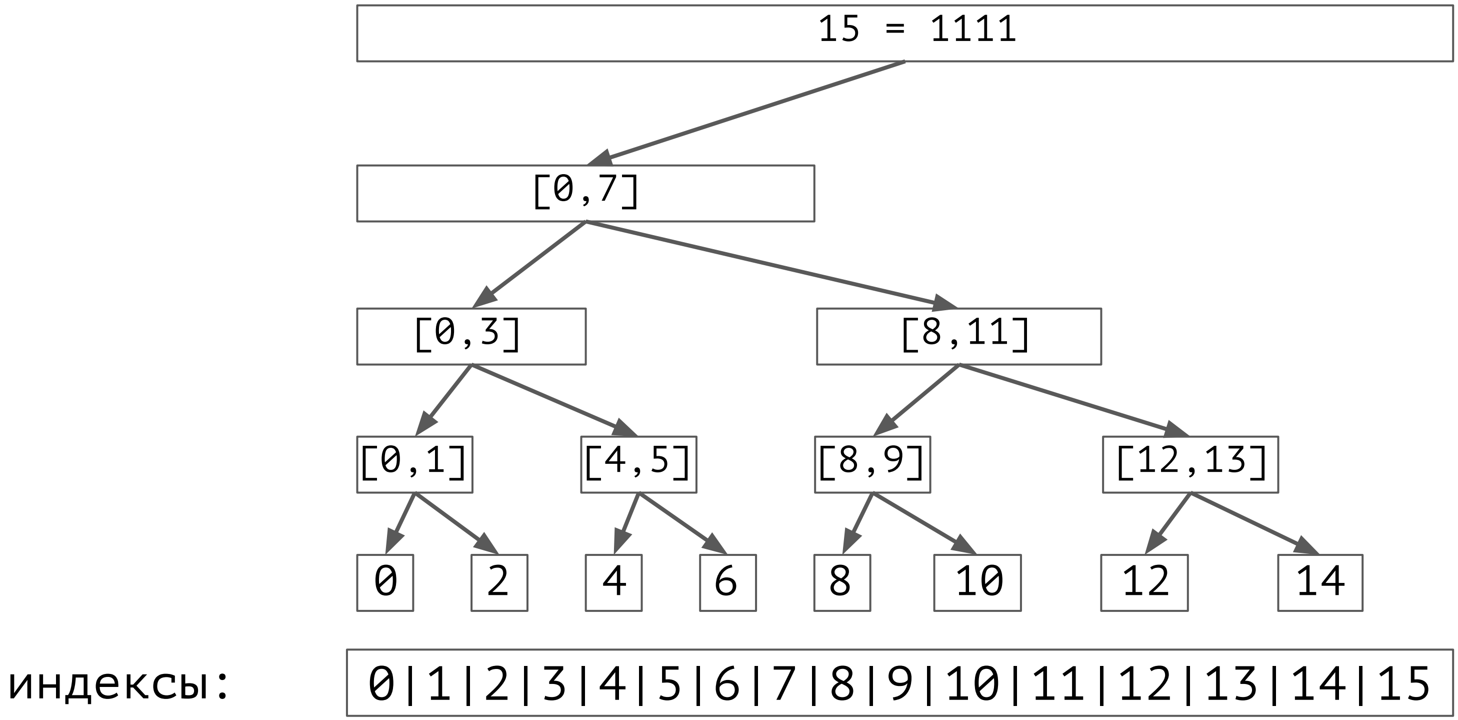


индексы:

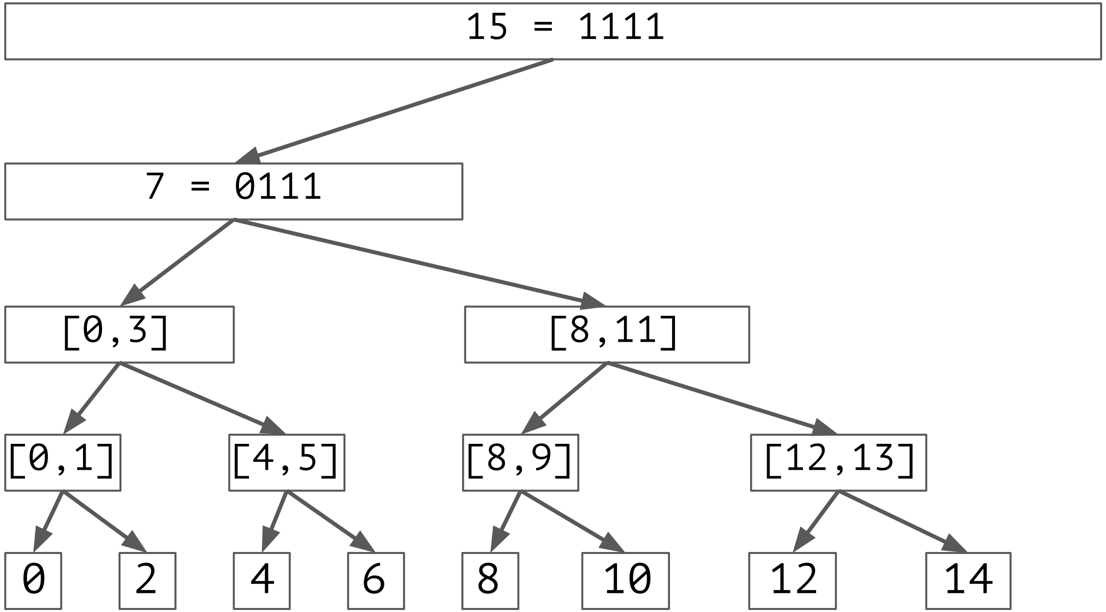
Дерево Фенвика (Binary indexed tree)



Дерево Фенвика (Binary indexed tree)

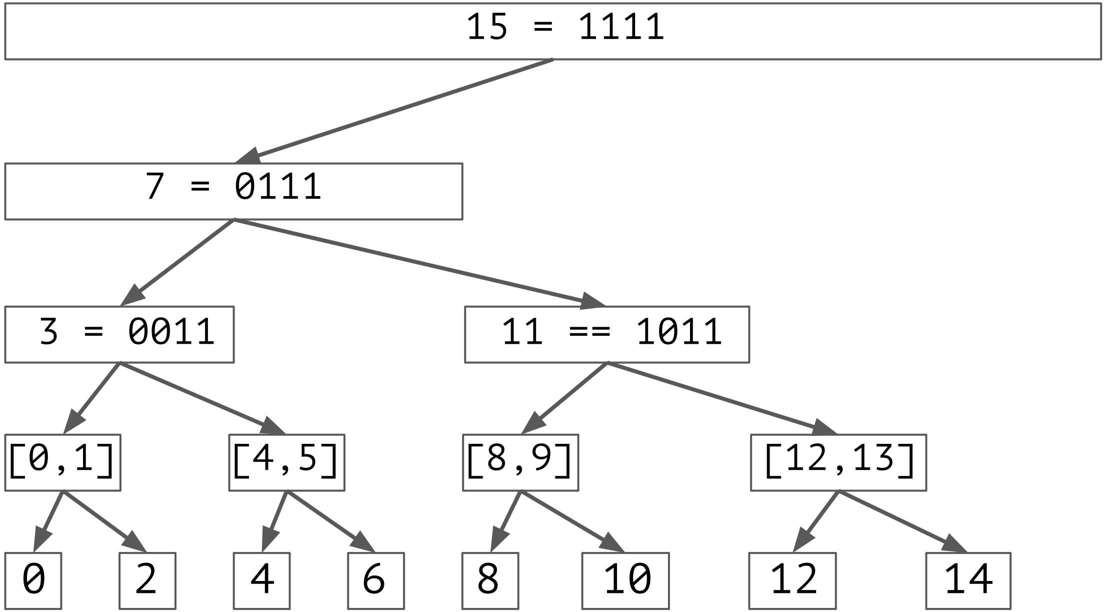


Дерево Фенвика (Binary indexed tree)



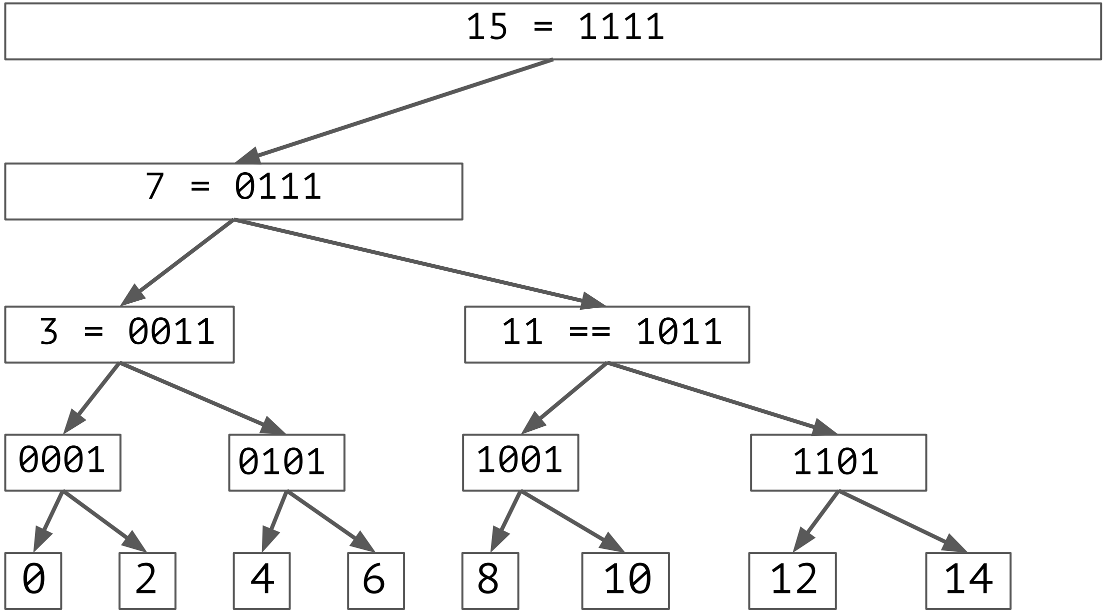
индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)



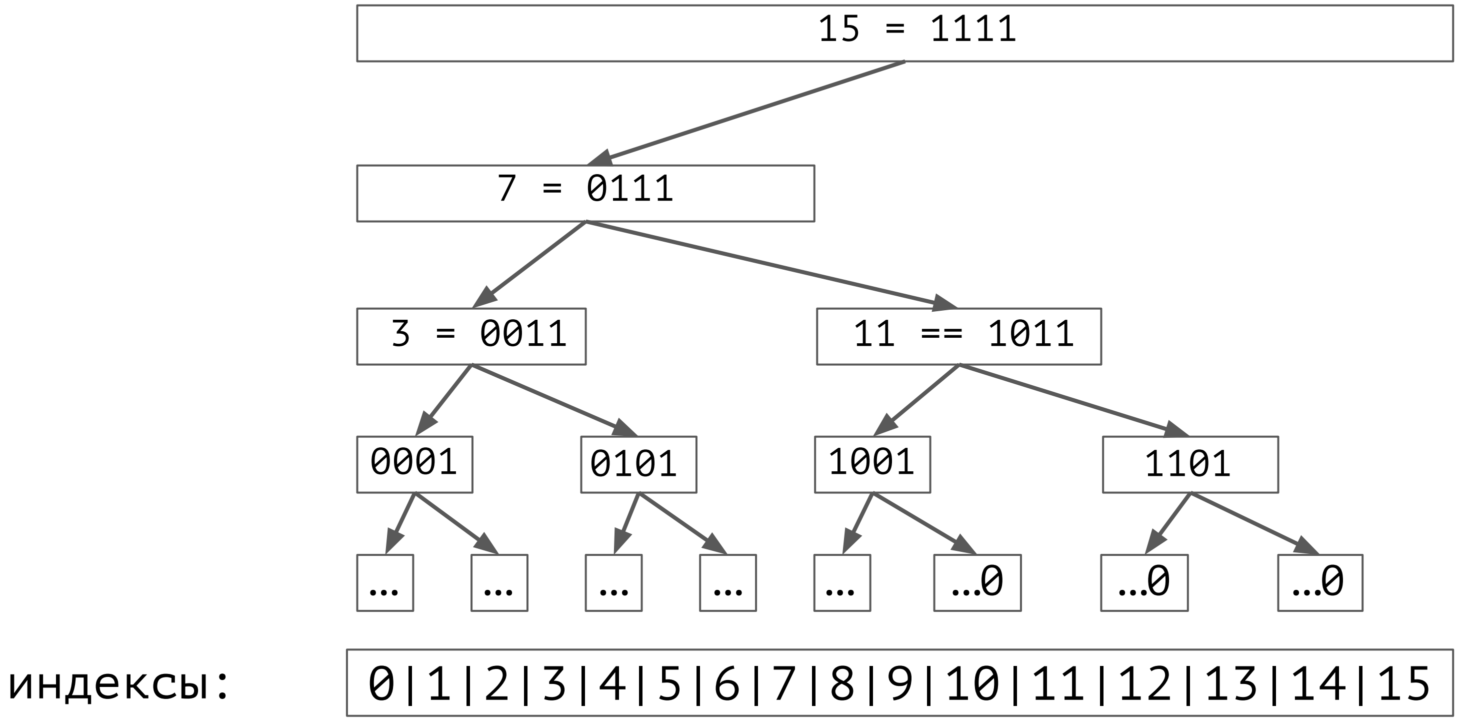
индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)



