セグメント木, 遅延評価セグメント木を理解する

BV21013 堀毛晴輝

2022 年 5 月 20 日

目次

1　研究背景

競技プログラミングで戦うために必要なデータ構造のなかに，セグメント木と遅延評価セグメント木がある．原理・実装を知り，理解することで実際に必要な場面で使いこなすことができる．

これらを使いこなすことができる高度 IT 人材となるべく，今回の研究の題材とした．

2　セグメント木

2.1　セグメント木とは

セグメント木とは，モノイドを満たす代数構造に対し使用できるデータ構造である．

モノイド

集合 S とその上の二項演算 $\cdot : S \times S \to S$ に対し，

(1). 結合律: 任意の $a, b, c \in S$ に対して，$(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(2). 単位元の存在: ある $e \in S$ が存在して，任意の $a \in S$ に対して $e \cdot a = a \cdot e = a$

を満たすならば，組 $(S, \cdot, e)$ をモノイドという．

長さ n の S の配列に対し，

- 要素の 1 点変更
- 区間の要素の総積の取得

を $O(\log n)$ で行うことができる．

今回参考とした AtCoder Library での実装では，

- 型 S
- 二項演算 $S\ op(S\ a,\ S\ b)$
- 単位元 $S\ e()$

を定義し，

```
1 segtree<S, op, e> seg(int n)
```

もしくは，

```
1 segtree<S, op, e> seg(vector<S> v)
```

とすることで前者は初期値がすべて $e()$ であり，長さ n の数列 a を，後者は初期値が v であり，長さ $n = v.size()$ の配列 a を作成する．また，以下の操作をできる．

- set

```
1 void seg.set(int p, S x)
```

$a[p]$ に x を代入する.
制約：$0 \leq p < n$
計算量：$O(\log n)$

- get

```
1 S seg.get(int p)
```

$a[p]$ を返す.
制約：$0 \leq p < n$
計算量：$O(1)$

- prod

```
1 S seg.prod(int l, int r)
```

$op(a[l], \cdots, a[r-1])$ をモノイドの性質を満たしていると仮定し計算する． $l = r$ 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

```
1 S seg.all_prod()
```

$op(a[0], \cdots a[n-1])$ を計算する． $n = 0$ のときは $e()$ を返す.
計算量：$O(1)$

2.2 原理

セグメント木では， 完全二分木で区間を管理している.

完全二分木

二分木であり葉以外の節天がすべて 2 つの子を持ち， 根から葉までの深さが等しい木構造を完全二分木という．深さが 1 だけ異なる葉が存在し， その葉が木全体の左側に詰めてあるような二分木も (おおよそ) 完全二分木という.

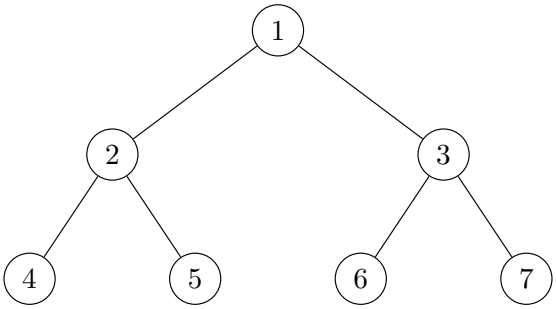


図 1: 完全二分木

木の根の添え字を 1 とし， 節点の添え字 i が与えられたとき， その親， 左の子， 右の子の添え字はそれぞれ $\lfloor i/2 \rfloor$, $i \times 2$, $i \times 2 + 1$ で計算できる．

根は区間全体を管理し， 各節点は区間を管理する． 図 1 と節点の形が違うが， これは各節点がどの範囲を管理するかをわかりやすくしたものであり， 親子関係などは図 1 と同じである．

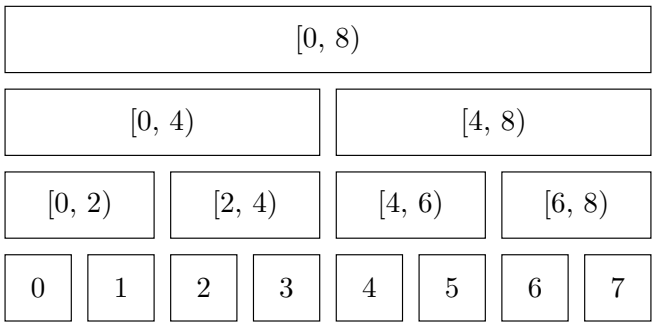


図 2: 完全二分木が管理する区間

このように持つことで， たとえば区間 $[1, 6)$ の積の値は， 図 3 の色を付けた部分の積をとることで求めることができる．

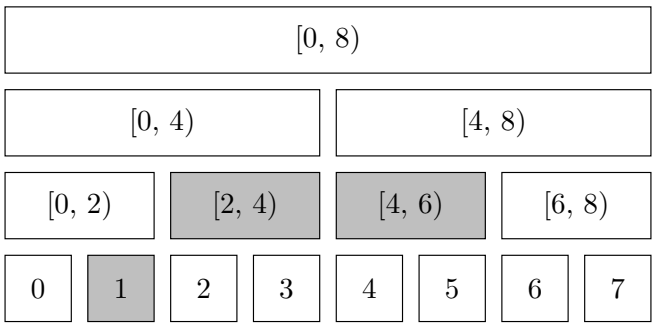


図 3: 区間 $[1, 6)$ の例

2.3 実装

2.3.1 構築

```
1  template <class S, (*op)(S, S), S (*e)()> struct segtree {
2    public:
3      segtree() : segtree(0) {}
4      segtree(int n) : segtree(vector<S> (n, e())) {}
5      segtree(const vector<S> &v) : _n(int(v.size())) {
6          int x = 0;
7          while((1 << x) < _n) x++;
8          log = x;
9          size = 1 << log;
10         d = vector<S> (2 * size, e());
11         for(int i = 0; i < _n; i++) d[size + i] = v[i];
12         for(int i = size - 1; i >= 1; i--) update(i);
13     }
14
15   private:
16     int _n, size, log;
17     vector<S> d;
18     void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
19 };
```

_n, size, log はそれぞれ配列 a の長さ， 配列 a の長さが 2 の冪になるように単位元を追加したときの長さ， 作成する完全二分木の深さである．d は完全二分木で， 以下のように区間を管理する．

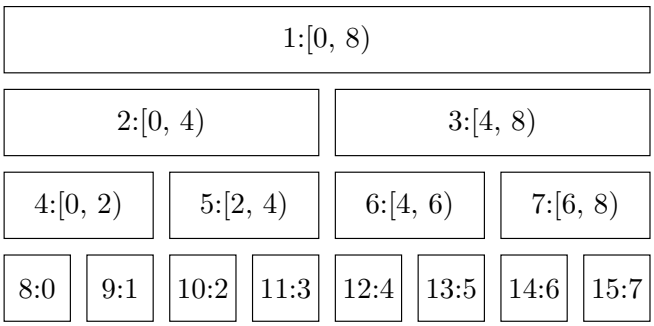


図 4: d が管理する区間

$a[i]$ の値は d[size + i] が持つため， 11 行目で与えられた配列 v をそれが対応する箇所に代入していく．また， 節点の値は子の積であるため， 18 行目の update を各節点に下から行っていくことで構築が完了する．構築の計算量は $O(n)$ である．

