



Cấu trúc dữ liệu nâng cao

LẬP TRÌNH THI ĐẦU

Đỗ Phan Thuận

Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 9 tháng 10 năm 2015

Nội dung



- Cấu trúc dữ liệu các tập không giao nhau (Union-Find/ Disjoint Set) và các ứng dụng
- Truy vấn trong khoảng (Range query)
- Cây phân đoạn (Segment/Interval Tree)

Union-Find



- Cho n phần tử
- Cần quản lý vào các tập không giao nhau
- Mỗi phần tử ở trong đúng 1 tập
- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $collections = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$
- Hai toán tử hiệu quả: $find(x)$ và $union(x, y)$.

Union-Find



- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $find(x)$ trả về phần tử đại diện của tập chứa x
 - ▶ $find(1) = 1$
 - ▶ $find(4) = 1$
 - ▶ $find(3) = 5$
 - ▶ $find(5) = 5$
 - ▶ $find(6) = 5$
 - ▶ $find(2) = 2$
- a và b thuộc cùng một tập khi và chỉ khi $find(a) == find(b)$

Union-Find



- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $union(x, y)$ trộn tập chứa x và tập chứa y vào nhau
 - ▶ $union(4, 2)$
 - ▶ $collections = \{1, 2, 4\}, \{3, 5, 6\}$
 - ▶ $union(3, 6)$
 - ▶ $collections = \{1, 2, 4\}, \{3, 5, 6\}$
 - ▶ $union(2, 6)$
 - ▶ $collections = \{1, 2, 3, 4, 5, 6\}$

Cài đặt Union-Find



- Hợp nhất nhanh với kỹ thuật nén đường (Quick Union with path compression)
- Cực kỳ dễ cài đặt
- Cực kỳ hiệu quả

```
struct union_find {
    vector<int> parent;
    union_find(int n) {
        parent = vector<int>(n);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }

        // find and union
    };
};
```

Cài đặt Union-Find



```
// find and union

int find(int x) {
    if (parent[x] == x) {
        return x;
    } else {
        parent[x] = find(parent[x]);
        return parent[x];
    }
}

void unite(int x, int y) {
    parent[find(x)] = find(y);
}
```

Cài đặt Union-Find phiên bản rút gọn



- Nếu vội...

```
#define MAXN 1000
int p[MAXN];

int find(int x) {
    return p[x] == x ? x : p[x] = find(p[x]);
}
void unite(int x, int y) { p[find(x)] = find(y); }

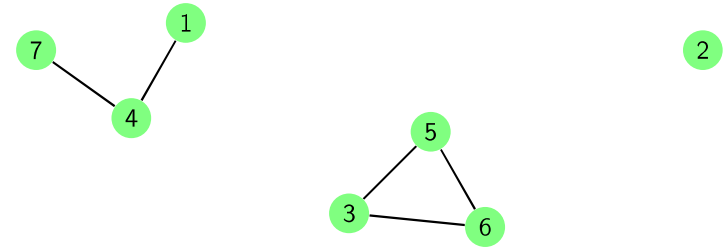
for (int i = 0; i < MAXN; i++) p[i] = i;
```

Ứng dụng của Union-Find

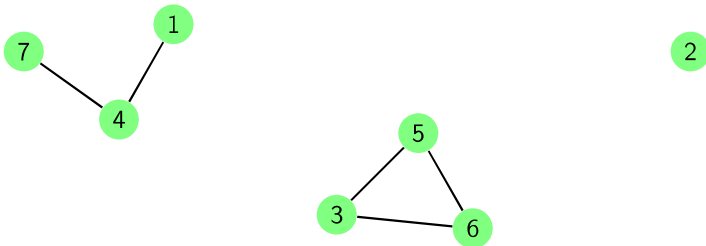


- Union-Find quản lý các tập không giao nhau
- Xử lý từng loại các tập không giao nhau tùy thời điểm
- Các bài toán ứng dụng quen thuộc thường trên đồ thị