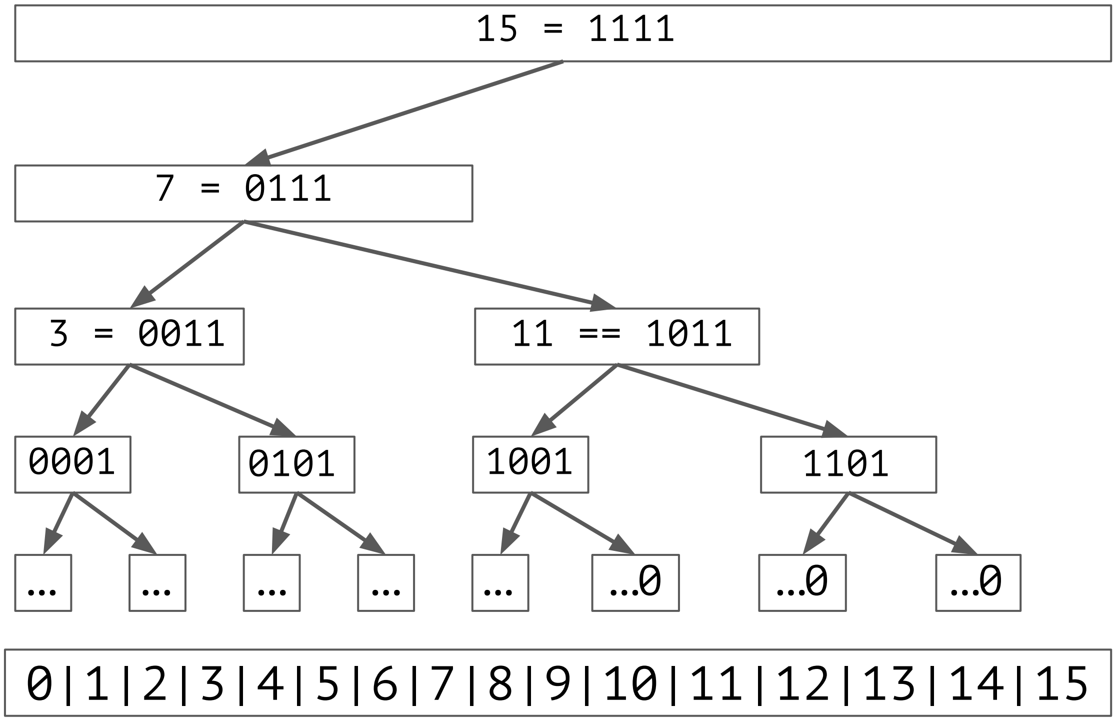
индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)



Дерево Фенвика (Binary indexed tree)

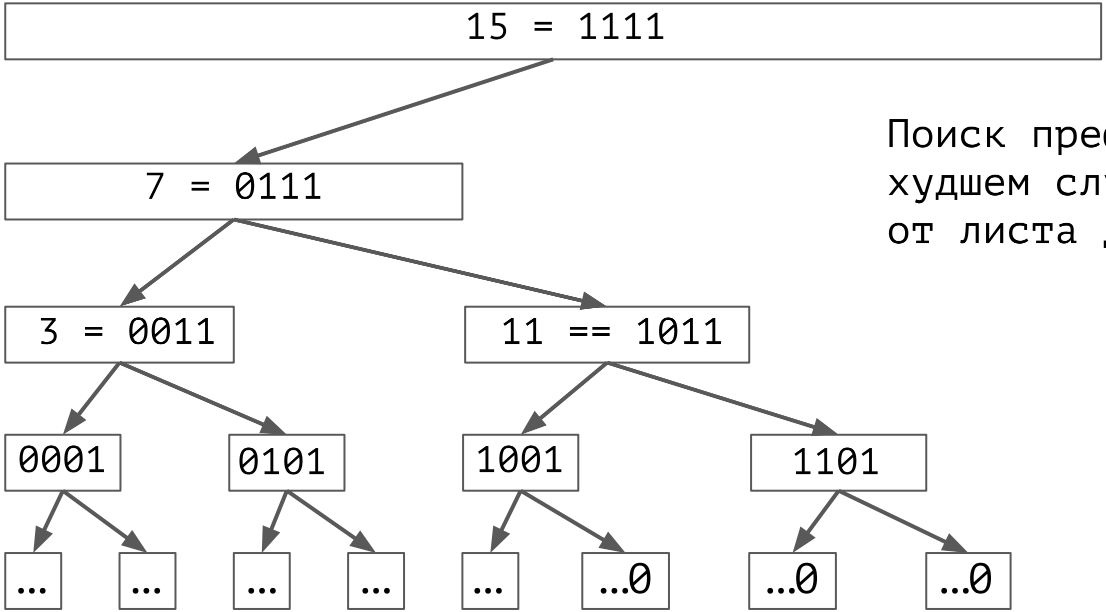
Уровни по количеству единиц в последнем блоке



индексы:

Дерево Фенвика (Binary indexed tree)

Уровни по количеству единиц в последнем блоке

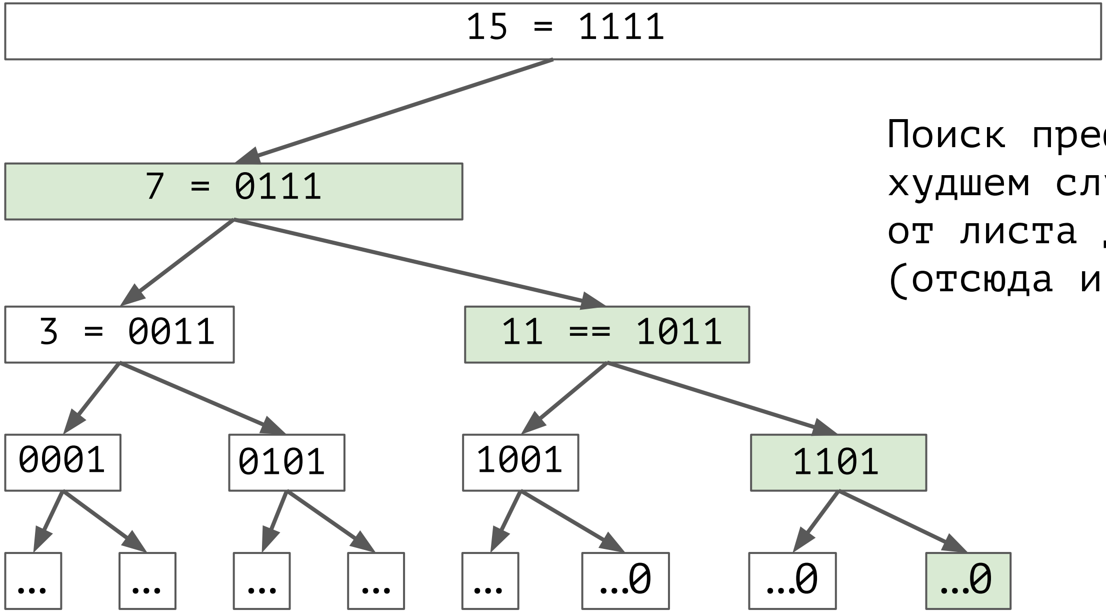


Поиск префиксный суммы в худшем случае - проход от листа до корня

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)

Уровни по количеству единиц в последнем блоке

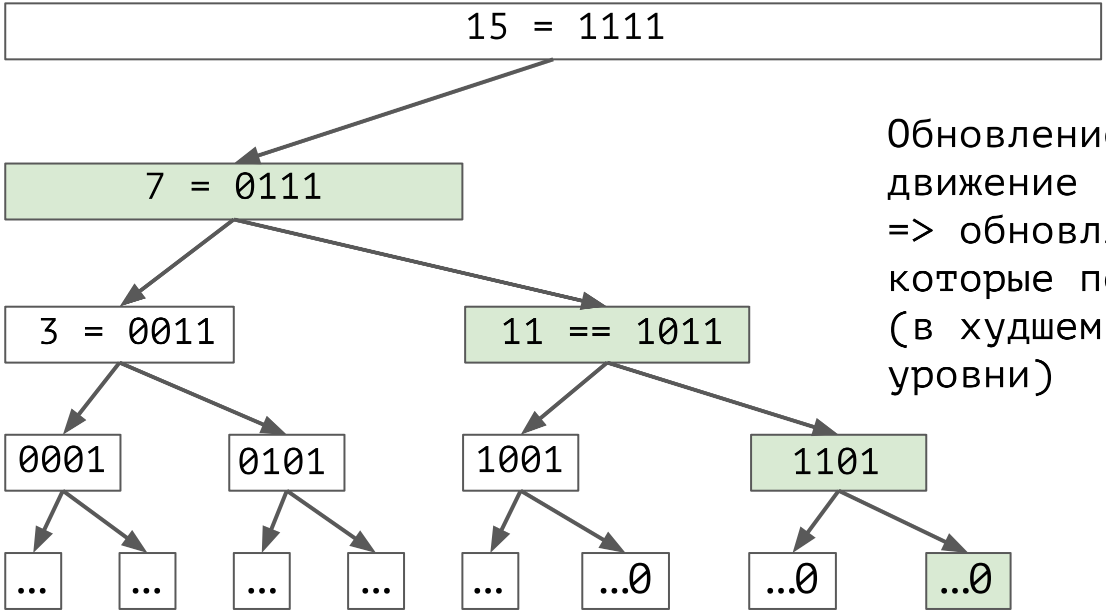


Поиск префиксный суммы в худшем случае - проход от листа до корня (отсюда и логарифм)

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)

Уровни по количеству единиц в последнем блоке

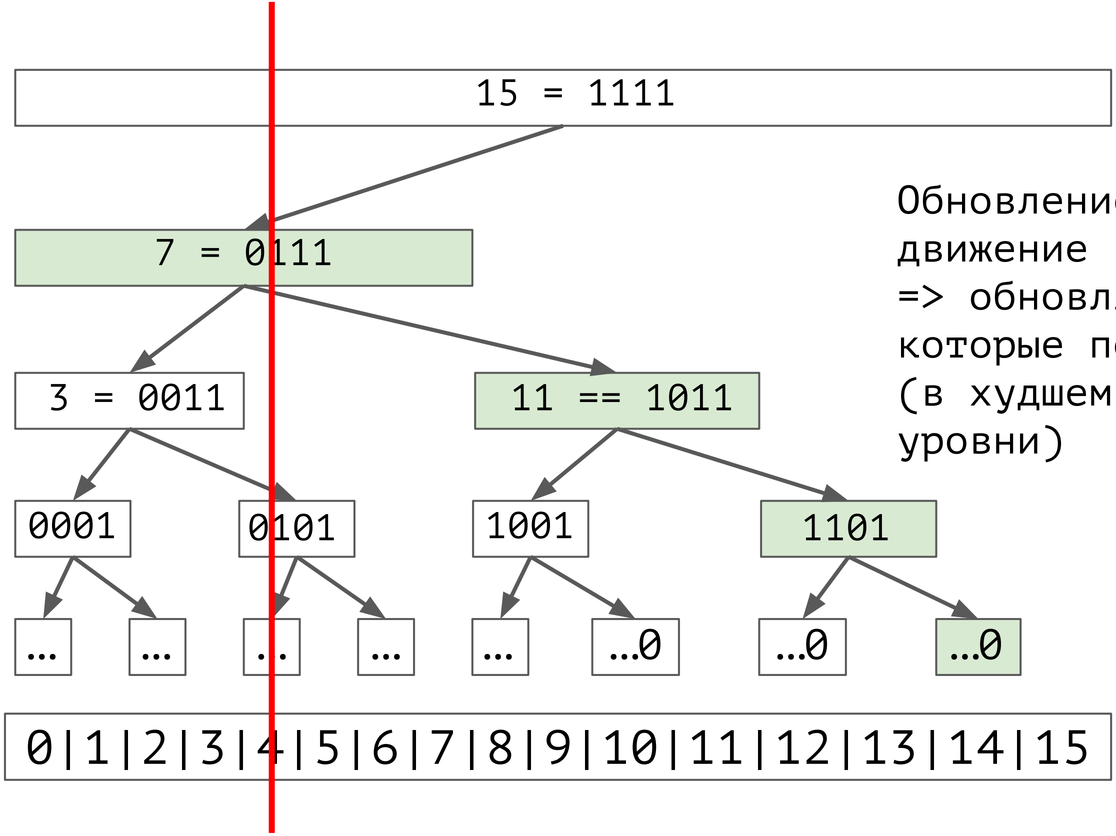


Обновление сумм - тоже движение по дереву вверх => обновляете все суммы, которые покрывают индекс (в худшем случае - все уровни)

индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)

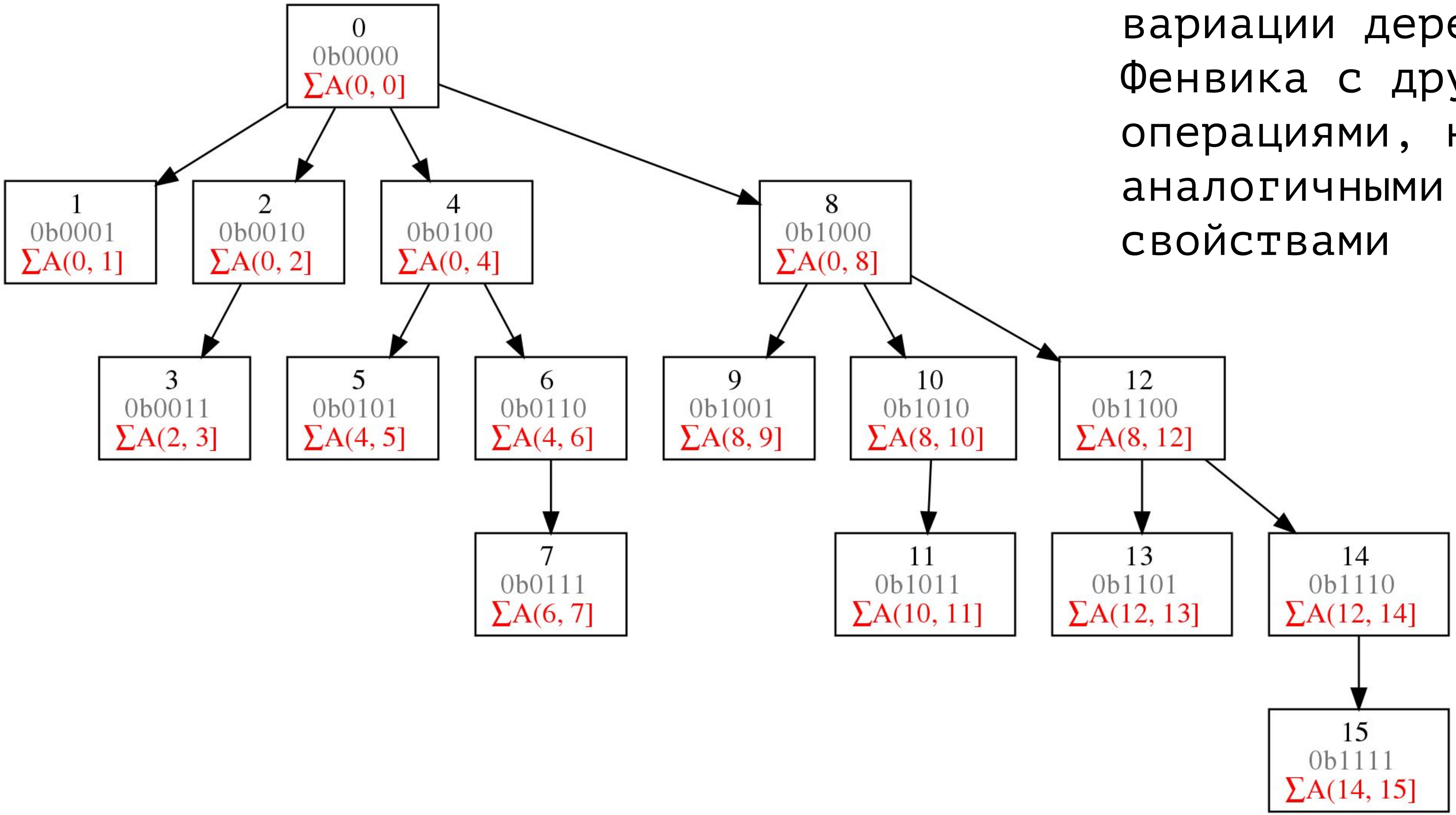
Уровни по количеству единиц в последнем блоке