2.3.2 set

```
1    void set(int p, S x) {
2        p += size;
3        d[p] = x;
4        for(int i = 1; i <= log; i++) update(p >> i);
5    }
```

2, 3 行目で $a[p]$ の値を持つ $\mathrm{d}[\mathrm{size}+p]$ を x に更新したのち， その区間を含む節点を更新していく．たとえば， $a[2]$ を更新したとき，

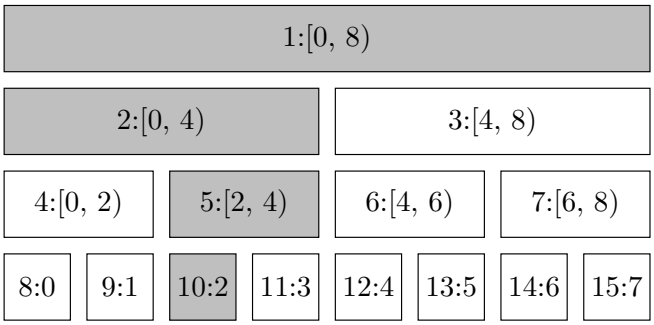


図 5: $a[2]$ を更新したときに更新する必要がある節点

色を付けた節点を下から順に更新していけばよい． 計算量は $O(\log n)$ である．

2.3.3 get

```
1    S get(int p) {
2        return d[p + size];
3    }
```

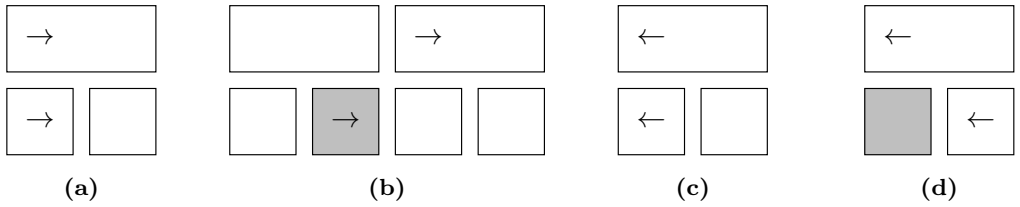
$a[p]$ の値は $\mathrm{d}[\mathrm{size}+p]$ であるから， それを返す． 計算量は $O(1)$ である．

2.3.4 prod

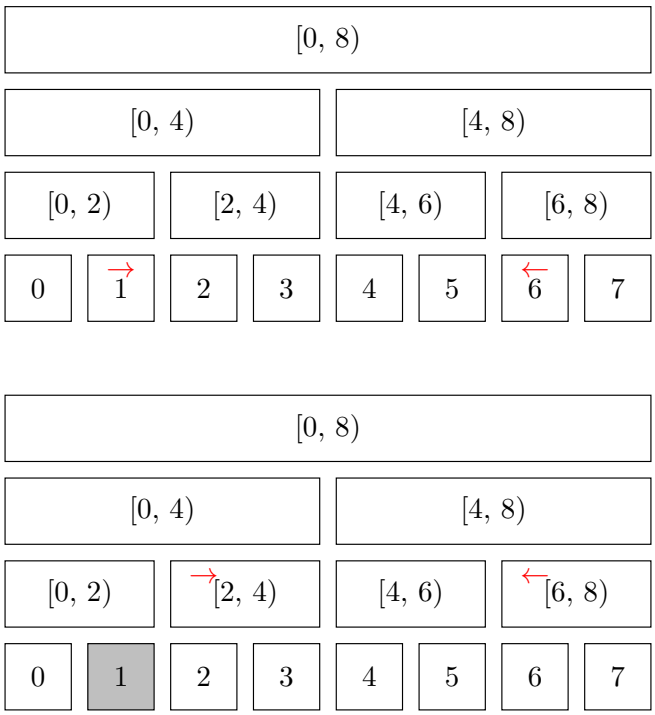
```
1    S prod(int l, int r) {
2        S sml = e(), smr = e();
3        l += size;
4        r += size;
5        while(l < r){
6            if(l & 1) sml = op(sml, d[l++]);
7            if(r & 1) smr = op(d[--r], smr);
8            l >>= 1;
9            r >>= 1;
```

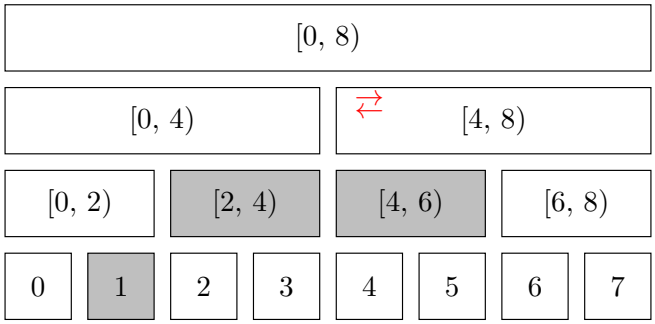
```
10          }
11          return op(sml, smr)
12      }
```

$[l, r)$ の値を求める． ここで， 少ない演算で求める区間の値を計算するためにできるだけ上の要素で足すことが必要である．



(a) は左の子が左端， (b) は右の子が左端， (c) は左の子が右端， (d) は右の子が右端の場合の上り方である．ここで， 色を付けた部分はその要素の積をとることを意味する． これを $l < r$ を満たす間続ければ $[l, r)$ の総積を求めることができる．たとえば， 区間 $[1, 6)$ は，





という手順で求めることができる．計算量は $O(\log n)$ である．

2.3.5 all_prod

```
1    S all_prod() {
2        return d[1];
3    }
```

$[0, n)$ の値は根が持つので， それを返す． 計算量は $O(1)$ である．

2.4 RMQ

AOJ の Range Minimum Query(RMQ) を解く．`https://onlinejudge.u-aizu.ac.jp/problems/DSL_2_A`

注． スライドの RMQ とは制約が若干異なっているので注意．

Range Minimum Query

数列 $A = a_0, a_1, \cdots, a_{n-1}$ に対し， 次の 2 つの操作を行うプログラムを作成せよ．

- update(i, x): a_i を x に変更する
- find(s, t): $a_s, a_{s+1}, \cdots, a_t$ の最小値を出力する

ただし， $a_i (i = 0, 1, \cdots, n-1)$ は $2^{31} - 1$ で初期化されているものとする．

n q
$\mathrm{com}_0\ x_0\ y_0$
$\mathrm{com}_1\ x_1\ y_1$
$\cdots$
$\mathrm{com}_{q-1}\ x_{q-1}\ y_{q-1}$

