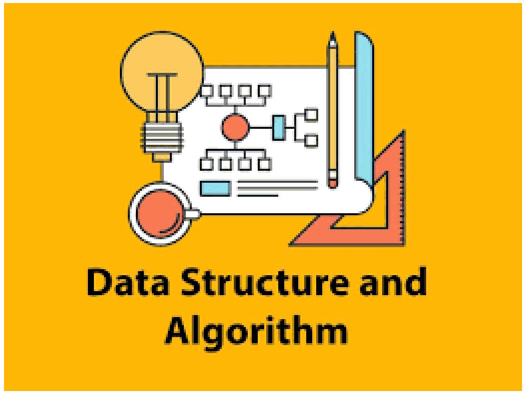
ĐỘI TUYỂN OLP Tin học PTIT 2025

CHỮA BÀI TUẦN 2 (Oct 6 – Oct 12)



Contest cá nhân – Thứ 7

- Bài 1: Hợp số (Nguyên lý inclusive – exclusive)
- Bài 2: Tổng chuỗi dãy số (Cây Segment tree)
- Bài 3: Chữ số 2, 3, 5, 7 (QHĐ + nhân ma trận)
- Bài 4: Nhà máy (QHD trên cây)

Bài 1: Hợp số

- Liệt kê các số nguyên tố trong phạm vi [1 50]
- Áp dụng nguyên tắc inclusive-exclusive cho việc đếm số lượng các số chia hết cho một trong những số A, B, C, D, E, …
- [L R] = [1 15], với 3 số nguyên tố 2, 3, 5
  - Số các số chia hết cho 2 là 15/2 = 7
  - Số các số chia hết cho 3 là 15/3 = 5
  - Số các số chia hết cho 5 là 15/5 = 3
  - Số các số chia hết cho 2 và 3 là 15/6 = 2
  - Số các số chia hết cho 2 và 5 là 15/10 = 1
  - Số các số chia hết cho 3 và 5 là 15/15 = 1
  - Số các số chia hết cho 2, 3 và 5 là 15/30 = 0
  - Số các số chia hết cho ít nhất một số bằng = 7+5+3 – 2 – 1 – 1 + 0 = 11

Bài 2: Tổng chuỗi dãy số

- Diễn giải công thức truy hồi của Dx:

$$D_x = \sum_{k=1}^{x} A_k \binom{x-k+2}{2} = \frac{1}{2}\left[(x+2)(x+1)\sum_{k\le x} A_k \;-\; (2x+3)\sum_{k\le x} kA_k \;+\; \sum_{k\le x} k^2 A_k\right]$$

- Dùng 3 cây segment trees quản lý tổng của
  - Cây 1 = sum(A_k) → S1
  - Cây 2 = sum(k*A_k) → S2
  - Cây 3 = sum(k*k*A_k) → S3
- Truy vấn tính tổng đoạn (1, x) trên 3 cây segment trees

Ans = ½ [(x+2)(x+1)*S1 - (2x+3)*S2 + S3]

- Thao tác update A[p] = val, point update cho 3 cây

Bài 3: Chữ số 2, 3, 5, 7

- Bài toán yêu cầu đếm số x có không quá N chữ số, sao cho số lượng các chữ số 2,3,5,7 có parity mod 2 lần lượt bằng A, B, C, D.
- Như vậy, trạng thái chỉ phụ thuộc vào parity (mod 2) của 4 chữ số này, tức có tổng cộng 2^4 = 16 trạng thái.

- Subtask 1: dp[i][cnt2][cnt3][cnt5][cnt7]

Bài 3: Chữ số 2, 3, 5, 7

- Bài toán yêu cầu đếm số x có không quá N chữ số, sao cho số lượng các chữ số 2,3,5,7 có parity mod 2 lần lượt bằng A, B, C, D.
- Như vậy, trạng thái chỉ phụ thuộc vào parity (mod 2) của 4 chữ số này, tức có tổng cộng 2^4 = 16 trạng thái.

- Gọi F[i][mask] là số lượng các số có i chữ số và trạng thái hiện tại là mask. Bitset trong mask thể hiện cho:
  - bit0 = parity(2)
  - bit1 = parity(3)
  - bit2 = parity(5)
  - bit3 = parity(7)
- Ta thêm 1 chữ số d mới, thu được F[i+1][mask’].

