# Mencari Leluhur Bersama Terkecil pada Pohon

Chalvin 13514032[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*[1]13514032@std.stei.itb.ac.id*

***Abstract*—Teori Graf adalah ilmu yang mempelajari graf yaitu suatu struktur yabg memodelkan hubungan antar objek. Pohon merupakan salah satu bentuk graf yang memiliki hubungan orangtua dan anak. Lowest Common Ancestor mencari leluhur terkecil yang menghubungkan dua buah simpul. RMQ merupakan permasalahan mencari nilai paling minimum dari sebuah sublarik.**

***Keywords*—Graf, pohon, Lowest Common Ancestor, RMQ**

## I. PENDAHULUAN

Pada ilmu komputer dan teori graf, mencari leluhur terkecil yang sama (*lowest common ancestor*) merupakan salah satu masalah klasik yang dicanangkan oleh Alfred Aho, John Hopcroft, dan Jeffrey Ullman pada tahun 1973, namun baru pada tahun 1984 algoritma yang efisien untuk masalah ini dikembangkan oleh Dov Harel dan Robert Tarjan. Sayangnya, struktur data yang mereka pakai sangat kompleks dan masih sulit untuk diimplementasikan. Pada tahun 1988, Baruch Schieber dan Uzi Vishkin menyederhanakan struktur data dari Harel dan Tarjan, menghasilkan struktur data yang lebih gampang untuk diimplementasikan, dengan kompleksitas waktu yang sama.

*Lowest common ancestor (*selanjutnya disingkat LCA) dari u dan v pada pohon T adalah simpul terjauh dari akar teratas yang merupakan leluhur dari u dan juga v. Menghitung LCA berguna untuk beberapa hal seperti menjadi jarak antar simpul pada suatu pohon. Jarak antara u dan v dapat dihitung dengan menjumlahkan jarak dari LCA u dan v ke u, lalu jarak LCA u dan v ke v.

Pada struktur data pohon dimana setiap simpul menunjuk ke orangtua mereka, LCA dapat ditentukan dengan mudah dengan mencari lintasan dari u ke akar teratas, juga lintasan dari v ke akar teratas, dan mencari perpotongan lintasan mereka yang paling pertama.

## II. DASAR TEORI

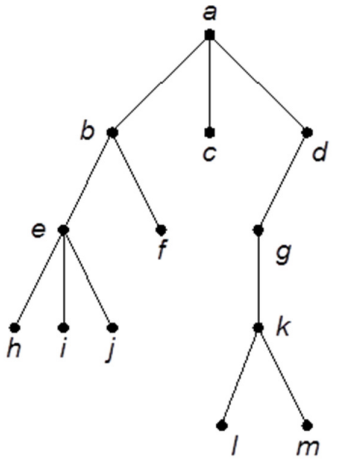
### A. Pohon dan Pohon Berakar

Pohon dalam ilmu komputer merupakan Struktur Data Abstrak yang sangat umum dan memliki banyak aplikasi dalam kehidupan sehari-hari.

Pohon didefinisikan sebagai suatu graf tak-berarah yang terhubung dan tidak mengandung sirkuit. Misalkan G = (V, E) yaitu graf tak-berarah sederhana yang memiliki n buah simpul. Maka sifat-sifat dibawah ini berlaku untuk G :

1. G merupakan suatu pohon.
2. Setiap pasang simpul di dalam pohon G dihubungkan oleh sebuah lintasan.
3. G terhubung dan memiliki sisi sebanyak m = (n-1) buah.
4. G tidak mengandung sirkuit dan memiliki sisi sebanyak m = (n-1) buah.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan menyebabkan munculnya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

Pada kebanyakan aplikasi pohon, simpul tertentu diperlakukan sebagai akar (*root)*. Sekali sebuah simpul ditetapkan sebagai akar, maka simpul-simpul lainnya dapat dicapai dari akar dengan memberi arah pada sisi-sisi pohon yang mengikuti. Pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah dinamakan pohon berakar (rooted tree).