Обновление сумм - тоже движение по дереву вверх => обновляете все суммы, которые покрывают индекс (в худшем случае - все уровни)

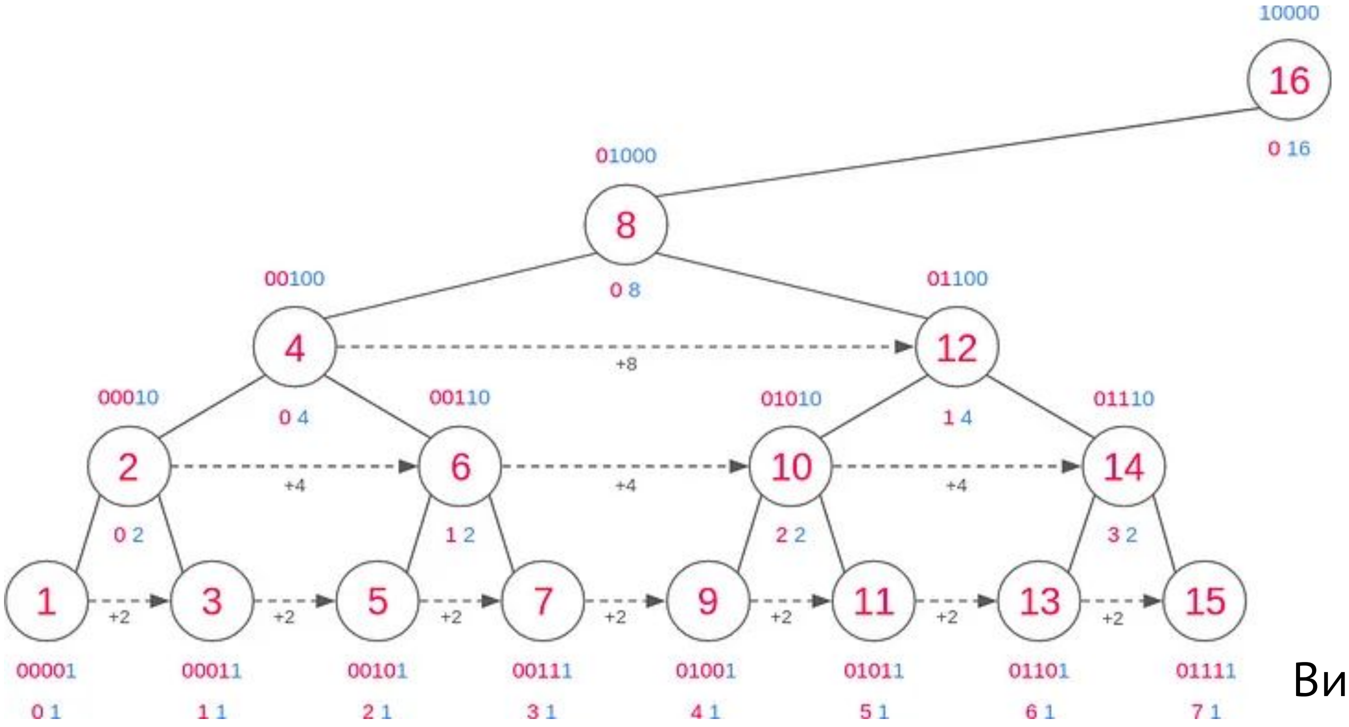
индексы: 0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15

Дерево Фенвика (Binary indexed tree)



Есть другие вариации дерева Фенвика с другими операциями, но аналогичными свойствами

Дерево Фенвика (Binary indexed tree)



Есть другие вариации дерева Фенвика с другими операциями, но аналогичными свойствами

Визуализация для них тоже отличается

Дерево Фенвика (Binary indexed tree)

Почему оно binary indexed tree?

Дерево Фенвика (Binary indexed tree)

Почему оно binary indexed tree?

Потому, что вершины связаны друг с другом через бинарное представление индексов. Так не только в этом дереве, но часто именно Фенвика так называют (в западной литературе)

Дерево Фенвика

Почему оно Фенвика?

Дерево Фенвика

Почему оно Фенвика?

- Впервые описано Борисом Рябко в 1989 году

Дерево Фенвика

Почему оно Фенвика?

- Впервые описано Борисом Рябко в 1989 году
- Позже описано Питером Фенвиком в 1994 году, именно после его статьи это название получило распространение

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

Введем две функции:

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Заметим, что $\forall x \geq 0 : f(x) \leq x$ (конъюнкция только съедает биты)

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

```
def getPrefixSum(pos: int) -> int:
  ans = 0
  current_pos = pos
  while current_pos >= 0:
    ans += S_{current_pos}
    current_pos = f(current_pos) - 1
  return ans
```

```
def sum(l, r: int) -> int:
  return getPrefixSum(r) -
         getPrefixSum(l - 1)
```

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

А как насчитать?

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

А как насчитать?

1) Можно изначально предположить, что массив содержит нули, и N раз позвать update(pos, val). Сложность?

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

А как насчитать?

1) Можно изначально предположить, что массив содержит нули, и N раз позвать update(pos, val). За O(NlogN)

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum\limits_{j=f(i)}^{i} a_j$

А как насчитать?

1) Можно изначально предположить, что массив содержит нули, и N раз позвать update(pos, val). За O(NlogN)

2) А можно воспользоваться префиксными суммами!

Prefix sum

```
def buildPrefixes(n: int, arr: int[]) -> int[]:
  prefixes = int[n]

  currentSum = 0
  for i in 0..n-1:
    currentSum += arr[i]
    prefixes[i] = currentSum

  return prefixes
```

Работает за O(N) времени и O(N) памяти

Препроцессинг запустили всего один раз, а запросов может быть много!

```
def sum(l, r: int, prefixes: int[]) -> int:
  return prefixes[r] - prefixes[l - 1]
```

Работает за O(1)

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

А как насчитать?

1) Можно изначально предположить, что массив содержит нули, и N раз позвать update(pos, val). За O(NlogN)

2) А можно воспользоваться префиксными суммами!

$$S_i = prefixes[i] - prefixes[f(i) - 1]$$

Дерево Фенвика: построение

a_0, a_1, a_2, …, a_{n-1}

…

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

А как насчитать?

1) Можно изначально предположить, что массив содержит нули, и N раз позвать update(pos, val). За O(NlogN)

2) А можно воспользоваться префиксными суммами!

   $S_i = prefixes[i] - prefixes[f(i) - 1]$

   Сначала считаем префиксы за O(N), потом S за O(N)

Дерево Фенвика: многомерный случай

Дерево Фенвика: многомерный случай

sum(3, 5) = ?

46 11 40 [8 2 42] 65 10

Дерево Фенвика: многомерный случай

sum(3, 5) = ?

46 11 40 [8 2 42] 65 10

sum(0, 2) = 97

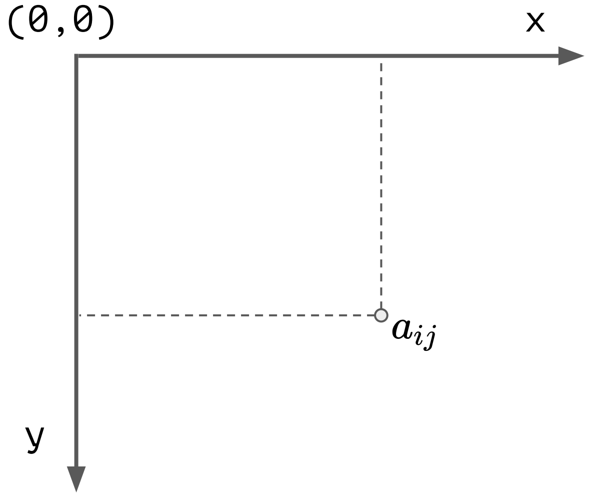
$\overbrace{46\ 11\ 40}\ 8\ 2\ 42\ 65\ 10$

$\underbrace{46\ 11\ 40\ 8\ 2\ 42}\ 65\ 10$

sum(0, 5) = 149

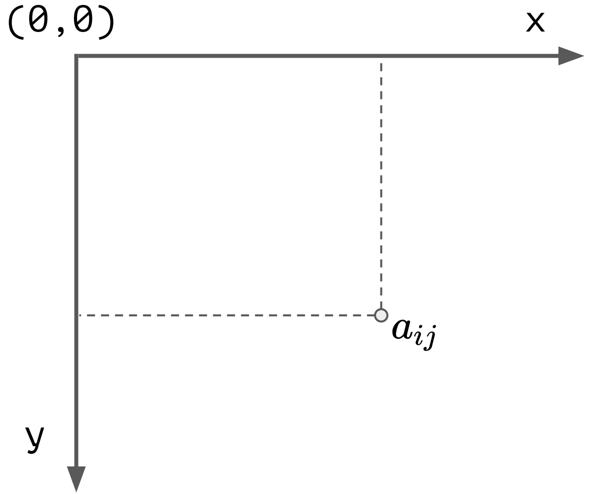
sum(3, 5) = 149 - 97 = 52

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Дерево Фенвика: многомерный случай

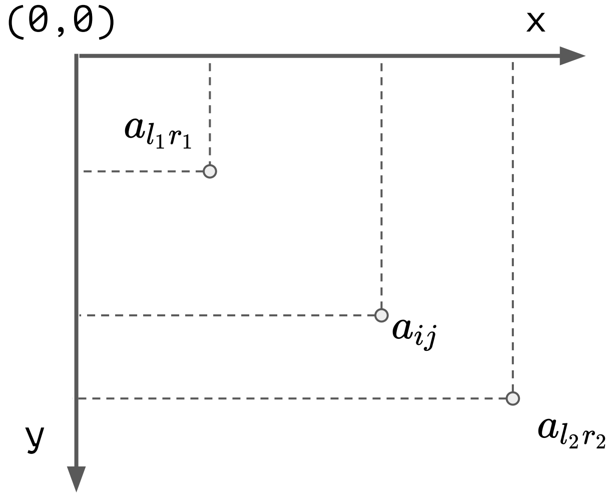


Пусть теперь есть матрица элементов.

Задача:

1) update(i, j, val)

Дерево Фенвика: многомерный случай

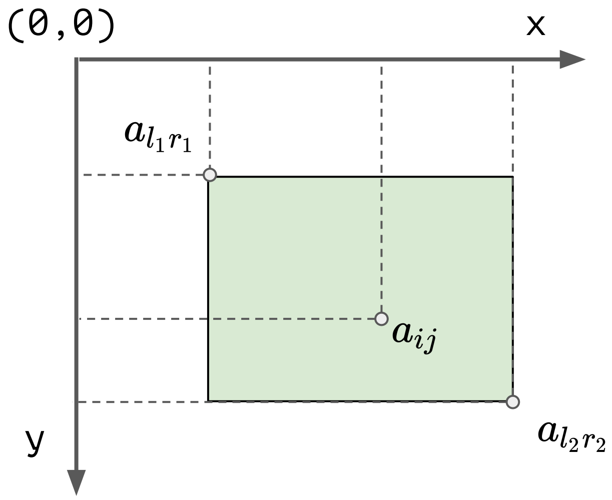


Пусть теперь есть матрица элементов.

Задача:

1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Дерево Фенвика: многомерный случай



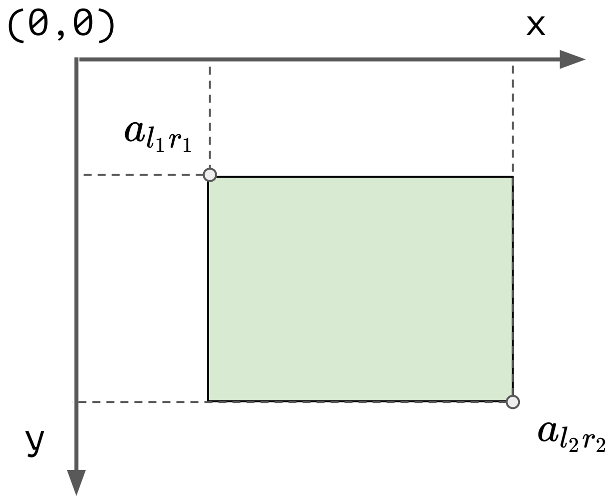
Пусть теперь есть матрица элементов.

