Optimasi Query dengan Binary Indexed Tree

Ensof Barhami 13516096
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516096@students.itb.ac.id

***Abstrak*—Binary Indexed Tree (BIT) atau Fenwick Tree adalah struktur data yang dapat mengupdate elemen dari tabel frekuensi kumulatif dengan efisien. Struktur data ini pertama kali dikenalkan oleh Peter M. Fenwick pada 1994. Struktur data ini dapat digunakan dalam permasalahan yang memerlukan banyak query, baik itu update nilai maupun melihat nilai di index tertentu. BIT mengoptimasi akses elemen dengan kompleksitas O(n) menjadi O(log n) seperti pada struktur data Segmented Tree yang implementasinya lebih sulit. Paper ini menampilkan tiga bentuk BIT dalam mengatasi query yaitu Point Update and Range Query, Range Update and Point Query, dan Range Update and Range Query untuk mengoptimasi permasalah query yang banyak dalam Computer Science.**

***Keywords*—Binary Indexed Tree, Fenwick Tree, Tabel Frekuensi Kumulatif, Query and Update, Segmented Tree.**

I. PENDAHULUAN

Kita biasanya memerlukan tipe data tertentu untuk mempercepat kompleksitas waktu algoritma kita. Salah satu tipe data tersebut adalah Binary Indexed Tree atau disingkat BIT. BIT mempercepat algoritma yang menangani tabel frekuensi kumulatif yang biasanya kita tampung dalam array prefix sum. Dengan prefix sum, kita dapat mengupdate nilai dari suatu elemen ke-k dengan cara iterasi semua elemen frekuensi kumulatif ke-1 sampai ke-k dan update masing-masing nilainya. Jika update nilai sangat sering dilakukan, tentu hal ini tidak efisien karena satu kali update memerlukan iterasi O(n). BIT dapat melakukan hal serupa hanya dengan iterasi sebanyak O(log n).

II. PREFIX SUM, FENWICK TREE

A. PREFIX SUM

Misal diberikan persoalan sebagai berikut.

Diberikan elemen sebanyak n buah dan query sebanyak q buah. Setiap query dapat berupa dua hal yaitu outputkan berapa hasil penjumlahan elemen dari index ke-a sampai index ke-b, dan update nilai dari elemen ke-k.

Permasalahan ini bisa kita selesaikan dengan BruteForce menggunakan array biasa. Jika diberikan query update, ubah nilai elemen ke-1 sampai ke-k sesuai yang diinginkan. Jika diberikan query penjumlahan, lakukan iterasi dari index ke-a sampai ke-b lalu jumlahkan semua elemennya. Kedua query dapat dilakukan dengan iterasi masing-masing O(n). Namun, hal ini dapat dioptimasi dengan mengubah array kita menjadi array prefix sum. Yaitu array P yang berisi penjumlahan elemen-elemen sebelumnya pada array A.

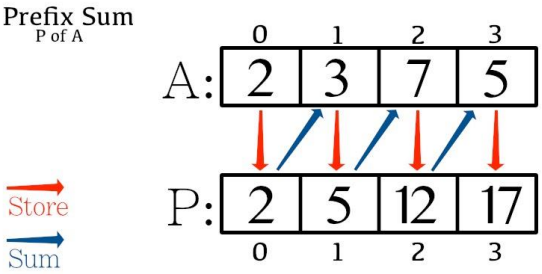

Figure 1. Prefix Sum Illustration

Dengan struktur seperti ini, kita perlu melakukan *precompute* array di awal. Tetapi kita tidak memerlukan iterasi saat akan melakukan query penjumlahan. Untuk mendapatkan penjumlahan elemen ke-a sampai elemen ke-b, kita hanya perlu melakukan operasi P[b] – P[a-1] untuk mendapatkan jumlah A[a] + A[a+1] + A[a+2] + ... + A[b]. Sehingga untuk update elemen memerlukan O(n) dan untuk menjumlah elemen-elemen **hanya** memerlukan O(1) dengan precompute di awal dan setiap kali melakukan update sebesar O(n).

B. FENWICK TREE

Permasalahan update nilai dan menjumlahkan elemen-elemen ini dapat dikembangkan, bisa saja update nya berupa update elemen ke-a sampai ke-b (range update bukan point update). Bisa juga penjumlahannya hanya di satu elemen saja (point query bukan range query) Sehingga ada banyak variasi masalah yaitu point update, range update, point query, dan range query.