GAMBAR 2.1 Contoh Pohon
Sumber : Slide Kuliah IF2120 Matematika Diskrit materi Pohon oleh Rinaldi Munir

Pohon berakar memiliki beberapa terminologi umum yang akan dipakai dalam makalah ini, diantaranya :

1. Anak (child atau children) dan Orangtua (parent)
   Misalkan x adalah sebuah simpul di dalam pohon berakar. Simpul y dikatakan anak dari x jika ada sisi

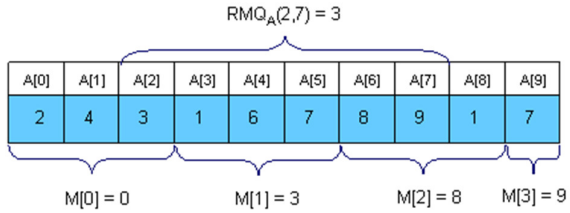
dari x ke y. Dalam hal demikian, x disebut orangtua dari y. Pada gambar 2.1, b,c,d merupakan anak dari a, e dan f adalah anak dari b dan seterusnya.

2. Lintasan (*path*)
   Lintasan dari simpul $v_1$ ke simpul $v_k$ adalah runtutan simpul-simpul $v_1$, $v_2$, ... ,$v_k$ sedemikian sehingga $v_i$ adalah orangtua dari $v_{i+1}$ untuk 1 <= i <= k. Pada gambar 2.1, lintasan dari a ke j adalah a, b, e, j.
3. Keturunan (descendant) dan leluhur(*ancestor*)
   Jika terdapat lintasan dari simpul x ke simpul y di dalam pohon, maka x adalah leluhur dari simpul y, dan y adalah keturunan dari simpul x. Pada gambar 2.1, b adalah leluhur dari simpul h, dengan demikian h merupakan keturunan dari simpul b.

B. Range Minimum Query

Pada ilmu komputer, *range minumum query* (selanjutnya disingkat RMQ) adalah permasalahan mencari nilai paling minimum dari sebuah sublarik pada sebuah larik. Solusi *naive* melibatkan sebuah matriks misalkan B dimana B[i,j] merupakan nilai paling minimum dari sublarik A[i..j]. Kompleksitas dari algoritma ini yaitu $O(n^3)$ untuk pemrosesan matriks dan O(1) untuk query. Namun, dengan memanfaatkan *dynamic programming*, kompleksitas algoritma untuk pemrosesan matriksnya dapat direduksimenjadi $O(n^2)$ saja.

Aproksimasi lain adalah dengan membagi larik A dengan panjang N menjadi akar N buah sublarik. Kita menyimpan posisi minimum dari tiap sublarik pada larik M[0..akar N-1]. M dapat diproses dalam O(N). Perhatikan contoh dibawah




GAMBAR 2.2 – Aproksimasi kedua *RMQ*
Sumber : https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/ diakses pada 11 Desember 2015 jam 3:40

Sekarang mari kita hitung RMQ(i,j). Idenya adalah mencari nilai paling minimum dari sublarik yang berada dalam interval, dan dari awal hingga akhir sublarik yang memotong batas interval. Untuk mendapat RMQ(2,7), kita cukup membandingkan A[2], A[M[1]], A[6], dan A[7] dan mendapatkan indeks dengan nilai paling minimum. Jika diperhatikan, algoritma ini tidak melakukan lebih dari 3*akar N buah operasi tiap querynya.

Aproksimasi terakhir yang akan dibahas adalah pencarian RMQ dengan memanfaatkan pohon segmen, yang mana merupakan cara lain untuk menyusun data pada pohon biner. Ada beberapa cara untuk mengimplementasikan pohon segmen. Implementasi kami memakai vector<int> *st* pada bahasa C++ untuk merepresentasikan pohon biner. Inter 1 (melewati indeks 0) adalah akar dan anak kiri dan anak kanan dari indeks p adalah indeks 2 x p dan (2 x p) + 1 secara berurutan. Nilai dari st[p] adalah nilai RMQ dari segmen yang diasiosiasikan dengan indeks p.

Akar dari pohon segmen merepresentasikan segmen [0, n-1]. Untuk setiap segmen [L,R] yang disimpan pada indeks p dimana L != R, segmennya akan dipecah menjadi segmen [L, (L+R)/2] dan [(L+R)/2 + 1, R]. sub-segmen tersebut akan disimpan secara berurutan pada indeks 2 x p dan (2 x p) +1. Ketika L = R, jelas bahwa st[p] = L.