Задача:

1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.

Дерево Фенвика: многомерный случай



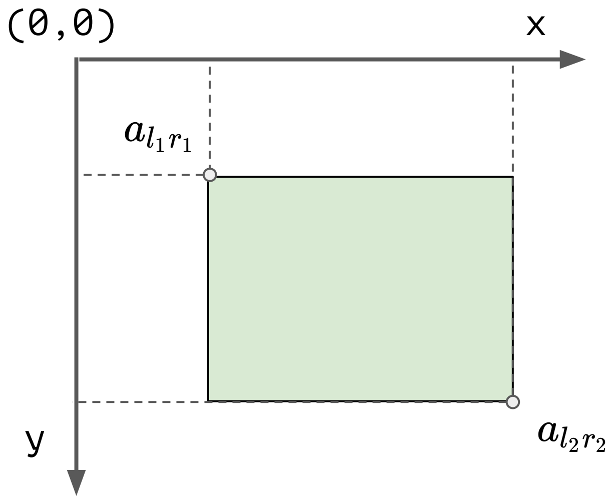
Пусть теперь есть матрица элементов.

Задача:

1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

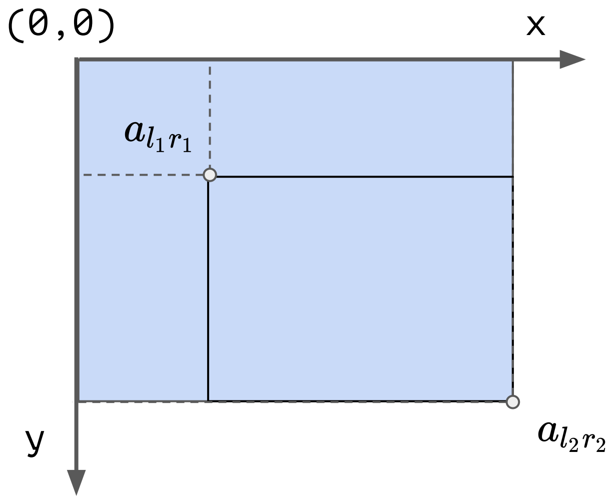
Задача:

1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

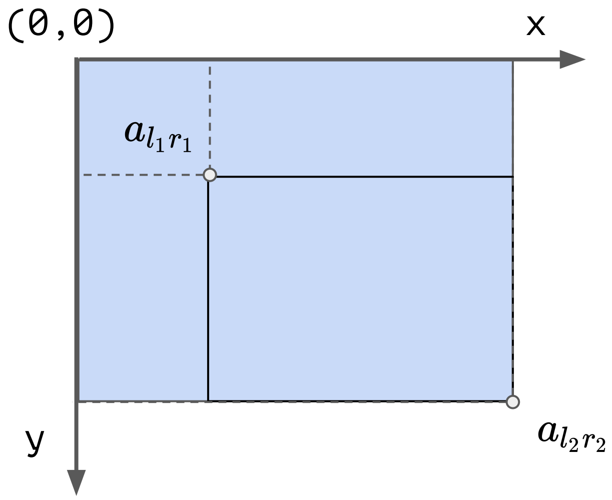
Задача:

1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Задача:

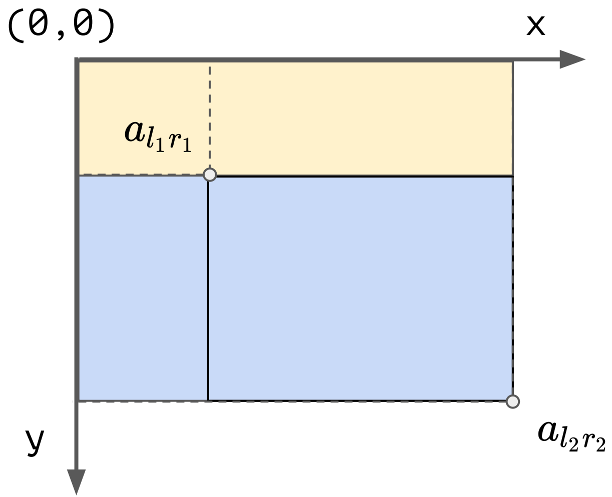
1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

sum(0, 0, l2, r2) -

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Задача:

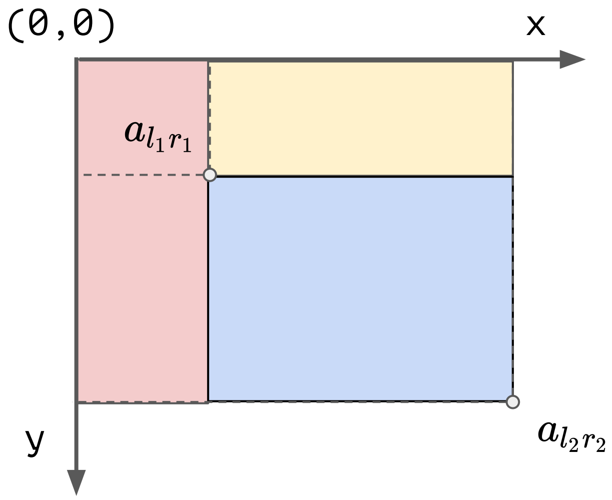
1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

sum(0, 0, l2, r2) - sum(0, 0, l2, r1 - 1) -

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Задача:

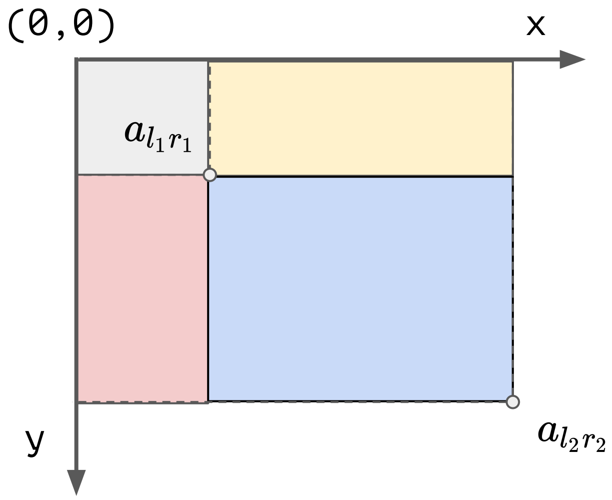
1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

```
sum(0, 0, l2, r2) - sum(0, 0, l2, r1 - 1) - sum(0, 0, l1 - 1, r2)
```

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Задача:

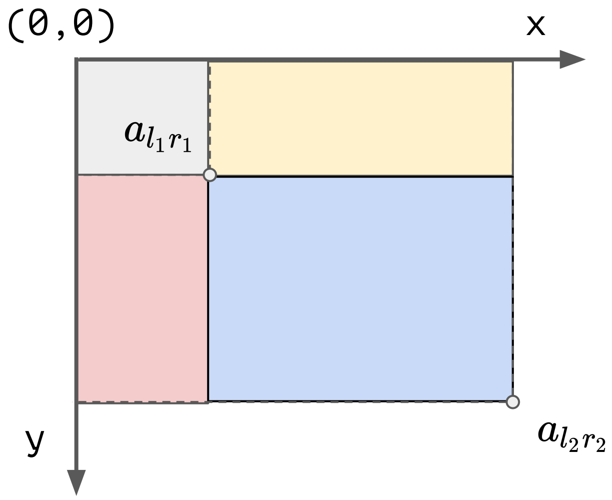
1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

Префиксными прямоугольниками назовем те, у которых левый верхний угол в (0, 0).

```
sum(0, 0, l2, r2) - sum(0, 0, l2, r1 - 1) - sum(0, 0, l1 - 1, r2)
+ sum(0, 0, l1 - 1, r1 - 1)
```

Дерево Фенвика: многомерный случай



Пусть теперь есть матрица элементов.

Задача:

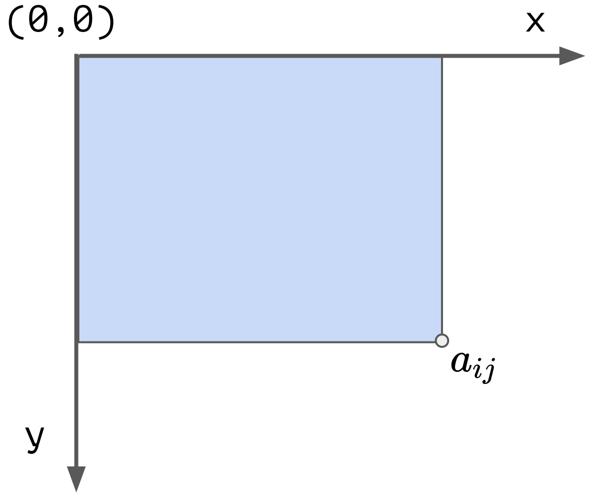
1) update(i, j, val)
2) sum(l1, r1, l2, r2)

Считаем сумму на прямоугольнике.
Как свести к префиксам?

```
sum(0, 0, l2, r2) - sum(0, 0, l2, r1 - 1) - sum(0, 0, l1 - 1, r2)
+ sum(0, 0, l1 - 1, r1 - 1)
```

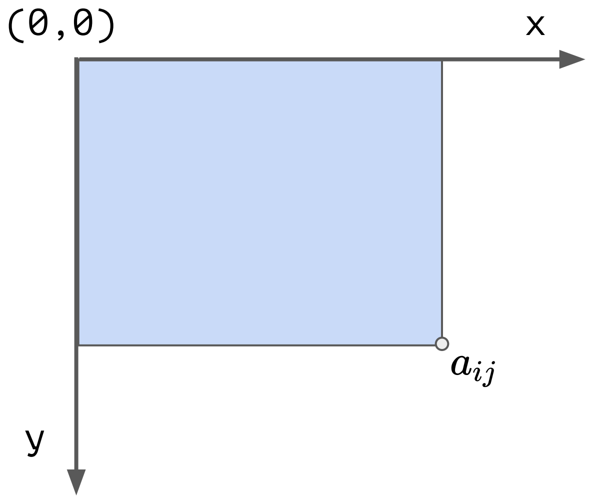
Сумма элементов любого прямоугольника сводится к комбинации суммы префиксных.

Дерево Фенвика: многомерный случай




```
sumPrefix(i, j) = ?
```

Дерево Фенвика: многомерный случай



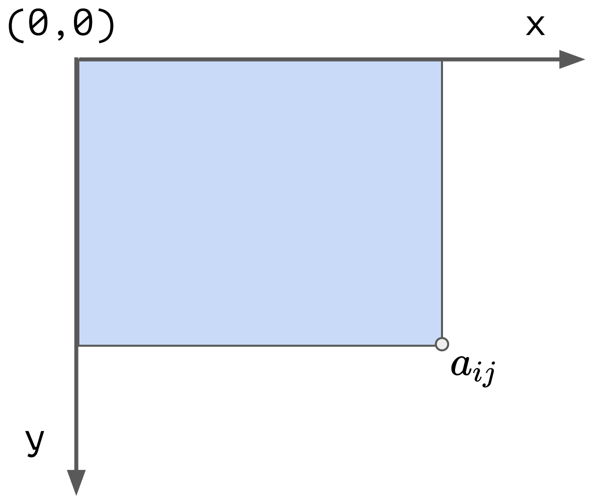
`sumPrefix(i, j) = ?`

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

$$S_{ij}$$

Дерево Фенвика: многомерный случай



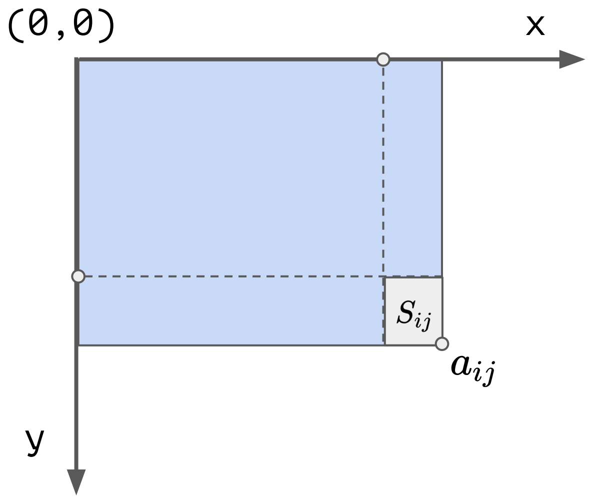
sumPrefix(i, j) = ?

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



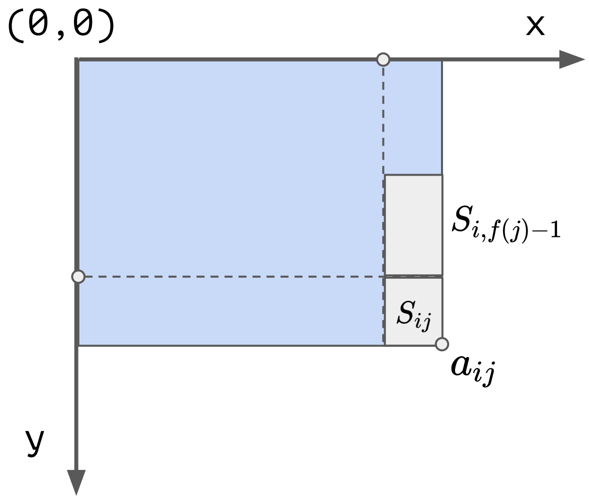
sumPrefix(i, j) = ?

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



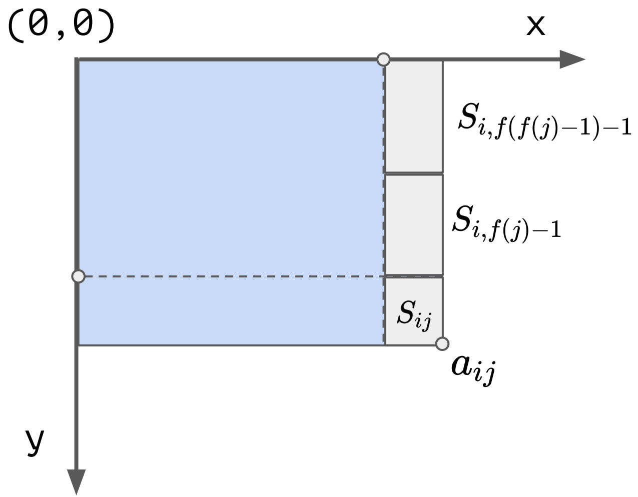
sumPrefix(i, j) = ?