1 行に A の要素数 n, クエリの数 q が与えられる． 続く q 行にクエリが与えられる． com はクエリの種類を表し， 0 なら update, 1 なら find を表す．find クエリについて， 最小値を 1 行に出力せよ．

- $1 \leq \mathrm{n} \leq 100000$
- $1 \leq \mathrm{q} \leq 100000$
- com_i が 0 のとき， $0 \leq x_i < n, 0 \leq y_i < 2^{31} - 1$
- com_i が 1 のとき， $0 \leq x_i < n, 0 \leq y_i < n$

```
#include <bits/stdc++.h>
using namespace std;

template <class S, S (*op)(S, S), S (*e)()> struct segtree {
  public:
    segtree() : segtree(0) {}
    segtree(int n) : segtree(vector<S> (n, e())) {}
    segtree(const vector<S> &v) : _n(int(v.size())) {
        int x = 0;
        while((1 << x) < _n) x++;
        log = x;
        size = 1 << log;
        d = vector<S> (2 * size, e());
        for(int i = 0; i < _n; i++) d[size + i] = v[i];
        for(int i = size - 1; i >= 1; i--) update(i);
    }

    void set(int p, S x){
        p += size;
        d[p] = x;
```

```
21          for(int i = 1; i <= log; i++) update(p >> i);
22      }
23
24      S get(int p) {
25          return d[p + size];
26      }
27
28      S prod(int l, int r){
29          S sml = e(), smr = e();
30          l += size;
31          r += size;
32          while (l < r) {
33              if (l & 1) sml = op(sml, d[l++]);
34              if (r & 1) smr = op(d[--r], smr);
35              l >>= 1;
36              r >>= 1;
37          }
38          return op(sml, smr);
39      }
40
41      S all_prod() { return d[1];}
42
43    private:
44      int _n, size, log;
45      vector<S> d;
46      void update(int k){ d[k] = op(d[2 * k], d[2 * k + 1]);}
47  };
48
49  using S = int;
50  S op(S a, S b){ return min(a, b); }
51  S e(){ return (1LL << 31) - 1; }
52
53  int main(){
54
55      int n, q; cin >> n >> q;
56      segtree<S, op, e> seg(n);
57
58      for(int i = 0; i < q; i++){
59          int com, x, y; cin >> x >> y;
60          if(com == 0){
61              seg.set(x, y);
62          }else{
```

```
63              cout << seg.get(x, y + 1) << endl;
64          }
65      }
66
67      return 0;
68  }
```

二項演算 min(a, b) は結合律を満たし， また制約上単位元を $2^{31}-1$ とできる． よって，これらをセグメント木に乗せることで Range Minimum Query を解くことができる．

3 遅延評価セグメント木

3.1 遅延評価セグメント木とは

遅延評価セグメント木とは， モノイド S と， S への作用素モノイド F に対し使用できるデータ構造である．

右作用/作用素モノイド

モノイド G と集合 X に対し， X への G による右作用とは， 写像

$$\psi : G \times X \to X,\ (g, x) \mapsto \psi(g, x)$$

であって， 次の 2 条件を満たすもののことをいう．

(1). 任意の $x \in X$ に対し， $\psi(e, x) = x$. ただし， e は G の単位元.

(2). 任意の $g, h \in G, x \in X$ に対し， $\psi(gh, x) = \psi(h, \psi(g, x))$

また， 作用に関するモノイドを作用素モノイドという．

また， $x_i, x_j \in S, f \in F$ に対して， $\psi(f, x_i \cdot x_j) = \psi(f, x_i) \cdot \psi(f, x_j)$ が成り立つ必要がある．

長さ n の S の配列に対し，

- 要素の 1 点変更
- 区間の要素の総積の取得
- 区間に $f \in F$ を作用させる

を $O(\log n)$ で行うことができる．

今回参考とした AtCoder Library での実装では，

- 型 S
- S の二項演算 S op(S a, S b)
- S の単位元 S e()
- 型 F
- F の二項演算 F composition(F f, F g)

- F の単位元 F id()
- $\psi(f,x)$ を返す関数 S mapping(F f, S x)

を定義し,

```
lazy_segtree<S, op, e, F, mapping, composition, id> seg(int n)
```

もしくは,

```
lazy_segtree<S, op, e, F, mapping, composition, id> seg(vector<S> v)
```

とすることで前者は初期値がすべて $e()$ であり， 長さ n の数列 a を， 後者は初期値が v であり，長さ $n = v.size()$ の配列 a を作成する．また， 以下の操作が可能である．

- set

  ```
  void seg.set(int p, S x)
  ```

  $a[p]$ に x を代入する．
  制約：$0 \leq p < n$
  計算量：$O(\log n)$
- get

  ```
  S seg.get(int p)
  ```

  $a[p]$ を返す．
  制約：$0 \leq p < n$
  計算量：$O(1)$
- prod

  ```
  S seg.prod(int l, int r)
  ```

  $op(a[l], a[l+1], \cdots, a[r-1])$ をモノイドの性質を満たしていると仮定し計算する． $l = r$ 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

  ```
  S seg.all_prod()
  ```

  $op(a[0], \cdots a[n-1])$ を計算する． $n = 0$ のときは $e()$ を返す．
  計算量：$O(1)$
- apply

  ```
  (1) void seg.apply(int p, F f)
  (2) void seg.apply(int l, int r, F f)
  ```

(1)：$a[p]$ に f を作用させる．
(2)：$i = l, l+1, \cdots, a[r-1]$ に f を作用させる．
制約：(1):$0 \leq p < n$, (2):$0 \leq l \leq r \leq n$
計算量：$O(\log n)$

3.2 原理

x_i, y_i を S の元， f, g, h を F の元， id を F の単位元 とする． 各モノイドの二項演算を，

$$S \times S \to S; (x_i, x_j) \mapsto x_i x_j$$
$$F \times F \to F; (f, g) \mapsto fg$$

と書き， 作用は

$$S \times F \to S; (x_i, f) \mapsto x_i^f$$

と書く．

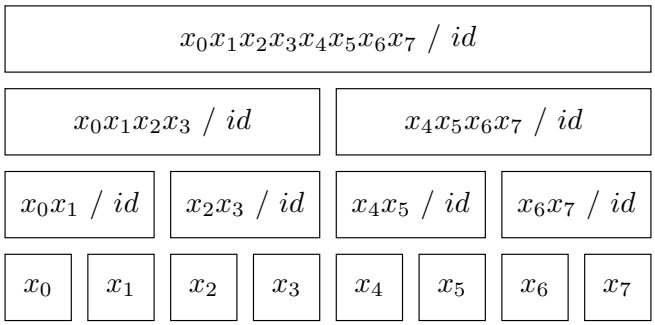


図 7: 初期状態

構築時はこのような状態となる． ここで， 左側はセグメント木同様 S の区間を管理する完全二分木， 右側は作用を管理する完全二分木である．これら 2 つの完全二分木を用いて管理する．

