Aplikasi *Binary Search Tree* Sebagai Metode Pencarian Dalam Kamus

Nurul Utami Amaliah Widodo - 13517132
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517132@itb.ac.id

***Abstract*—Pencarian adalah salah satu masalah yang paling umum dan memiliki banyak implementasi dalam kehidupan sehari-hari. Salah satu metode yang paling efektif digunakan untuk melakukan proses pencarian adalah dengan *binary search tree.* Dengan *binary searh tree* dan pencarian secara rekursif, metode ini dapat dimanfaatkan untuk melakukan proses pencarian data yang sangat besar dan banyak dalam kamus.**

***Keywords—binary search tree*, algoritma, metode pencarian, struktur data.**

I. PENDAHULUAN

Pencarian atau *searching* adalah salah satu masalah yang paling umum ditemukan dan memiliki banyak implementasi dalam kehidupan sehari-hari. *Searching* ini sendiri pun memiliki 2 metode, yaitu pencarian sekunsial (*sequential search*) dan pencarian biner (*binary search*). *Binary search* adalah suatu metode yang digunakan untuk melakukan pencarian pada data yang telah diurutkan.

Binary search ini merupakan metode yang digunakan pada pohon pencarian biner atau *binary search tree* (BTS). *Binary Search Tree* adalah pohon biner yang memiliki aturan tersendiri dalam mengatur keterurutan data sehingga dapat digunakan untuk melakukan pencarian. Pohon biner (*binary tree*) adalah bentuk pohon *n-ary* dengan n=2, dengan setiap simpul cabangnya mempunyai paling banyak dua buah anak. *Binary search tree* dapat dikatakan sebagai pohon biner paling penting, khususnya dalam operasi pencarian, penyisipan, dan penghapusan elemen karena kinerjanya yang baik ditinjau dari waktu pencarian.

Masalah pencarian merupakan masalah yang sangat sering dilakukan pada penggunaan kamus, dalam bentuk apapun. Mulai dari kamus bentuk buku (konvensional) sehingga kita harus melakukan secara manual, hingga kamus berbentuk digital sehingga teknik pencariannya pun dikembangkan bagaimana agar pencarian ini dilakukan dengan seefetif dan seefisien mungkin.

Kamus memiliki data yang sangat banyak karena dalam setiap bahasa bisa memiliki ribuan bahkan hingga jutaan kosakata. Kamus sendiri setidaknya memiliki dua bahasa yang berarti jumlah data akan mencapai dua kali lipat pasangan kosakata dan artinya dari salah satu bahasa. Dengan banyaknya data yang ada dalam kamus, maka sangat perlu penerapan metode pencarian seefisien mungkin dalam melakukan pencarian suatu kosakata di dalamnya.

Kamus merupakan suatu kumpulan data terurut secara menaik (*ascending*). Dengan begitu teknik pencarian dengan metode binary search dapat diterapkan pada pencarian kosakata di kamus. Maka, melalui makalah ini, penulis tertarik untuk membahas pengaplikasian *binary search tree* sebagai metode teknik pencarian di kamus.

II. LANDASAN TEORI

A. Kamus

Kata kamus diserap dari Bahasa Arab Qamus (قاموس), dengan bentuk jamaknya qawamis. Kata Arab itu sendiri berasal dari kata Yunani Ωκεανός (okeanos) yang berarti ‘samudra’. Makna dasar yang terkandung dalam kata kamus adalah wadah pengetahuan, khususnya pengetahuan bahasa, yang tidak terhingga dalam dan luasnya.

Menurut Kamus Besar Bahasa Indonesia kamus adalah buku acuan yang memuat kata dan ungkapan, biasanya disusun menurut abjad berikut keterangan tentang makna, pemakaian, atau terjemahannya. Menurut KBBI juga, kamus adalah buku yang memuat kumpulan istilah atau nama yang disusun menurut abjad beserta penjelasan tentang makna, pemakaian, atau terjemahannya.

Dari pengertian kamus yang diuraikan sebelumnya diketahui bahwa kamus merupakan kumpulan data yang tersusun secara terurut, baik secara menaik (*ascending*) maupun menurun (*descending*) menurut abjad.

