Maturski rad

Segmentno stablo

Učenik:

Marko Baković

Profesor mentor:

Zoran Vasiljević

Sadržaj

Uvod

Kada imamo neke podatke, recimo niz brojeva, često nam je potrebno da radimo samo sa delom tog niz, odnosno potrebno nam je da što brže nadjemo neku osobinu dela niza (minimum, maksimum...). To lako možemo uraditi u linearnom vremenu, prolaskom kroz traženi podniz. Međutim, problem nastaje kada je potrebno više puta naći osobinu različitih delova niza. Tada linearno vreme više nije dovoljno. Segmentno stablo nam omogućava da tu složenost dovedemo do logaritamske.

Segmentno stablo je stablo koje ima sledeće osobine:

1. Svaki čvor stabla ima tačno dva sina. Neka čvor ima indeks i. Njegov levi sin ima indeks 2*i, a desni 2*i + 1.
2. Svaki čvor obuhvata neki interval i čuva neki podatak za taj interval. Ako čvor obuhvata interval [L, R], neka je $M = \frac{L+R}{2}$, tada levi sin obuhvata [L, M], a desni [M + 1, R].
3. Podatak koji čuva svaki čvor mora biti takav da se vrednost podatka za oca može izvesi iz podataka njegovih sinova. Na primer, ukoliko je podatak maksimum, vrednost oca bi bila maksimum vrednosti sinova.

Najčešći postupci koje primenjujemo nad segmentnim stablom jesu promena člana (nekad i intervala) niza i traženje podatka za neki interval (retko i član niza).

Ukoliko imamo neku promenu člana niza, sve što treba da promenimo u segmentnom stablu jeste čvor koji obuhvata samo tog člana, i sve pretke tog čvora (do korena). Kako ima najviše log(N) predaka, jasno je da je složenost promene logaritamska.

Isto tako, kada nam je potreban podatak za neki deo niz, kombinovaćemo podatke na onim čvorovima čiji interval potpuno pripada posmatranom delu, ali ukoliko smo uzeli podatke oca, nećemo se spuštati na sinove. Vremenska složenost ovoga je takođe logaritamska.

Implementacija segmentnog stabla

Segmentno stablo je samo struktura koja može da pomogne pri rešavanju problema, tako da ne postoji tačno određena implementacija segmentnog stabla. Razlikuje se od problema do problema. Takođe, postoje različite implementacije segmentnog stabla. Ali ovde će biti reči o rekurzivnoj implementaciji.

Iako stalno koristimo reč stablo, segmentno stablo je samo niz koji mi zamišljamo kao stablo. Svaki čvor stabla ima svoj indeks, počev od indeksa korena koji je 1, i vodeći se osobinom 1. datom u osobinama stabla (uvod). Niz treba definisati tako da ima $2^{\lceil logN \rceil + 1}$ članova.

Funkcije koje su najčešće u upotrebi jesu init, update i get.

Init

Init funkcija konstruiše segmentno stablo nad zadatim nizom u vremenskoj složenosti O(N). Funkcija ima 4 argumenta:

- idx (indeks čvora na kome se trenutno nalazimo)
- left (leva granica intervala koji obuhvata taj čvor)
- right (desna granica intervala koji obuhvata taj čvor)
- queue (niz nad kojim konstruišemo segmentno stablo)

Funkciju pozivamo kao init(1, najmanji indeks u nizu, najveći indeks u nizu, niz). Funkcija merge dobija podatak za oca na osnovu podataka sinova i razlikuje se od problema do problema. Vremenska složenost sledeće implementacije je O(N).

```
void init( int idx, int left, int right, int queue[] )
{
   if ( left == right )
   {
      tree[ idx ] = queue[];
      return;
   }
   int mid = ( left + right ) / 2;
   init( 2 * idx, left, mid );
   init( 2 * idx + 1, mid + 1, right );
   tree[ idx ] = merge ( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}
```

Update

Update funkcija unosi promenu nekog čvora u segmentno stablo. Poziva se za idx, left, right, i za indeks čvora koji se menja k i novu vrednost val. Vremenska složenost sledeće implementacije je O(logN).

```
void update( int idx, int left, int right, int k, int val )
{
   if ( left == right )
   {
      tree[ idx ] = val;
      return;
   }
   int mid = ( left + right ) / 2;
   if ( k <= mid ) update_tree( 2 * idx, left, mid, k, val );
   else update_tree( 2 * idx + 1, mid + 1, right, k, val );
   tree[ idx ] = merge( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}
```

Get

Funkcija get izvlači vrednost za neki interval iz segmentnog stabla. Pored idx, left, right, poziva se i za levu i desnu granicu traženog intervala (left_bound i right_bound). Vremenska složenost sledeće implementacije je O(log N).

```
tip_podatka get( int idx, int left, int right, int left_bound, int right_bound )
{
   if ( left_bound == left && right_bound == right ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( right_bound <= mid ) return get( 2 * idx, left, mid, left_bound, right_bound );
   else if ( left_bound > mid ) return get( 2 * idx + 1, mid + 1, right, left_bound, right_bound );
   else
   {
      tip_podatka ret1 = get( 2 * idx, left, mid, left_bound, mid );
      tip_podatka ret2 = get( 2 * idx + 1, mid + 1, right, mid + 1, right_bound );
      return merge( ret1, ret2 );
   }
}
```

Primena segmentnog stabla

Kao što smo rekli, segmentno stablo je samo struktura koja može da pomogne i njegova primena (kao i način primene) nije jasno određena i razlikuje se od problema do problema. Sledeći problemi predstavljaju samo neke od najznačajnijih i najrazličitijih primena segmentnog stabla.

Maximum Sum

izvor: http://www.spoj.com/problems/KGSS/

Tekst zadatka

Dat je niz od N članova (2 <= N <= 10^5) čiji su članovi pozitivni prirodni brojevi manji od 10^8. Date su dve operacije:

- U i x (1 <= i <= N, 0 <= x <= 10^8) – na poziciju i u nizu staviti broj x
- Q x y (1 <= x < y <= N) – naći zbir dva najveća člana niza koji se nalaze u [x, y]

Odgovoriti na svaki upit tipa Q.