Prefix sum biasa tidak akan efektif untuk menangani variasi update dan query ini. Sehingga kita membutuhkan struktur data yang lebih cepat. Ada dua alternatif, yaitu Segmented Tree dan Fenwick Tree atau biasa disebut BIT. Namun kali ini kita hanya akan membahas Fenwick Tree.

Secara umum, fenwick tree berbentuk sebagai berikut.

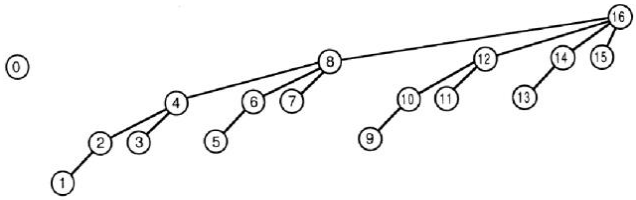

Figure 2. Fenwick Tree

Untuk mempermudah, fenwick tree atau BIT dapat digambar sebagai berikut.

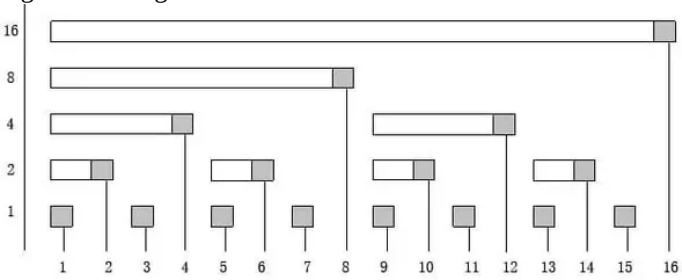

Figure 3. Fenwick Tree

Dari ilustrasi diatas dapat dibaca sebagai berikut :
Node-1 menyimpan informasi dirinya sendiri.
Node-2 menyimpan informasi dirinya dan node-1.
Node-3 menyimpan informasi dirinya sendiri.
Node-6 menyimpan informasi dirinya dan node-5.
Node-16 menyimpan informasi node-1 sampai node-16.
Dst.

Sehingga, ketika kita akan mengetahui penjumlahan elemen ke-9 sampai elemen ke-14 kita hanya perlu mengakses node-14 dan node-12.

Sekilas, BIT mirip seperti prefix sum namun indeks yang disimpannya bukan dari indeks elemen pertama.

Prefix sum mengikuti aturan :
- **P[x] = penjumlahan dari A[1] sampai A[x].**

Sedangkan BIT mengikuti aturan :
- **BIT[x] = penjumlahan dari A[x-k+1] sampai A[x]**
- **dengan k = x&(-x) dalam aljabar boolean.**
- **S[k] = BIT[k] + S[k - k & (-k)]**
- **Dengan S=penjumlahan elemen ke-1 sampai ke-k**

x = 1, k = 1, BIT[1] = A[1]
x = 2, k = 2, BIT[2] = A[1] + A[2]
x = 3, k = 1, BIT[3] = A[3]
x = 4, k = 4, BIT[4] = A[1] + A[2] + A[3] + A[4]
x = 5, k = 1, BIT[5] = A[5]
x = 6, k = 2, BIT[6] = A[5] + A[6]
x = 7, k = 1, BIT[7] = A[7]
x = 8, k = 8, BIT[8] = A[1] + A[2] + … + A[8]

S[1] = BIT[1]
S[2] = BIT[2]
S[3] = BIT[3] + BIT[2]
S[4] = BIT[4]
S[5] = BIT[5] + BIT[4]
S[7] = BIT[7] + BIT[6] + BIT[4]
Dst.