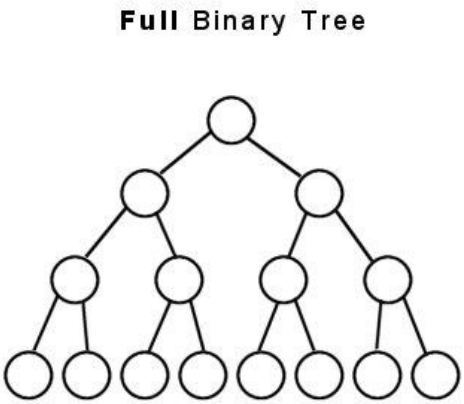
Menurut Pusat Bahasa Departemen Pendidikan Nasional, ada lima prosedur umum yang harus dilalui dalam melakukan penyusunan kamus, yaitu perancangan, pembinaan data korpus, pengisihan dan pengabjadan data, pengolahan data, dan pemerian makna.

B. Pohon Biner

Pohon biner (*binary tree*) merupakan pohon n-ary jika n=2, dengan setiap simpul cabangnya mempunyai paling banyak dua buah anak yang disebut anak kiri (*left child*) dan anak kanan (*right child*). Pohon yang akarnya adalah anak kiri disebut upapohon kiri (*left subtree*), sedangkan pohon yang akarnya adalah anak kanan disebut upapohon kanan (*right subtree*).

Kunci pada subpohon kiri dan kanan berbeda, karena perbedaan itu maka pohon biner adalah pohon terurut. Pohon

yang simpulnya hanya terletak di salah satu sisi disebut pohon condong (*skewed tree*). Jika simpulnya semua di kiri disebut pohon condong-kiri (*skew left*), sedangkan jika simpulnya semua di kanan disebut pohon condong-kanan (*skew right*). Pohon biner dikatakan pohon biner penuh (*full binary* tree) ketika semua simpulnya memiliki dua anak kecuali anak yang paling terakhir (daun).



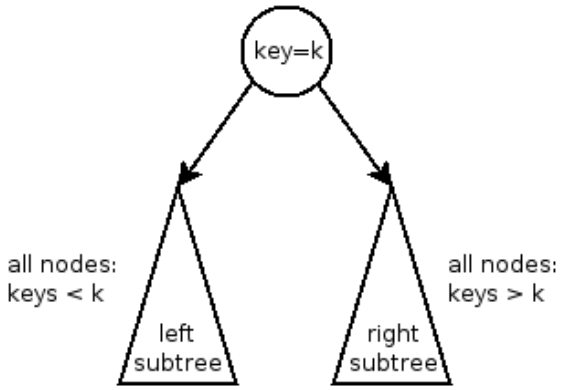
Gambar 1*: Full Binary Tree*
(Sumber: http://web.cecs.pdx.edu/~sheard/course/Cs163/Doc/FullvsComplete.html)

C. Pohon Pencarian Biner

Pohon pencarian biner (*binary search tree*) adalah pohon biner dengan aturan struktur data tertentu, yaitu:

1. Subpohon kiri mengandung simpul dengan kunci yang nilainya lebih kecil daripada nilai kunci akarnya.
2. Subpohon kanan mengandung simpul dengan kunci yang nilainya lebih besar daripada nilai kunci akarnya.
3. Baik subpohon kiri maupun subpohon kanan merupakan pohon biner.

Simpul dari pohon pencarian biner dapat berupa kunci pada data *record* atau data itu sendiri. Pohon biner dapat dibentuk menjadi *binary search tree* dengan mengikuti aturan yang disebutkan di atas.



Gambar 2: Skema *Binary Search Tree*
(Sumber: http://faculty.ycp.edu/~dhovemey/spring2007/cs201/info/binarySearchTrees.html)

Pohon pencarian biner juga biasa disebut *ordered/sorted binary tree* karena datanya yang terurut. Karena keterurutan itu maka *binary search tree* menjadi pohon yang paling sering digunakan khususnya dalam operasi pencarian, penyisipan dan penghapusan elemen karena memungkinkan kerja yang lebih cepat jika dibandingkan dengan struktur data yang lain. Operasi ini dapat dijalankan dengan menggunakan prinsip pencarian biner karena datanya yang terurut.