Ulaz sadrži broj N, u sledećem redu N članova niza. Treći red sadrži Q i potom Q upita u svakom redu po jedan. Upiti su gore navedenog oblika.

Izlaz sadrži odgovor na svaki upit tipa Q u novom redu.

```
5
1 2 3 4 5
6
Q 2 4
Q 2 5
U 1 6
Q 1 5
U 1 7
Q 1 5

7
9
11
12
```

Objašnjenje

Svaki čvor segmentnog stabla koje treba da napravimo nad datim nizom sadržaće dva najveća broja iz niza na intervalu koji taj čvor obuhvata. Jasno je da za čvorove koji obuhvataju jedan član niza važi da drugi broj treba da bude nula (ili neki negativni broj).

Kod

```
#include <cstdio>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 100005;
const int size = 131072 + 5;

int n, m, queue[ maxn ], ans1, ans2, tempQ[ 5 ];

pair < int, int > tree[ 2 * size ];

pair < int, int > merge_nodes( pair < int, int > A, pair < int, int > B )
{
tempQ[ 0 ] = A.first; tempQ[ 1 ] = A.second;
tempQ[ 2 ] = B.first; tempQ[ 3 ] = B.second;
sort( tempQ, tempQ + 4 );
return make_pair( tempQ[ 3 ], tempQ[ 2 ] );
}

void init_tree( int idx, int left, int right )
{
if ( left == right )
{
tree[ idx ] = make_pair( queue[ left ], 0 );
return;
}
int mid = ( left + right ) / 2;
init_tree( 2 * idx, left, mid );
init_tree( 2 * idx + 1, mid + 1, right );
tree[ idx ] = merge_nodes( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

void update_tree( int idx, int left, int right, int k, int val )
{
if ( left == right )
{
tree[ idx ] = make_pair( val, 0 );
return;
}
int mid = ( left + right ) / 2;
if ( k <= mid ) update_tree( 2 * idx, left, mid, k, val );
else update_tree( 2 * idx + 1, mid + 1, right, k, val );
tree[ idx ] = merge_nodes( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

void add( int A )
{
if ( A > ans1 )
{
ans2 = ans1;
ans1 = A;
}
else if ( A > ans2 ) ans2 = A;
```

```
}

void get_tree( int idx, int left, int right, int left_bound, int right_bound )
{
if ( left_bound == left && right_bound == right )
{
add( tree[ idx ].first );
add( tree[ idx ].second );
return;
}
int mid = ( left + right ) / 2;
if ( right_bound <= mid ) get_tree( 2 * idx, left, mid, left_bound, right_bound );
else if ( left_bound > mid ) get_tree( 2 * idx + 1, mid + 1, right, left_bound, right_bound );
else
{
get_tree( 2 * idx, left, mid, left_bound, mid );
get_tree( 2 * idx + 1, mid + 1, right, mid + 1, right_bound );
}
}

int main()
{
scanf( "%d", &n );
for ( int i = 0; i < n; i++ ) scanf( "%d", &queue[ i ] );
scanf( "%d", &m );

init_tree( 1, 0, n - 1 );
getchar();
while ( m-- )
{
char temp = getchar();
if ( temp == 'U' )
{
int i, x;
scanf( "%d %d", &i, &x );
update_tree( 1, 0, n - 1, i - 1, x );
}
else
{
int x, y;
scanf( "%d %d", &x, &y );
ans1 = 0;
ans2 = 0;
get_tree( 1, 0, n - 1, x - 1, y - 1 );
printf( "%d\n", ans1 + ans2 );
}
getchar();
}

return 0;
}
```

Snjeguljica

izvor: Hrvatska informatička olimpijada 2013. godine

Tekst zadatka

Snežana živi sa N patuljaka (2 <= N <= 200000). Svakog dana, Snežana natera patuljke da stanu u vrstu po visini. Za potrebe zadatka, pretpostavimo da su visine patuljaka 1, 2,..., N (ne postoje dva patuljka iste visine). Kako patuljci ne umeju sami da se poređaju, Snežana im pomaže postavljajući upite:

- 1 x y (1 <= x, y <= N, x ≠ y) – patuljci na pozicijama x i y treba da zamene mesta
- 2 a b (1 <= a <= b <= N) – da li se patuljci sa visinama a, a + 1, ..., b nalaze na uzastopnim mestima u vrsti (u bilo kojem poretku)

Odgovoriti na svaki upit tipa 2 sa „YES“ za da, odnosno „NO“ za ne (bez navodnika).

Ulaz u prvom redu sadrži N i M (2 <= M <= 200000), broj patuljaka i broj upita. U drugom redu nalaze se brojevi od 1 do N koji predstavljaju početni raspored patuljaka. Potom se u M redova nalazes upiti gore navedenog oblika.

Izlaz sadrži „YES“ ili „NO“ za svaki upit tipa 2, u svakom redu po jedan odgovor.

```
7 7
4 7 3 5 1 2 6
2 1 7
1 3 7
2 4 6
2 4 7
2 1 4
1 1 4
2 1 4

YES
NO
YES
NO
YES
```

Objašnjenje

Neka je dati niz, niz x. Napravimo prvo niz pos na sledeći način: pos[x[i]] = i. Očigledno niz pos predstavlja poziciju patuljka neke visine. Kada će patuljci sa visinama od a do b zauzimati uzastopne pozicije? Ako nađemo najmanji i najveći indeks patuljaka čija je visina izmedju a i b, neka su to L i R, jasno je da će patuljci zauzimati uzastopne pozicije ako važi da je R – L + 1 jednako sa brojem patuljaka, odnosno b – a + 1. Kada ovo znamo, možemo konstruisati segmentno stablo tako da svaki čvor stabla, recimo da obuhvata interval [i, j], sadrži minimalnu i maksimalnu poziciju patuljaka čija je visina iz [i, j]. Funkcija get_tree poziva se za date visine a i b, i vraća minimum i maksimum niza pos na intervalu [a, b].