Union-find trên đồ thị

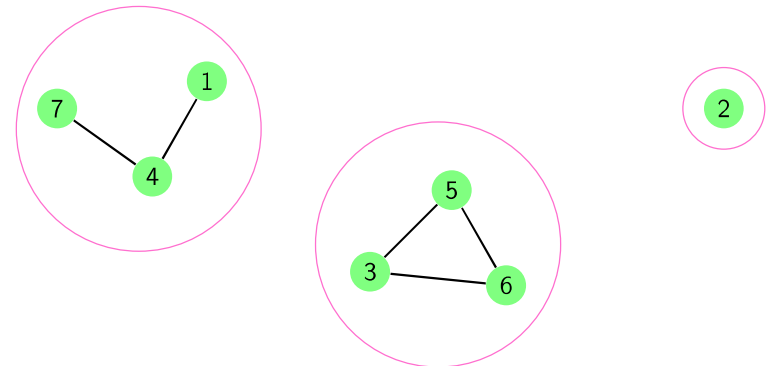


Union-find trên đồ thị



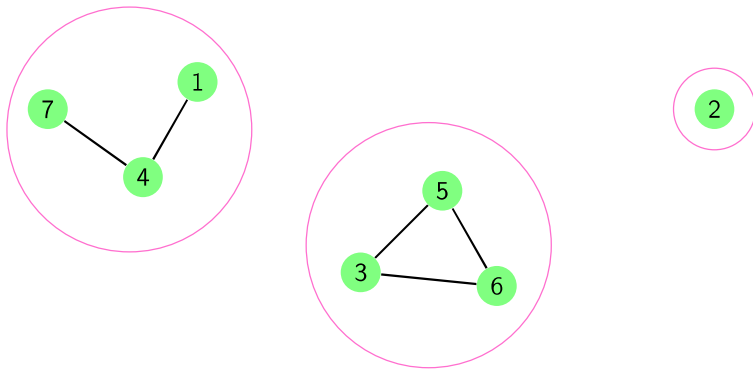
- $items = \{1, 2, 3, 4, 5, 6, 7\}$

Union-find trên đồ thị



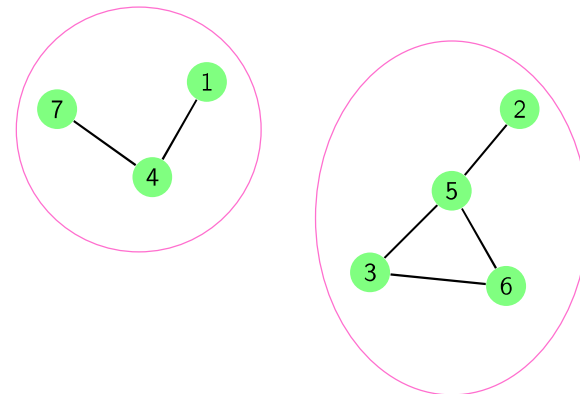
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$

Union-find trên đồ thị



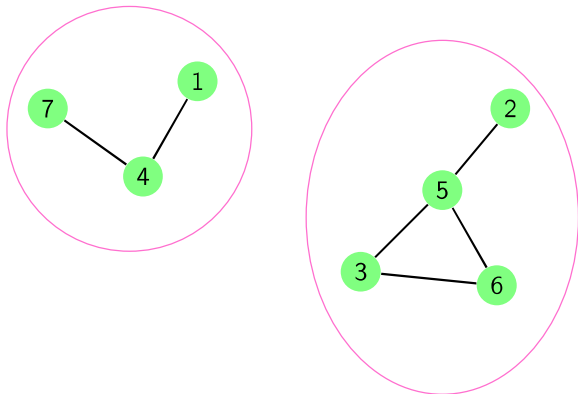
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$
- $union(2, 5)$

Union-find trên đồ thị



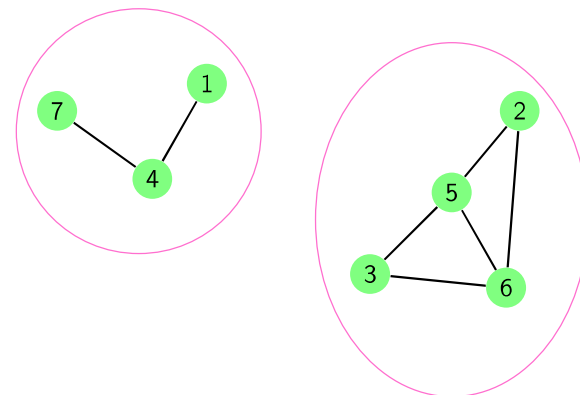
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

Union-find trên đồ thị



- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$
- $union(6, 2)$

Union-find trên đồ thị



- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

Bài toán ví dụ: Friends



- <http://uva.onlinejudge.org/external/106/10608.html>

Truy vấn trong khoảng (Range Query)



