이태한(lastknight00)

Fenwick Tree(Binary Indexed Tree(BIT))

1. Index의 LSB를 구한다.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
LSB	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16

Fenwick Tree(Binary Indexed Tree(BIT))

※LSB(Least Significant Bit) 구하는 방법

-음수 표현 방법

①원래 수의 1의 보수를 구한다(0->1, 1->0)

②1의 보수에 1을 더한다.

-위의 방법을 적용하였을 때, 작은 수(오른쪽 Bit)부터 큰 수로 이동하며 처음 1이 나온 Bit(LSB)만 1로 유지되며 나머지 Bit는 수가 바뀌어 있게 된다.

-0이 계속 나오게 되면 1의 보수를 구하며 1로 변경됨.

-1을 더하며 0으로 변하며 자릿수 올림이 발생하여 다음 자리로 1이 넘어감.

-이때 다음 자릿수가 0(원래 수가 1(LSB)인 경우만 1이 되며, LSB가 아닌 경우 계속 0으로 변경되고 자릿수 올림이 발생됨)

-위 규칙으로 LSB 우측 Bit는 0을 유지하게 됨.

-LSB는 1로 유지되며 LSB 좌측 Bit는 1의 보수를 구하는 과정에 의해 1과 0이 바뀌어 있다.

-원래 수를 v라고 했을 때, v&-v를 하면 LSB만 1로 하는 수를 구할 수 있다.

Fenwick Tree(Binary Indexed Tree(BIT))

2. Tree[i]에 값을 저장한다.
   -Tree[i]에 저장되는 값은 v[i-[LSB[i]]+1] ~ v[i] 의 값들의 합이 된다.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
v	3	2	5	7	10	3	2	7	8	2	1	9	5	10	7	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
LSB	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16
Tree	3	5	5	17	10	13	2	39	8	10	1	20	5	15	7	85

이태한(lastknight00)

Fenwick Tree(Binary Indexed Tree(BIT))

3. 1~i까지 prefix sum을 구한다.

13까지의 합을 구할 경우, Tree[13]+Tree[12]+Tree[8]을 구하면 된다.

∵Tree[13] = v[13]

Tree[12] = v[12] + v[11] + v[10] + v[9]

Tree[8] = v[8] + v[7] + v[6] + v[5] + v[4] + v[3] + v[2] v[1]

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
v	3	2	5	7	10	3	2	7	8	2	1	9	5	10	7	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
LSB	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16
Tree	3	5	5	17	10	13	2	39	8	10	1	20	5	15	7	85

이태한(lastknight00)

Fenwick Tree(Binary Indexed Tree(BIT))

3. 1~i까지 prefix sum을 구한다.

13까지의 합을 구할 경우, Tree[13]+Tree[12]+Tree[8]을 구하면 된다.

13 = 1101

12 = 1100

8 = 1000

=> index의 LSB를 빼면서 0이 되기 전까지 Tree[i]의 합을 구하면 된다.

★ 왜 13, 12, 8인가??

13의 LSB는 1이다. 2번 과정에서 Tree[13]에는 v[13]의 값만 저장이 되어 있다.
그럼 우리는 v[1]~v[12] + Tree[13]을 구해야 한다.
이것은 다시 Tree[12]를 이용하여 구할 수 있는데, Tree[12]의 LSB는 4이다.
따라서 v[1]~v[8]+Tree[12]+Tree[13]으로 바꿀 수 있다.
이것을 다시 Tree[8]을 이용하여 구하게 되면 Tree[8]+Tree[12]+Tree[13]으로 바꿔서 구할 수 있다.

Fenwick Tree(Binary Indexed Tree(BIT))

4. 결론

○시간복잡도
- -트리를 만드는 과정 : O(nlogn)
- -합을 구하는 과정 : O(logn)
- -총 O(nlogn)

○우리가 알고 있는 부분합 구하는 방법
- -t[i]=v[1]~v[i]까지의 합
- -합을 구하는 과정(배열을 구성하는 과정 포함) : O(n)

○우리가 알고있는것 보다 느리다.

○단, 값의 수정이 빈번하게 일어날 경우.
- -우리가 알고 있는 방법 : i번재 수정시, i~n까지의 합을 모두 구해야 함(O(n))
- -BIT의 경우, 해당 i를 포함해서 합을 구한 index만 수정하면 됨(O(logn))

이태한(lastknight00)

Fenwick Tree(Binary Indexed Tree(BIT))

4. 코드(백준 2042 구간합 구하기)

```
#include<cstdio>
long long a[1000001],t[1000001],d;
int N, M, K,b,c;
long long sum(int p) {
    long long retval = 0;
    while (p) {
        retval += t[p];
        p -= (p&-p);
    }
    return retval;
}
void update(int p, long long v) {
    while (p<=N) {
        t[p] += v;
        p += (p&-p);
    }
}
int main() {
    scanf("%d %d %d", &N, &M, &K);
    M += K;
    for (int i = 1; i <= N; i++) {
        scanf("%lld", &a[i]);
        update(i, a[i]);
    }
    while (M--) {
        scanf("%d %d %lld", &b, &c, &d);
        if (b == 1) {
            update(c, d-a[c]);
            a[c] = d;
        }
        else {
            printf("%lld\n", sum(d)-sum(c-1));
        }
    }
}
```

이태한(lastknight00)