Segtree

snippet/Segtree[*1] は ACL[*2] の Segtree を移植したものである（ただし、min—max_left—right は未実装）。
Segtree はモノイドに対して使用できるデータ構造である。モノイドとは、以下を満たす代数構造である。

- 任意の $a, b, c \in S$ に対して $(a \cdot b) \cdot c \ = a \cdot (b \cdot c)$ が成り立つ （結合律）
- 任意の $a \in S$ に対して、$a \cdot e = e \cdot a = a$ を満たすような $e \in S$ が存在する（単位元の存在）

長さ N の列に対し、要素の 1 点変更、および、区間の要素の総積の取得をいずれも $\mathrm{O}(\log N)$ で行うことができる。（`op`, `e` がいずれも定数時間の場合）

```
data (PrimMonad m, Unbox a, Show a) => Segtree m a
```

Segtree は mutable なデータ構造で、IO または ST モナド上で用いることができる。要素は Unboxed な値をとることが可能（内部構造として Data.Vector.Unboxed.Mutable[*3] を利用）である。ライブラリのインタフェース上は、モノイドであることを明には要求していない。

コンストラクタ (1)

コンストラクタは 2 種類ある。ひとつめは、長さ `n` を指定して生成するものである。

```
newSEGT :: (PrimMonad m, Unbox a, Show a) => (a -> a -> a) -> a -> Int -> m (Segtree m a)
```

`newSEGT op e n` のように、二項演算 `op :: a -> a -> a`、単位元 `e :: a`　および要素数 `n :: Int` を指定する。これにより、長さ `n` の数列 $\{a_i\}$ が作られ、その初期値はすべて `e` となる。

制約

- $0 \leq n \leq 10^8$

計算量

- $\mathrm{O}(n)$

コンストラクタ (2)

ふたつめのコンストラクタは、初期値をベクターで与えるもの。

[*1] https://github.com/unnohideyuki/AtHaskell/blob/master/snippets/Segtree.hs
[*2] https://atcoder.jp/posts/517
[*3] http://hackage.haskell.org/package/vector-0.12.1.2/docs/Data-Vector-Unboxed-Mutable.html#t:Unbox

```
fromVecSEGT :: (PrimMonad m, Unbox a, Show a) => (a -> a -> a) -> a -> (V.Vector a) -> m (Segtree m
```

`fromVecSEGT op e v` のように、二項演算 `op :: a -> a -> a`、単位元 `e :: a`　および初期値ベクトル `v :: Data.Vector.Unboxed.Vector a` を指定する。これにより、`v` と同じ長さの数列 $\{a_i\}$ が作られ、`v` の内容が初期値となる。

制約

- $0 \leq n \leq 10^8$

計算量

- $\mathrm{O}(n)$

set

数列 $\{a_i\}$ の 1 要素を変更する。

```
setSEGT :: (PrimMonad m, Unbox a, Show a) => (Segtree m a) -> Int -> a -> m ()
```

`setSEGT segtree p x` は、セグメント木 `segtree` において数列の 1 要素 a_p の値を `x` で置き換える。

制約

- $0 \leq p < n$

計算量

- $\mathrm{O}(\log n)$

get

数列の 1 要素の値を返す。

```
getSEGT :: (PrimMonad m, Unbox a, Show a) => (Segtree m a) -> Int -> m a
```

`getSEGT segtree p` は、セグメント木 `segtree` における数列の 1 要素 a_p の値を返す。

制約

- $0 \leq p < n$

計算量

- O(1)

prod

指定された区間 $[l, r)$ の要素の総積

```
prodSEGT :: (PrimMonad m, Unbox a, Show a) => (Segtree m a) -> Int -> Int -> m a
```

`prodSEGT segtree l r` は、`a[l] ‘op‘ a[l+1] ‘op‘ ... ‘op‘ a[r-1]` を、モノイドの性質を満たしていると仮定して計算する。なお、$l = r$ のときは `e` を返す。

制約

- $0 \leq l \leq r \leq n$

計算量

- O($\log n$)

all_prod

全区間の総積

```
all_prodSEGT :: (PrimMonad m, Unbox a, Show a) => (Segtree m a) -> m a
```

`a[0] ‘op‘ a[1] ‘op‘ ... ‘op‘ a[n-1]` を計算する。$n = 0$ のときは `e` を返す。

計算量

- O(1)