Proses pencarian dimulai dari akar kemudian membandingkan elemen yang dicari dengan kunci pada akar. Jika elemen lebih kecil daripada nilai kunci akar, proses pencarian akan dilanjutkan pada subphon kiri. Sedagkan jika elemen lebih besar daripada nilai kunci akar, proses pencarian akan dilanjutkan pada subpohon kanan. Begitu seterusnya proses tersebut dilakukan hingga mencapai simpul atau daun yang nilainya sama dengan kunci yang sedang diproses.

Metode ini memungkinkan untuk melakukan proses hanya pada setengah dari keseluruhan pohon sehingga setiap operasi pencarian, penyisipan, dan penghapusan elemen membutuhkan waktu yang sebanding dengan nilai dari logaritma jumlah *item* yang disimpan dalam pohon. Proses ini lebih cepat jika dibandingan dengan struktur data lainnya, seperti data linear pada array tidak terurut.

Terdapat tiga jenis traversal pada *binary search tree*, yaitu:

1. Pre-order Traversal
   Pre-order traversal dilakukan dengan urutan proses data pada akar, lakukan traversal pada subpohon kiri secara keseluruhan, kemudian lakukan traversal pada subpohon kanan secara keseluruhan.

```
1   void perorder(struct node*root)
2   {
3       if(root)
4       {
5           printf("%d ",root-&gt;data);    //Printf root-&gt;data
6           preorder(root-&gt;left);    //Ke left subtree
7           preorder(root-&gt;right);     //Ke right subtree
8       }
9   }
10
```

Gambar 3: Algoritma Pre-Order Traversal *Binary Search Tree*
(Sumber: https://indonesia.hackerearth.com/konsep-dasar-binary-search-tree-pohon-pencarian-biner/)

2. In-oder Traversal
   In-order traversal dilakukan dengan urutan proses melakukan traversal pada subpohon kiri secara kecara keseluruhan, proses data pada akar, kemudian lakukan traversal pada subpohon kanan secara keseluruhan.

```
1   void inorder(struct node*root)
2   {
3       if(root)
4       {
5           inorder(root-&gt;left);    //Ke left subtree
6           printf("%d ",root-&gt;data);    //Printf root-&gt;data
7           inorder(root-&gt;right);     //Ke right subtree
8       }
9   }
10
```

Gambar 4: Algoritma In-Order Traversal *Binary Search Tree*
(Sumber: https://indonesia.hackerearth.com/konsep-dasar-binary-search-tree-pohon-pencarian-biner/)

3. Post-order Traversal
   Post-order traversal dilakukan dengan urutan proses melakukan traversal pada subpohon kiri secara keseluruhan, melakukan traversal pada subphon kanan secara keseluruhan, kemudian proses data pada akar.

```
    void postorder(struct node*root)
    {
        if(root)
        {
            postorder(root-&gt;left);    //Ke left subtree
            postorder(root-&gt;right);     //ke right subtree
            printf("%d ",root-&gt;data);    //Printf root-&gt;data
        }
    }

```

Gambar 5: Algoritma Post-Order Traversal Binary Search Tree
(Sumber: https://indonesia.hackerearth.com/konsep-dasar-binary-search-tree-pohon-pencarian-biner/)

Namun ketika kasusnya telah dihadapkan pada sebuah pohon kita harus melakukan verifikasi apakah pohon tersebut telah membentuk pohon pencarian biner atau bukan. Verifikasi dilakukan dengan menegcek apakah pohon tersebut mengikuti aturan pohon pencarian biner.

```
struct TreeNode {
    int key;
    int value;
    struct TreeNode *left;
    struct TreeNode *right;
};

bool isBST(struct TreeNode *node, int minKey, int maxKey) {
    if (node == NULL) return true;
    if (node->key < minKey || node->key > maxKey) return false;

    return isBST(node->left, minKey, node->key-1) && isBST(node->right, node->key+1, maxKey);
}
```

Gambar 6: Algoritma Verifikasi *Binary Search Tree*
(Sumber: https://en.wikipedia.org/wiki/Binary_search_tree)

