درخت بازه‌ای

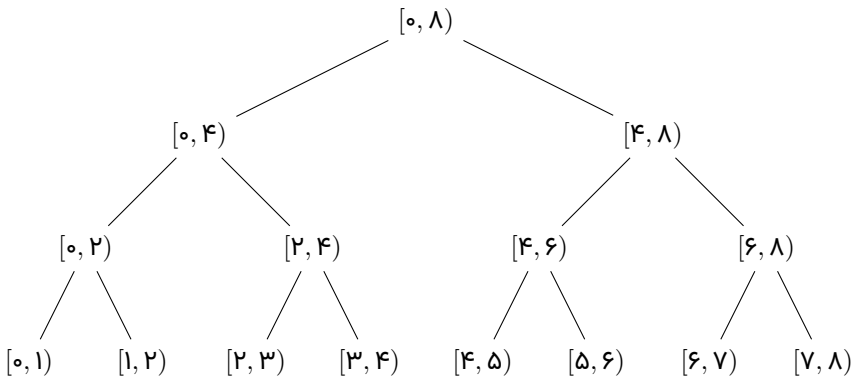
با استفاده از fenwick تونستیم به پرسش‌های عوض کردن یک خانه و پیدا کردن جمع یک بازه جواب بدیم. دیدیم که fenwick برای اینکه جواب پرسش یک بازه رو پیدا کنه، باید پرسش این ویژگی رو داشته باشه که partially جواب داده بشه. یعنی بتونیم جواب برای بازه $[l, r]$ رو از روی جواب $[۱, l-۱]$ و $[۱, r]$ پیدا کنیم. برای مثال این ویژگی رو پرسش‌های جمع و ضرب دارند ولی خیلی از عملیات‌ها مثل ماکسیمم و gcd این ویژگی رو ندارند . در ادامه مسئله‌ای که حل می‌کنیم رو توضیح می‌دیم.

آرایه‌ای از اعداد داریم. تعدادی پرسش به یکی از حالات زیر می‌آید:

- change $idx\ value$: مقدار خانه idx را به $value$ تغییر بده.
- getmax $L\ R$: ماکسیمم بازه $[L, R]$ را خروجی بده.

هدف انجام هر پرسش در $\mathcal{O}(\log n)$ می‌باشد که n طول آرایه است.

فرض کنید که $n = ۸$ است. به درخت زیر توجه کنید.



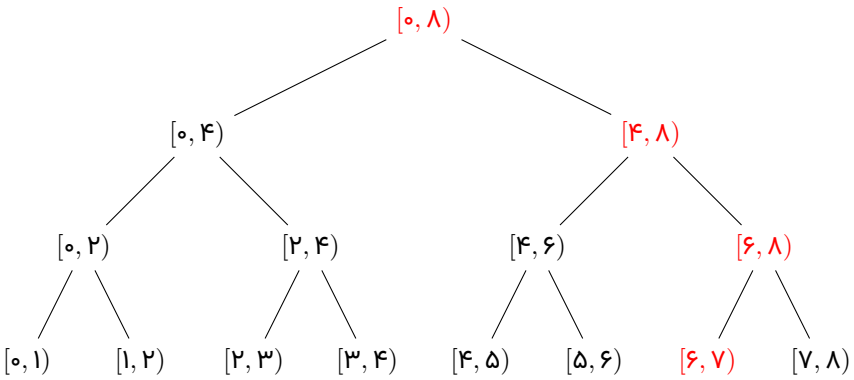
هر راس این درخت دودویی کامل یک بازه از آرایه را نشان می‌دهد. برگ‌های آن بازه‌های تک عضوی هستند و بازه هر راس اجتماع بازع دو بچه‌ی آن است. تعداد راس‌های این درخت دقیقا $۲n - ۱$ است و ارتفاع آن نیز $\lceil \log_n rceil$ می‌باشد.

در هر راس ماکسیمم بازه مربوط به آن را ذخیره می‌کنیم. حال بررسی می‌کنیم هر تغییر در آرایه چه تغییری در درخت می‌دهد و برای جواب دادن به هر پرسش باید چه کنیم.

تغییر یک خانه

ابتدا فرض کنید در مثال $n = ۸$ مقدار خانه ۶ تغییر کند. بدیهی است که راس‌هایی از درخت ممکن است مقدارشان تغییر کند که ۶ در بازه‌ی مربوط به آن حضور داشته باشد. این خانه‌ها در شکل زیر علامت زده شده‌اند.

ویژگی این درخت این است که هر عضو از آرایه در مسیر برگ آن به ریشه حضور دارد. اگر از ریشه به پایین بیاییم، هر اندیسی در بازه مربوط به دقیقا یکی از بچه‌های ریشه وجود دارد. به طور بازگشتی می‌توان همه راس‌هایی که دچار تغییر می‌شوند را آپدیت کرد. اگر راس دلخواهی از درخت باید تغییر کند، ابتدا بچه‌ی مربوطه‌اش را تغییر می‌دهیم و سپس مقدار آن را ماکسیمم دو بچه‌اش قرار می‌دهیم.