- Cho mảng A có n phần tử
- Cho i, j , yêu cầu trả lời:
 - ▶ $\max(A[i], A[i+1], \dots, A[j-1], A[j])$
 - ▶ $\min(A[i], A[i+1], \dots, A[j-1], A[j])$
 - ▶ $\text{sum}(A[i], A[i+1], \dots, A[j-1], A[j])$
- Cần trả lời các truy vấn trên thật hiệu quả, có nghĩa là không cần duyệt qua toàn bộ các phần tử
- Nhiều khi cũng cần phải cập nhật các phần tử

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6)$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$
- $\text{sum}(2, 5)$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$
- $\text{sum}(2, 5) = 29$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$
- $\text{sum}(2, 5) = 29$
- $\text{sum}(2, 2)$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$
- $\text{sum}(2, 5) = 29$
- $\text{sum}(2, 2) = 7$

Tổng trong khoảng (Range sum) trên mảng tĩnh



- Quan sát các tổng trong khoảng trên mảng tĩnh (nghĩa là không có chức năng cập nhật)

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$
- $\text{sum}(2, 5) = 29$
- $\text{sum}(2, 2) = 7$

- Làm thế nào để thực hiện các truy vấn này thật hiệu quả?

Tổng trong khoảng trên mảng tĩnh



- Đơn giản hóa: chỉ cần truy vấn dạng $\text{sum}(0, j)$
- Lưu ý $\text{sum}(i, j) = \text{sum}(0, j) - \text{sum}(0, i - 1)$

1	0	7	8	5	9	3
---	---	---	---	---	---	---

=

1	0	7	8	5	9	3
---	---	---	---	---	---	---

-

1	0	7	8	5	9	3
---	---	---	---	---	---	---

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1						

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1					

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8				

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16			

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16	21		

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16	21	30	

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16	21	30	33

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16	21	30	33

- $O(n)$ cho tiền chuẩn bị
- $O(1)$ cho mỗi truy vấn
- Liệu có thể cập nhật một cách hiệu quả?

Tổng trong khoảng trên mảng tĩnh



- Vậy chỉ cần quan tâm đến các tổng tiền tố (prefix sums)
- Tuy nhiên chỉ có n tổng như vậy...
- Chỉ cần chuẩn bị n tổng này một lần từ đầu

1	0	7	8	5	9	3
1	1	8	16	21	30	33

- $O(n)$ cho tiền chuẩn bị
- $O(1)$ cho mỗi truy vấn
- Liệu có thể cập nhật một cách hiệu quả? Không, nếu như không có thay đổi gì

Tổng trong khoảng trên mảng động



- Để làm gì?:
 - ▶ tổng trên một khoảng
 - ▶ cập nhật một phần tử

1	0	7	8	5	9	3
---	---	---	---	---	---	---

Tổng trong khoảng trên mảng động



- Để làm gì?:
 - ▶ tổng trên một khoảng
 - ▶ cập nhật một phần tử

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6)$

Tổng trong khoảng trên mảng động



- Để làm gì?:
 - ▶ tổng trên một khoảng
 - ▶ cập nhật một phần tử

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$

Tổng trong khoảng trên mảng động



- Để làm gì?:

- ▶ tổng trên một khoảng
- ▶ cập nhật một phần tử

1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 6) = 33$

- $\text{update}(3, -2)$

Tổng trong khoảng trên mảng động



- Để làm gì?:

- ▶ tổng trên một khoảng
- ▶ cập nhật một phần tử

1	0	7	-2	5	9	3
---	---	---	----	---	---	---

- $\text{sum}(0, 6) = 33$

- $\text{update}(3, -2)$

Tổng trong khoảng trên mảng động



- Để làm gì?:

- ▶ tổng trên một khoảng
- ▶ cập nhật một phần tử

1	0	7	-2	5	9	3
---	---	---	----	---	---	---

- $\text{sum}(0, 6) = 33$

- $\text{update}(3, -2)$

- $\text{sum}(0, 6)$

Tổng trong khoảng trên mảng động



- Để làm gì?:

- ▶ tổng trên một khoảng
- ▶ cập nhật một phần tử

1	0	7	-2	5	9	3
---	---	---	----	---	---	---

- $\text{sum}(0, 6) = 33$

- $\text{update}(3, -2)$

- $\text{sum}(0, 6) = 23$

Tổng trong khoảng trên mảng động



- Để làm gì?:

- ▶ tổng trên một khoảng
- ▶ cập nhật một phần tử

1	0	7	-2	5	9	3
---	---	---	----	---	---	---

- $\text{sum}(0, 6) = 33$

- $\text{update}(3, -2)$

- $\text{sum}(0, 6) = 23$

- Làm thế nào để thực hiện các truy vấn này một cách hiệu quả?