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



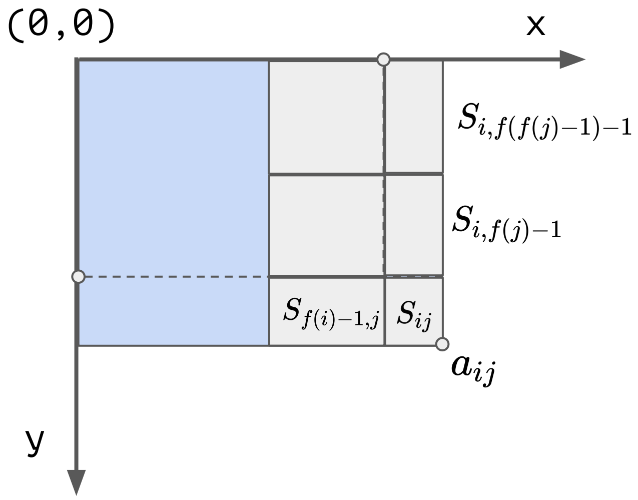
sumPrefix(i, j) = ?

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x,current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



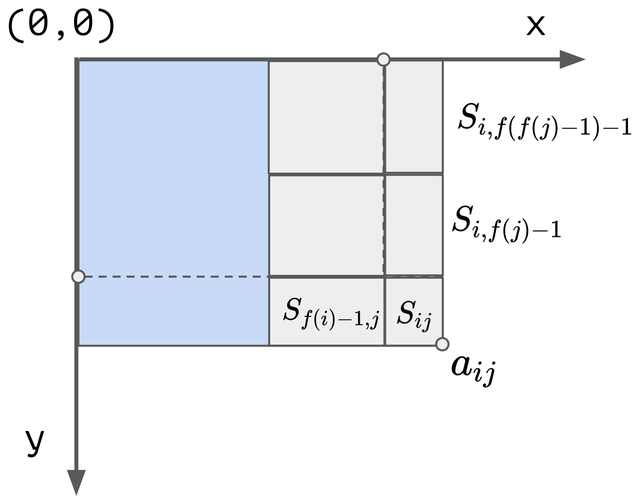

```
sumPrefix(i, j) = ?
```

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



Сложность?

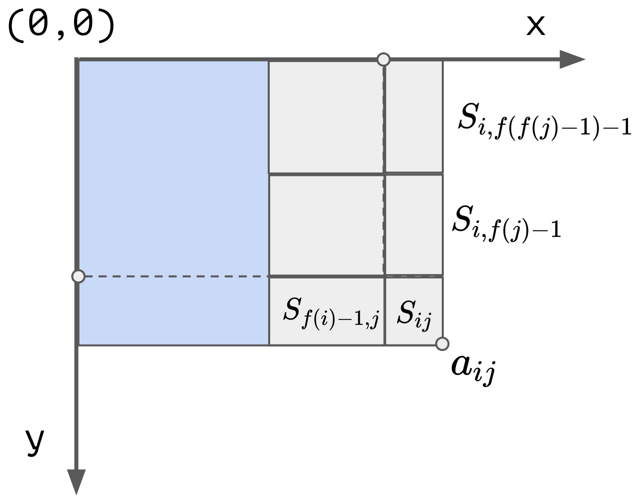
sumPrefix(i, j) = ?

Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

Дерево Фенвика: многомерный случай



sumPrefix(i, j) = ?

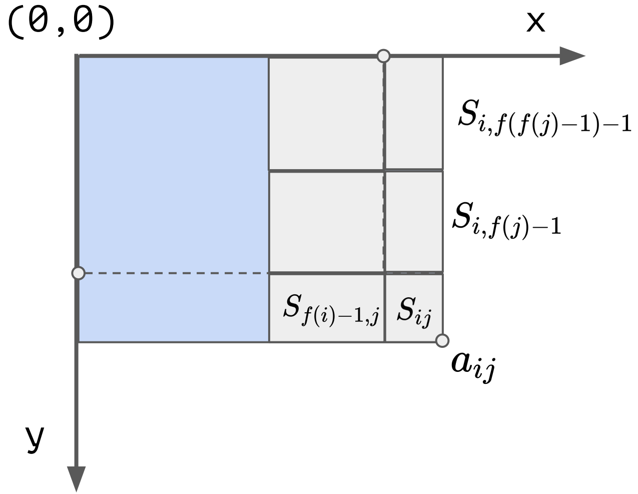
Повторим трюк с Фенвиком! Насчитываем:

$$S_{ij} = \sum_{v=f(i)}^{i} \sum_{u=f(j)}^{j} a_{vu}$$

```
def getPrefixSum(i, j: int) -> int:
  ans = 0, current_x = i
  while current_x >= 0:
    current_y = j
    while current_y >= 0:
      ans += S_{current_x, current_y}
      current_y = f(current_y) - 1
    current_x = f(current_x) - 1
  return ans
```

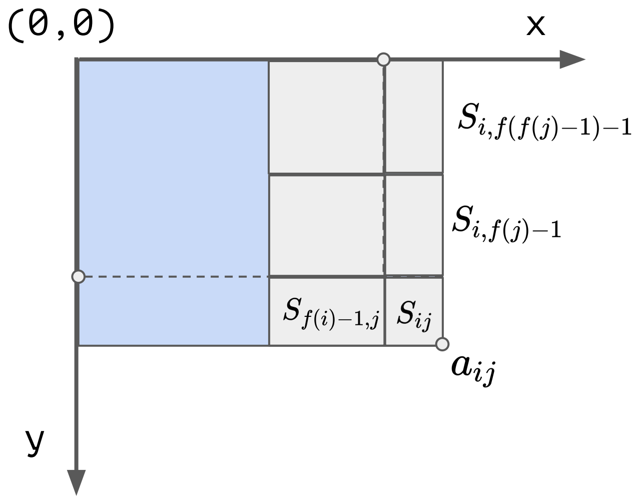
Сложность?
Если матрица квадратная,
то O(logN*logN)

Дерево Фенвика: многомерный случай



`update(x, y, val) = ?`

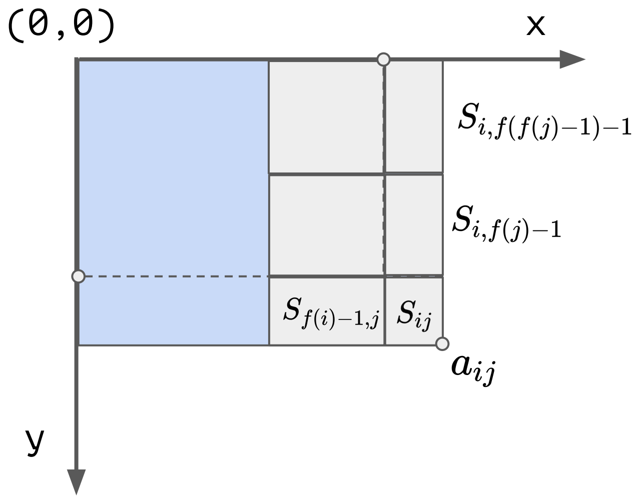
Дерево Фенвика: многомерный случай




```
update(x, y, val) = ?
```

```
def increment(x, y, val: int) -> int:
    i = x
    while i < n:
      j = y
        while j < n:
          S_{i,j} += val
          j = g(j)
     i = g(i)
```

Дерево Фенвика: многомерный случай

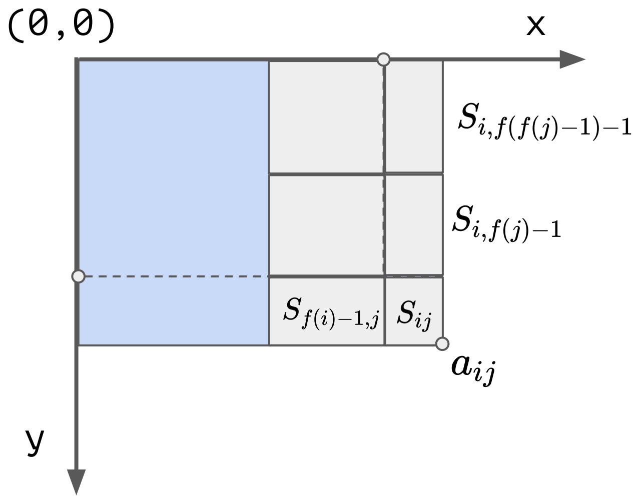



```
update(x, y, val) = ?
```

```
def increment(x, y, val: int) -> int:
    i = x
    while i < n:
      j = y
        while j < n:
          S_{i,j} += val
          j = g(j)
     i = g(i)
```

```
def update(x, y, val: int) -> int:
    increment(x, y, val - a[x][y])
```

Дерево Фенвика: многомерный случай



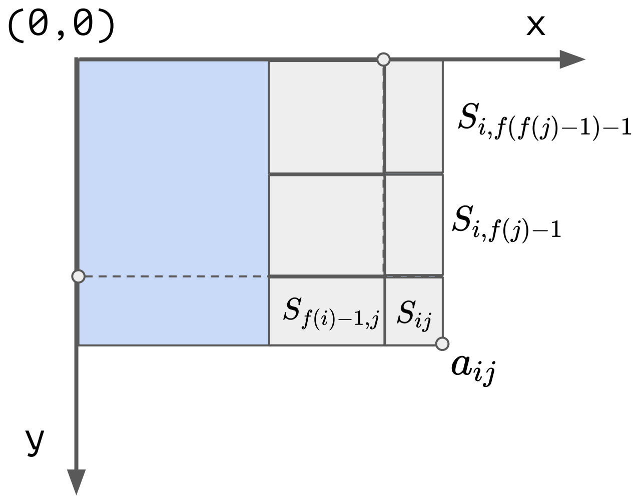
Сложность: O(logN*logN)

update(x, y, val) = ?

```
def increment(x, y, val: int) -> int:
    i = x
    while i < n:
      j = y
        while j < n:
          S_{i,j} += val
          j = g(j)
     i = g(i)
```

```
def update(x, y, val: int) -> int:
    increment(x, y, val - a[x][y])
```

Дерево Фенвика: многомерный случай




```
update(x, y, val) = ?
```

```
def increment(x, y, val: int) -> int:
    i = x
    while i < n:
      j = y
        while j < n:
          S_{i,j} += val
          j = g(j)
     i = g(i)
```

Сложность: O(logN*logN)

```
def update(x, y, val: int) -> int:
    increment(x, y, val - a[x][y])
```

Аналогично обобщается на большие размерности 🎉

Дерево Фенвика: другие операции

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

max(0, 3) = 46

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

max(0, 3) = 46
max(0, 5) = 46

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

max(0, 3) = 46
max(0, 5) = 46
max(3, 5) = ???

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

max(0, 3) = 46
max(0, 5) = 46
max(3, 5) = ???

А если не можем перейти к префиксным максимумам, то и наш трюк с хождением по f(i) не сработает!

Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы:

max(3, 5) = ?

46 11 40 [8 2 42] 65 10

max(0, 3) = 46
max(0, 5) = 46
max(3, 5) = ???

А если не можем перейти к префиксным максимумам, то и наш трюк с хождением по f(i) не сработает!

Здесь можно выкрутиться, используя специальную структуру данных "Обратное дерево Фенвика"

Обратное дерево Фенвика

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

$$a_0, a_1, a_2, \ldots, a_{n-1}$$

Необходимо поддержать новые операции следующего вида:

1. max(l, r) - найти максимум элементов на полуинтервале (a_l, a_r]

2. update(pos, val) - обновляем значение элемента a_pos

```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \sum_{j=f(i)}^{i} a_j$

a_0, a_1, a_2, …, a_{n-1}

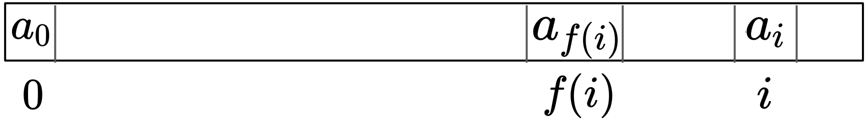
```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$

a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

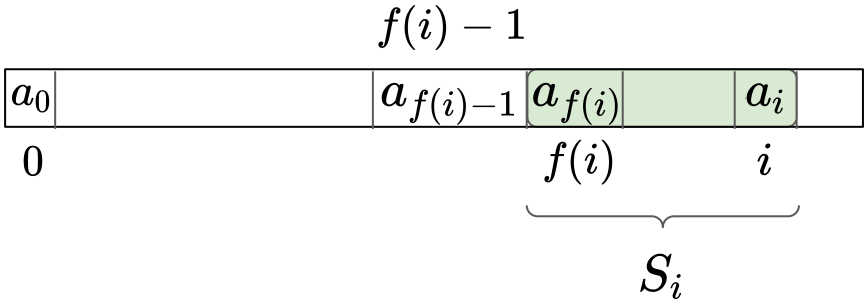
Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max_{j=f(i)}^{i} a_j$



a_0, a_1, a_2, …, a_{n-1}

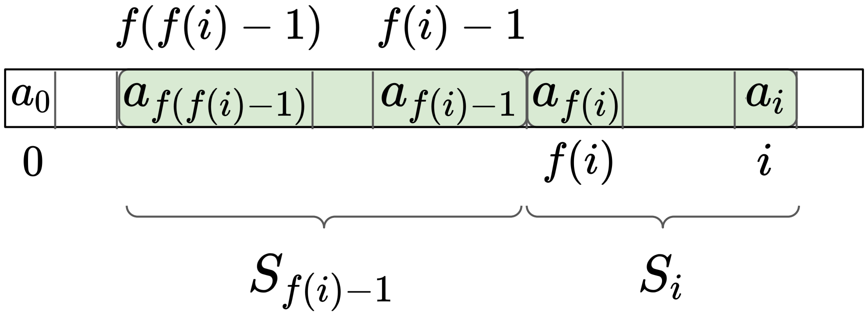
```
f(x) = x & (x + 1)        & - побитовое "и"
g(x) = x | (x + 1)        | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



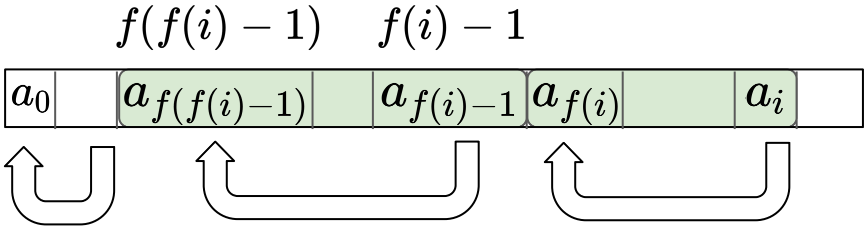
```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