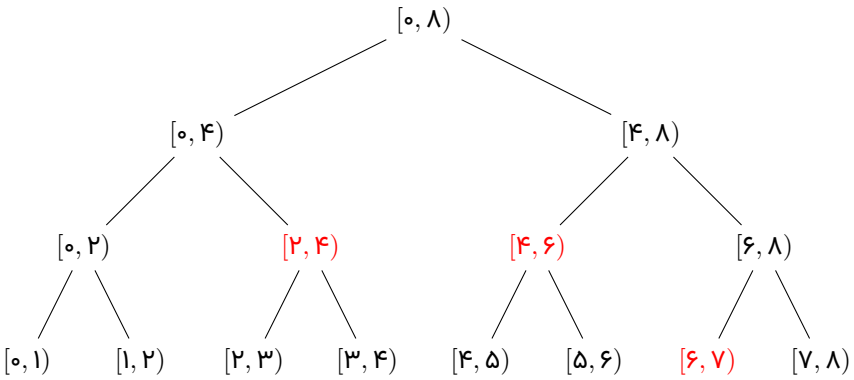
شبه‌کد آن به این صورت است.

```
struct Node {
    Node* lc;
    Node* rc;
    int value;

    void change(int L, int R, int idx, int new_value) {
        if (L + 1 == R) {
            value = new_value;
            return;
        }
        int mid = (L + R) >> 1;
        if (idx < mid)
            lc->change(L, mid, idx, new_value);
        else
            rc->change(mid, R, idx, new_value);
        value = max(lc->value, rc->value);
    }
};
```

گرفتن یک بازه

دوباره در مثال $n = ۸$ فرض کنید ماکسیمم بازه $[۲, ۷)$ را می‌خواهیم پیدا کنیم. از ریشه شروع می‌کنیم و در هر راسی که هستیم به دنبال پیدا کردن ماکسیمم اشتراک راس و بازه مربوطه هستیم. اگر بازه درخت کاملا درون بازه پرسش قرار داشت، مقدار ذخیره شده راس درخت را برمی‌گردانیم و در غیر این صورت به صورت بازگشتی از دو بچه‌ی آن ماکسیمم اشتراکشان را پیدا می‌کنیم و سپس ماکسیمم دو مقدار بازگردانده شده را برمی‌گردانیم.



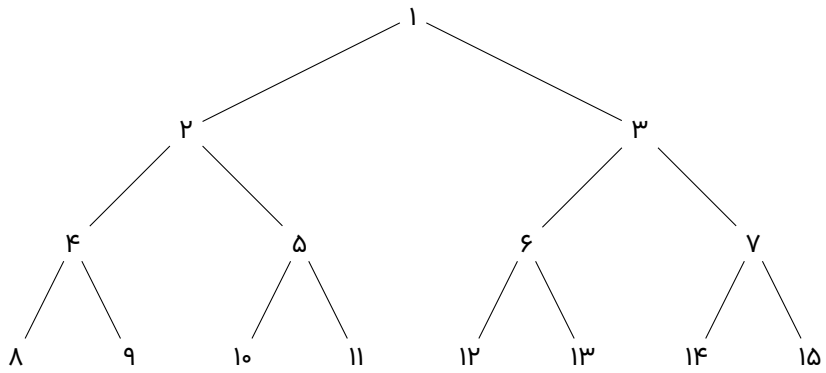

```
const int INF = 1e9;

struct Node {
    Node* lc;
    Node* rc;
    int value;

    int get(int L, int R, int st, int en) {
        if (R <= st or en <= L)
            return -INF;
        if (st <= L and R <= en)
            return value;
        int mid = (L + R) >> 1;
        return max(lc->get(L, mid, st, en), rc->get(mid, R, st, en));
    }
};
```

جزئیات پیاده‌سازی

شماره رئوس درخت مانند شکل زیر است. خاصیت این شماره‌گذاری این است که شماره‌ی بچه‌های راس x برابر $2x$ و $2x+1$ می‌باشد.




```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAXN = 1e5 + 10;

int seg[4 * MAXN];

int get(int id, int L, int R, int l, int r) {
    if (r <= L or R <= l)
        return;
    if (l <= L and R <= r)
        return seg[id];
    int mid = (L + R) >> 1;
    return max(get(2 * id + 0, L, mid, l, r),
               get(2 * id + 1, mid, R, l, r));
}

void change(int id, int L, int R, int idx, int val) {
    if (idx < L or R <= idx)
        return;
    if (L + 1 == R) {
        seg[id] = val;
        return;
    }
    int mid = (L + R) >> 1;
    change(2 * id + 0, L, mid, idx, val);
    change(2 * id + 1, mid, R, idx, val);
```

```
    seg[id] = max(seg[2 * id + 0], seg[2 * id + 1]);
}

int a[MAXN];

int main() {
    int n, q;
    cin >> n >> q;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        change(1, 0, n, i, a[i]);
    }
    while (q--) {
        int type;
        cin >> type;
        if (type == 1) {
            int idx, val;
            cin >> idx >> val;
            change(1, 0, n, idx, val);
        }
        else {
            int l, r;
            cin >> l >> r;
            cout << get(1, 0, n, l, r) << '\n';
        }
    }
}
```