Segment Tree



1	0	7	8	5	9	3
---	---	---	---	---	---	---

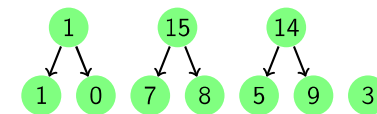
Segment Tree



1	0	7	8	5	9	3
---	---	---	---	---	---	---

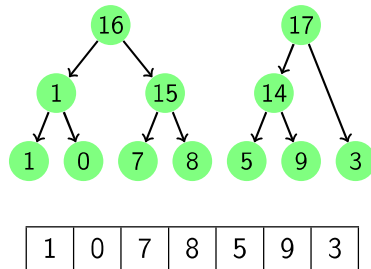
1	0	7	8	5	9	3
---	---	---	---	---	---	---

Segment Tree

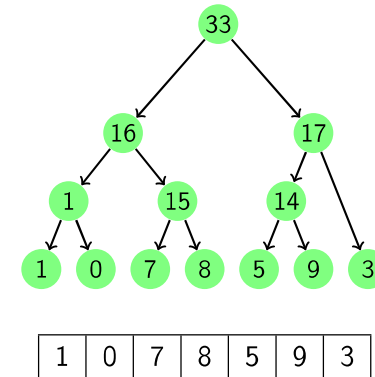


1	0	7	8	5	9	3
---	---	---	---	---	---	---

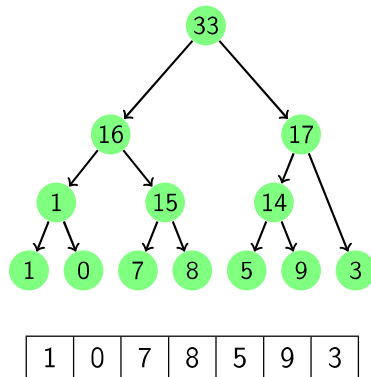
Segment Tree



Segment Tree



Segment Tree



- Mỗi đỉnh lưu tổng của một đoạn (tương ứng) trong mảng

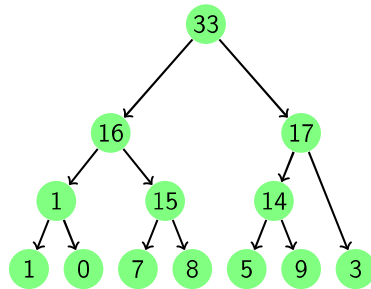
Cây phân đoạn - Cài đặt



```
struct segment_tree {
    segment_tree *left, *right;
    int from, to, value;
    segment_tree(int from, int to)
        : from(from), to(to), left(NULL), right(NULL), value(0) { }
};

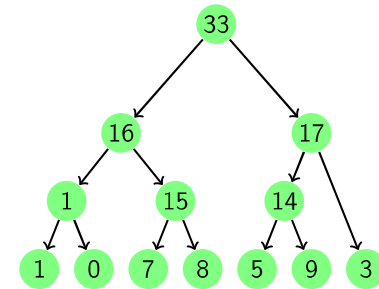
segment_tree* build(const vector<int> &arr, int l, int r) {
    if (l > r) return NULL;
    segment_tree *res = new segment_tree(l, r);
    if (l == r) {
        res->value = arr[l];
    } else {
        int m = (l + r) / 2;
        res->left = build(arr, l, m);
        res->right = build(arr, m + 1, r);
        if (res->left != NULL) res->value += res->left->value;
        if (res->right != NULL) res->value += res->right->value;
    }
    return res;
}
```

Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

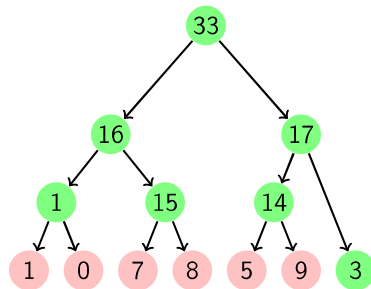
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