具体例とともに， どのような手順を踏んでいるのかを見る．

I. $[1, 6)$ に f を作用させる．

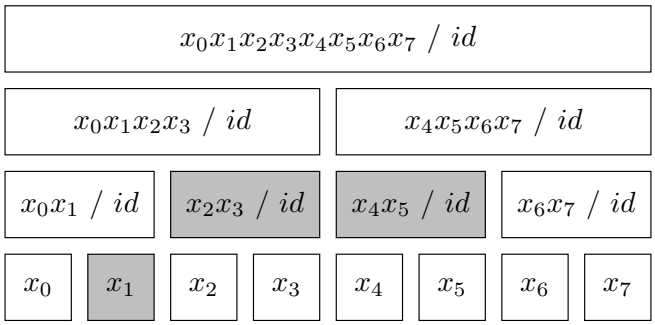


図 8: $[1, 6)$ に f を作用

1. 上から伝搬 (上から下へ)

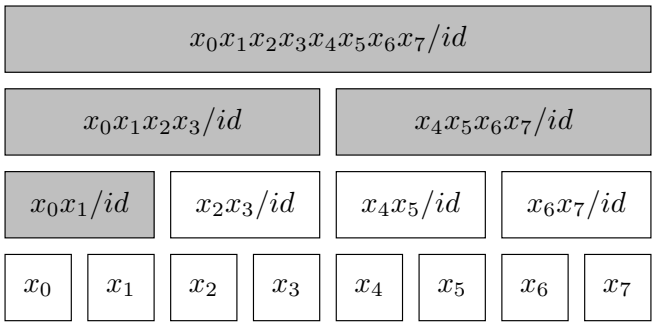


図 9: 上から伝搬

2. 区間に f を作用

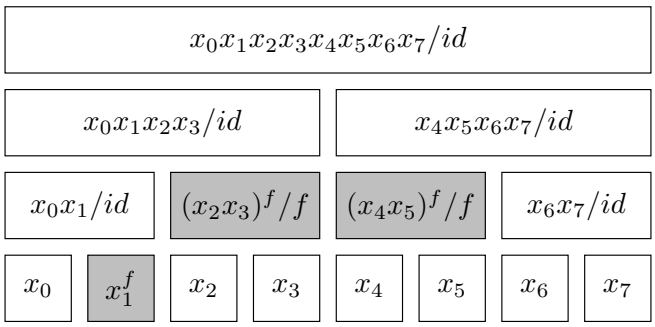


図 10: 区間に f を作用

3. 上側を計算しなおし (下から上へ)

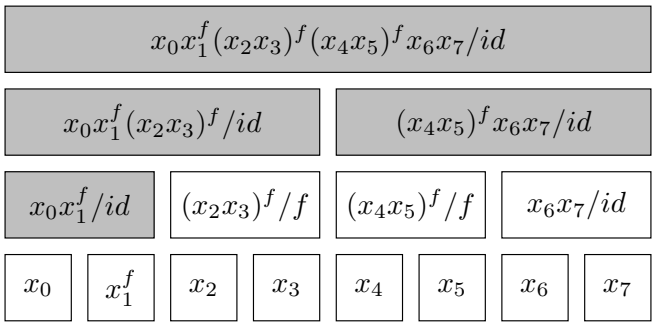


図 11: 上側を計算しなおし

II. $[3, 8)$ に g を作用させる.

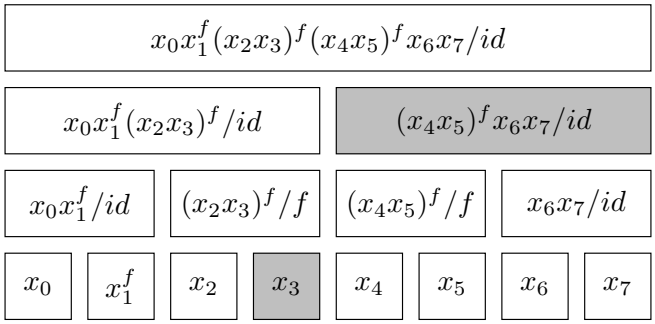


図 12: $[3, 8)$ に g を作用

1. 上から伝搬 (上から下へ)

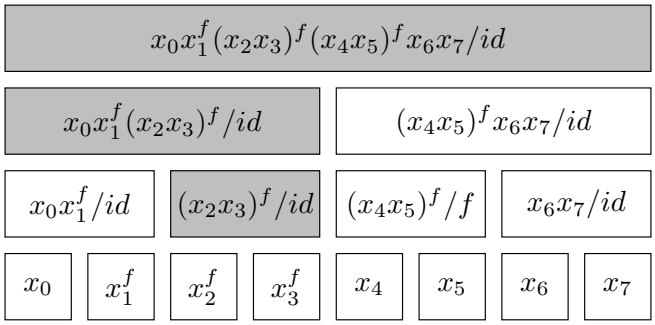


図 13: 上から伝搬

2. 区間に g を作用

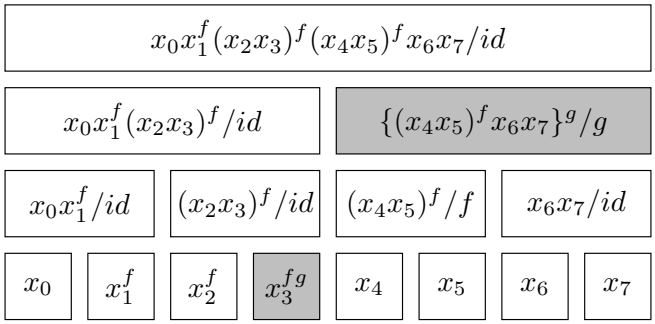


図 14: 区間に g を作用

3. 上側を計算しなおし (下から上へ)

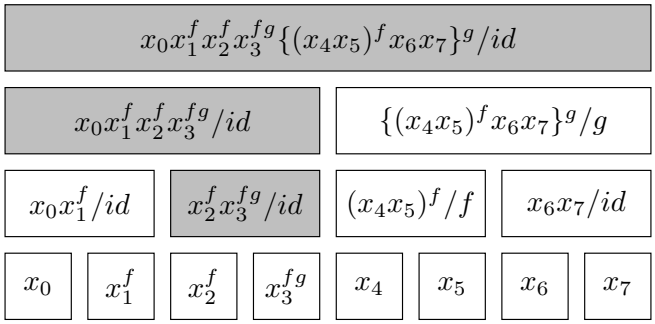


図 15: 上側を計算しなおし

III. $[4,7)$ を取得する.

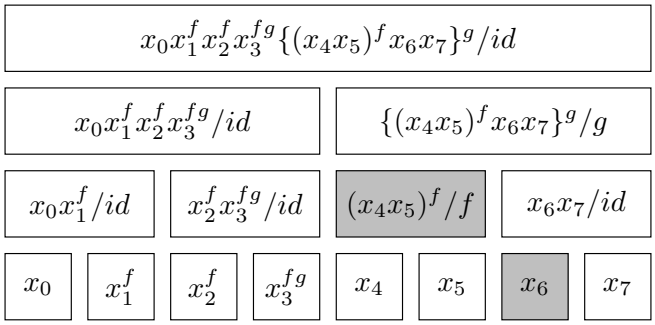


図 16: $[4,7)$ を取得

1. 上から伝搬 (上から下へ)