Bài 3: Chữ số 2, 3, 5, 7

- Công thức quy hoạch động:

$$f[i+1][mask'] = \sum_{mask} f[i][mask] \times \text{trans}(\text{mask} \to \text{mask'})$$

- Từ mask, ta có 10 cách chọn chữ số d để thu được mask', trong đó chỉ có chữ số 2, 3, 5, 7 làm thay đổi parity

- Ma trận chuyển trạng thái:

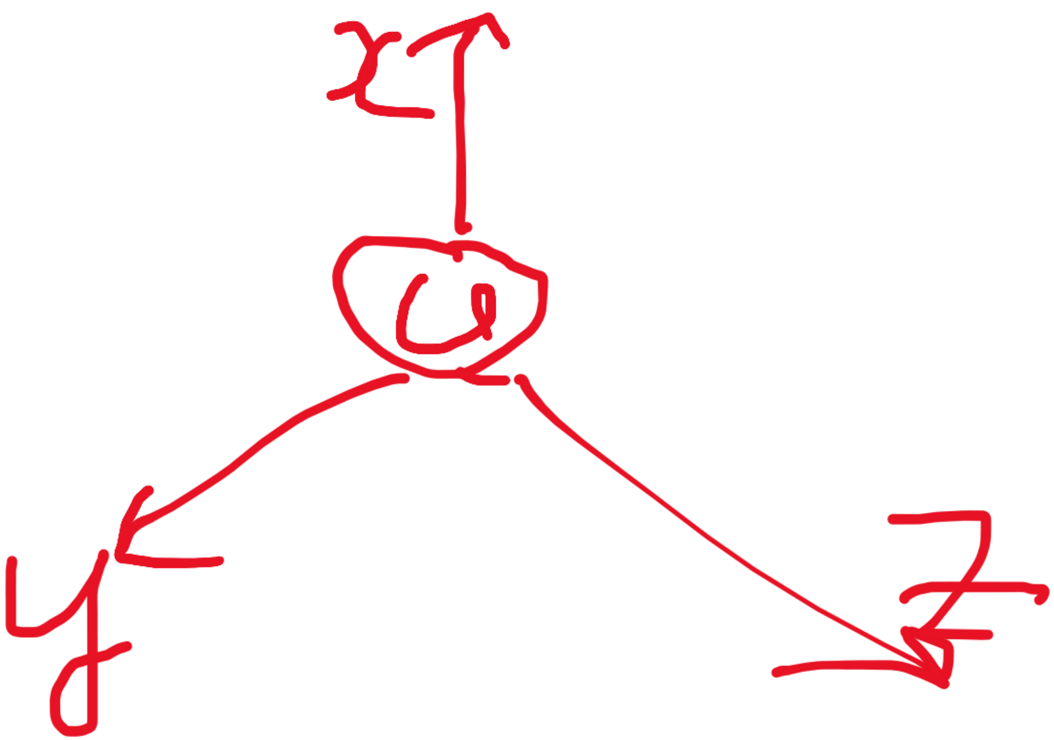
Trans[mask][mask'] bằng:

  - 1 nếu như mask' = mask XOR bit(d) với d = {2, 3, 5, 7}
  - 6 nếu như mask' = mask, thể hiện việc chọn d = 0, 1, 2, 4, 6, 8, 9
  - 0 trong các trường hợp khác

- Sử dụng nhân ma trận để tăng tốc độ cho N <= 10^{18}

Bài 4: Nhà máy

- **Đề bài:** Tìm 3 vị trí x, y, z cho UCLN gcd(x → y), gcd(y → z), gcd(z → x) > 1 và d(x, y) + d(y, z) + d(z, x) lớn nhất.



Bài 4: Nhà máy

- **Đề bài:** Tìm 3 vị trí x, y, z cho UCLN gcd(x → y), gcd(y → z), gcd(z → x) > 1 và d(x, y) + d(y, z) + d(z, x) lớn nhất.

- Với số C[i] <= 100000, mỗi C[i] chỉ chứa tối đa 7 thừa số nguyên tố → với mỗi thừa số nguyên tố p, ta xây dựng cây con G_p, nối các cạnh u, v nếu như gcd(C[u], C[v]) chia hết cho p

  → Sau đó xử lí quy hoạch động trên cây cho từng cây con G_p này

- Với mỗi đỉnh u, lấy u làm giao điểm của 3 đường đi x → y, y → z và z → x, ta cần duy trì 3 đường đi dài nhất từ u xuống dưới và 1 đường đi dài nhất từ u lên root
- Ans[u] = max(down[u][1] + down[u][2] + down[u][3],

