Range updates with BIT / Fenwick Tree

December 2, 2013 by kartik kukreja

I described implementation of BIT/Fenwick tree in an earlier post as a way of maintaining cumulative frequency table, which allows operations like updating any single element and querying sum of elements in a range [a…b] in logarithmic time. I recently found out that this is only one of the ways of using a BIT. A BIT can in fact be operated in one of three modes:

1. Point Updates and Range Queries

Given an array A of N numbers, we need to support adding a value v to any element A[p] and querying the sum of numbers A[a] + A[a+1] + … + A[b], both operations in O(log N). Let ft[N+1] denotes the underlying fenwick tree.

```
# Add v to A[p]
update(p, v):
  for (; p <= N; p += p&(-p))
    ft[p] += v

# Return sum A[1...b]
query(b):
  sum = 0
  for(; b > 0; b -= b&(-b))
    sum += ft[b]
  return sum

# Return sum A[a...b]
query(a, b):
  return query(b) - query(a-1)
```

Point Updates and Range Queries.py hosted with ❤ by **GitHub** **view raw**

Take a look at C++ implementation.

2. Range Updates and Point queries

Given an array A of N numbers, we need to support adding a value v to each element A[a…b] and querying the value of A[p], both operations in O(log N). Let ft[N+1] denote the underlying fenwick tree.

```
# Add v to A[p]
update(p, v):
  for (; p <= N; p += p&(-p))
    ft[p] += v

# Add v to A[a...b]
update(a, b, v):
  update(a, v)
  update(b + 1, -v)

# Return A[b]
```

```
query(b):
  sum = 0
  for(; b > 0; b -= b&(-b))
    sum += ft[b]
  return sum
```

Range Updates and Point Queries.py hosted with ❤ by **GitHub** **view raw**

Explanation: update(p, v) will affect all p' >= p. To limit the effect to a given range [a…b], we subtract -v from all p' > b by performing the operation update(b+1, -v).

See problem UPDATEIT which uses this idea.

Take a look at C++ implementation.

3. Range Updates and Range Queries

Given an array A of N numbers, we need to support adding a value v to each element A[a…b] and querying the sum of numbers A[a…b], both operations in O(log N). This can be done by using two BITs B1[N+1], B2[N+1].

```
update(ft, p, v):
  for (; p <= N; p += p&(-p))
    ft[p] += v

# Add v to A[a...b]
update(a, b, v):
  update(B1, a, v)
  update(B1, b + 1, -v)
  update(B2, a, v * (a-1))
  update(B2, b + 1, -v * b)

query(ft, b):
  sum = 0
  for(; b > 0; b -= b&(-b))
    sum += ft[b]
  return sum

# Return sum A[1...b]
query(b):
  return query(B1, b) * b - query(B2, b)

# Return sum A[a...b]
query(a, b):
  return query(b) - query(a-1)
```

Range Updates and Range Queries.py hosted with ❤ by **GitHub** **view raw**

Explanation:

BIT B1 is used like in the earlier case with range updates/point queries such that query(B1, p) gives A[p].

Consider a range update query: Add v to [a…b]. Let all elements initially be 0. Now, Sum(1…p) for different p is as follows:

- 1 <= p < a : 0
- a <= p <= b : v * (p – (a – 1))
- b < p <= N : v * (b – (a – 1))

Thus, for a given index p, we can find Sum(1…p) by subtracting a value X from p * Sum(p,p) (Sum(p,p) is the actual value stored at index p) such that

- 1 <= p < a : Sum(1..p) = 0, X = 0
- a <= p <= b : Sum(1…p) = (v * p) – (v * (a-1)), X = v * (a-1)
- b < p <= N: Sum(1…p) = (v * b) – (v * (a-1)), X = -(v * b) + (v * (a-1))

To maintain this extra factor X, we use another BIT B2 such that

- Add v to [a…b] -> Update(B2, a, v * (a-1)) and Update(B2, b+1, -v * b)
- Query(B2, p) gives the value X that must be subtracted from A[p] * p

See problem HORRIBLE which uses this idea.

Take a look at C++ implementation.

References:

- http://apps.topcoder.com/forums/?module=Thread&threadID=715842&start=0&mc=8
- http://programmingcontests.quora.com/Tutorial-Range-Updates-in-Fenwick-Tree