Kod

```
#include <cstdio>
#include <algorithm>

using namespace std;

const int maxn = 2e5 + 5;
const int tree_size = 1 << 19;

int n, m, x[ maxn ], pos[ maxn ], a, b, type;

pair < int, int > tree[ tree_size ];

void init_tree( int idx, int left, int right )
{
   if ( left == right )
   {
      tree[ idx ] = make_pair( pos[ left ], pos[ left ] );
      return;
   }
   int mid = ( left + right ) / 2;
   init_tree( 2 * idx, left, mid );
   init_tree( 2 * idx + 1, mid + 1, right );
   tree[ idx ].first = max( tree[ 2 * idx ].first, tree[ 2 * idx + 1 ].first );
   tree[ idx ].second = min( tree[ 2 * idx ].second, tree[ 2 * idx + 1 ].second );
}

void update_tree( int idx, int left, int right, int k )
{
   if ( left == right )
   {
      tree[ idx ] = make_pair( pos[ left ], pos[ left ] );
      return;
   }
   int mid = ( left + right ) / 2;
   if ( k <= mid ) update_tree( 2 * idx, left, mid, k );
   else update_tree( 2 * idx + 1, mid + 1, right, k );
   tree[ idx ].first = max( tree[ 2 * idx ].first, tree[ 2 * idx + 1 ].first );
   tree[ idx ].second = min( tree[ 2 * idx ].second, tree[ 2 * idx + 1 ].second );
}

pair < int, int > get_tree( int idx, int left, int right, int L, int R )
{
   if ( left == L && right == R ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( R <= mid ) return get_tree( 2 * idx, left, mid, L, R );
   else if ( L > mid ) return get_tree( 2 * idx + 1, mid + 1, right, L, R );
   else
   {
      pair < int, int > ret1 = get_tree( 2 * idx, left, mid, L, mid );
      pair < int, int > ret2 = get_tree( 2 * idx + 1, mid + 1, right, mid + 1, R );
      return make_pair( max( ret1.first, ret2.first ), min( ret1.second, ret2.second ) );
   }
}
```

```
int main()
{
   scanf( "%d %d", &n, &m );
   for ( int i = 0; i < n; i++ )
   {
      scanf( "%d", &x[ i ] );
      x[ i ]--;
      pos[ x[ i ] ] = i;
   }

   init_tree( 1, 0, n - 1 );
   while ( m-- )
   {
      scanf( "%d %d %d", &type, &a, &b );
      a--; b--;
      if ( type == 1 )
      {
         swap( pos[ x[ a ] ], pos[ x[ b ] ] );
         swap( x[ a ], x[ b ] );
         update_tree( 1, 0, n - 1, x[ a ] );
         update_tree( 1, 0, n - 1, x[ b ] );
      }
      else
      {
         pair < int, int > curr = get_tree( 1, 0, n - 1, a, b );
         if ( curr.first - curr.second == b - a ) printf( "YES\n" );
         else printf( "NO\n" );
      }
   }

   return 0;
}
```

Can you answer these queries III

izvor: http://www.spoj.com/problems/GSS3/

Tekst zadatka

Dat je niz A od N elemenata (1 <= N <= 50000) koji čine celi brojevi između -10000 i 1000. Nad nizom treba izvesti M operacija (0 <= M <= 50000). Operacije su sledećeg oblika:

- 0 x y (1 <= x <= N, -10000 <= y <= 10000) – na poziciju x staviti y
- 1 x y (1 <= x <= y <= N) – naći maksimalni zbir podniza uzastopnih članova na intervalu [x, y]

Ulaz sadrži broj N, u sledećem redu niz A. U trećem redu nalazi se M, a potom M upita oblika ili 0 ili 1.

Izlaz sadrži redom odgovore na svaki upit tipa 1 u novom redu.

```
4
1 2 3 4
4
1 1 3
0 3 -3
1 2 4
1 3 3

6
4
-3
```

Objašnjenje

Čvorove segmentnog stabla treba organizovati tako da čvor (recimo da obuhvata interval [L, R]) sadrži:

- leftMax (najveću sumu uzastopnih počev od kraja L)
- rightMax (najveću sumu uzastopnih počev od kraja R)
- all (sumu svih članova na intervalu)
- value (maksimalni zbir nekog podniza uzastopnih na intervalu, odnosno rešenje za interval [L, R])

Neka je trenutni čvor V, i neka su njegovi sinovi A i B. Tada je V.all jednako zbiru A.all i B.all. V.leftMax može biti ili A.leftMax ili zbir A.all i B.leftMax. Slično za V.rightMax. V.value biće ili A.value ili B.value ili deo u sredini odnosno A.rightMax + B.leftMax.