Untuk melakukan penyisipan, aturannya sama ketika akan membuat atau mengubah pohon biner biasa (atau bentuk lainnya) menjadi sebuah pohon pencarian biner. Yang dilakukan adalah membandingkan antara elemen yang akan dimasukkan/disisipkan dengan kunci akar. Jika elemen lebih kecil daripada nilai kunci akar disisipkan ke subpohon kiri sedangkan jika elemen nilainya lebih besar daripada nilai kunci akar, elemen akan disisipkan di subpohon kanan.

```
    struct node* insert(struct node* root, int data)
    {
        if (root == NULL)    //Jika pohon kosong, lakukan return pada satu node tunggal
            return newNode(data);
        else
        {
            //Selain kondisi tersebut, ulangi pohon
            if (data &lt;= root-&gt;data)
                root-&gt;left  = insert(root-&gt;left, data);
            else
                root-&gt;right = insert(root-&gt;right, data);
            //return root pointer yang tidak diubah
            return root;
        }
    }

```

Gambar 7: Algoritma *Insertion Binary Search Tree*
(Sumber: https://indonesia.hackerearth.com/konsep-dasar-binary-search-tree-pohon-pencarian-biner/)

III. ANALISIS DAN PEMBAHASAN

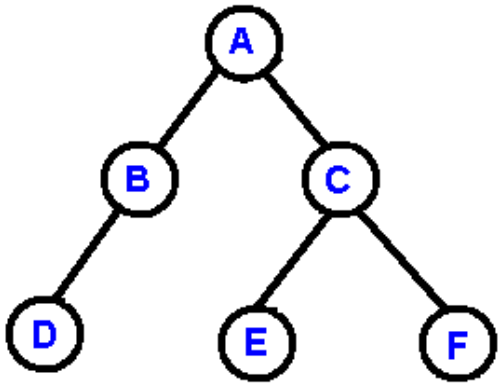
A. Batasan Masalah

Dari definisi kamus sendiri yang terlalu luas memungkinkan banyaknya klasifikasi data yang dalam implemetasinya butuh lebih banyak struktur data selain *binary search tree* dalam melakukan proses pencarian. Selain itu metode pengurutan dalam kamus memungkinkan untuk melakukan pengurutan secara *ascending* maupun *descending*.

Maka yang akan dibahas dalam makalah ini adalah kamus istilah. Dimungkinkan untuk kamus bahasa namun hanya untuk satu arah translasi. Untuk pengurutan data diambil konvensi paling umum yaitu pengurutan menaik atau *ascending*. Data pada *binary search tree* merupakan kunci pada data record ke makna dari kosakata tersebut.

B. Analisa Pencarian Data Dalam Kamus

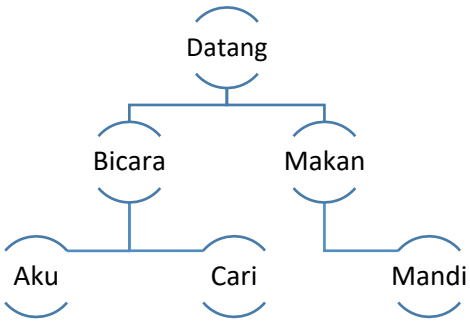
Prinsip dari pencarian data menggunakan *binary search tree* adalah membentuk data menjadi *binary search tree* agar data menjadi terurut.



Gambar 8: Contoh Pengurutan Pada Kamus dengan *Binary Search Tree*
(Sumber: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/pix/binaryTree.bmp)

Misalnya akan dilakukan pencarian terhadap elemen E.Akan ada empat kondisi pada proses pencarian:

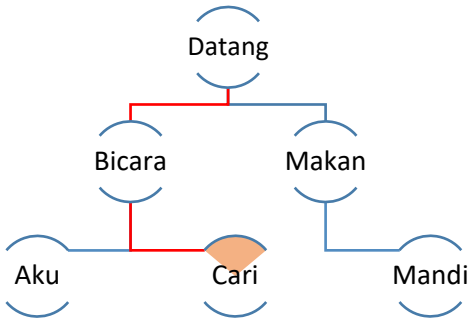
1. Jika data target E langsung ditemukan (nilai kunci akar bernilai sama dengan elemen E), maka proses akan berhenti dan menghasilkan kunci record ke data makna dari kata tersebut.
2. Jika data target E < nilai kunci akar, maka pencarian hanya akan dilakukan pada subpohon kiri. Seluruh elemen yang terdapat pada subpohon kiri dapat diabaikan.
3. Jika data target E > nilai kunci akar, maka pencarian hanya akan dilaukan pada subpohon kanan. Seluruh elemen yang terdapat pada subpohon kiri dapat diabaikan.
4. Jika data target E tidak ditemukan dimanapun maka akan diberikan pesan kesalahan.