III. IMPLEMENTASI

Seperti yang sudah disebutkan diawal. Terdapat tiga macam BIT yang akan dibahas. Berikut implementasi yang akan digunakan pada tiga bentuk BIT dengan prerequisite array BIT sudah dibuat dengan aturan-aturan diatas.

```
void pointUpdate (int idx,int val){
   for(int i=idx;i>=MAX;i+=(i&(-i))){
       BIT[i]+=val;
   }
}
int pointQuery (int x){
   int ans=0;
   for(int i=x;i>=1;i-=(i&(-i))){
       ans+=BIT[i];
   }
   return ans;
}
int rangeQuery (int a,int b){
   return pointQuery(b) - pointQuery(a-1);
}
void rangeUpdate(int a, int b, int val){
       pointUpdate(a,val);
       pointUpdate(b+1, -val);
}
```

1. Point Update dan Range Query

Update hanya pada satu elemen dan query penjumlahan dari elemen ke-a sampai elemen ke-b setiap query nya. Gunakan dua fungsi rangeQuery dan pointUpdate saja. Dengan prerequisite array BIT sudah dibuat.

2. Range Update dan Point Query

Update berupa range dari elemen ke-a sampai elemen ke-b. Sedangkan query/info penjumlahan hanya pada satu elemen. Gunakan dua fungsi rangeUpdate dan pointQuery. Perhatikan prosedur rangeUpdate memanggil dua prosedur pointUpdate. pointUpdate pertama digunakan untuk menjumlahkan elemen ke-a sampai ke-n dengan nilai sebesar val. Sedangkan pointUpdate yang kedua digunakan untuk meniadakan efek update yang pertama pada elemen ke-(b+1) sampai elemen ke-n dengan cara menjumlahkan nilai sebesar -val (negatif val) ke setiap elemennya.

3. Range Update dan Range Query

Biasanya, range update dan range query diimplementasikan menggunakan Segmented Tree, namun sebenarnya kita juga bisa mengimplementasikannya menggunakan Binary Indexed Tree. Untuk melakukannya kita memerlukan array BIT tambahan untuk menyimpan informasi tambahan nilai pada range tertentu.

Misalkan kita melakukan rangeUpdate(a,b,v). Saat kita melakukan query(q), akan ada tiga kemungkinan tambahan nilai yang harus ditambahkan ke hasil query(q) :
a. 1 <= q < a : 0
b. a <= q <= b : v*(p-(a-1))
c. b < q <= n : v*(b-(a-1))

Sehingga, untuk mendapatkan nilai query(q) kita perlu mengalikannya dengan nilai yang ditambahkan lalu mengurangkan nilai x dari hasilnya : q*query(q,q) – X. Dengan X :
a. X=0 ; 1 <= q < a.
b. X=v*(a-1) ; a <= q <= b.
c. X=-(v*b) + (v*(a-1)) ; b < q <= n.

Oleh karena itu kita harus mengubah fungsi Update menjadi sebagai berikut.

```
void range_update(int i, int j, ll){
   update(BIT1, i, v);
   update(BIT1, j + 1, -v);
   update(BIT2, i, v * (i - 1));
   update(BIT2, j + 1, -v * j);
}
```

IV. Application

BIT banyak digunakan untuk optimasi pada permasalahan yang menggunakan banyak query dalam waktu yang berdekatan. Biasanya digunakan untuk algoritma-algoritma yang butuh optimasi pada struktur datanya. Pada awalnya Fenwick Tree atau BIT ini diciptakan untuk mengimplementasikan Arithmetic Coding Algorithm, yaitu salah satu algoritma kompresi seperti Huffman.

V. Conclusion

BIT dapat berjalan dengan kompleksitas waktu O(log n) untuk query, dan O(log n) pula untuk update. Sehingga struktur data ini dapat melakukan optimasi yang sangat besar pada permasalahan-permasalahan yang menggunakan query-query untuk update.

.

VI. Acknowledgment

Penulis berterimakasih kepada Dr. Ir. Rinaldi Munir, MT sebagai dosen penulis atas bimbingannya selama kelas Strategi Algoritma sehingga penulis dapat membuat makalah ini.

References

1. http://williamrjribeiro.com/?p=132

[1] http://williamrjribeiro.com/?p=132 (Retrieved May 13th 2018)
[2] https://sanugupta.wordpress.com/2014/08/29/binary-indexed-tree-fenwick-tree/ (Retrieved May 14th 2018)
[3] https://kartikkukreja.wordpress.com/2013/12/02/range-updates-with-bit-fenwick-tree/ (Retrieved May 13th 2018)
[4] https://www.quora.com/Which-is-the-most-amazing-data-structure?redirected_qid=787364#!n=12 (Retrieved May 14th 2018).

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

Ttd

Ensof Barhami 13516096