Kod

```
#include <cstdio>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 50005;
const int size = 65536 + 5;

struct node
{
   int leftMax, rightMax, value, all;
};

node tree[ 2 * size ];

int n, m, queue[ maxn ];

node merge_nodes( node A, node B )
{
   node ret;
   ret.all = A.all + B.all;
   ret.leftMax = max( A.all + B.leftMax, A.leftMax );
   ret.rightMax = max( B.all + A.rightMax, B.rightMax );
   ret.value = max( max( A.value, B.value ), A.rightMax + B.leftMax );
   return ret;
}
```

```
void init_tree( int idx, int left, int right )
{
   if ( left == right )
   {
      tree[ idx ].leftMax = tree[ idx ].rightMax = tree[ idx ].value = tree[ idx ].all = queue[ left ];
      return;
   }
   int mid = ( left + right ) / 2;
   init_tree( 2 * idx, left, mid );
   init_tree( 2 * idx + 1, mid + 1, right );
   tree[ idx ] = merge_nodes( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

void update_tree( int idx, int left, int right, int k, int val )
{
   if ( left == right )
   {
      tree[ idx ].leftMax = tree[ idx ].rightMax = tree[ idx ].value = tree[ idx ].all = val;
      return;
   }
   int mid = ( left + right ) / 2;
   if ( k <= mid ) update_tree( 2 * idx, left, mid, k, val );
   else update_tree( 2 * idx + 1, mid + 1, right, k, val );
   tree[ idx ] = merge_nodes( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

node get_tree( int idx, int left, int right, int left_bound, int right_bound )
{
   if ( left_bound == left && right_bound == right ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( right_bound <= mid ) return get_tree( 2 * idx, left, mid, left_bound, right_bound );
   else if ( left_bound > mid ) return get_tree( 2 * idx + 1, mid + 1, right, left_bound, right_bound );
   node tmp1, tmp2;
   tmp1 = get_tree( 2 * idx, left, mid, left_bound, mid );
   tmp2 = get_tree( 2 * idx + 1, mid + 1, right, mid + 1, right_bound );
   node ans = merge_nodes( tmp1, tmp2 );
   return ans;
}

int main()
{
   scanf( "%d", &n );
   for ( int i = 0; i < n; i++ ) scanf( "%d", &queue[ i ] );
   scanf( "%d", &m );

   init_tree( 1, 0, n - 1 );
   while ( m-- )
   {
      int x, y, type;
      scanf( "%d %d %d", &type, &x, &y );
      if ( type == 0 ) update_tree( 1, 0, n - 1, x - 1, y );
      else printf( "%d\n", get_tree( 1, 0, n - 1, x - 1, y - 1 ).value );
   }

   return 0;
}
```

Frequent values

izvor: http://www.spoj.com/problems/FREQUENT/

Tekst zadatka

Dat je niz A od N brojeva (1 <= N <= 10^5) u neopadajućem redosledu čiji su članovi celi brojevi po apsolutnoj vrednosti manji od 10^5. Treba odgovoriti na Q upita (1 <= Q <= 10^5) oblika x y (1 <= x <= y <= N), koji je to broj koji se najčešće pojavljuje na intervalu [x, y].

Ulaz sadrži u prvom redu N i Q, u sledećem niz A, a u sledećih Q redova nalaze se po dva broja.

Izlaz sadrži Q redova, u svakom redu po jedan broj, odgovor na upit.

```
10 3
-1 -1 1 1 1 1 3 10 10 10
2 3
1 10
5 10

1
4
3
```

Objašnjenje

Na početku, važno je da pomenemo da je niz sortiran, pa se iste vrednosti uvek nalaze u grupi. Pretpostavimo da imamo upit od i do j. Neka je $P_{i,j}$ broj pojavljivanja najučestalijeg broja u datom intervalu. Podelimo interval na dva dela, [i, k] i [k + 1, j]. Lako je primetiti sledeće:

- Ako je a_k razlicito od a_{k+1} tada je $P_{i,j} = \max(P_{i,k} , P_{k+1,j})$
- Ako je $a_k = a_{k+1}$:

  $P_{i,j} = \max(P_{i,k} , P_{k+1,j} ,$
  broj pojavljivanja a_k u [i, k] + broj pojavljivanja a_{k+1} u [k + 1, j])

Sada možemo konstruisati segmentno stablo tako da svaki čvor sadrži:

- Broj pojavljivanja najučestalijeg broja u svom intervalu (cnt)
- Krajnji levi broj i broj njegovog pojavljivanja (leftValue, leftCnt)
- Krajnji desni broj i broj njegovog pojavljivanja (rightValue, rightCnt)

Inicijalizaciju segmentnog stabla obavljamo na sledeći način:

leftValue[i, j] predstavlja leftValue čvora u segmentnom stablu koji obuhvata interval [i, j], isto vazi i za druge vrednosti čvorova (rightValue, ...).

Za svaki čvor koji obuhvata interval oblika [i, i], jasno je da je cnt = leftCnt = rightCnt = 1, leftValue = rightValue = a_i. Ako smo inicijalizovali čvorove koji obuhvataju intervale [i, k] i [k + 1, j], mozemo da inicijalizujemo čvor koji obuhvata interval [i, j]. Očigledno je da je

leftValue[i, j] = leftValue[i, k] i da je rightValue[i, j] = rightValue[k + 1, j]. Ako je leftValue[i, k] = leftValue[k + 1, j], tj. ako je $a_i = a_{k+1}$ jasno je da se od i do k+1 u nizu nalazi isti broj (jer je niz sortiran) pa je leftCnt [i, j] = leftCnt[i, k] + leftCnt[k + 1, j], u suprotnom je leftCnt[i, j] = leftCnt[i, k]. Na isti način se odredjuje i rightCnt. I poslednje, broj pojavljivanja najučestalijeg broja:

- Ako je rightValue[i, k] razlicito od leftValue[k + 1, j], tj. ako je a_k razlicito od a_{k+1}, vidimo da delovi [i, k] i [k + 1, j] nemaju istih brojeva, pa je cnt[i , j] = max(cnt[i, k], cnt[k + 1, j]).
- U suprotnom, cnt[i, j] = max(cnt[i, k], cnt[k + 1, j], rightCnt[i, k] + leftCnt[k + 1, j]).

Preostaje nam jos odgovaranja na upite koje je, kada posedujemo sve informacije iz segmentnog stabla, veoma jednostavno. Sve sto treba jeste da delimo interval dok ne dobijemo intervale za koje imamo podatke iz segmentnog stabla, a potom da slozimo te podatke na isti nacin kao da inicijalizujemo segmentno stablo nad tim čvorovima.