  down[u][1] + down[u][2] + up[u]) * 2 – S[u] * 3

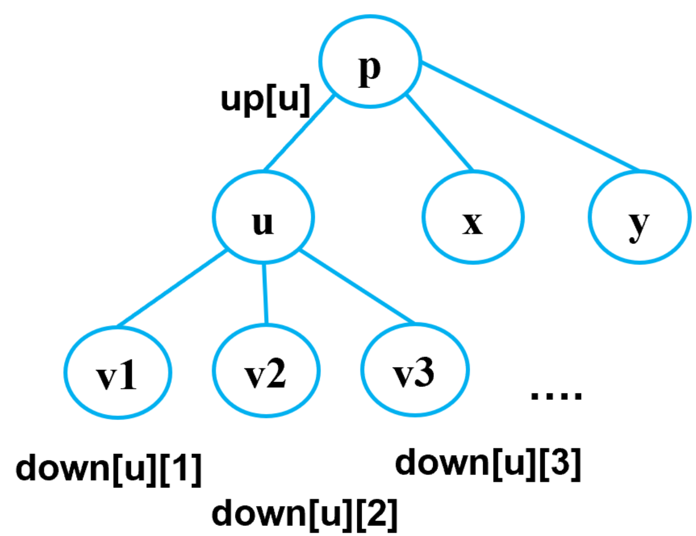
Bài 4: Nhà máy

- Ans[u] = max(down[u][1] + down[u][2] + down[u][3],
  down[u][1] + down[u][2] + up[u])
- DFS1 cho down[u][1 .. 2 .. 3]
- DFS2 cho up[u]: cần chọn 1 đường đi tốt nhất từ p lên phía root hoặc các con của p khác với đỉnh u.
  - u thuộc down[p][1]: nhánh con tốt nhất của p

  Update up[u] = max(up[u], down[p][2] + S[u])

  - U không thuộc down[p][1], ví dụ node x.

  Update up[x] = max(up[x], down[p][1] + S[u])



Bài 4: Nhà máy

- Với số C[i] <= 100000, mỗi C[i] chỉ chứa tối đa 7 thừa số nguyên tố
- → với mỗi thừa số nguyên tố p, ta xây dựng cây con G_p, nối các cạnh u, v nếu như gcd(C[u], C[v]) chia hết cho p

- Nếu không tách riêng từng G_p, các mảng quy hoạch động sẽ thêm chỉ số cho số nguyên tố, down[u][1][p] với p <= 7.
- Khi quy hoạch động, duyệt 7*7 cặp để để update cho đúng thành phần chung cho số nguyên tố p
- Cách cài đặt này sẽ phức tạp hơn một chút.

2025 ICPC vòng miền Nam

- D, C: anh Dương
  F: Nam + thầy Kiên

Problem D: The Alchemist's Diminishing Gold

Tóm tắt đề

Cho một số n và k thao tác. Tại mỗi thao tác, ta chọn một số n' là ước của n và thay $n = n'$ với xác suất bằng $\frac{1}{số\ ước\ của\ n}$. Tính giá trị kì vọng của n sau k thao tác với modulo $10^9 + 7$.

Problem D: The Alchemist's Diminishing Gold

- Phân tích n thành tích các thừa số nguyên tố.
- Giả sử $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_m^{a_m}$
- Giả sử sau k thao tác ta thu được giá trị:

$$n' = p_1^{b_1} \cdot p_2^{b_2} \cdot \dots \cdot p_m^{b_m}$$

với $b_i \leq a_i \; \forall \; i \leq m$

-> Đáp án của bài toán sẽ là $\prod_{n' \leq n, n' | n} P(n, n', k) \cdot n'$ với $P(n, n', k)$ là xác suất để đưa n về n' sau k thao tác

Problem D: The Alchemist's Diminishing Gold

Bài toán đưa về: Cho một mảng $a = \{a_1, a_2, \dots, a_m\}$ và một mảng $b = \{b_1, b_2, \dots, b_m\}$ với $b_i \le a_i \forall i \le m$. Tính xác suất để đưa a về b sau k thao tác, nếu tại mỗi thao tác ta có thể thay $a_i = a_i'$ với $a_i' \le a_i$.

Nhận xét: việc đưa a_i về b_i là độc lập với mỗi i, nên xác suất để chuyển mảng a về b sau k thao tác bằng $\prod_{i=1}^{m} p(a_i, b_i, k)$ với $p(x, y, k)$ là xác suất để chuyển x về y sau k thao tác.

Problem D: The Alchemist's Diminishing Gold

p(x, y, k) có thể được tính bằng quy hoạch động:

- Trường hợp cơ sở: p(x, x, 0) = 1
- Chuyển trạng thái: Với giá trị y ở thao tác thứ k, ta có thể chuyển y thành y' với $y' \le y$ ở thao tác k + 1 với xác suất $\frac{1}{y+1}$. Do đó với mỗi $y' \le y$, cập nhật: $\mathrm{p}(\mathrm{x}, \mathrm{y}', \mathrm{k}+1) \mathrel{+}= \frac{1}{y+1} \cdot p(x, y, k)$. Có thể tối ưu đoạn này bằng suffix sum để chuyển trạng thái trong $O(1)$.