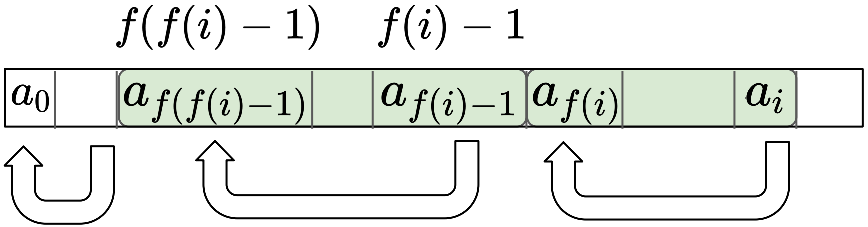
```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$

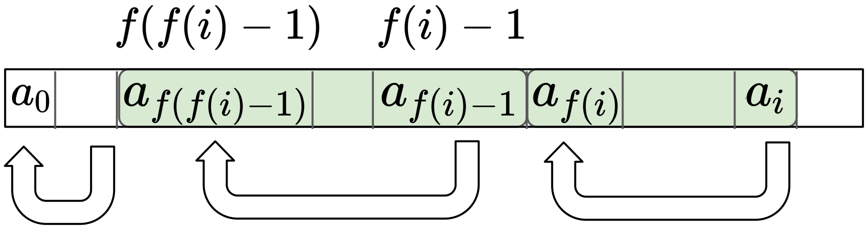


И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

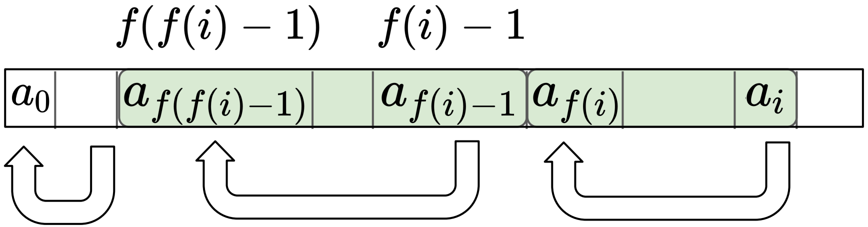
Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$

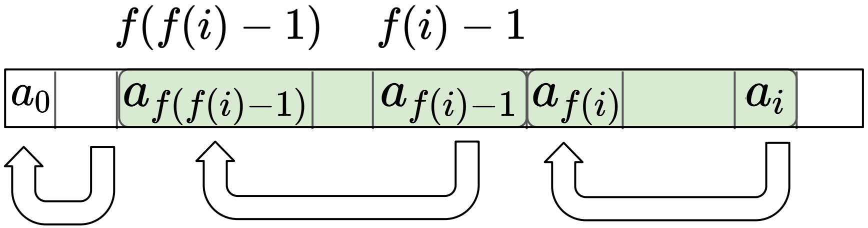


И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$

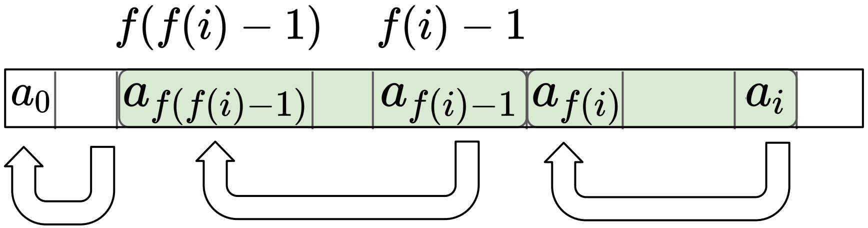


И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)   & - побитовое "и"
g(x) = x | (x + 1) | - побитовое "или"

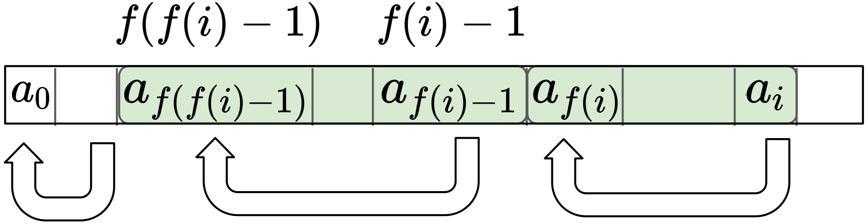
Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

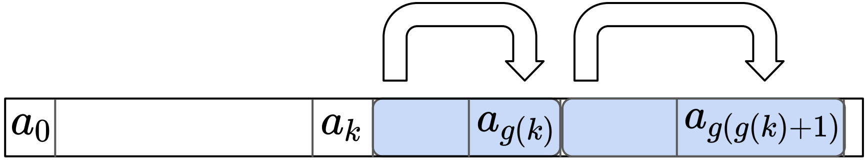
```
a_0, a_1, a_2, …, a_{n-1}

f(x) = x & (x + 1)          & - побитовое "и"
g(x) = x | (x + 1)          | - побитовое "или"
```

Тогда насчитаем: $\forall i \in [0, n-1] : S_i = \max\limits_{j=f(i)}^{i} a_j$



И еще насчитаем: $\forall i \in [0, n-1] : S'_i = \max\limits_{j=i+1}^{g(i)} a_j$

Т.е. идем в обратную сторону, для этого храним второе дерево (встречное)



Обратное дерево Фенвика

Задача: пусть есть массив фиксированного размера n. Пусть для простоты n - это степень двойки.

a_0, a_1, a_2, …, a_{n-1}

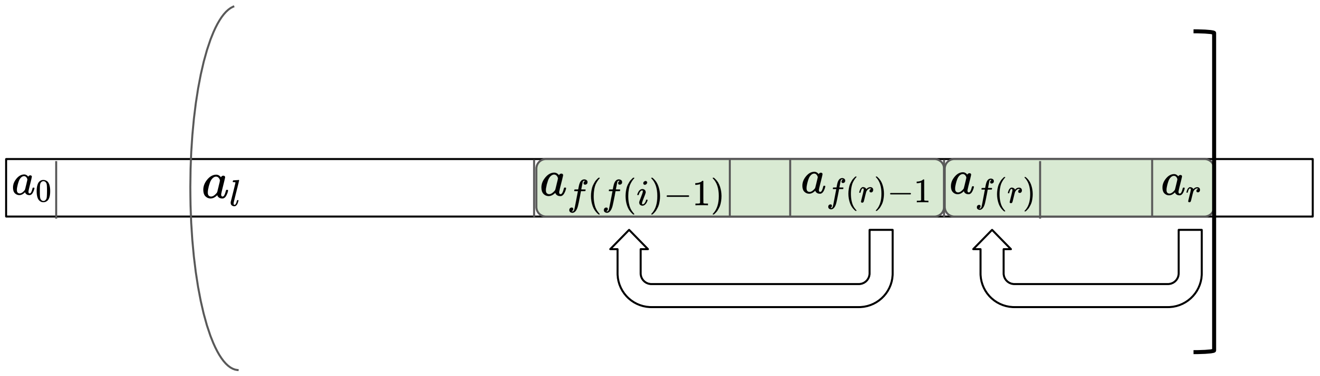
Необходимо поддержать новые операции следующего вида:

1. max(l, r) - найти максимум элементов на полуинтервале (a_l, a_r]
2. update(pos, val) - обновляем значение элемента a_pos

Обратное дерево Фенвика

Решение для max(l, r):

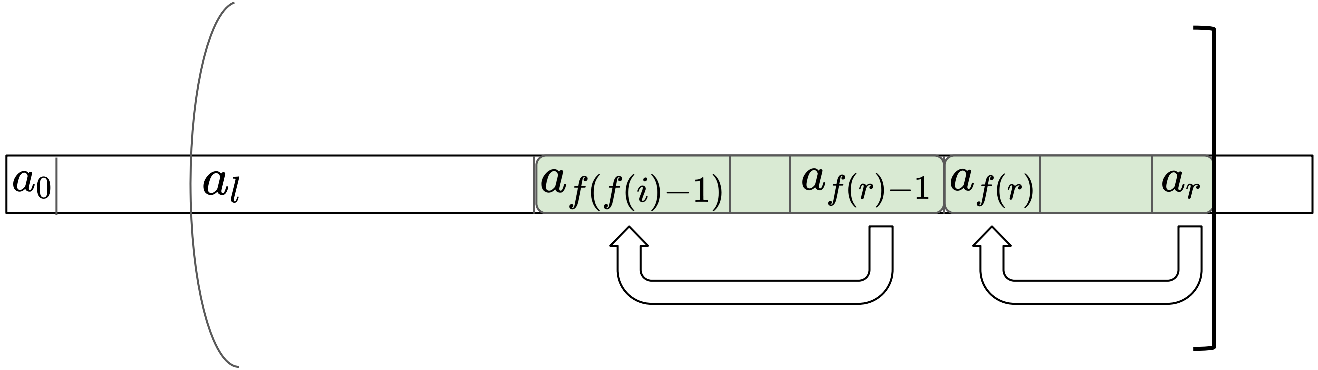
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума



Обратное дерево Фенвика

Решение для max(l, r):

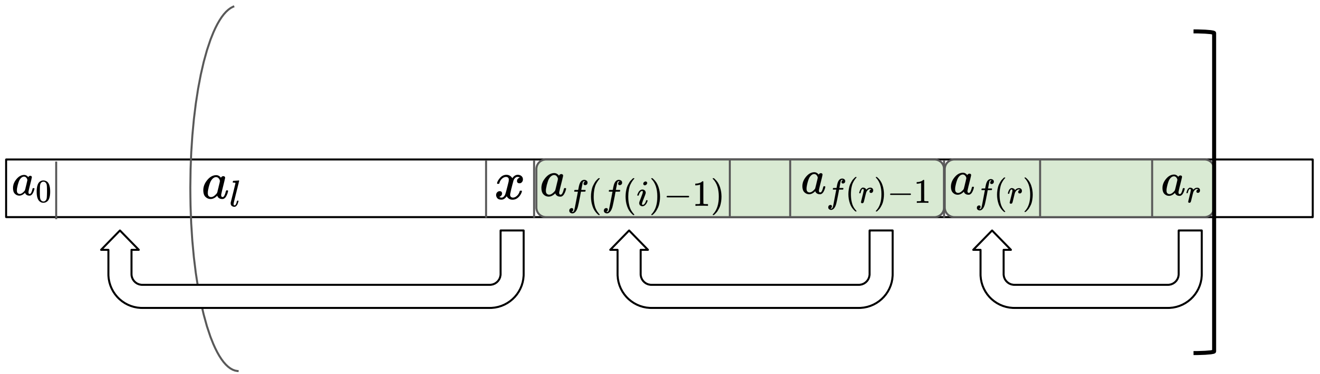
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l



Обратное дерево Фенвика

Решение для max(l, r):

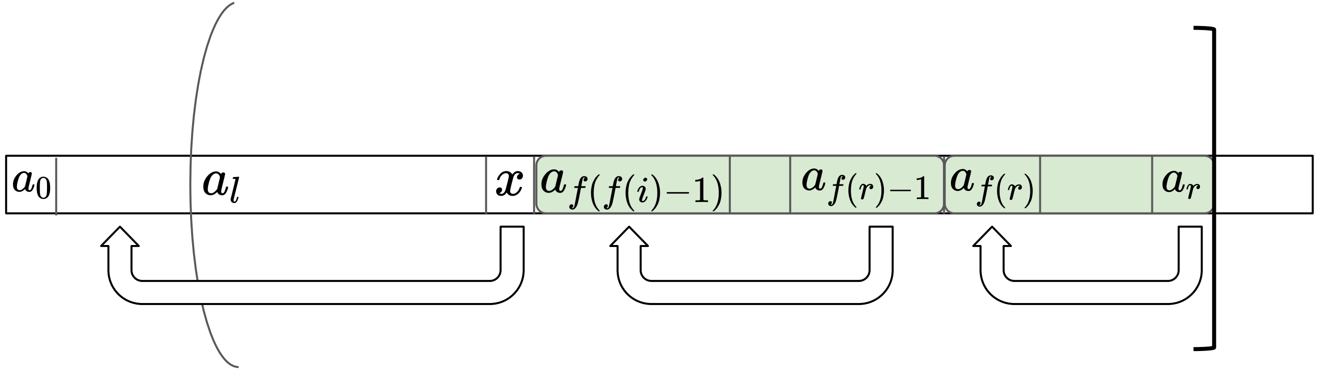
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l



Обратное дерево Фенвика

Решение для max(l, r):

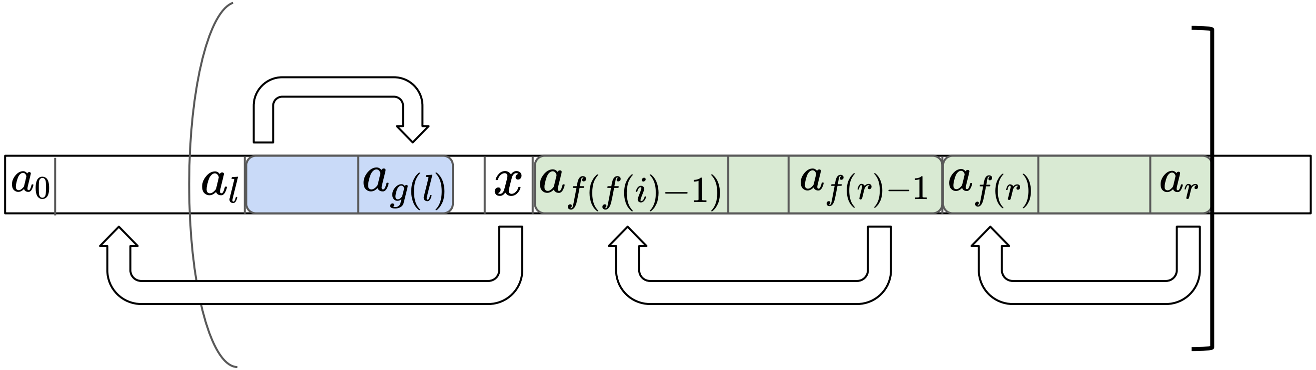
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l
3) Начинаем двигаться влево от a_l до x, используем S_i'



Обратное дерево Фенвика

Решение для max(l, r):

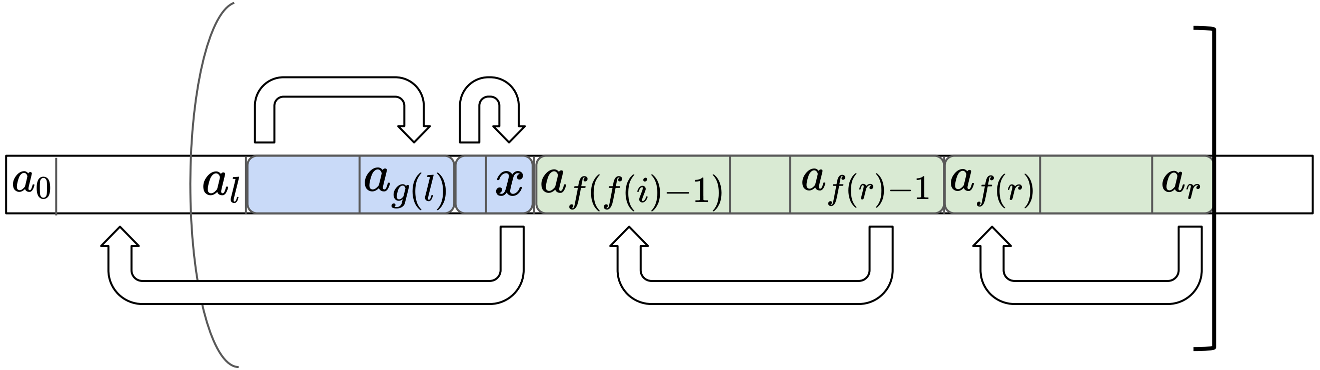
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l
3) Начинаем двигаться влево от a_l до x, используем S'_i



Обратное дерево Фенвика

Решение для max(l, r):

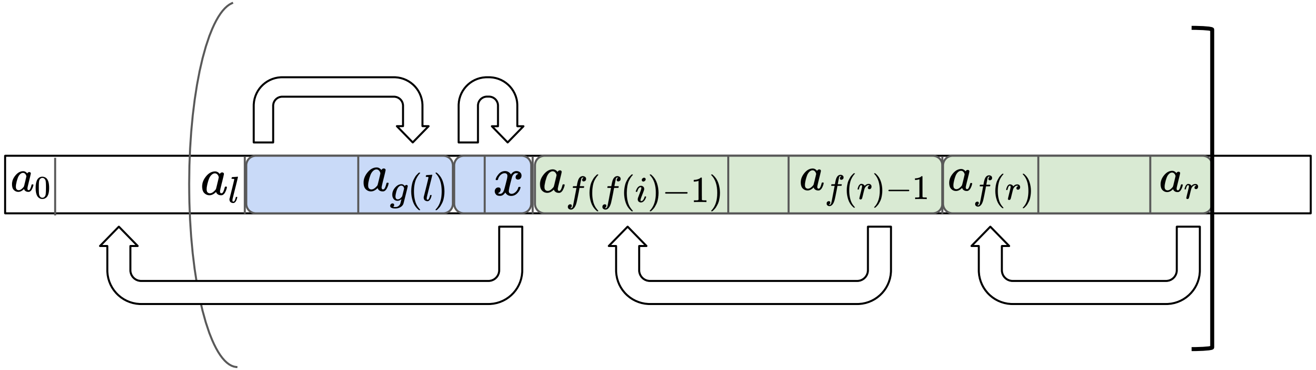
1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l
3) Начинаем двигаться влево от a_l до x, используем S_i'
4) Утверждается, что в какой-то момент мы дойдем до x



Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x

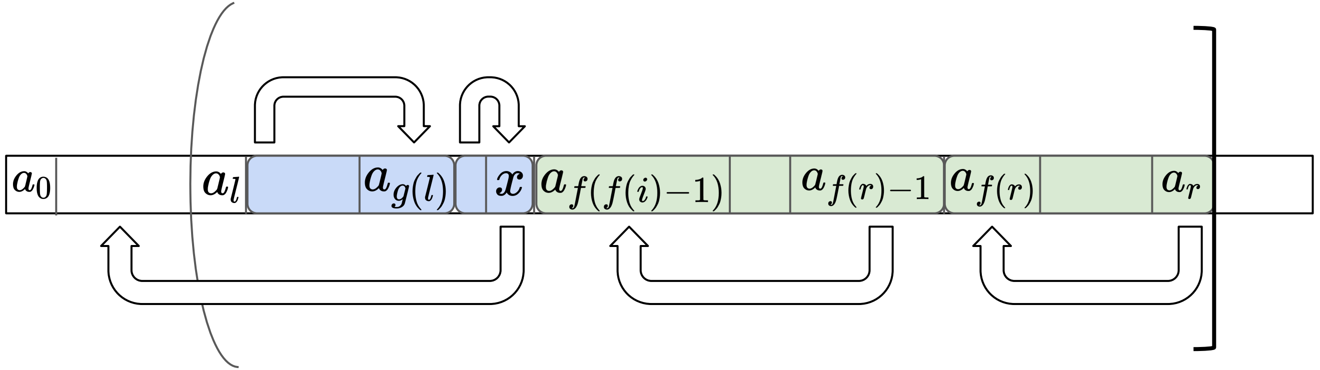


Обратное дерево Фенвика

Решение для `max(l, r)`:

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$



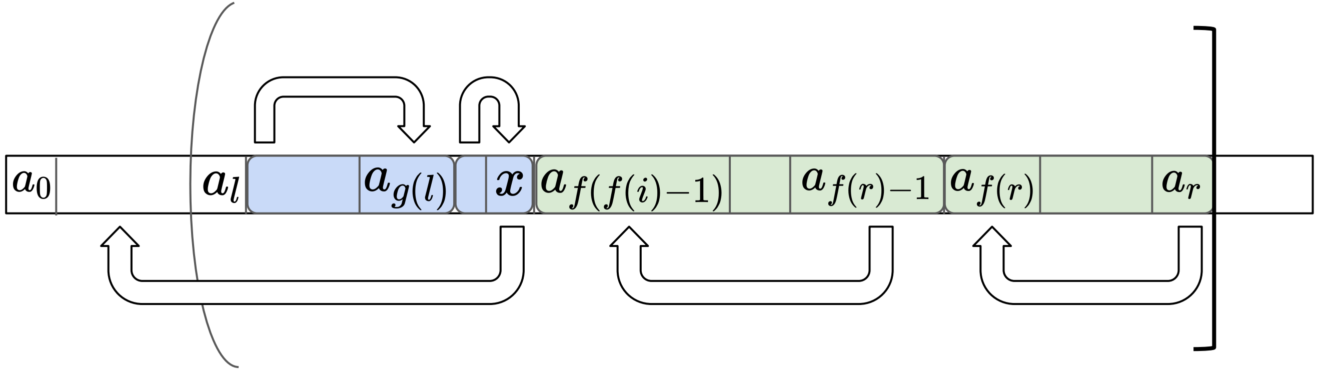
Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \leq a_l \end{cases} \Rightarrow g(a_l) \leq x$$