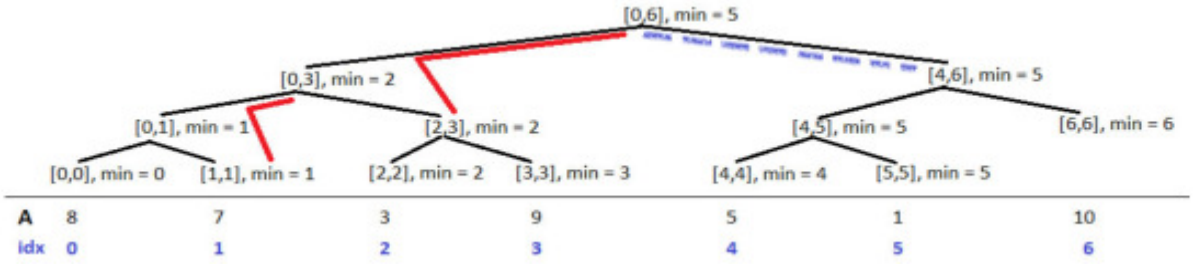
Sebaliknya, secara rekursif kita bangun pohon segmen, menbandingkan nilai dari subsegmen kiri dan kanan dan meng*update* nilai st[p].

Setelah pohon segmen selesai dibangun, query dapat dilakukan dengan kompleksitas O(log n). Misalkan kita mencari nilai RMQ(1,3). Kita mulai dari akar (indeks 1) yang merepresentasikan segmen [0,6]. Kita tidak dapat menggunakan nilai yang disimpan pada segmen [0,6] = st[1] = 5 ksebagai jawaban dari RMQ(1,6) karna nilai yang disimpan merupakan nilai minimum dari segmen yang lebih besar dari pada yang kita cari pada RMQ(1,3). Dari akar, kita hanya menjelajahi anak kirinya karena anak kanannya merepresentasikan segmen [4,6] yang mana sudah diluar dari segmen yang kita cari pada RMQ(1,3).

Sekarang kita berada pada akar dari anak kiri yang merepresentasikan segmen [0,3]. Segmen ini [0,3] masih lebih besar daripada segmen yang kita cari pada RMQ(1,3). Malahan, RMQ(1,3) berpotongan dengan kedua anak kiri (segmen[0,1]) ]dan anak kanan (segmen[2,3]) dari segmen [0,3], sehingga kita harus menulusuri anak kiri dan anak kanan.

Pada anak kiri, kita berada pada segmen [0,1] yang mana masih diluar dari segmen [1,3] yang kita inginkan,

Gambar 2.3 Segment Tree
Sumber : Competitive Programming 1 oleh Steven dan Felix Halim



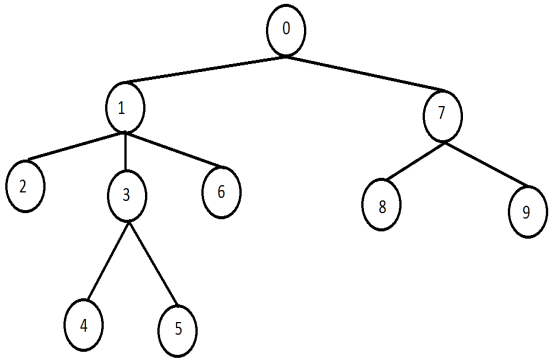

sehingga dari segmen [0,1] kita menuju ke anak kanannya yaitu segmen [1,1], yang mana sekarang berada pada segmen [1,3]. Pada titik ini, kita tau nilai dari RMQ(1,1) = st[9] = 1 dan kita bisa mengembalikan nilai

ini. Segmen kanan [2,3] dari segmen [0,3] merupakan bagian dari segmen [1,3]. Dari nilai yang disimpan pada simpul ini, kita tau bahwa RMQ(2,3) = st[5] = 2. Kita tidak perlu lagi menulusuri anak-anak dari segmen ini.