Problem D: The Alchemist's Diminishing Gold

- Quay về bài toán ban đầu, với mỗi $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_m^{a_m}$, xét tất cả các $n' = p_1^{b_1} \cdot p_2^{b_2} \cdot \ldots \cdot p_m^{b_m}$ là giá trị của n sau khi thực hiện k thao tác.
- Đáp án của bài toán tăng thêm một lượng: $\mathrm{n}' \cdot \sum_{i=1}^{m} p(a_i, b_i, k)$
- Độ phức tạp: $O(k \cdot M^2 + \sqrt{n})$ với M là mũ lớn nhất trong các ước nguyên tố, M lớn nhất có thể lên đến 50 (2^{49}).

Problem C: Weighted Substring Queries

Tóm tắt đề

Cho một từ điển gồm N xâu phân biệt, xâu s_i có trọng số c_i. Cho một xâu T và Q truy vấn. Mỗi truy vấn có dạng [L, R] yêu cầu tính giá trị:

$$val(L, R) = \sum_{i=1}^{N} c_i \cdot k_i$$

với k_i là tần suất xuất hiện của xâu s_i trong khoảng [L, R] của xâu T.

Problem C: Weighted Substring Queries

- Giả sử với mỗi xâu s_i, ta có thể tìm tất cả các khoảng trong xâu T mà nó là xâu con thì bài toán trở thành: Cho n đoạn $[l_i, r_i, c_i]$ và q truy vấn, mỗi truy vấn có dạng [L, R] và ta cần tính tổng tất cả các c_i mà $L \le l_i$ và $r_i \le R$.
- Ta có thể giải quyết bài toán bằng cách xử lý truy vấn offline + BIT xử lý truy vấn tính tổng: Xử lý truy vấn theo L tăng dần. Ban đầu, với mỗi i, tăng vị trí r_i lên c_i. Trước mỗi truy vấn với L, cập nhật giảm các vị trí r_i có $l_i = L - 1$ đi c_i. Khi đó đáp án với mỗi truy vấn [L, R] sẽ là tổng trong khoảng từ L tới R.

Problem C: Weighted Substring Queries

- Vấn đề còn lại là làm thế nào để lấy được tất các khoảng mà mỗi s_i là xâu con trong T.
- Bài toán này có thể được xử lý sử dụng Aho-corasick: Xây một cây Trie dựa trên tập các xâu s_i với mỗi cạnh của cây biểu thị cho 26 kí tự. Ở mỗi nút của cây, đánh dấu nút hiện tại có phải là kết thúc của một xâu s_i nào đó hay không.
- Sau khi dựng xong Trie, tại mỗi nút của cây ta xây thêm các cạnh (gọi là suffix link) để nối một nút u với một nút v nào đó thỏa mãn: gọi p_u là xâu được tạo từ đường đi từ gốc đến u, p_v là xâu được tạo từ đường đi từ gốc đến v. Có cạnh nối từ u tới v nếu suffix của p_u là prefix của p_v, và độ dài của prefix (hay suffix) này là lớn nhất có thể.

Problem C: Weighted Substring Queries

Giả sử tập xâu ban đầu:

a

ab

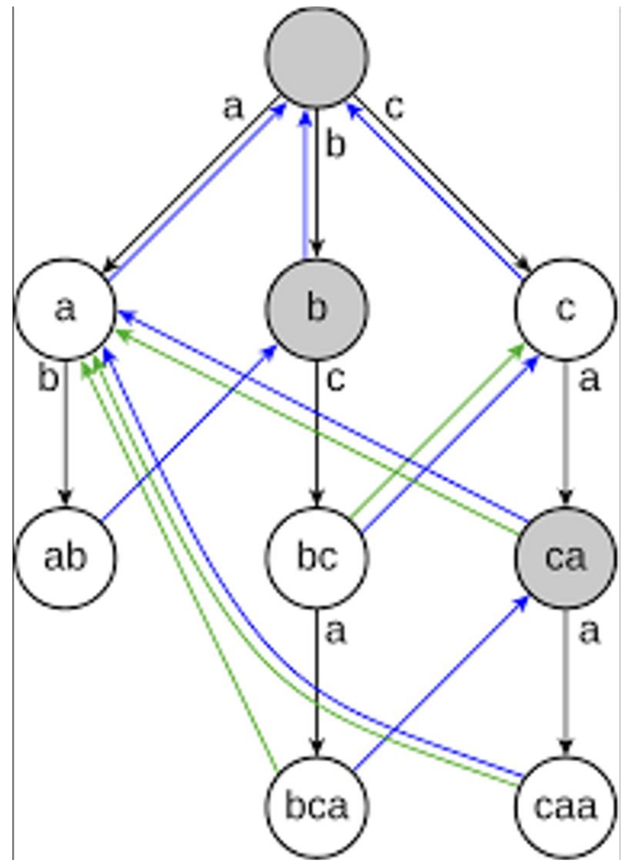
bc

bca

c

caa

Ta có hình ảnh của Trie sau khi đã xây các suffix link:



Problem C: Weighted Substring Queries

- Sau khi đã dựng Trie + suffix link, duyệt từng kí tự của T từ trái sang phải, đồng thời duy trì một con trỏ u ban đầu đặt ở gốc của cây. u sẽ di chuyển dựa trên kí tự tiếp theo của T.
- Giả sử đang ở vị trí i của xâu T. Nếu đỉnh u hiện tại là một nút "kết thúc" thì ta lấy ra xâu s_j ứng với nút đó, và thêm đoạn $\left[i - len(s_j) + 1, i, c_j\right]$ vào danh sách, sau đó đi theo suffix link ứng tại nút u để xem còn xâu s_j nào kết thúc ở i hay không.
- Khi duyệt các kí tự c của xâu T, nếu cạnh (u, c) tồn tại thì ta cho u đi qua cạnh này, nếu không ta cho u đi theo suffix link của đỉnh u để đến đỉnh v cho tới khi tồn tại cạnh (v, c) thì cho u đi qua cạnh này (đoạn này tiền xử lý bằng dp).

Problem C: Weighted Substring Queries

- to: Lưu Trie
- d[u][i]: Nút tiếp theo mà u đi tới nếu kí tự tiếp theo là i
- Term[u]: term[u] = u nếu u là đỉnh kết thúc tại của một xâu nào đó, nếu không thì term[u] = 0
- sl[u]: suffix link của đỉnh u

```cpp
void add_string(string &s){
    int u = 0;
    for(char c: s){
        if(!to[u][c - 'a'])
            to[u][c - 'a'] = ++cur;
        u = to[u][c - 'a'];
    }
    term[u] = u;
}


void push_sl(){
    queue<int> q;
    q.push(0);
    while(q.size()){
        int u = q.front(); q.pop();
        for(int i = 0; i < 26; ++i){
            int v = to[u][i];
            if(v){
                sl[v] = d[sl[u]][i];
                if(!term[v])
                    term[v] = term[sl[v]];
                d[u][i] = v;
                q.push(v);
            }else
                d[u][i] = d[sl[u]][i];

        }
    }
}
```

Problem C: Weighted Substring Queries

Duyệt T:

```
for(int i = 1; i <= T.size(); ++i){
    u = d[u][T[i] - 'a'];
    int v = term[u];
    // v là đỉnh kết thúc của một xâu nào đó
    while(v){
        // ...
        v = term[sl[v]];
    }
```

Độ phức tạp: $O(26 \cdot \sum_i len(s_i) + len(T) + K + qlog\big(len(T)\big)$ với K là tổng số khoảng mà các s_i là xâu con của T

Bài F: Power Absorption

Tóm tắt đề

- Cho n con quái vật , quái vật i xuất hiện từ thời gian L[i] đến thời gian R[i] có sức mạnh là P[i]
- Ban đầu power=1, có m câu hỏi câu hỏi thứ j có dạng t,D,A,F
  - Gọi E = 1 + (D * power + A) % F.
  - In ra ans là tổng sức mạnh của E con quái vật có sức mạnh nhỏ nhất xuất hiện tại thời điểm t.
  - Cập nhật power = ans .

- Giới hạn : 1 <= n, m, L[i], R[i], D, A, F, t <= 10^5 , 1<=P[i]<=1e7

Bài F: Power Absorption

Giải:

- Xét bài toán con tính tổng k con quái vật có P nhỏ nhất không quan trọng thời gian :
  - Sort các con quái vật theo P[i] , sau đó thêm theo thứ tự vào segment tree , mỗi nút lưu số lượng các con quái vật và tổng trọng số các con quái vật.
  - Khi đó ta có thể sử dụng kĩ thuật Walk on Segmenttree để query

Bài F: Power Absorption

Giải:

- Bây giờ vấn đề chỉ nằm ở thời gian t , ta có thể cải tiến bằng cách sử dụng Persistant Segment tree lưu lại mọi trạng thái tại mọi thời điểm.
- Khi truy vấn ta chỉ cần truy cập lại thời điểm t và walk tại thời điểm đó .
- Độ phức tạp : O(1e5 * log(1e5))

Bài F: thầy Kiên chữa bài

Tài liệu tham khảo: Persistent tree

- https://wiki.vnoi.info/algo/data-structures/persistent-data-structures
- https://codeforces.com/blog/entry/15890

Cây Persistent tree

- Ý tưởng: tích lũy nhiều cây segment tree lại với nhau.
- Mỗi version là một gốc (root) của cây segment tree
- Khi cập nhật một vị trí, ta chỉ tạo lại các node trên đường đi tới vị trí đó, phần còn lại dùng chung với version cũ.