Kod

```
#include <cstdio>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 100005;
const int size = 131072 + 5;

struct node
{
   int leftCnt, rightCnt, cnt, leftValue, rightValue;
};

node tree[ 2 * size ];

int n, q, queue[ maxn ];

node merge_nodes( node A, node B )
{
   node ret;
   ret.leftValue = A.leftValue;
   ret.rightValue = B.rightValue;
   if ( A.leftValue == B.leftValue ) ret.leftCnt = A.leftCnt + B.leftCnt;
   else ret.leftCnt = A.leftCnt;
   if ( B.rightValue == A.rightValue ) ret.rightCnt = B.rightCnt + A.rightCnt;
   else ret.rightCnt = B.rightCnt;
   ret.cnt = max( A.cnt, B.cnt );
   if ( A.rightValue == B.leftValue ) ret.cnt = max( ret.cnt, A.rightCnt + B.leftCnt );
   return ret;
}
```

```
void init_tree( int idx, int left, int right )
{
   if ( left == right )
   {
      tree[ idx ].leftCnt = tree[ idx ].rightCnt = tree[ idx ].cnt = 1;
      tree[ idx ].leftValue = tree[ idx ].rightValue = queue[ left ];
      return;
   }
   int mid = ( left + right ) / 2;
   init_tree( 2 * idx, left, mid );
   init_tree( 2 * idx + 1, mid + 1, right );
   tree[ idx ] = merge_nodes( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

node get_tree( int idx, int left, int right, int leftBound, int rightBound )
{
   if ( leftBound == left && rightBound == right ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( rightBound <= mid ) return get_tree( 2 * idx, left, mid, leftBound, rightBound );
   if ( leftBound > mid ) return get_tree( 2 * idx + 1, mid + 1, right, leftBound, rightBound );
   node tmp1, tmp2;
   tmp1 = get_tree( 2 * idx, left, mid, leftBound, mid );
   tmp2 = get_tree( 2 * idx + 1, mid + 1, right, mid + 1, rightBound );
   node ans;
   ans = merge_nodes( tmp1, tmp2 );
   return ans;
}

int main()
{
   scanf( "%d %d", &n, &q );
   for ( int i = 0; i < n; i++ ) scanf( "%d", &queue[ i ] );

   init_tree( 1, 0, n - 1 );
   while ( q-- )
   {
      int x, y;
      scanf( "%d %d", &x, &y );
      printf( "%d\n", get_tree( 1, 0, n - 1, x - 1, y - 1 ).cnt );
   }

   return 0;
}
```

Lazy propagation

Nekad se od nas traži da promenu izvedemo ne samo na jednom članu već na nekom intervalu. Tada koristimo tehniku „lazy propagation“. Opet treba napomenuti da ne postoji tačno određen način primene ove tehnike već da zavisi od toga šta se od nas traži.

Ideja iza lazy propagation tehnike jeste da update-ujemo samo ono što moramo i kad nam to treba. Na primer, zamislimo segmentno stablo koje pokriva niz od 20 članova. Ako update-ujemo interval [0, 19], update-ujemo samo vrednost korena stable (jer taj čvor obuhvata [0, 19]), a za sinove koji obuhvataju [0, 9] i [10, 19] kažemo da treba tek da budu update-ovani (markiramo ih na neki način). Kada budemo imali neki upit, spuštajući se niz stablo prema tom upitu kada stanemo na neki čvor koji je markiran, update-ujemo ga, a njegove sinove markiramo. Pretpostavimo da imamo upit za interval [5, 6], evo šta bi se dešavalo:

- [0, 9] : markiramo [0, 4] i [5, 9], a update-ujemo vrednost [0, 9]
- [5, 9] : markiramo [5, 7] i [8, 9], a update-ujemo [5, 9]
- [5, 7] : markiramo [5, 6] i [7, 7], a update-ujemo [5, 7]
- [5, 6] : markiramo [5, 5] i [6, 6], a update-ujemo [5, 6]

Kako je [5, 6] update-ovano imamo njegovu vrednost koja nam se tražila, a čvorovi koju su markirani zahvataju sledeće intervale [10, 19], [0, 4], [8, 9], [7, 7], [5, 5], [6, 6].

Način na koji markiramo i update-ujemo čvorove zavisi od onoga šta nam se traži u zadatku.

Copying data

izvor: http://codeforces.com/problemset/problem/292/E

Tekst zadatka

Data su dva niza A i B, oba sa N članova (1 <= N <= 100000). Nad nizom treba izvesti M upita (1 <= M <= 100000), sledećeg tipa:

- Kopirati deo niza A dužine k počev od pozicije x u niz B počev od pozicije y, odnosno $B_{y+q} = A_{x+q}$ (0 <= q < k). Ovaj upit zadat je sa 1 x y k i upit je validan (sigurno obuhvata postojeće elemente).
- Odrediti vrednost na poziciji x u nizu B, odnost B_x. Ovaj upit zadat je sa 2 x.

Ulaz sadrži u prvom redu N i M, potom u drugom redu niz A, a u trećem niz B. Potom sledi M upita tipa ili 1 ili 2.

Izlaz sadrži odgovor na svaki upit tipa 2, svaki odgovor štampati u novom redu.

```
5 10
1 2 0 -1 3
3 1 5 -2 0
2 5
1 3 3 3
2 5
2 4
2 1
1 2 1 4
2 1
2 4
1 4 2 1
2 2

0
3
-1
3
2
3
-1
```

Objašnjenje

Da bi smo odredili trenutnu vrednost člana niza B, neophodno nam je da znamo koji je to upit tipa 1 koji je poslednji zahvatio posmatranog člana. Dakle, ako je upit tipa 1, ubacimo u niz x i y tog upita, a potom update-ujmo interval [y, y + k - 1] za vrednost indeksa na koji smo postavili x i y u niz. Svaki čvor segmentnog stabla sadrži indeks onog upita tipa 1 koji je poslednji zahvatio ceo interval koji obuhvata taj čvor stable. Ako je upit tipa dva, odredimo indeks koji je poslednji zahvatio člana sa indeksom x. Kako? Spuštajmo se po segmentnom stablu, počev od čvora sa indeksom 1, tako što uvek idemo u onog sina koji zahvata indeks x. Kako svaki čvor sadrži indeks upita koji je zahvatio ceo njegov interval, jasno je da na svakom čvoru na koji zgazimo imamo indeks nekog upita koji je zahvatio traženog člana. Kako je nama potreban poslednji upit, potrebno je da nadjemo maksimum na tom putu u segmentnom stablu. Ukoliko je indeks koji smo dobili -1, to znači da ni jedan upit tipa 1 nije zahvatio traženog člana, pa samo štampamo B_x. Ako nije, recimo da su Q.x i Q.y, x i y onog upita koji je poslednji zahvatio našeg člana sa indeksom x, lako možemo da dodjemo do zaključka da treba da štampamo član niza A sa indeksom Q.x + x – Q.y. U ovom zadatku promenu ne spuštamo sa oca na sinove, već sve promene kombinujemo na putu do traženog čvora.