```
a_l   = 1…0101…0…
x     = 1…0101…1…
```



Обратное дерево Фенвика

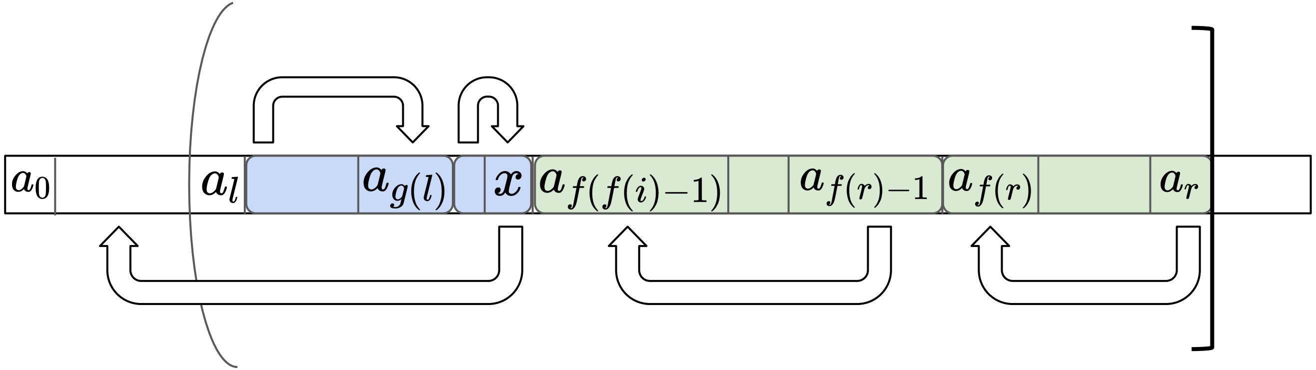
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l   = 1…0101… 0…. . .
x     = 1…0101… 1…011
```

Первая позиция (слева), в которой x строго больше, т.е. содержит 1, а a_l - 0



Обратное дерево Фенвика

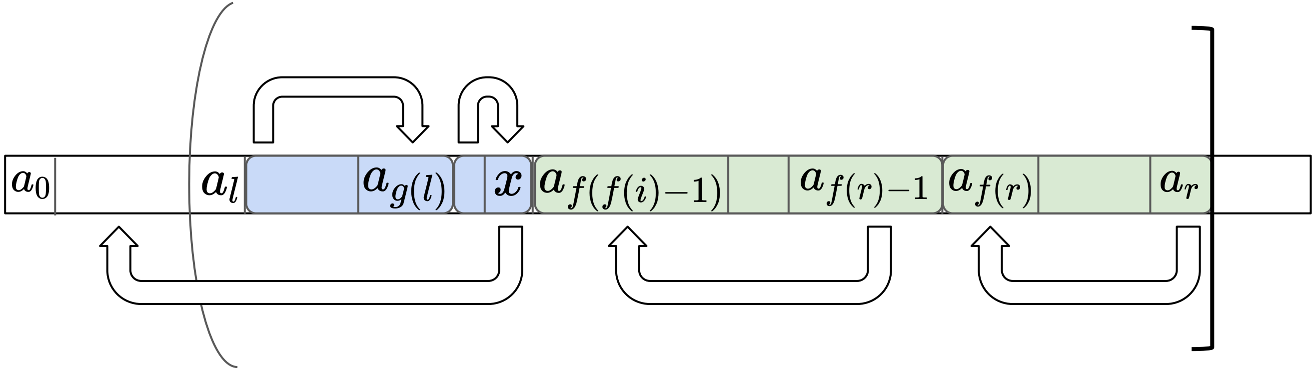
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \leq a_l \end{cases} \Rightarrow g(a_l) \leq x$$

```
a_l  = 1…0101… 0… . . .
x    = 1…0101… 1…011
f(x) = 1…0101… 1…000
```

По определению f(x) зануляет последнюю группу единиц.