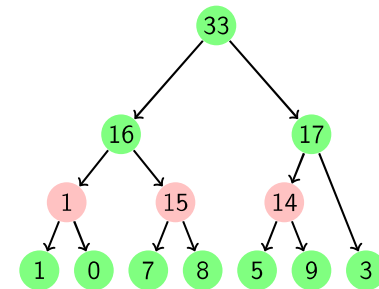
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

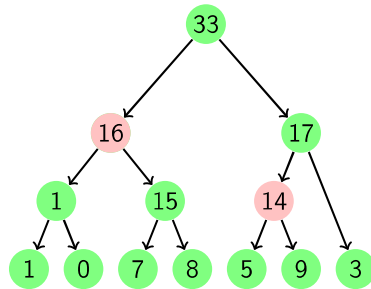
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

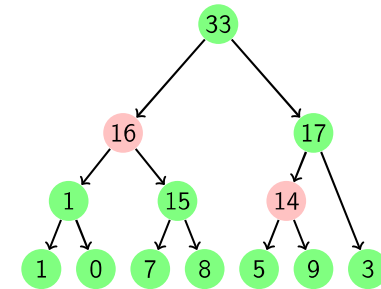
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 5)$

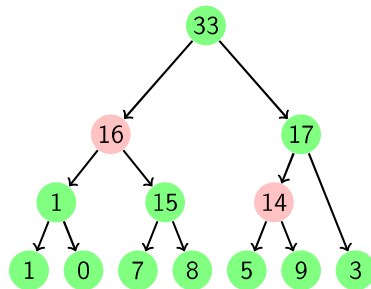
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 5) = 16 + 14 = 30$

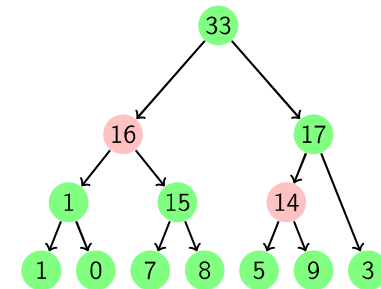
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 5) = 16 + 14 = 30$
- Chỉ cần quan tâm một số ít đỉnh mà vẫn có được thông tin toàn đoạn

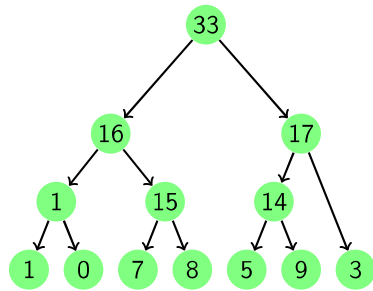
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

- $\text{sum}(0, 5) = 16 + 14 = 30$
- Chỉ cần quan tâm một số ít đỉnh mà vẫn có được thông tin toàn đoạn
- Nhưng làm thế nào để tìm thấy chúng?

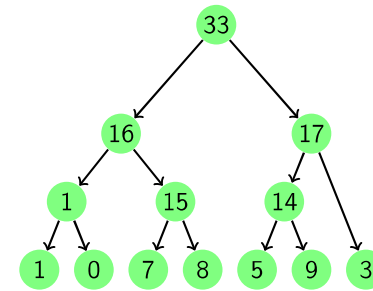
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

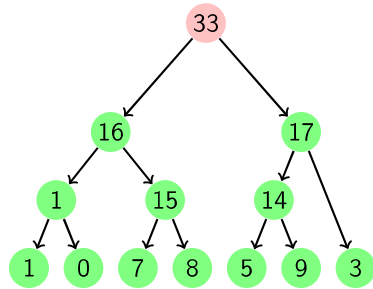
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

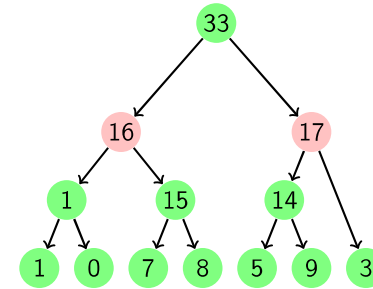
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $\text{sum}(0, 5)$

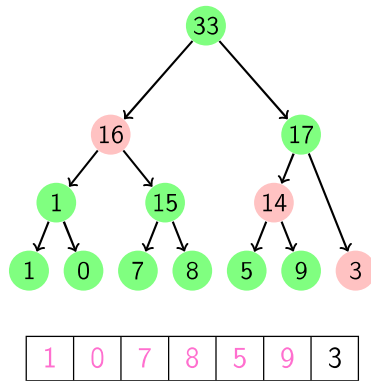
Truy vấn trên Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