→ Nhờ đó, ta lưu được lịch sử mọi trạng thái của dãy số.

- Ví dụ

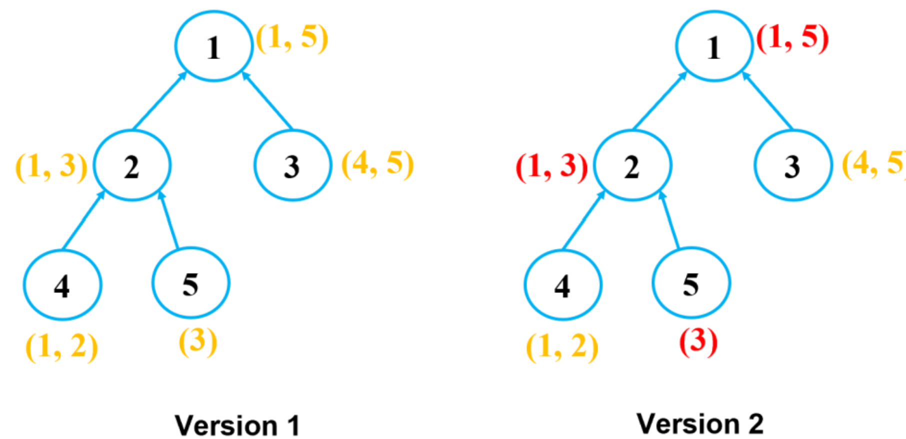
A[version 1] = (5 1 3 2 4)

A[version 2] = (5 1 6 2 4)

Cây Persistent tree

- A[version 1] = (5 1 3 2 4)
- A[version 2] = (5 1 6 2 4)
- cập nhật a[3] = 6

→ tạo version cây mới:

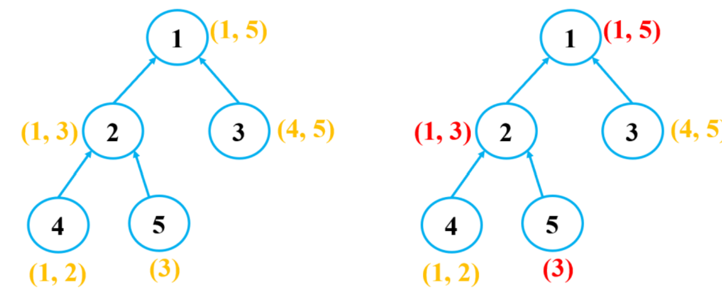


- Chỉ đường đi đến index 3 bị thay đổi.Các node khác được tái sử dụng.
  - root1 là version 1 (mảng cũ)
  - root2 là version 2 (mảng mới)
- Node mới chỉ tạo ra dọc đường: root (1) → 2([1..3]) → 5([3]).

Cây Persistent tree

```
struct Node {
    int cnt;
    LL sum;
    Node *l, *r;
}
```

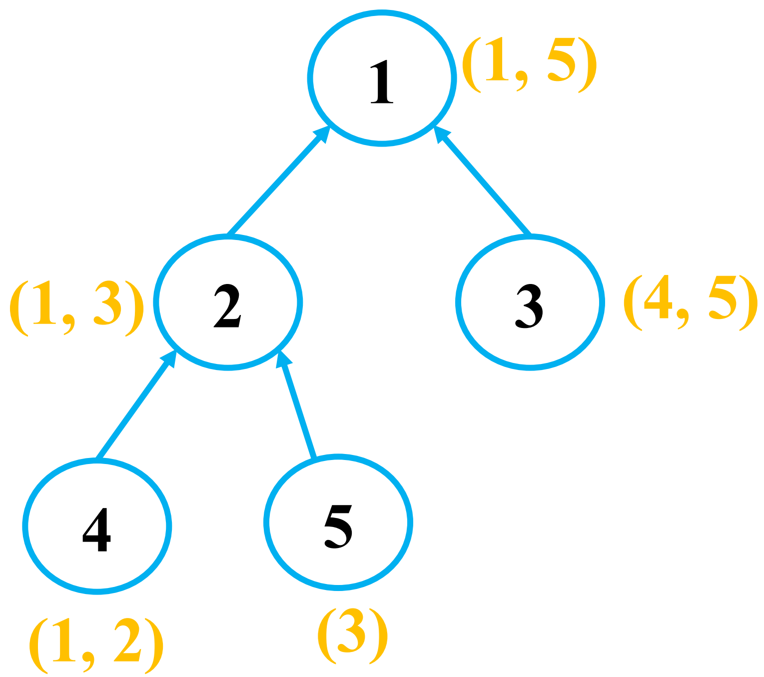
- Code cây segment tree bằng con trỏ
- Hàm update: root version mới từ root của version mới
- Cơ bản giống update thường của segment tree
- Chỉ các nút trên đường đi từ root cũ → lá được tạo node mới