Gambar 9: Contoh Proses Pencarian Data

Untuk contoh pohon pencarian biner pada gambar di atas, jika kita mencari kata “Cari” maka:

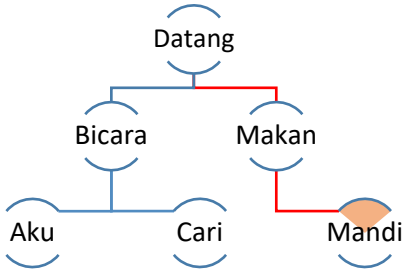
1. Nilai kunci akar = Datang.
   Mengecek apakah kata Cari = Datang. Karena tidak sama melanjutkan pengecekan apakah Cari < Datang ataukah Cari > Datang. Karena Cari < Datang maka pencarian dilanjutkan ke subpohon kiri.
2. Nilai kunci akar = Bicara.
   Mengecek apakah kata Cari = Bicara. Karena tidak sama melanjutkan pengecekan apakah Cari < Bicara ataukah Cari > Bicara. Karena Cari > Datang maka pencarian dilanjutkan ke subpohon kanan.
3. Nilai kunci akar = Cari.
   Mengecek apakah kata Cari = Cari. Karena sama maka akan melanjutkan ke data *record* dengan makna untuk kata Cari.



Gambar 10: Contoh Proses Pencarian Data

Jika mencari kata Mandi, maka:

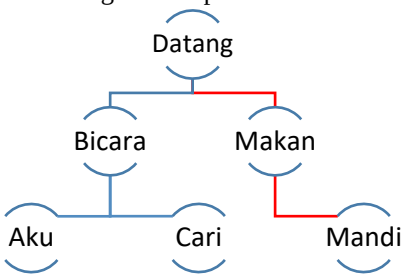
1. Nilai kunci akar = Datang.
   Mengecek apakah kata Mandi = Datang. Karena tidak sama melanjutkan pengecekan apakah Mandi < Datang ataukah Mandi > Datang. Karena Mandi > Datang maka pencarian dilanjutkan ke subpohon kanan.
2. Nilai kunci akar = Makan.
   Mengecek apakah kata Mandi = Makan. Karena tidak sama melanjutkan pengecekan apakah Mandi < Makan ataukah Mandi > Makan. Karena Mandi > Makan maka pencarian dilanjutkan ke subpohon kanan.
3. Nilai kunci akar = Mandi.
   Mengecek apakah kata Mandi = Mandi. Karena sama maka akan melanjutkan ke data *record* dengan makna untuk kata Mandi.



Gambar 11: Contoh Proses Pencarian Data

Jika mencari kata Minum, maka:

1. Nilai kunci akar = Datang.
   Mengecek apakah kata Minum = Datang. Karena tidak sama melanjutkan pengecekan apakah Minum < Datang ataukah Minum > Datang. Karena Minum > Datang maka pencarian dilanjutkan ke subpohon kanan.
2. Nilai kunci akar = Makan.
   Mengecek apakah kata Minum = Makan. Karena tidak sama melanjutkan pengecekan apakah Minum < Makan ataukah Minum > Makan. Karena Minum > Makan maka pencarian dilanjutkan ke subpohon kanan.
3. Nilai kunci akar = Mandi.
   Mengecek apakah kata Minum = Mandi. Karena tidak sama melanjutkan pengecekan apakah Minum < Makan ataukah Minum > Makan. Karena Minum > Makan maka pencarian dilanjutkan ke subpohon kanan. Namun tidak ada subpohon kanan, maka kata Minum tidak ditemukan dan akan mengirimkan pesan kesalahan.