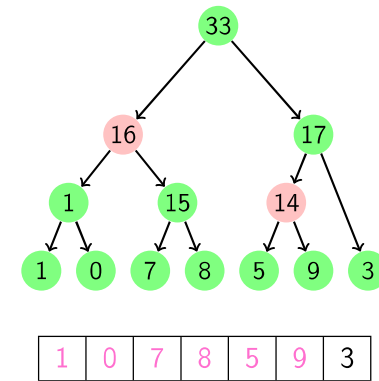
• $\text{sum}(0, 5)$

Truy vấn trên Cây phân đoạn



• $\text{sum}(0, 5)$

Truy vấn trên Cây phân đoạn



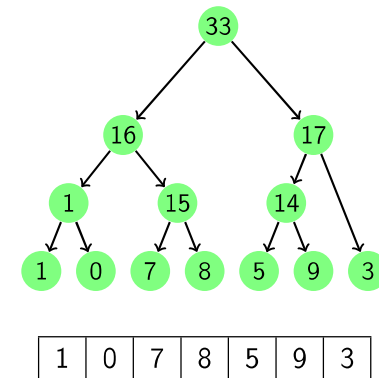
• $\text{sum}(0, 5)$

Truy vấn một Cây phân đoạn - Cài đặt

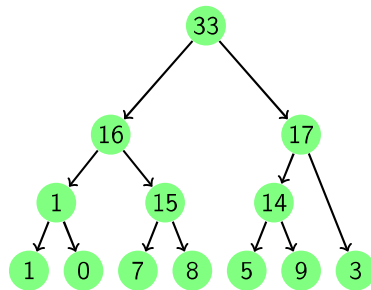


```
int query(segment_tree *tree, int l, int r) {
    if (tree == NULL) return 0;
    if (l <= tree->from && tree->to <= r) return tree->value;
    if (tree->to < l) return 0;
    if (r < tree->from) return 0;
    return query(tree->left, l, r) + query(tree->right, l, r);
}
```

Cập nhật Cây phân đoạn



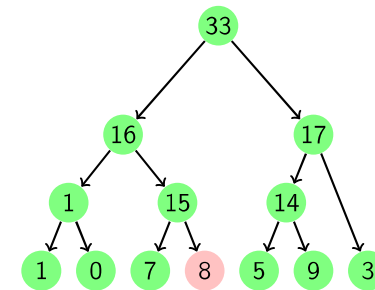
Cập nhật Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

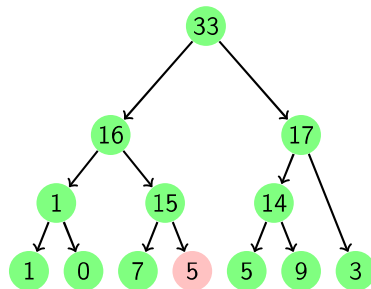
Cập nhật Cây phân đoạn



1	0	7	8	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

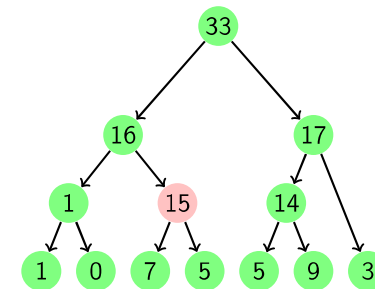
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

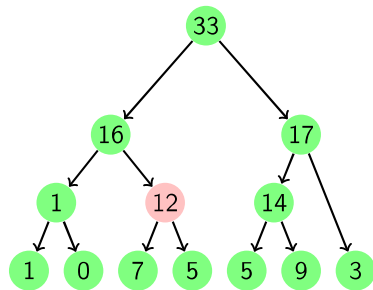
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

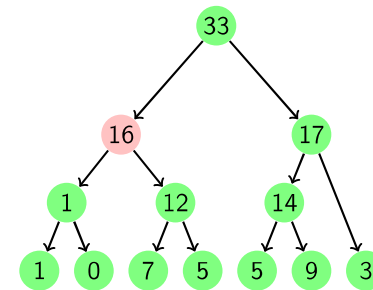
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

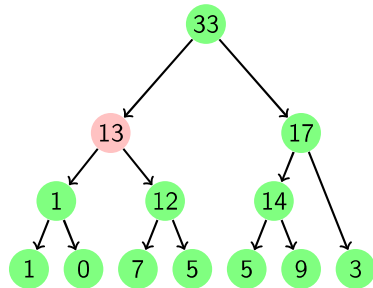
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