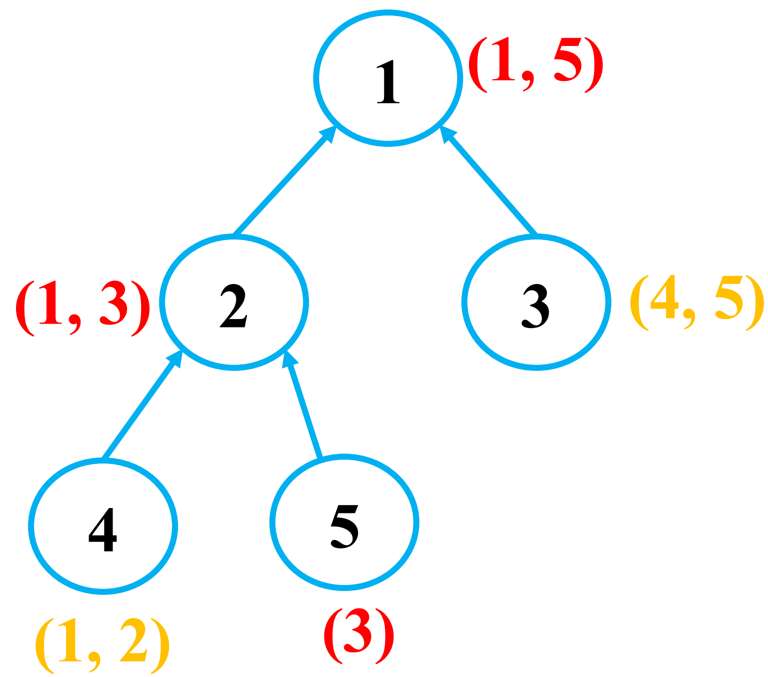
Version 1 Version 2

```
Node* cloneNode(Node* u) {
    if (!u) return new Node();
    return new Node(u);
}


Node* update(Node* u, int L, int R, int pos, int val) {
    Node* v = cloneNode(u);
    if (L == R) {
        v->cnt += 1;
        v->sum += val;
        return v;
    }
    int mid = (L + R) >> 1;
    if (pos <= mid)
        v->l = update(v->l, L, mid, pos, val);
    else
        v->r = update(v->r, mid + 1, R, pos, val);

    v->cnt = getCnt(v->l) + getCnt(v->r);
    v->sum = getSum(v->l) + getSum(v->r);
    return v;
}
```