Kod

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
#include <string>
#include <vector>

using namespace std;

typedef long long lld;
typedef pair < int, int > pii;
typedef pair < string, int > psi;

const int maxn = 1e5 + 5;
const int tree_size = ( 1 << 18 ) + 5;

int n, m, A[ maxn ], B[ maxn ], x, y, k, type, tree[ tree_size ];

vector < pii > Q;

void init_tree()
{
   memset( tree, -1, sizeof( tree ) );
}

void update_tree( int idx, int left, int right, int L, int R, int val )
{
   if ( left == L && right == R )
   {
      tree[ idx ] = val;
      return;
   }
   int mid = ( left + right ) / 2;
   if ( R <= mid ) update_tree( 2 * idx, left, mid, L, R, val );
   else if ( L > mid ) update_tree( 2 * idx + 1, mid + 1, right, L, R, val );
   else
   {
      update_tree( 2 * idx, left, mid, L, mid, val );
      update_tree( 2 * idx + 1, mid + 1, right, mid + 1, R, val );
   }
}

int get_tree( int idx, int left, int right, int k )
{
   if ( left == right ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( k <= mid ) return max( tree[ idx ], get_tree( 2 * idx, left, mid, k ) );
   else return max( tree[ idx ], get_tree( 2 * idx + 1, mid + 1, right, k ) );
}

void copy_array( int stA, int stB, int len )
{
   Q.push_back( make_pair( stA, stB ) );
   update_tree( 1, 0, n - 1, stB, stB + len - 1, Q.size() - 1 );
}
```

```
int solve( int st )
{
   int idx = get_tree( 1, 0, n - 1, st );
   if ( idx == -1 ) return B[ st ];
   return A[ Q[ idx ].first + st - Q[ idx ].second ];
}

int main()
{
   scanf( "%d %d", &n, &m );
   for ( int i = 0; i < n; i++ ) scanf( "%d", &A[ i ] );
   for ( int i = 0; i < n; i++ ) scanf( "%d", &B[ i ] );

   init_tree();
   while ( m-- )
   {
      scanf( "%d", &type );
      if ( type == 1 )
      {
         scanf( "%d %d %d", &x, &y, &k );
         copy_array( x - 1, y - 1, k );
      }
      else
      {
         scanf( "%d", &x );
         printf( "%d\n", solve( x - 1 ) );
      }
   }

   return 0;
}
```

Multiples of 3

izvor: http://www.spoj.com/problems/MULTQ3/

Tekst zadatka

Dat je niz od N brojeva (1 <= N <= 100000) čiji su svi članovi na početku jednaki 0. Na vama je da izvedete sledeće operacije:

- 0 A B (0 <= A <= B <= N – 1) – povećati sve brojeve na intervalu [A, B] za 1
- 1 A B (0 <= A <= B <= N – 1) – odgovoriti na pitanje koliko ima brojeva iz intervala [A, B] deljivih sa 3

Ulaz sadrži u prvom redu brojeve N i Q (1 <= Q <= 100000). U sledećih Q redova nalaze se upiti tipa 1 ili 2.

Izlaz sadrži odgovore na svaki upit tipa 1, svaki odgovor u novom redu.

```
4 7
1 0 3
0 1 2
0 1 3
1 0 0
0 0 3
1 3 3
1 0 3

4
1
0
2
```

Objašnjenje

Neka svaki čvor segmentnog stabla čuva koliko brojeva iz njegovog intervala ima ostatak 0, 1 ili 2 pri deljenju sa 3, zvaćemo ih niz ostataka tog čvora. Kako sada update-ujemo interval? Spuštajući se niz segmentno stablo, kada se nađemo na čvoru koji je ceo zahvaćen promenom, rotiram njegov niz ostataka u desno za 1. Ali to nije sve. Neka taj čvor ima indeks k. Povećamo shift[k] za 1. Shift[k] predstavlja koliko je mesta u desno rotiran niz ostataka čvora segmentnog stabla sa indeksom k. Ovo nam je potrebno kada budemo hteli da dodjemo do nekih potomaka čvora k. Tada ćemo iskoristiti to da su oni rotirani za shift[k]. Kada smo uradili ovo, spustimo promenu trenutnog čvora na njegove sinove, odnosno rotiramo nizove ostataka sinova za shift[k], i uvećamo shift[2 * k] i shift[2 * k + 1] za shift[k], a potom kažemo da je shift[k] = 0. Sada se na čvoru k nalazi tačno stanje za interval koji obuhvata čvor k, i ukoliko budemo imali upit koji zahvata ceo taj interval, iskoristićemo to stanje. Kako sada dobijamo rešenje za neki interval? Opet spuštajući se niz stablo, kada stanemo na čvor koji potpuno upada u interval, recimo da opet ima indeks k, znamo da u nizu ostataka imamo tačno stanje za taj interval. Međutim, taj interval mogao je biti rotiran kao deo intervala nekog čvora koji je predak čvoru k. Sve što treba jeste da znamo sumu svih shift-ova na putu od čvora 1 do čvora k. I recimo da je suma svih šiftova po modulu 3 jednaka s, tada ćemo na rešenje dodati tree[k][(3 – s) % 3], kao da smo rotirali niz ostataka čvora k za s.

Kod

