MEX (Minimal Excluded) của một dãy số

1. Định nghĩa

MEX (Minimal Excluded) của một dãy số là phần tử **không âm nhỏ nhất** không xuất hiện trong dãy đó. Ví dụ:

$$\text{mex}(\{1, 2, 3\}) = 0$$
$$\text{mex}(\{0, 1, 2, 4\}) = 3$$
$$\text{mex}(\{0, 1, 2, 3\}) = 4$$

Lưu ý rằng MEX của một mảng có kích thước N không bao giờ lớn hơn N.

2. Truy vấn một lần

2.1. Cài đặt cơ bản

Cách tiếp cận đơn giản nhất là sử dụng một tập hợp (set) để lưu trữ tất cả các phần tử trong mảng, sau đó kiểm tra các số từ 0 đến N xem số nào đầu tiên không có trong tập hợp.

```
int find_MEX(const vector<int>& nums) {
    set<int> s(nums.begin(), nums.end());

    int result = 0;
    while (s.find(result) != s.end())
        result++;
    return result;
}
```

Độ phức tạp: $O(N \log N)$.

2.2. Cài đặt tối ưu hơn

Ta hoàn toàn có thể sử dụng một `vector<boolean>` thay vì set:

```
int find_MEX(const vector<int>& nums) {
    int max_ele = *max_element(nums.begin(), nums.end());
    vector<bool> marked(max_ele + 1, false);

    for (auto num : nums)
        marked[num] = true;

    int result = 0;
    while (marked[result])
        result++;
    return result;
}
```

Độ phức tạp: $O(N)$

Cách này nhanh hơn nhưng chỉ hiệu quả khi tính MEX một lần. Nếu cần tính MEX nhiều lần với các thay đổi trên mảng, ta cần một cấu trúc dữ liệu tốt hơn.

3. Truy vấn nhiều lần

3.1. MEX với cập nhật mảng và truy vấn $O(1)$

- Khởi tạo:
  - Sử dụng cấu trúc dữ liệu set để xác định các giá trị bị mất và map để xác định tần số xuất hiện.
  - Thêm vào set các giá trị từ 1 đến n sau đó duyệt các phần tử trong mảng và xóa các phần tử có giá trị tương ứng trong set. Khi đó giá trị MEX lúc nào cũng là giá trị đầu tiên của set.
- Cập nhật:
  - Thực hiện update đơn giản như đoạn code phía dưới với độ phức tạp $O(\log n)$.
- Truy vấn:
  - Đơn giản lúc nào cũng là giá trị đầu tiên của set.

```
class Mex {
private:
    map<int, int> frequency;
    set<int> missing_numbers;
    vector<int> A;

public:
    Mex(const vector<int>& A) : A(A) {
        for (int i = 0; i <= A.size(); i++)
            missing_numbers.insert(i);

        for (int x : A) {
            ++frequency[x];
            missing_numbers.erase(x);
        }
    }

    int mex() {
        return *missing_numbers.begin();
    }

    void update(int idx, int value) {
        if (--frequency[A[idx]] == 0)
            missing_numbers.insert(A[idx]);
        A[idx] = value;
        ++frequency[value];
        missing_numbers.erase(value);
    }
};
```

Độ phức tạp:

- Khởi tạo: $O(N \log N)$
- Truy vấn: $O(1)$
- Cập nhật: $O(\log N)$

3.2. MEX trên một đoạn [l, r] bất kỳ

```
const int N = 1e5 + 5;

class SegmentTree {
private:
```

```
    int seg[N * 4];
public:
    SegmentTree() {
        memset(seg, 0, sizeof seg);
    }

    void set(int id, int l, int r, int i, int val) {
        if (l > i || r < i || l > r) return;

        if (l == r) {
            seg[id] = val;
            return;
        }

        int mid = (l + r) / 2;
        set(id*2, l, mid, i, val);
        set(id*2 + 1, mid + 1, r, i, val);
        seg[id] = min(seg[id*2], seg[id*2 + 1]);
    }

    int find_mex(int id, int l, int r, int val) {
        if (l > r) return 0;

        if (l == r) return l;

        int left_node = seg[id*2];
        int right_node = seg[id*2 + 1];
        int mid = (l + r) / 2;
        if (val > left_node) {
            return find_mex(id*2, l, mid, val);
        }
        return find_mex(id*2 + 1, mid + 1, r, val);
    }
};

int main() {
    vector<int> A = {0, 0, 1, 2, 4, 6};
    int n = A.size();
    int q;
    cin >> q;
    vector<vector<pair<int, int>>> queries(n + 1);
    for (int i = 0; i < q; ++i) {
        int l, r;
        cin >> l >> r;
        queries[r].push_back({l, i});
    }
    SegmentTree tree;
    vector<int> res(q);
    int m = *max_element(A.begin(), A.end());
    for (int i = 0; i < n; ++i) {
        tree.set(1, 0, m, A[i], i + 1);
        for (auto [l, idx] : queries[i]) {
            res[idx] = tree.find_mex(1, 0, m, l);
        }
    }
    for (auto r : res)
```

```
        cout << r << endl;
}
```

4. References

CP-Algorithms
Offline Range MEX queries in O(log n)
MEX of an array