Cây Persistent tree

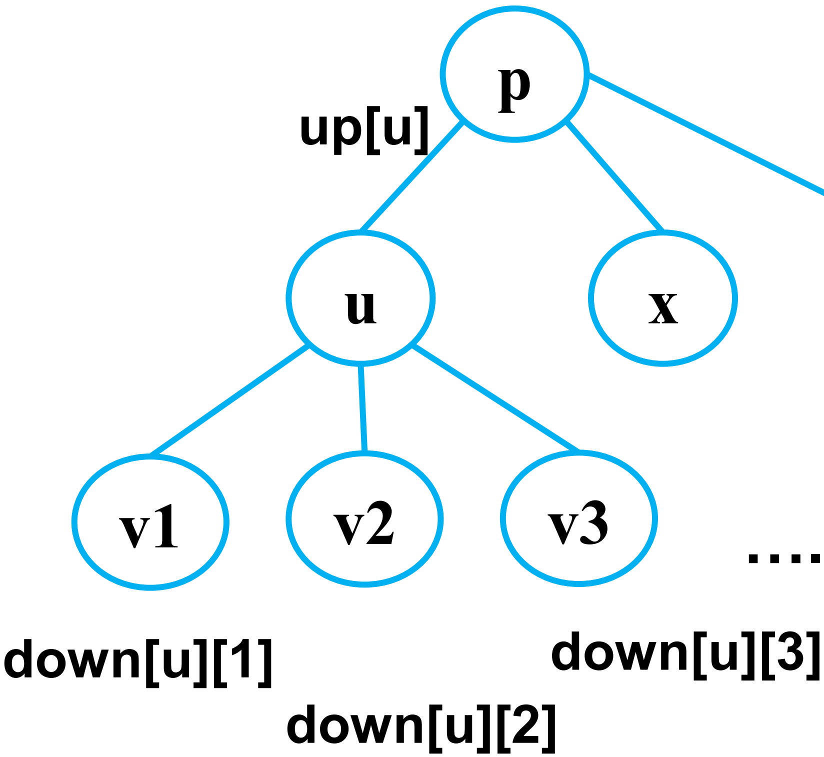


Version 1



Version 2

Cây Persistent tree



Cây Persistent tree

- Bài toán 1: Cho dãy số A[] và Q truy vấn,
  - truy vấn loại 1 là update A[p] = x,
  - truy vấn 2 yêu cầu tìm tổng K số nhỏ nhất của dãy?
- Ta xây dựng **Persistent tree** trên miền giá trị của dãy số A[].
- Tạo ra N cây segment tree, cây thứ i đại diện là root[i], quản lý đoạn A[1→i]
- mỗi nút lưu 2 thông tin: số lượng các số có trong miền quản lý (cnt) + tổng của chúng (sum).
- root[i] được xây dựng từ root[i-1] và update nút mới có pos = A[i],

giá trị (1, A[i])

Cây Persistent tree

- Bài toán 1: Cho dãy số A[] và Q truy vấn,
  - truy vấn loại 1 là update A[p] = x,
  - truy vấn 2 yêu cầu tìm tổng K số nhỏ nhất của dãy?

- **Persistent tree – thao tác truy vấn**
- Hàm truy vấn tính tổng K phần tử nhỏ nhất:
- Nếu K ≤ leftCnt (root->Left->cnt)

→ ta chỉ cần đi trái (vì K nhỏ nhất đều nằm bên trái).

Ngược lại → cộng leftSum và đi sang phải với K − leftCnt.

- Khi đến lá (một giá trị cụ thể): kết quả là K * value[lá]

Cây Persistent tree

- Bài toán 1: Cho dãy số A[] và Q truy vấn, truy vấn loại 1 là update A[p] = x, truy vấn 2 yêu cầu tìm tổng K số nhỏ nhất của dãy?

- **Persistent tree – thao tác update**
- Update A[p] = x, xóa phần tử cũ bằng update(A[p], -1), thêm phần tử bằng update(x, +1) trên root[N].

Bài F: Hấp thu năng lượng

- Quái vật xuất hiện tại [L, R] với năng lượng P

Sự kiện:

  - (P, +1): update(A[P], +1) trên root[L]
  - (P, -1): update(A[P], -1) trên root[R+1]

- Yêu cầu: tại mỗi mốc thời điểm T, tìm K phần tử nhỏ nhất trên miền giá trị [1 maxValue]
- Root[T] = cây version T, quản lý tất cả các quái tại thời điểm T
- Xử lí truy vấn tìm K phần tử nhỏ nhất trên miền giá trị [1 maxValue], giống như bài toán phía trên.

Cây Persistent tree - 2

- Bài toán 2: Cho dãy số A[] và Q truy vấn, mỗi truy vấn yêu cầu tìm tổng K số nhỏ nhất của đoạn [L R]?

Cây Persistent tree - 2

- Bài toán 2: Cho dãy số A[] và Q truy vấn, mỗi truy vấn yêu cầu tìm tổng K số nhỏ nhất của đoạn [L R]?

- **Persistent tree:** Root[i] chứa thông tin (cnt, sum) cho từng prefix[i], tức toàn bộ phần tử a[1..i], có thể sử dụng thêm nén dữ liệu nếu giá trị lớn.
- Truy vấn một đoạn [L, R]: xử lí bằng prefix[R] – prefix[L-1]

Vì root[R] = quản lý miền giá trị của a[1..R],

root[L-1] quản lý miền giá trị của a[1..L-1]

- Cách lấy hiệu 2 version rootR và rootL:
  - cnt_in_segment = seg[rootR].cnt - seg[rootL].cnt;
  - sum_in_segment = seg[rootR].sum - seg[rootL].sum;
- “lấy hiệu” ở đây là để so sánh giữa 2 cây version này có bn phần tử

Cây Persistent tree - 2

- Bài toán 2: Cho dãy số A[] và Q truy vấn, mỗi truy vấn yêu cầu tìm tổng K số nhỏ nhất của đoạn [L R]?

- **Persistent tree**
- Hàm truy vấn tính tổng K phần tử nhỏ nhất của hiệu rootR và rootL:
- Tại mỗi node, tính số phần tử ở nửa con trái của hiệu 2 cây:
  - leftCnt;
  - leftSum;
- Nếu K ≤ leftCnt → ta chỉ cần đi trái (vì K nhỏ nhất đều nằm bên trái).

Ngược lại → cộng leftSum và đi sang phải với K − leftCnt.

- Khi đến lá (một giá trị cụ thể): kết quả là K * value[lá]

QUESTIONS & ANSWERS