Gambar 12: Contoh Proses Pencarian Data

C. Algoritma Pencarian Data di Kamus

Skema:

1. Jika data target E langsung ditemukan (nilai kunci akar bernilai sama dengan elemen E), maka proses akan berhenti dan menghasilkan kunci record ke data makna dari kata tersebut.
2. Jika data target E < nilai kunci akar, maka pencarian hanya akan dilakukan pada subpohon kiri. Seluruh elemen yang terdapat pada subpohon kiri dapat diabaikan.
3. Jika data target E > nilai kunci akar, maka pencarian hanya akan dilaukan pada subpohon kanan. Seluruh elemen yang terdapat pada subpohon kiri dapat diabaikan.

Seluruh proses dari skema tersebut terus dilakukan secara berulang (rekursif).

```
def search_recursively(key, node):

    if node is None or node.key == key:
        return node
    if key < node.key:
        return search_recursively(key, node.left)
    # key > node.key
    return search_recursively(key, node.right)
```

Gambar 12: Algoritma Proses Pencarian Data Rekursif

Algoritma tersebut juga dapat dituliskan dalam bentuk lain:

```
def search_iteratively(key, node):
    current_node = node
    while current_node is not None:
        if key == current_node.key:
            return current_node
        if key < current_node.key:
            current_node = current_node.left
        else: # key > current_node.key:
            current_node = current_node.right
    return current_node
```

Gambar 13: Algoritma Proses Pencarian Data Iteratif

IV. KESIMPULAN

Ada banyak struktur data yang dapat digunakan untuk melakukan proses pencarian data seperti *sequential search* berbasis array hingga table hash. Namun salah satu strutur data dengan efektifitas waktu yang paling baik adalah menggunakan metode pencarian dengan *binary search tree*.

Penggunaan *binary search tree* pada pencarian data di kamus mampu memangkas setengah jalan pencarian pada kasus terburuk dalam proses pencariannya. Metode menggunakan *binary search tree* ini tidak hanya dapat digunakan pada pencarian di kamus, namun dapat juga digunakan dalam proses pencarian lainnya yang lebih umm.

V. UCAPAN TERIMA KASIH

Pertama, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas segala limpahan nikmat dan kesempatan sehingga penulis mampu menyelesaikan makalah ini. Kemudian penulis mengucapkan terima kasih kepada seluruh dosen pengampu mata kuliah IF2120 Matematika Diskrit, khususnya kepada Bapak Judhi Santoso selaku dosen Matematika Diskrit untuk K03. Selanjutnya penulis juga mengucapkan terima kasih untuk seluruh dukungan dan dorongan dari keluarga. Terakhir penulis mengucapkan terima kasih kepada seluruh pihak yang tidak dapat disebutkan satu per satu yang telah berkontribusi, baik secara langsung maupun tidak langsung dalam penyelesaian makalah ini.

REFERENSI

[1] Munir, Rinaldi. *Matematika Diskrit*, Bandung: Informatika, 2012, edisi kelima.
[2] https://kbbi.kemdikbud.go.id/, diakses tanggal 9 Desember 2018, pukul 07:10 GMT+7.
[3] Ahmad, Ibrahim. *Perkamusan Melayu: Suatu Pengenalan*. Kuala Lumpur: Dewan Bahasa dan Pustaka, 2002.
[4] Pusat Bahasa Departemen Pendidikan Nasional. *Kamus Besar Bahasa Indonesia*, Jakarta: Balai Pustaka, 2002.
[5] https://www.geeksforgeeks.org/binary-search-tree-data-structure/, diakses tanggal 9 Desember 2018, pukul 08:04 GMT+7.
[6] https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/tutorial/, diakses tanggal 9 Desember 2018, pukul 09:07 GMT+7.
[7] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms: 2009 (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4.
[8] Robert Sedgewick, Kevin Wayne: Algorithms Fourth Edition. Pearson Education, 2011, ISBN 978-0-321-57351-3, p. 410.
[9] Mehlhorn, Kurt; Sanders, Peter (2008). Algorithms and Data Structures: The Basic Toolbox (PDF). Springer.
[10] Heger, Dominique A. (2004), "A Disquisition on The Performance Behavior of Binary Search Tree Data Structures" (PDF), European Journal for the Informatics Professional, 5 (5): 67–75
[11] Gonnet, Gaston. "Optimal Binary Search Trees". Scientific Computation. ETH Zürich. Archived from the original on 12 October 2014. Retrieved 1 December 2013.

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, 10 Desember 2018

Nurul Utami Amaliah Widodo - 13517132