```
#include <cstdio>
#include <algorithm>

using namespace std;

const int size_tree = ( 1 << 18 ) + 5;

int n, q, type, a, b, tree[ size_tree ][ 3 ], shift[ size_tree ];

void init_tree( int idx, int left, int right )
{
   if ( left == right )
   {
      tree[ idx ][ 0 ] = 1;
      return;
   }
   int mid = ( left + right ) / 2;
   init_tree( 2 * idx, left, mid );
   init_tree( 2 * idx + 1, mid + 1, right );
   tree[ idx ][ 0 ] = tree[ 2 * idx ][ 0 ] + tree[ 2 * idx + 1 ][ 0 ];
}

void rotate_node( int idx )
{
   swap( tree[ idx ][ 0 ], tree[ idx ][ 1 ] );
   swap( tree[ idx ][ 0 ], tree[ idx ][ 2 ] );
   shift[ idx ] = ( shift[ idx ] + 1 ) % 3;
}

void update_tree( int idx, int left, int right, int L, int R )
{
   if ( left == L && right == R )
   {
      rotate_node( idx );
      return;
   }
   int mid = ( left + right ) / 2;
   if ( R <= mid ) update_tree( 2 * idx, left, mid, L, R );
   else if ( L > mid ) update_tree( 2 * idx + 1, mid + 1, right, L, R );
   else
   {
      update_tree( 2 * idx, left, mid, L, mid );
      update_tree( 2 * idx + 1, mid + 1, right, mid + 1, R );
   }
   for ( int i = 0; i < shift[ idx ]; i++ )
   {
      rotate_node( 2 * idx );
      rotate_node( 2 * idx + 1 );
   }
   shift[ idx ] = 0;
   for ( int i = 0; i < 3; i++ ) tree[ idx ][ i ] = tree[ 2 * idx ][ i ] + tree[ 2 * idx + 1 ][ i ];
}
```

```
int get_tree( int idx, int left, int right, int L, int R, int k )
{
   if ( left == L && right == R ) return tree[ idx ][ ( 3 - k ) % 3 ];
   int mid = ( left + right ) / 2;
   if ( R <= mid ) return get_tree( 2 * idx, left, mid, L, R, ( k + shift[ idx ] ) % 3 );
   else if ( L > mid ) return get_tree( 2 * idx + 1, mid + 1, right, L, R, ( k + shift[ idx ] ) % 3 );
   else return get_tree( 2 * idx, left, mid, L, mid, ( k + shift[ idx ] ) % 3 ) +
            get_tree( 2 * idx + 1, mid + 1, right, mid + 1, R, ( k + shift[ idx ] ) % 3 );
}

int main()
{
   scanf( "%d %d", &n, &q );

   init_tree( 1, 0, n - 1 );
   while ( q-- )
   {
      scanf( "%d %d %d", &type, &a, &b );
      if ( ! type ) update_tree( 1, 0, n - 1, a, b );
      else printf( "%d\n", get_tree( 1, 0, n - 1, a, b, 0 ) );
   }

   return 0;
}
```

Upiti nad stablom

Ovo je nešto drugačiji tip problem od svih prethodnih, prvenstveno zato što zahteva primenu segmentnog stabla „nad“ stablom. Najbolje je da odmah predjemo na primer i sve objasnimo na primeru.

Query on a tree

izvor: http://www.spoj.com/problems/QTREE/)

Tekst zadatka

Dato je stablo (neusmeren povezan graf bez ciklusa) sa N čvorova i granama obeleženim sa 1, 2, ..., N – 1. Od vas se traži da sprovedete sledeće operacije:

- CHANGE i x – premeniti cenu i-te grane na x
- QUERY a b – koliko košta najskuplja grana na putu od a do b?

Odgovoriti na svaki QUERY.

Ulaz sadrći broj N (1 <= N <= 10000). U sledećih N – 1 redova nalaze se tri broja a, b, c (1 <= a, b <= N, 1 <= c <= 1000000). U i + 1 redu nalazi se opis i-te grane koja spaja a i b i ima cenu c. Potom se nalaze upiti oblika CHANGE i x ili QUERY a b. Unos upita završava se linijom koja sadrži DONE.

Izlaz sadrži odgovor na svaki QUERY u novom redu.

```
3
1 2 1
2 3 2
QUERY 1 2
CHANGE 1 3
QUERY 1 2
DONE

1
3
```

Kako sada nad zadatim stablom konstruisati segmentno stablo i šta će čvorovi stabla da čuvaju? Pre svega, potrebno je dato stablo razložiti na lance i od tih lanaca napraviti niz. To se radi pomoću Heavy-light dekompozicije.

Heavy-light dekompozicija (HLD)

HLD je način rastavljanja stabla na lance i to na sledeći način:

- Odaberimo neki čvor i postavimo ga za koren celog stabla.
- Neka je subtree[idx] broj čvorova u stablu koje za koren ima čvor idx. Niz subtree možemo da odredimo koristeći dfs u složenosti O(N), gde je N broj čvorova stabla.
- Recimo da čvor idx pripada lancu sa indeksom takođe idx. Istom lancu pripadaće i onaj sin čvor idx koji ima najveću vrednost subtree-a. Svi ostali sinovi činiće nove lance koji će imati indekse po njima.

HLD se koristi i za nalaženje LCA (najdubljeg zajedničkog pretka dva čvora). Evo kako izgleda implementacija algoritma HLD:

```
void hld( int curr, int prev, int type )
{
   chain[ curr ] = type;
   int idxMax = -1;
   for ( int i = 0; i < adj[ curr ].size(); i++ )
   {
      edge x = adj[ curr ][ i ];
      if ( x.node == prev ) continue;
      if ( idxMax == -1 || subtree[ x.node ] > subtree[ idxMax ] ) idxMax = x.node;
   }
   if ( idxMax != -1 ) hld( idxMax, curr, type );
   for ( int i = 0; i < adj[ curr ].size(); i++ )
   {
      edge x = adj[ curr ][ i ];
      if ( x.node == prev || x.node == idxMax ) continue;
      hld( x.node, curr, x.node );
   }
}
```

Objašnjenje

Vratimo se sada na zadatak. Odaberimo čvor sa indeksom 0 kao koren stabla i napravimo HLD. Kako pravimo HLD, potrebno je i sve čvorove indeksirati tako da se u istom lancu nalaze čvorovi za uzastopnim indeksima (ovo je vrlo lako postići, samo dodavanjem idx[curr] = cnt; cnt++; ispred chain[curr] = type; u nizu idx biće sadržani novi indeksi). Takođe, potrebno je odrediti val[idx] što predstavlja cenu puta od čvora idx do njegovog oca. Segmentno stablo treba konstruisati tako da svaki čvor čuva maksimum, uz uslov da čvorovi koji obuhvataju samo jednog člana sa indeksom idx[i] imaju vrednost val[i]. Ili drugačije rečeno, svaki čvor segmentnog stabla čuva maksimalnu cenu puta od nekog čvora do njegovor oca za sve čvorove iz njegovog intervala, s tim što čvorovi nisu indeksirani kao na ulazu, već koristimo njihove nove indekse dobijene u HLD-u. Kada imamo upit CHANGE sve što treba jeste da nadjemo za dati put ko je sin, a ko otac od ta dva čvora, i update-ujemo segmentno stablo tako da čvor koji obuhvata [idx[sin], idx[sin]] bude jednak novoj vrednosti puta. U slučaju QUERY-ja potrebno je put od jednog do drugog čvora podeliti na puteve za koje važi da su svi čvorovi na putu u istom lancu. Za svaki deo puta, treba u segmentnom stablu naći maksimalnu vrednost neke grane na tom delu, koristeći funkciju get. Kada znamo sve maksimume, potrebno je naći njihov maksimum i to je rešenje.

Kod

```
#include <cstdio>
#include <algorithm>
#include <vector>
#include <cstring>

using namespace std;

const int maxn = 1e4 + 5;
const int size = 16384 + 5;

struct edge
{
   int node, cost, id;
   edge()
   {
   }
   edge( int _node, int _cost, int _id )
   {
      node = _node;
      cost = _cost;
      id = _id;
   }
};

vector < vector < edge > > adj;

int n, cnt, tree[ 2 * size ], father[ maxn ], subtree[ maxn ], chain[ maxn ], idx[ maxn ], val[ maxn ],
match[ maxn ], depth[ maxn ];

void dfs( int curr, int prev )
{
   father[ curr ] = prev;
   subtree[ curr ] = 1;
   for ( int i = 0; i < adj[ curr ].size(); i++ )
   {
      edge x = adj[ curr ][ i ];
      if ( x.node == prev ) continue;
      match[ x.id ] = x.node;
      val[ x.node ] = x.cost;
      father[ x.node ] = curr;
      depth[ x.node ] = depth[ curr ] + 1;
      dfs( x.node, curr );
      subtree[ curr ] += subtree[ x.node ];
   }
}

void heavyLight( int curr, int prev, int type )
{
   idx[ curr ] = cnt;
   cnt++;
   chain[ curr ] = type;
   int idxMax = -1;
   for ( int i = 0; i < adj[ curr ].size(); i++ )
   {
      edge x = adj[ curr ][ i ];
```

```
      if ( x.node == prev ) continue;
      if ( idxMax == -1 || subtree[ x.node ] > subtree[ idxMax ] ) idxMax = x.node;
   }
   if ( idxMax != -1 ) heavyLight( idxMax, curr, type );
   for ( int i = 0; i < adj[ curr ].size(); i++ )
   {
      edge x = adj[ curr ][ i ];
      if ( x.node == prev || x.node == idxMax ) continue;
      heavyLight( x.node, curr, x.node );
   }
}

void update( int idx, int left, int right, int k, int val )
{
   if ( left == right )
   {
      tree[ idx ] = val;
      return;
   }
   int mid = ( left + right ) / 2;
   if ( k <= mid ) update( 2 * idx, left, mid, k, val );
   else update( 2 * idx + 1, mid + 1, right, k, val );
   tree[ idx ] = max( tree[ 2 * idx ], tree[ 2 * idx + 1 ] );
}

int get( int idx, int left, int right, int leftBound, int rightBound )
{
   if ( left >= leftBound && right <= rightBound ) return tree[ idx ];
   int mid = ( left + right ) / 2;
   if ( rightBound <= mid ) return get( 2 * idx, left, mid, leftBound, rightBound );
   else if ( leftBound > mid ) return get( 2 * idx + 1, mid + 1, right, leftBound, rightBound );
   int tmp1 = get( 2 * idx, left, mid, leftBound, rightBound );
   int tmp2 = get( 2 * idx + 1, mid + 1, right, leftBound, rightBound );
   return max( tmp1, tmp2 );
}

int solve( int x, int y )
{
   int ret = 0;
   if ( x == y ) return 0;
   while ( chain[ x ] != chain[ y ] )
   {
      if ( depth[ chain[ x ] ] < depth[ chain[ y ] ] ) swap( x, y );
      ret = max( ret, get( 1, 0, n - 1, idx[ chain[ x ] ], idx[ x ] ) );
      x = father[ chain[ x ] ];
   }
   if ( depth[ x ] > depth[ y ] ) swap( x, y );
   if ( x != y ) ret = max( ret, get( 1, 0, n - 1, idx[ x ] + 1, idx[ y ] ) );
   return ret;
}
```

```
int main()
{
   scanf( "%d", &n );
   adj.resize( n );
   for ( int i = 0; i < n - 1; i++ )
   {
      int x, y, cost;
      scanf( "%d %d %d", &x, &y, &cost );
      x--;   y--;
      adj[ x ].push_back( edge( y, cost, i ) );
      adj[ y ].push_back( edge( x, cost, i ) );
   }
   cnt = 0;
   dfs( 0, -1 );
   heavyLight( 0, -1, 0 );
   for ( int i = 0; i < n; i++ ) update( 1, 0, n - 1, idx[ i ], val[ i ] );
   char query[ 15 ];
   scanf( "%s", query );
   while ( query[ 0 ] != 'D' )
   {
      int x, y;
      scanf( "%d %d", &x, &y );
      if ( query[ 0 ] == 'C' ) update( 1, 0, n - 1, idx[ match[ x - 1 ] ], y );
      else printf( "%d\n", solve( x - 1, y - 1 ) );
      scanf( "%s", query );
   }

   return 0;
}
```