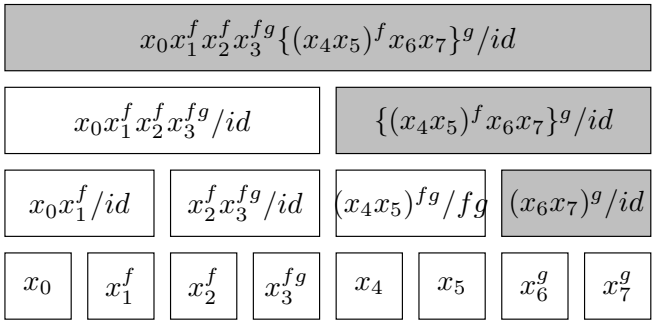


図 17: 上から伝搬

2. 対応する区間積を取る

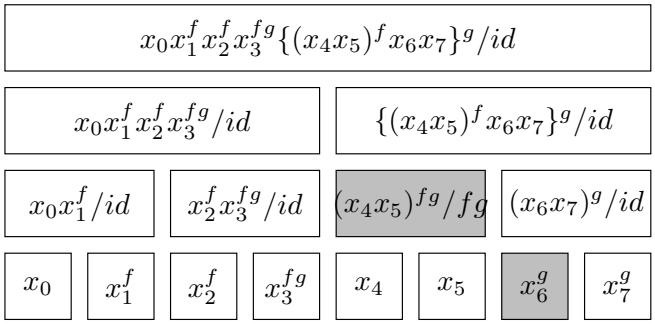


図 18: 対応する区間積を取る

$(x_4x_5)^{fg}x_6^g$ が得られる.

IV. x_5 を y_5 に更新する.

1. 上から伝搬 (上から下へ)

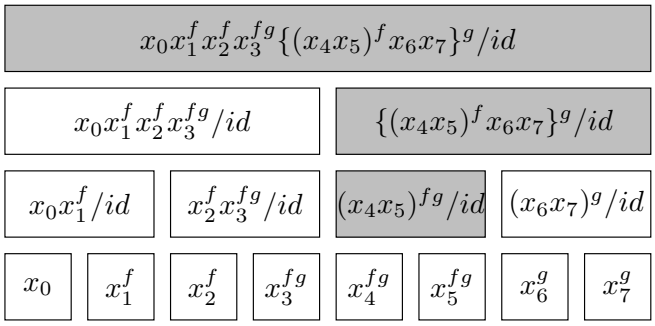


図 19: 上から伝搬

2. 更新する

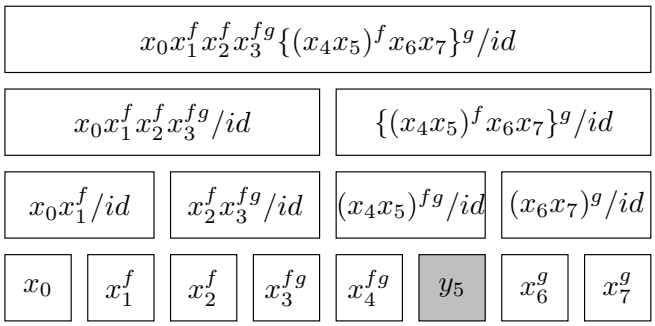


図 20: 更新する

3. 上側を計算しなおし (下から上へ)

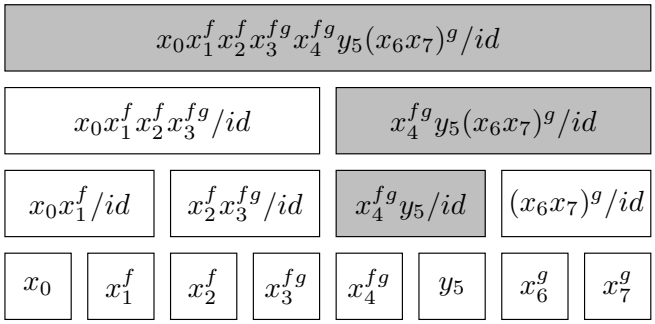


図 21: 更新する

このような手順を踏みながらクエリを処理していく.

- 区間作用
  1. 上から伝搬 (上から下へ)
  2. 区間に作用
  3. 上側を計算しなおし (下から上へ)
- 区間取得
  1. 上から伝搬 (上から下へ)
  2. 区間を取得
- 1 点更新
  1. 上から伝搬 (上から下へ)
  2. 更新する
  3. 上側を計算しなおし (下から上へ)

を順に行うことでクエリを処理できる.

3.3 実装

3.3.1 構築

```
template<class S,
         S (*op)(S, S),
         S (*e)(),
         class F,
         S (*mapping)(F, S),
         F (*composition)(F, F),
         F (*id)()>
struct lazy_segtree{
  public:
    lazy_segtree() : lazy_segtree(0) {}
    lazy_segtree(int n) : lazy_segtree(vector<S> (n, e())) {}
```

```
12      lazy_segtree(const vector<S> &v) : _n(int(v.size())) {
13          int x = 0;
14          while((1 << x) < _n) x++;
15          log = x;
16          size = 1 << log;
17          d = vector<S> (2 * size, e());
18          for(int i = 0; i < _n; i++) d[size + i] = v[i];
19          for(int i = size - 1; i >= 1; i--) update(i);
20      }
21
22    private:
23      int _n, size, log;
24      vector<S> d;
25      vector<F> lz;
26
27      void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
28      void all_apply(int k, F f) {
29          d[k] = mapping(f, d[k]);
30          if(k < size) lz[k] = composition(f, lz[k]);
31      }
32      void push(int k){
33          all_apply(2 * k, lz[k]);
34          all_apply(2 * k + 1, lz[k]);
35          lz[k] = id();
36      }
37  };
```

lz は各節点に作用されている値である． push はその区間に作用している値を子へと伝播させる関数， all_apply は k の部分木に f を作用させる関数である．構築はセグメント木と同じであり，下から順に更新をしていく． 計算量は $O(n)$ である．

3.3.2 set

```
1       void set(int p, S x) {
2           p += size;
3           for(int i = log; i >= 1; i--) push(p >> i);
4           d[p] = x;
5           for(int i = 1; i <= log; i++) update(p >> i);
6       }
```

3 行目で上から下へ作用を伝搬， 4 行目で値の更新， 5 行目で下から上へ計算しなおしを行っている． 計算量は $O(\log n)$ である．

3.3.3 get

```
1    S get(int p) {
2        p += size;
3        for(int i = log; i >= 1; i--) push(p >> i);
4        return d[p];
5    }
```