Sekarang, pada segmen [0,3] misalkan p1 = RMQ(1,1) = 1 dan p2 = RMQ(2,3) = 2. Karena A[p1] = 17 > A[p2] = 13, maka nilai RMQ(1,3) = p2 = 2.

Untuk mencari nilai RMQ(4,6), kita mulai lagi dari akar teratas yaitu segmen[0,6]. Karena segmen [0,6] lebih besar dibanding segmen yang kita inginkan pada RMQ(4,6), kita menulusuri anak kanan segmen [0,6], yaitu segmen [4,6]. Karena segmen [4,6] merupakan segmen yang kita inginkan, kita tinggal mengembalikan nilai yang disimpan oleh segmen [4,6] yaitu st[3] = 5.

## III. PEMBAHASAN




Gambar 3.1

### A. Aproksimasi trivial/naif

Metode trivial untuk mencari LCA(u,v) adalah mencari lintasan dari u sampai ke akar paling atas dan mencatat lintasan tersebut. Lalu kita lakukan hal yang sama dari v, hanya saja kita langsung berhenti bila menemukan simpul yang terdapat pada lintasan u ke akar teratas. Simpul tersebut merupakan LCA dari u dan v.

Misalkan kita ingin mencari LCA dari(2,5) dari gambar 3.1. Pertama, kita cari lintasan dari 2 ke 0 yaitu 2 -> 1 -> 0 dan mencatat lintasan ini. Kemudian kita cari lintasan dari 2 ke 1 yaitu 5 -> 3 -> 1 lalu berhenti karena 1 ada pada lintasan 2 ke 0. Sehingga LCA(2,5) adalah 1.
Algoritma ini membutuhkan waktu O(N) tiap querynya.

### B. Reduksi menjadi *Range Minimum Query*

Kita dapat mereduksi masalah LCA ini menjadi masalah RMQ. Kuncinya adalah memperhatikan bahwa LCA(u,v) adalah simpul terdangkal pada pohon yang dilalui oleh u dan v ketika mencari jalan ke akar teratas. Jadi, yang perlu kita lakukan hanyalah melakukan DFS(Depth First Search) pada pohon dan mencatat informasi tentang kedalaman (*depth* pada *DFS)* dan waktu kita melewati setiap simpul. Perhatikan bahwa kita akan mengunjungi 2 x N – 1 buah simpul dalam DFS kita karena ada beberapa simpul yang akan dikunjungi lebih dari sekali. Kita perlu membuat tiga buah larik ketika menjalankan DFS ini : E[0..2 x N - 2] untuk mencatat simpul yang kita kunjungi, L[0..2 x N - 2] untuk mencatat kedalaman tiap simpul ayng dikunjungi, dan H[0..N-1] untuk mencatat index pertama i muncul di E. Kunci dari implementasinya adalah sebagai berikut

```
int L[2*MAX_N], E[2*MAX_N], H[MAX_N], idx;
void dfs(int cur, int depth){
        H[cur] = idx;
        E[idx] = cur;
        L[idx++] = depth;
        For(int i = 0; i < children[cur].size(); i++)
        {
                Dfs(children[cur][i], depth+1);
                E[idx] = cur;
                L[idx++] = depth;
        }
}
```

Sebagai contoh, jika kita memanggil fungsi dfs(0,0) pada pohon di gambar 3.1, kita akan mempunyai

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| H | 0 | 1 | 2 | 4 | 5 | 7 | 10 | 13 | 14 | 16 |
| E | 0 | 1 | 2 | 1 | 3 | 4 | 3 | 5 | 3 | 1 |
| L | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 2 | 1 |

| Index | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| H | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| E | 6 | 1 | 0 | 7 | 8 | 7 | 9 | 7 | 0 |
| L | 2 | 1 | 0 | 1 | 2 | 1 | 2 | 1 | 0 |

Setelah kita mendapatkan 3 larik tersebut, kita dapat menyelesaikan LCA menggunakan RMQ. Asumsikan H[u] < H[v]. Perhatikan bahwa masalahnya tereduksi menjadi mencari vertex dengan kedalaman terkecil pada E[H[u],H[v]]. Jadi, solusinya menjadi LCA(u,v) = E[RMQ(H[u],H[v])] dimana RMQ(i,j) dijalankan pada larik L.