Обратное дерево Фенвика

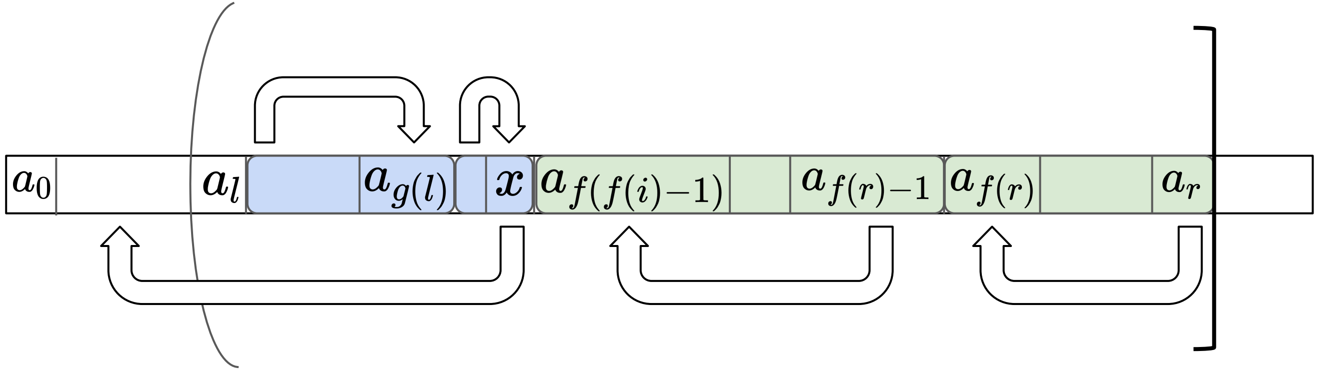
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l  = 1…0101… 0…. . .
x    = 1…0101… 1…011
f(x) = 1…0101… 1…000
```

По определению f(x) зануляет последнюю группу единиц. Но раз он остается меньше либо равен a_l =>



Обратное дерево Фенвика

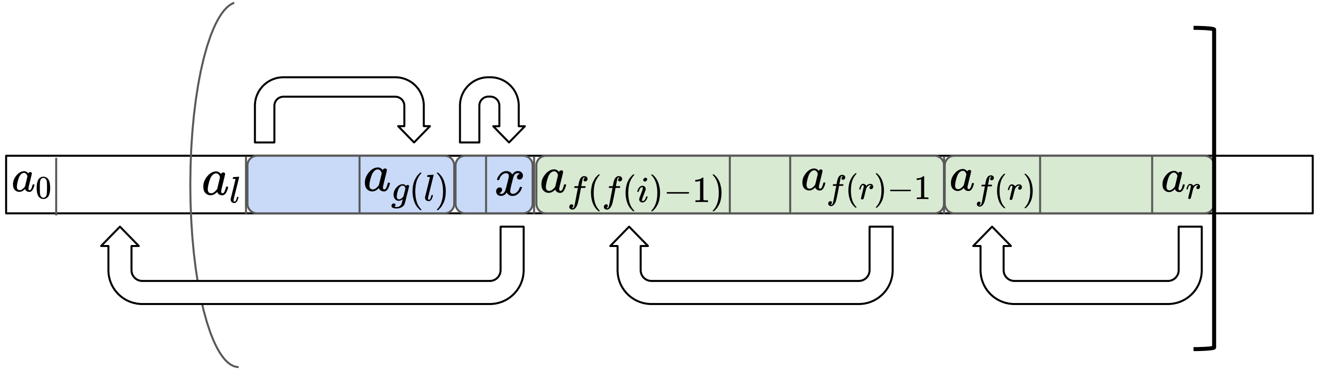
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \leq a_l \end{cases} \Rightarrow g(a_l) \leq x$$

```
a_l  = 1…0101…0…...
x    = 1…0101…11111
f(x) = 1…0101…00000
```

По определению f(x) зануляет последнюю группу единиц. Но раз он остается меньше либо равен a_l => то весь его хвост из 1!



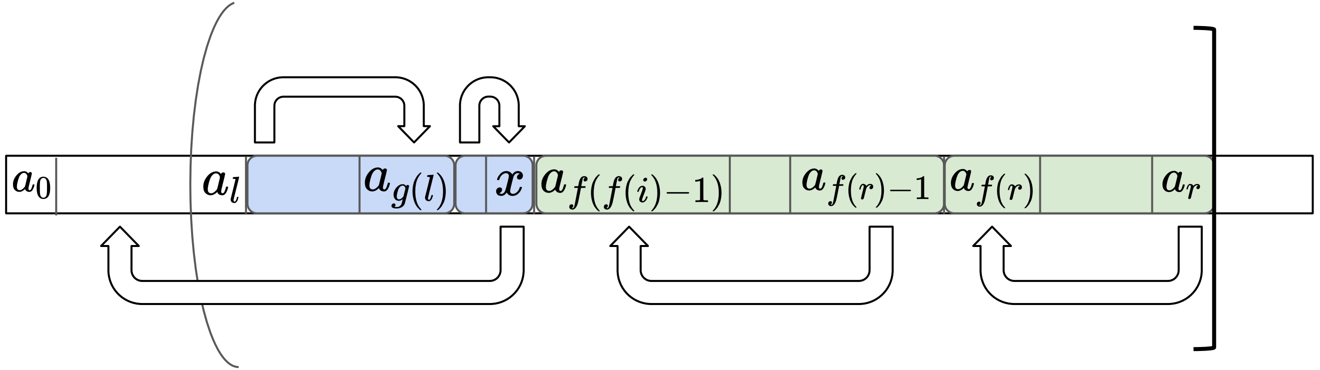
Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l    = 1…0101…0…011
g(a_l) =
x      = 1…0101…11111
```



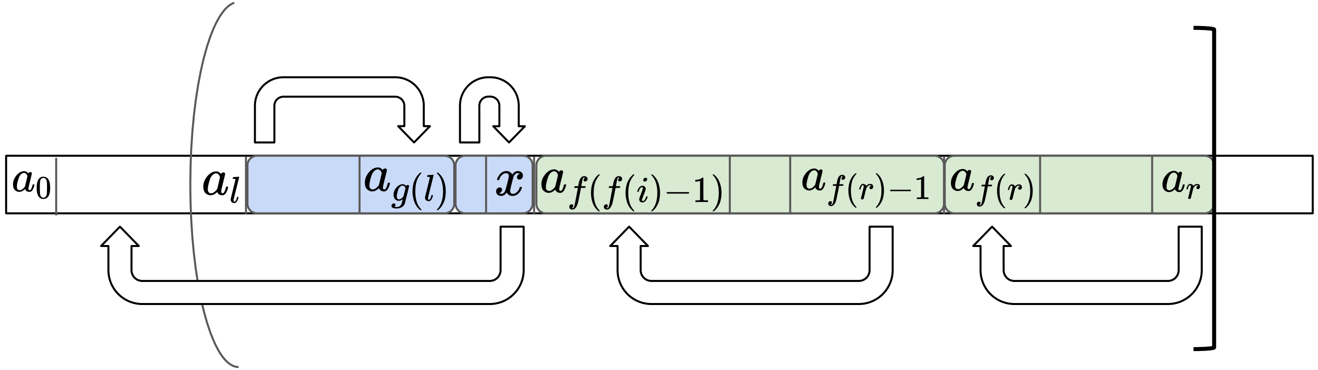
Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l    = 1…0101…0…011
g(a_l) = 1…0101…
x      = 1…0101…11111
```



Обратное дерево Фенвика

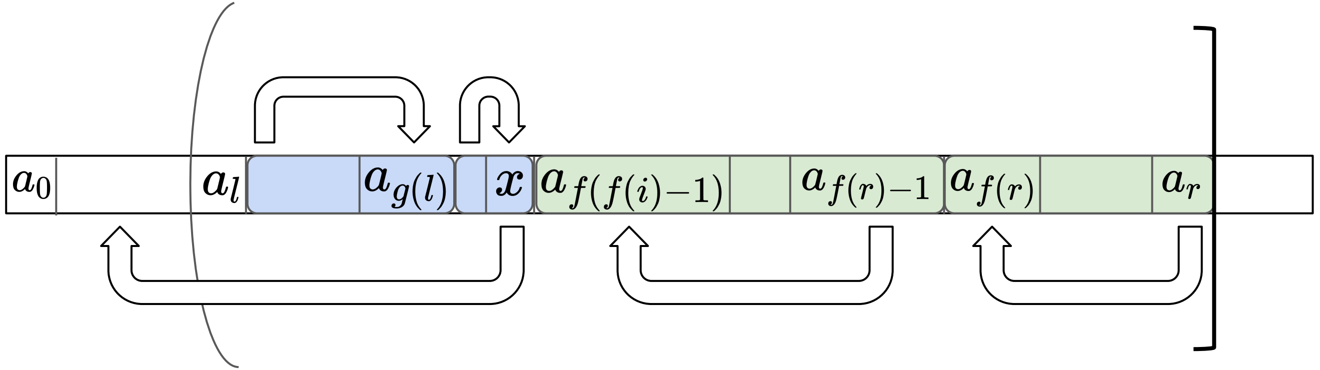
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l    = 1…0101…0…011
g(a_l) = 1…0101…0…111
x      = 1…0101…11111
```

Поставили 1 на место первого справа нуля.



Обратное дерево Фенвика

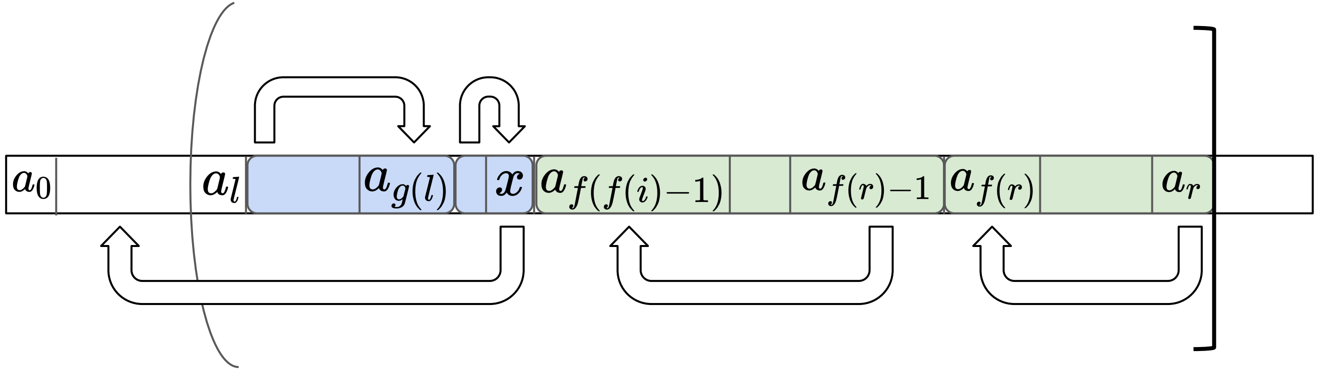
Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

```
a_l    = 1…0101…01111
g(a_l) = 1…0101…11111
x      = 1…0101…11111
```

Поставили 1 на место первого справа нуля.
В крайнем случае вот так:
g(a_l) == x



Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \le a_l \end{cases} \Rightarrow g(a_l) \le x$$

Тогда, если с первого же прыжка вправо попали в x => утверждение верно.

Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

$$\begin{cases} x > a_l \\ f(x) \leq a_l \end{cases} \Rightarrow g(a_l) \leq x$$

Тогда, если с первого же прыжка вправо попали в x => утверждение верно. Если нет, возьмем g(a_l) + 1 в качестве новой левой границы и повторим рассуждение (f(x), конечно, окажется меньше нее).

Обратное дерево Фенвика

Решение для max(l, r):

Утверждается, что в какой-то момент мы дойдем обратным ходом до x. Действительно, покажем, что:

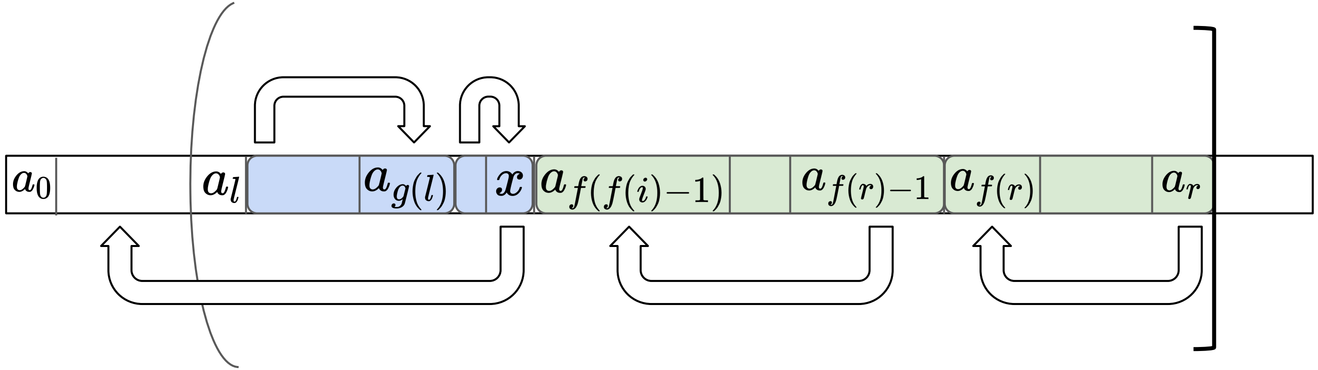
$$\begin{cases} x > a_l \\ f(x) \leq a_l \end{cases} \Rightarrow g(a_l) \leq x$$

Тогда, если с первого же прыжка вправо попали в x => утверждение верно. Если нет, возьмем g(a_l) + 1 в качестве новой левой границы и повторим рассуждение (f(x), конечно, окажется меньше нее). Повторять можно конечно число раз, т. к. g(x) за log доведет до всех 1 => утверждение верно.

Обратное дерево Фенвика

Решение для max(l, r):

1) Двигаемся по отрезку влево, начиная с r, используем S_i для поиска максимума
2) Останавливаемся, когда следующий прыжок перейдет через a_l
3) Начинаем двигаться влево от a_l до x, используем S_i'
4) Утверждается, что в какой-то момент мы дойдем до x $\square$



Дерево Фенвика: другие операции

Без проблем (почти) обобщается до умножения вместо суммы, где вместо вычитания будем делать префиксные перемножения (только учтем ноль).

Но вот уже с максимумом и минимумом на отрезки проблемы, которые можно решить обратным деревом Фенвика.

С другими операциями все еще хуже: например, как считать gcd непонятно, etc.

Дерево Фенвика vs Дерево отрезков

А зачем это все?
Есть ли плюсы по сравнению с деревом отрезков?

Дерево Фенвика vs Дерево отрезков

А зачем это все?
Есть ли плюсы по сравнению с деревом отрезков? Есть!

- ➕ Занимает меньше памяти
- ➕ Быстрее работает (лучше константы)
- ➕ Быстрее пишется

Дерево Фенвика vs Дерево отрезков

А зачем это все?
Есть ли плюсы по сравнению с деревом отрезков? Есть!

- ➕ Занимает меньше памяти
- ➕ Быстрее работает (лучше константы)
- ➕ Быстрее пишется
- ➕ Очень легко обобщается на большие размерности 🎉

Дерево Фенвика vs Дерево отрезков

А зачем это все?
Есть ли плюсы по сравнению с деревом отрезков? Есть!

- ➕ Занимает меньше памяти
- ➕ Быстрее работает (лучше константы)
- ➕ Быстрее пишется
- ➕ Очень легко обобщается на большие размерности 🎉

Минусы:

- ➖ Значительно меньше задач решает 😔
- ➖ Менее интуитивно

Мини-задача #45 (1 балл)

https://leetcode.com/problems/create-sorted-array-through-instructions/

Решите задачу, используя дерево Фенвика.

Takeaways

- Дерево Фенвика, как быстрая, компактная и короткая в записи альтернатива деревьям отрезков.
- Отлично обобщается на большие размерности.
- Плохо обобщается на другие запросы кроме суммы.