3 行目で上から下へ作用を伝搬させる． その後， p に対応する値を返す． 計算は $O(\log n)$ である．

3.3.4 prod

```
1    S prod(int l, int r) {
2        if(l == r) return e();
3        l += size; r += size;
4
5        for(int i = log; i >= i; i--) {
6            if(((l >> i) << i) != l) push(l >> i);
7            if(((r >> i) << i) != r) push(r >> i);
8        }
9
10       S sml = e(), smr = e();
11       while(l < r) {
12           if(l & 1) sml = op(sml, d[l++]);
13           if(r & 1) smr = op(d[--r], smr);
14           l >>= 1;
15           r >>= 1;
16       }
17       return op(sml, smr);
18   }
```

区間取得をする際に伝搬をしなくてはならない節点を考える． たとえば， [2,6) の区間取得をするとき，

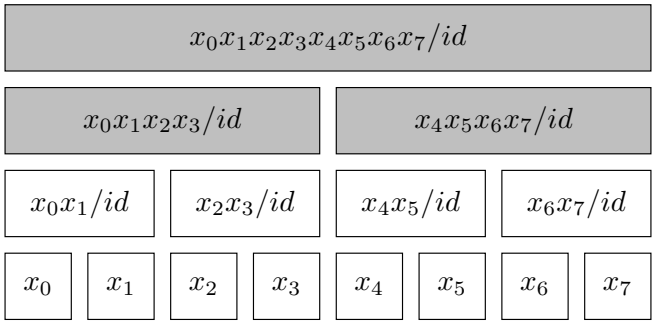


図 22: [2,6) の区間取得を行うときの伝搬を行わないといけない節点

x_2x_3 の節点と x_4x_5 の節点の二項演算を返すため， 伝搬は色をつけた節点でのみ行えばよい．つまり， x_2x_3 と x_6x_7 の節点では伝搬をする必要はない．左端/右端の真上にあるが， 伝搬をする必要がないときはどのような条件かを考える． [l,r) の区間について，

- l の真上にあり， 伝搬をする必要がない節点とは， その節点が持つ範囲の左端が l であるような節点である．
- r の真上にあり， 伝搬をする必要がない節点とは， その節点が持つ範囲の左端が r であるような節点である． (このとき， r より左を見たときに必要な部分までの伝搬がすべて完了する)

真上にある節点は $i = 1, 2, ...\log$ に対して (x >> i) で表される (x は見ているインデックス +size)．それらの節点の左端は ((x >> i) << i) であるから，それが x と一致する場合はその節点では伝搬をしなくてよい．よって， 5-8 行目のように書くことで伝搬が完了する．区間取得はセグメント木と同じ処理をおこなっている．

3.3.5 all_prod

```
1    S all_prod() { return d[1]; }
```

d[1] より上には何もないので， 伝搬されてくる作用もない． よってそのまま d[1] を返す． 計算量は $O(1)$ である．

3.3.6 apply

```
1    void apply(int p, F f) {
2        p += size;
3        for(int i = log; i >= 1; i--) push(p >> i);
4        d[p] = mapping(f, d[p]);
5        for(int i = 1; i <= log; i++) update(p >> i);
6    }
7    void apply(int l, int r, F f) {
8        if(l == r) return;
```

```
        l += size; r += size;

        for(int i = log; i >= 1; i--) {
            if(((l >> i) << i) != l) push(i >> i);
            if(((r >> i) << i) != r) push(r >> i);
        }

        {
            int l2 = l, r2 = r;
            while(l < r) {
                if(l & 1) all_apply(l++, f);
                if(r & 1) all_apply(--r, f);
                l >>= 1; r >>= 1;
            }
            l = l2; r = r2;
        }

        for(int i = 1; i <= log; i++) {
            if(((l >> i) << i) != l) update(l >> i);
            if(((r >> i) << i) != r) update(r >> i);
        }
    }
```

作用させる箇所が 1 つのとき， set と同じようにそれらの上にあるものを伝搬し， 変更し， 計算しなおしを行えばよい．区間に対しての作用では， prod と同じように上から伝搬をし， 取得する際の節点に作用させ， 計算しなおしを行えばよい．

3.4 RMQ and RUQ

AOJ の RMQ(Range Minimum Query) and RUQ(Range Update Query) を解く．`https://onlinejudge.u-aizu.ac.jp/problems/DSL_2_F`

注． スライドの RMQ and RUQ とは制約が若干異なっているので注意．

RMQ and RUQ

数列 $A = a_0, a_1, \cdots, a_{n-1}$ に対し， 次の 2 つの操作を行うプログラムを作成せよ．

- update(s, t, x): $a_s, a_{s+1}, \cdots, a_t$ を x に変更する
- find(s, t): $a_s, a_{s+1}, \cdots, a_t$ の最小値を出力する

ただし， $a_i (i = 0, i, \cdots, n-1)$ は $2^{31} - 1$ で初期化されているものとする．

n q
query_0
query_1
$\cdots$
query_{q-1}

1 行目に A の要素数 n, クエリの数 q が与えられる． 続く q 行にクエリが与えられる．クエリは以下のいずれかの形式で与えられる． 各 find クエリについて， 最小値を 1 行に出力せよ．

0 s t x : update(s, t, x)
1 s t : find(s, t)

