SCV/2021

구간 합 구하기

- Segment Tree -

안태진(taejin7824@gmail.com)
GitHub(github.com/taejin1221)
상명대학교 소프트웨어학과
201821002

Contents

Problem

- 구간 합 구하기
  - 단순 구간 합 구하기는 Prefix Sum으로 가능
    - 하지만 중간에 값이 바뀐다면?
    - Prefix Sum은 값이 바뀌면 그 뒤의 값도 모두 바꿔야하기 때문에 바꾸는데 $O(n)$ 이 소요
    - => 비효율!
  - Segment Tree
    - 구간 합을 $O(\log n)$만에 구하고, 값을 $O(\log n)$에 바꿀 수 있음!
  - 대표적인 Segment Tree 문제
    - Segment Tree: 만물 세그, 만물 DP라는 말이 있음...
      - 대회의 대부분 문제는 Segment Tree, DP...?

Contents

- Problem
- Approach
- Code

Approach

- 프로그래머는 코드로 대화한다.
- 바로 코드로 갑시다
- 절대 귀찮아서 안만든거 아님

Contents

- Problem
- Approach
- Code

Code

- 보기 전에...
  - 전체 코드 길이 108줄
  - main 20줄
    - 나머지가 세그 코드...
    - Segment Tree를 Class로 만듬

Code

- main
  - 입력 받고
  - 세그먼트 트리 만들고
  - a가 1이면 update하기
    - index는 0부터니 1을 빼주자
    - 여기 value를 int로 받아서 100%에 틀림...
    - 뭔가 부족하죠? 나중에
  - a가 2면 sum을 구하기
    - index는 0부터이니 둘 다 1씩 빼주기

```
int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n, m, k;
    cin >> n >> m >> k;

    ll arr[1'000'000];
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    SegmentTree segTree(arr, n);
    for (int i = 0; i < m + k; i++) {
        int a;
        cin >> a;
        if (a == 1) {
            int idx; ll value;
            cin >> idx >> value;
            segTree.update(idx - 1, value);
        } else {
            int start, end;
            cin >> start >> end;
            cout << segTree.getSum(start - 1, end - 1) << '\n';
        }
    }

    return 0;
}
```

Code

- SegmentTree Class
  - 멤버 변수 및 구간을 나타내는 구조체 선언

- 생성자 및 소멸자
  - tree 배열 생성 및 0으로 초기화
  - minIdx, maxIdx: arr 배열의 최소 idx, 최대 idx
  - makeTree: 배열을 이용해 세그먼트 트리를 만듬
    - 즉, 트리에 값들을 채워넣음

```
class SegmentTree {
    ll* tree;
    int minIdx, maxIdx;

    class Range {
    public:
        int start, end;
        Range(int s, int e): start(s), end(e) { }
        int getMid() {
            return (start + end) / 2;
        }
    };
```

```
public:
    SegmentTree(ll arr[], int size) {
        tree = new ll[size * 4];
        fill(tree, tree + size * 4, 0);
        minIdx = 0, maxIdx = size - 1;

        makeTree(1, Range(minIdx, maxIdx), arr);
    }

    ~SegmentTree() {
        delete[] tree;
    }
```

Code

- makeTree Method
  - treeIdx: 현재 노드 트리
  - curr: 현재 노드가 나타내는 범위

  - 만약 현재 노드가 한개의 idx를 나타낸다면 값 삽입
  - 아니라면 재귀적으로 왼쪽 만들고 오른쪽 만들기
  - 왼쪽 자식 오른쪽 자식 더해주기

```
void makeTree(int treeIdx, Range curr, ll arr[]) {
    if (curr.start == curr.end)
        tree[treeIdx] = arr[curr.start];
    else {
        int left = treeIdx * 2, right = treeIdx * 2 + 1;
        makeTree(left, Range(curr.start, curr.getMid()), arr);
        makeTree(right, Range(curr.getMid() + 1, curr.end), arr);

        tree[treeIdx] = tree[left] + tree[right];
    }
}
```

Code

- getSum Method
  - 첫번째 getSum
    - 간단히 범위만 주어지면 알맞는 메서드 호출
  - 두번째 getSum
    - treeIdx: 현재 노드
    - curr: 현재 노드의 범위
    - target: 구하려는 범위

```
ll getSum(int start, int end) {
    return getSum(1, Range(minIdx, maxIdx), Range(start, end));
}

ll getSum(int treeIdx, Range curr, Range target) {
    if (target.start <= curr.start && curr.end <= target.end)
        return tree[treeIdx];
    else if (curr.end < target.start || target.end < curr.start)
        return 0ll;
    else {
        return (
            getSum(treeIdx * 2, Range(curr.start, curr.getMid()), target) +
            getSum(treeIdx * 2 + 1, Range(curr.getMid() + 1, curr.end), target)
        );
    }
}
```

Code

- update Method

```
void update(int targetIdx, ll value) {
    update(1, Range(minIdx, maxIdx), targetIdx, value);
}

void update(int treeIdx, Range curr, int target, ll value) {
    if (curr.start == curr.end && curr.start == target)
        tree[treeIdx] = value;
    else if (curr.start <= target && target <= curr.end) {
        int left = treeIdx * 2, right = treeIdx * 2 + 1;
        update(left, Range(curr.start, curr.getMid()), target, value);
        update(right, Range(curr.getMid() + 1, curr.end), target, value);

        tree[treeIdx] = tree[left] + tree[right];
    }
}
```

감사합니다!