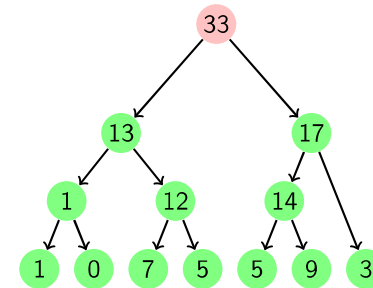
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

• $update(3, 5)$

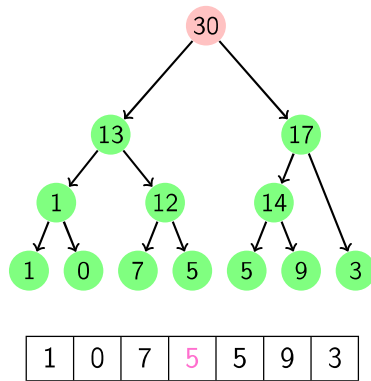
Cập nhật Cây phân đoạn



1	0	7	5	5	9	3
---	---	---	---	---	---	---

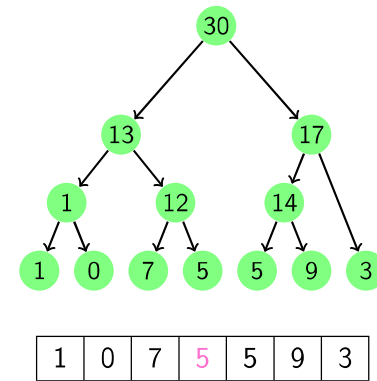
• $update(3, 5)$

Cập nhật Cây phân đoạn



- `update(3,5)`

Cập nhật Cây phân đoạn



- `update(3,5)`

Cập nhật Cây phân đoạn - Cài đặt



```
int update(segment_tree *tree, int i, int val) {
    if (tree == NULL) return 0;
    if (tree->to < i) return tree->value;
    if (i < tree->from) return tree->value;
    if (tree->from == tree->to && tree->from == i) {
        tree->value = val;
    } else {
        tree->value = update(tree->left, i, val)
            + update(tree->right, i, val);
    }
    return tree->value;
}
```

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn
 - ▶ truy vấn trong một khoảng
 - ▶ cập nhật một giá trị

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn
 - ▶ truy vấn trong một khoảng
 - ▶ cập nhật một giá trị
- Nhưng độ hiệu quả của các toán tử này thế nào?

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn trong $O(n)$
 - ▶ truy vấn trong một khoảng
 - ▶ cập nhật một giá trị
- Nhưng độ hiệu quả của các toán tử này thế nào?

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn trong $O(n)$
 - ▶ truy vấn trong một khoảng trong $O(\log n)$
 - ▶ cập nhật một giá trị
- Nhưng độ hiệu quả của các toán tử này thế nào?

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn trong $O(n)$
 - ▶ truy vấn trong một khoảng trong $O(\log n)$
 - ▶ cập nhật một giá trị trong $O(\log n)$
- Nhưng độ hiệu quả của các toán tử này thế nào?

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn trong $O(n)$
 - ▶ truy vấn trong một khoảng trong $O(\log n)$
 - ▶ cập nhật một giá trị trong $O(\log n)$
- Nhưng độ hiệu quả của các toán tử này thế nào?
- Dễ dàng sử dụng Cây phân đoạn cho các toán tử min, max, gcd, và các toán tử tương tự khác, nói chung là cài đặt tương tự

Cây phân đoạn



- Bây giờ ta có thể
 - ▶ xây dựng một Cây phân đoạn trong $O(n)$
 - ▶ truy vấn trong một khoảng trong $O(\log n)$
 - ▶ cập nhật một giá trị trong $O(\log n)$
- Nhưng độ hiệu quả của các toán tử này thế nào?
- Dễ dàng sử dụng Cây phân đoạn cho các toán tử min, max, gcd, và các toán tử tương tự khác, nói chung là cài đặt tương tự
- Cũng có thể cập nhật một khoảng giá trị trong thời gian $O(\log n)$ (Tra Google với từ khóa Segment Trees with Lazy Propagation nếu muốn mở rộng thêm)

Bài toán ví dụ: Potentiometers



- <http://uva.onlinejudge.org/external/120/12086.html>