- $1 \leq$ n ≤ 100000
- $1 \leq$ q ≤ 100000
- $0 \leq$ s $\leq$ t $<$ n
- $0 \leq x < 2^{31} - 1$

```
#include <bits/stdc++.h>
using namespace std;

template <class S,
          S (*op)(S, S),
          S (*e)(),
          class F,
          S (*mapping)(F, S),
          F (*composition)(F, F),
          F (*id)()>
struct lazy_segtree {
  public:
    lazy_segtree() : lazy_segtree(0) {}
    lazy_segtree(int n) : lazy_segtree(std::vector<S>(n, e())) {}
    lazy_segtree(const std::vector<S>& v) : _n(int(v.size())) {
        int x = 0;
        while((1 << x) < _n) x++;
        log = x;
```

```
19          size = 1 << log;
20          d = vector<S>(2 * size, e());
21          lz = vector<F>(size, id());
22          for (int i = 0; i < _n; i++) d[size + i] = v[i];
23          for (int i = size - 1; i >= 1; i--) {
24              update(i);
25          }
26      }
27
28      void set(int p, S x) {
29          p += size;
30          for (int i = log; i >= 1; i--) push(p >> i);
31          d[p] = x;
32          for (int i = 1; i <= log; i++) update(p >> i);
33      }
34
35      S get(int p) {
36          p += size;
37          for (int i = log; i >= 1; i--) push(p >> i);
38          return d[p];
39      }
40
41      S prod(int l, int r) {
42          if (l == r) return e();
43
44          l += size;
45          r += size;
46
47          for (int i = log; i >= 1; i--) {
48              if (((l >> i) << i) != l) push(l >> i);
49              if (((r >> i) << i) != r) push(r >> i);
50          }
51
52          S sml = e(), smr = e();
53          while (l < r) {
54              if (l & 1) sml = op(sml, d[l++]);
55              if (r & 1) smr = op(d[--r], smr);
56              l >>= 1;
57              r >>= 1;
58          }
59
60          return op(sml, smr);
```

```
61      }
62
63      S all_prod() { return d[1]; }
64
65      void apply(int p, F f) {
66          p += size;
67          for (int i = log; i >= 1; i--) push(p >> i);
68          d[p] = mapping(f, d[p]);
69          for (int i = 1; i <= log; i++) update(p >> i);
70      }
71      void apply(int l, int r, F f) {
72          if (l == r) return;
73
74          l += size;
75          r += size;
76
77          for (int i = log; i >= 1; i--) {
78              if (((l >> i) << i) != l) push(l >> i);
79              if (((r >> i) << i) != r) push(r >> i);
80          }
81
82          {
83              int l2 = l, r2 = r;
84              while (l < r) {
85                  if (l & 1) all_apply(l++, f);
86                  if (r & 1) all_apply(--r, f);
87                  l >>= 1;
88                  r >>= 1;
89              }
90              l = l2;
91              r = r2;
92          }
93
94          for (int i = 1; i <= log; i++) {
95              if (((l >> i) << i) != l) update(l >> i);
96              if (((r >> i) << i) != r) update(r >> i);
97          }
98      }
99
100 private:
101     int _n, size, log;
102     vector<S> d;
```

```
103     vector<F> lz;
104 
105     void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
106     void all_apply(int k, F f) {
107         d[k] = mapping(f, d[k]);
108         if (k < size) lz[k] = composition(f, lz[k]);
109     }
110     void push(int k) {
111         all_apply(2 * k, lz[k]);
112         all_apply(2 * k + 1, lz[k]);
113         lz[k] = id();
114     }
115 };
116 
117 using S = int;
118 S op(S a, S b){ return min(a, b); }
119 S e(){ return (1LL << 31) - 1; }
120 using F = int;
121 S mapping(F f, S x){ return (f == -1 ? x : f); }
122 F composition(F f, F g){ return (f == -1 ? g : f); }
123 F id(){ return -1; }
124 
125 int main(){
126 
127     int n, q; cin >> n >> q;
128     lazy_segtree<S, op, e, F, mapping, composition, id> seg(n);
129     for(int i = 0; i < q; i++){
130         int com; cin >> com;
131         if(com == 0){
132             int s, t, x; cin >> s >> t >> x;
133             seg.apply(s, t+1, x);
134         }else{
135             int s, t; cin >> s >> t;
136             cout << seg.prod(s, t+1) << endl;
137         }
138     }
139 
140 }
```

モノイド S を

$$(int, \cdot, 2^{31}-1), a \cdot b = min(a, b)$$

モノイド F を

$$(int, \times, -1), f \times g = \begin{cases} g & \text{if } g \neq -1, \\ f & \text{else.} \end{cases}$$

また,

$$\psi(f, x) = \begin{cases} f & \text{if } f \neq -1, \\ x & \text{else.} \end{cases}$$

とすることで， RMQ and RUQ を解くことができる．モノイド F について， 実際の単位元となる値は存在しないが， 各クエリで出てこない -1 を単位元とし， -1 のときとそうでないときの処理を分けることで単位元となるようにしている．

4 応用

4.1 LCA

LCA(Lowest Common Ancestor, 最近共通祖先)

根付き木において， 頂点 a と b を根に向かってたどっていったとき， 最初に合流する頂点を a, b の LCA(Lowest Common Ancestor, 最近共通祖先) という．

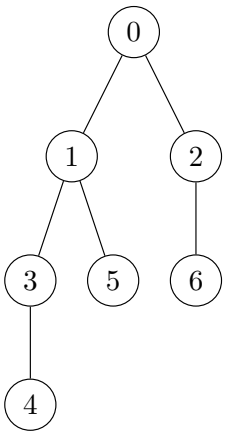


図 23: 0 を根とする根付き木の例

たとえば， この 0 を根とする根付き木に対して,

- 4, 5 の LCA は 1
- 5, 6 の LCA は 0
- 2, 6 の LCA は 2

である．

オイラーツアー

根から DFS し根に戻ってくる経路を， オイラーツアーという．

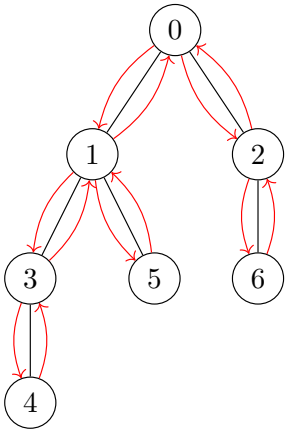


図 24: オイラーツアーの例

たとえば， この 0 を根とする根付き木に対して， オイラーツアーの経路の 1 つは，

$$0 \to 1 \to 3 \to 4 \to 3 \to 1 \to 5 \to 1 \to 0 \to 2 \to 6 \to 2 \to 0$$

である．

オイラーツアーをする際に， 各頂点にて訪れたときのステップ数を記録しておく． 図 24 の例では，

ステップ	0	1	2	3	4	5	6	7	8	9	10	11	12
経路	0	1	3	4	3	1	5	1	0	2	6	2	0

表 1: オイラーツアーの経路

頂点	0	1	2	3	4	5	6
in	0	1	9	2	3	6	10

表 2: 各頂点に初めて訪れた時のステップ数

オイラーツアーの経路に対して， 区間 [min(in[a], in[b]), max(in[a], in[b])] を考える． たとえば区間 [in[4], in[5]] は次のようになる．

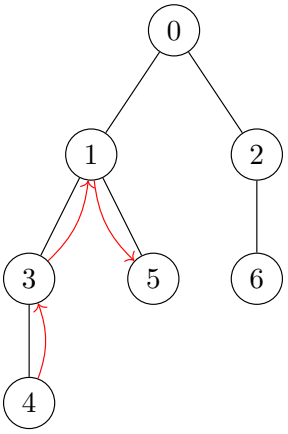


図 25: 区間 [in[4], in[5]]

[in[4], in[5]] は， $4 \to 3 \to 1 \to 5$ となる． この区間の中で深さが最小である頂点， すなわち頂点 1 が頂点 4, 5 の LCA となる．

もう 1 つ例を見る． [in[3], in[6]] では， 次のようになる．

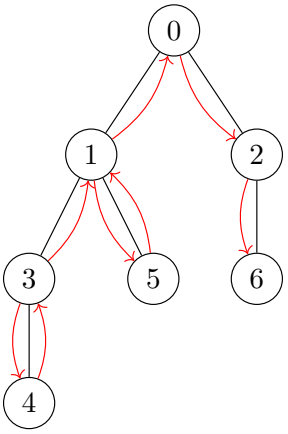


図 26: 区間 [in[3], in[6]]