## V. KESIMPULAN

LCA dapat direduksi menjadi permasalahan RMQ, juga mereduksi kompleksitasnya yang awalnya O(N) tiap querynya menggunakan aproksimasi naif, menjadi O(log N) tiap querynya.

## VI. REFERENSI


- "The LCA Problem Revisited" [PPT] oleh Michael A.Bender and Martin Farach-Colton
- Competitive Programming oleh Steven Halim dan Felix Halim
- Introduction to Algorithm oleh Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, dan Clifford Stein
- http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).ppt
- https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/
- http://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/


## VII. APPENDIX

1. Implementasi RMQ dengan Segment tree dalam C (dari http://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/)

```
// C program for range minimum query using segment tree
#include <stdio.h>
#include <math.h>
#include <limits.h>

// A utility function to get minimum of two numbers
int minVal(int x, int y) { return (x < y)? x: y; }

// A utility function to get the middle index from corner indexes.
int getMid(int s, int e) { return s + (e -s)/2; }

/* A recursive function to get the minimum value in a given
range
        of array indexes. The following are parameters for this
function.

        st --> Pointer to segment tree
        index --> Index of current node in the segment tree.
Initially
                        0 is passed as root is always at
index 0
        ss & se --> Starting and ending indexes of the segment
represented
                                by current node, i.e.,
st[index]
        qs & qe --> Starting and ending indexes of query
range */
int RMQUtil(int *st, int ss, int se, int qs, int qe, int index)
{
        // If segment of this node is a part of given range, then
return
        // the min of the segment
        if (qs <= ss && qe >= se)
                return st[index];

        // If segment of this node is outside the given range
        if (se < qs || ss > qe)
                return INT_MAX;

        // If a part of this segment overlaps with the given
range
        int mid = getMid(ss, se);
        return minVal(RMQUtil(st, ss, mid, qs, qe,
2*index+1),
                                RMQUtil(st, mid+1, se,
qs, qe, 2*index+2));
}

// Return minimum of elements in range from index qs (quey
start) to
// qe (query end). It mainly uses RMQUtil()
int RMQ(int *st, int n, int qs, int qe)
{
        // Check for erroneous input values
        if (qs < 0 || qe > n-1 || qs > qe)
        {
                printf("Invalid Input");
                return -1;
        }

        return RMQUtil(st, 0, n-1, qs, qe, 0);
}
```

```
// A recursive function that constructs Segment Tree for
array[ss..se].
// si is index of current node in segment tree st
int constructSTUtil(int arr[], int ss, int se, int *st, int si)
{
        // If there is one element in array, store it in current
node of
        // segment tree and return
        if (ss == se)
        {
                st[si] = arr[ss];
                return arr[ss];
        }

        // If there are more than one elements, then recur for
left and
        // right subtrees and store the minimum of two values
in this node
        int mid = getMid(ss, se);
        st[si] = minVal(constructSTUtil(arr, ss, mid, st,
si*2+1),

        constructSTUtil(arr, mid+1, se, st, si*2+2));
        return st[si];
}

/* Function to construct segment tree from given array. This
function
allocates memory for segment tree and calls constructSTUtil() to
fill the allocated memory */
int *constructST(int arr[], int n)
{
        // Allocate memory for segment tree

        //Height of segment tree
        int x = (int)(ceil(log2(n)));

        // Maximum size of segment tree
        int max_size = 2*(int)pow(2, x) - 1;

        int *st = new int[max_size];

        // Fill the allocated memory st
        constructSTUtil(arr, 0, n-1, st, 0);

        // Return the constructed segment tree
        return st;
}

// Driver program to test above functions
int main()
{
        int arr[] = {1, 3, 2, 7, 9, 11};
        int n = sizeof(arr)/sizeof(arr[0]);

        // Build segment tree from given array
        int *st = constructST(arr, n);

        int qs = 1; // Starting index of query range
        int qe = 5; // Ending index of query range

        // Print minimum value in arr[qs..qe]
        printf("Minimum of values in range [%d, %d] is =
%d\n",
                                                        qs,
qe, RMQ(st, n, qs, qe));

        return 0;
```

}

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, 8 Desember 2015

Chalvin 13514032