$3 \to 4 \to 3 \to 1 \to 5 \to 1 \to 0 \to 2 \to 6$ となる． この中で深さが最小である頂点は頂点 0 である． 頂点 3, 6 の LCA は頂点 0 である．

頂点 a, b の LCA は， オイラーツアーした経路の区間 [min(in[a], in[b]), max(in[a], in[b])] の深さが最小の頂点である．区間の最小値はセグメント木を用いることで $O(\log N)$(N は頂点の数) で求めることができる． したがって， 任意の 2 頂点の LCA を $O(\log N)$ で求めることができる．

木上の 2 点間の距離を求めるとき， 片方の頂点から DFS/BFS をした場合， 最悪計算量は $\Theta(N)$ となる．前処理として， 根からすべての頂点への距離を記録しておく． 根から頂点 i までの距離を $dist_i$ と書くことにする．すると， 頂点 a から b の距離は，

$$dist_a + dist_b - 2dist_{LCA(a,b)}$$

で求めることができる．

たとえば， 図 24 の例で頂点 4 から頂点 5 への距離を考える．

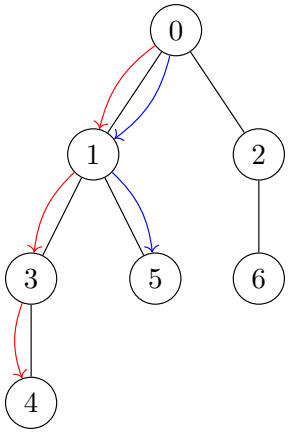


図 27: 頂点 4 から頂点 5 への距離

$dist_4$ を赤， $dist_5$ を青で示した． LCA(4, 5) = 1 である． LCA の定義より， $dist_4$, $dist_5$ は $dist_{LCA(4,5)}$ まで同じ経路をたどる．よって， 頂点 4 から頂点 5 への距離は，

$$dist_4 + dist_5 - 2dist_{LCA(4,5)} = 3 + 2 - 2 \cdot 1 = 3$$

と求めることができる．例として重みなしの木をあげたが， 木の定義より重み付きの木でも同じことがいえる．

```
#include <bits/stdc++.h>
using namespace std;

template<class T> struct LCA {
    public:
    LCA(int n) : _n(n) {
        G.resize(_n);
    }

    void add_edge(int s, int t){ add_edge(s, t, T(1)); }
    void add_edge(int s, int t, T c){
        G[s].push_back({t, c});
        G[t].push_back({s, c});
    }

    void build(int root = 0){
        in.resize(_n, -1);
        out.resize(_n, -1);
        dist.resize(_n);
        dist[root] = T(0);
```

```
21
22          auto dfs = [&](auto&& self, int now, int bef, int depth) -> void {
23
24              in[now] = euler_tour.size();
25              euler_tour.push_back({depth, now});
26
27              for(auto e: G[now]) if(e.to != bef){
28                  dist[e.to] = dist[now] + e.cost;
29                  self(self, e.to, now, depth + 1);
30                  euler_tour.push_back({depth, now});
31              }
32
33          };
34
35          dfs(dfs, root, -1, 0);
36
37          log = 0;
38          while((1 << log) < euler_tour.size()) log++;
39          size = 1 << log;
40          d.resize(2 * size, e());
41
42          for(int i = 0; i < euler_tour.size(); i++) d[size + i] = euler_tour[
                i];
43          for(int i = size - 1; i >= 1; i--) update(i);
44
45      }
46
47      int query(int u, int v){
48          return prod(min(in[u], in[v]), max(in[u], in[v]) + 1).index;
49      }
50
51      T get_path(int u, int v){
52          return dist[u] + dist[v] - 2 * dist[query(u, v)];
53      }
54
55      private:
56      struct edge{ int to; T cost; };
57      struct S { int depth, index; };
58      S op(S a, S b){ return (a.depth < b.depth ? a : b); }
59      S e(){ return {1 << 28, 1 << 28}; }
60
61      int _n, log, size;
```

```
    vector<vector<edge>> G;
    vector<T> dist;
    vector<int> in;
    vector<S> euler_tour;
    vector<S> d;

    void update(int k){ d[k] = op(d[2 * k], d[2 * k + 1]); }
    S prod(int l, int r){
        S smr = e(), sml = e();
        l += size; r += size;
        while(l < r){
            if(l & 1) sml = op(sml, d[l++]);
            if(r & 1) smr = op(d[--r], smr);
            l >>= 1; r >>= 1;
        }
        return op(sml, smr);
    }

};
```

- 初期化

```
LCA<Edge> lca(n);
```

Edge は辺の重みの型 (重みなしの場合は int), n は頂点の数. 計算量：$O(n)$

- add_edge

```
            (1) void lca.add_edge(int s, int t);
            (2) void lca.add_edge(int s, int t, Edge c);
```

(1)：頂点 s と頂点 t を結ぶ長さ 1 の辺を追加
(2)：頂点 s と頂点 t を結ぶ長さ c の辺を追加
計算量：$O(1)$

- build

```
void lca.build(int root = 0);
```

root を根として構築する.
計算量：$O(n)$

- query

```
int lca.query(int s, int t);
```

LCA(s, t) を求める.

計算量：$O(\log n)$

- get_path

```
Edge lca.get_path(int s, int t);
```

s から t の距離を求める.
計算量：$O(\log n)$

5 今後の課題

セグメント木上の二分探索には今回触れなかったため， 木上の二分探索についても今後学んでいきたい.

セグメント木・遅延評価セグメント木にのみ触れたが， このほかにも双対セグメント木や, Dinamic Segtree, Segtree Beats といったものもあるので， それらについても理解を深めていきたい.

参考文献

[1] AtCoder, ”AtCoder Library Document”, https://atcoder.github.io/ac-library/document_ja/index.html

[2] えびちゃん, ”非再帰セグ木サイコー！ 一番好きなセグ木です”, https://hcpc-hokudai.github.io/archive/structure_segtree_001.pdf

[3] keymoon, ”SegmentTree に載る代数的構造について”, https://qiita.com/keymoon/items/0f929a19ed30f34ae6e8

[4] maspy, ”SegmentTree のお勉強”, https://maspypy.com/segment-tree-%E3%81%AE%E3%81%8A%E5%8B%89%E5%BC%B72

[5] maspy, ”Euler Tour のお勉強”, https://maspypy.com/euler-tour-%E3%81%AE%E3%81%8A%E5%8B%89%E5%BC%B7

[6] Aizu Online Judge, ”RMQ”, https://onlinejudge.u-aizu.ac.jp/problems/DSL_2_A

[7] Aizu Online Judge, ”RMQ and RUQ”, https://onlinejudge.u-aizu.ac.jp/problems/DSL_2_F