Validasi Upapohon pada Struktur Data Pohon dengan Algoritma Pencocokan String

Sri Umay Nur'aini Sholihah (*13514007*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Infomatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung 40132, Indonesia
13514007@std.stei.itb.ac.id

***Abstrak*—Algoritma pencocokan string merupakan salah satu algoritma menarik yang bisa dieksplorasi penggunaannya, salah satunya dapat dimanfaatkan untuk mengetahui apakah sebuah pohon merupakan upapohon dari suatu pohon. Pohon (*tree*) adalah salah satu struktur data yang sering digunakan dalam dunia *programming*. Makalah ini akan membahas penggunaan algoritma pencocokan string untuk persoalan tersebut.**

Kata Kunci—pohon; upapohon; KMP, Boyer-Moore

I. PENDAHULUAN

Algoritma pencocokan string (*string matching algorithm*) merupakan salah satu algoritma yang sangat panyak penggunaanya saat ini. Penggunaan algoritma ini antara lain adalah pada mesin pencari, contohnya Google dan Bing, juga pada fitur pencarian kata yang terdapat pada editor *text* seperti Notepad dan Microsoft Word. Algoritma pencocokan string memang algoritma yang menarik karena dapat dieksplorasi untuk banyak hal. Pada makalah akan dijelaskan salah satu eksplorasi dari penggunaan algoritma pencocokan string yang dapat dimanfaatkan untuk mengetahui apakah suatu pohon merupakan upapohon dari suatu pohon lain.

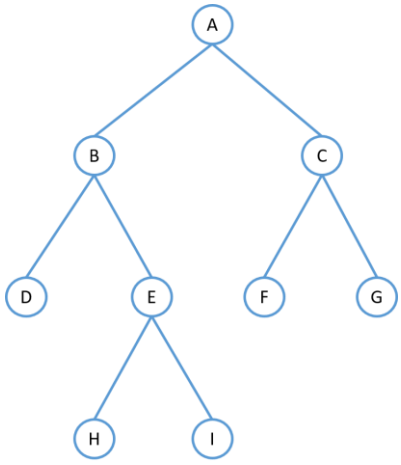
Struktur data pohon merupakan struktur data yang sering digunakan untuk menyimpan informasi dalam dunia *programming*. Struktur dari struktur data pohon ini relatif lebih rumit jika dibandingkan dengan struktur data seperti sederhana seperti *array*. Struktur data pohon bersifat rekursif, yaitu setiap bagian pohon juga merupakan suatu pohon. Dalam pengelolaannya struktur data pohon ini, ada kalanya pengguna struktur data pohon harus mengetahui apakah suatu ada pohon (bagian pohon yang kecil) yang merupakan upapohon dari pohon induk. Misal dalam suatu database yang tersimpan dalam bentuk struktur data pohon, pengguna struktur data ini ingin mengetahui apakah sebuah data (yang juga suatu pohon) merupakan bagian atau benar ada dalan database tersebut. Untuk menyelesaikan permasalahan tersebut, perlu algoritma yang cukup mangkus, sebab permasalah pencarian informasi dalam suatu database bisanya menuntut kecepatan dan keakuratan. Dengan menggunakan algoritma pencocokan string yang akan dibahas pada makalah ini, persoalan validasi apakah suatu pohon merupakan upapohon dari suatu pohon lain akan dapat diselesaikan dengan kompleksitas yang cukup baik. Selain itu dengan metode ini, program juga tidak membutuhkan banyak memori sebab pohon disimpan dalam bentuk string yang memiliki memori yang ringkas.

II. DASAR TEORI

A. Struktur Data Pohon

Struktur data pohon merupakan struktur data yang sering digunakan dam memiliki banyak manfaat. Kelebihan dari struktur data pohon ini adalah sifatnya yang rekursif, yaitu bagian dari pohon adalh juga sebuah pohon. Hal tersebut merupakan kelebihan karena struktur data pohon dapat dibangun dengan fungsi rekursif sehingga lebih cepat dan efisien, sementara merupakan kekurangan karena strukturnya menjadi lebih rumit, karena memiliki banyak cabang dibandingkan dengan struktur data yang linear seperti array.

Struktur data pohon merupakan pengembangan dari struktur data graf. Pada struktur data pohon terdapat simpul-simpul yang menyimpan informasi. Berikut adalah reperesentasi struktur data pohon dengan graf dan akan dijelaskan terminologi-terminologi pada struktur data pohon.



Gambar 1 Struktur data pohon

1) Simpul

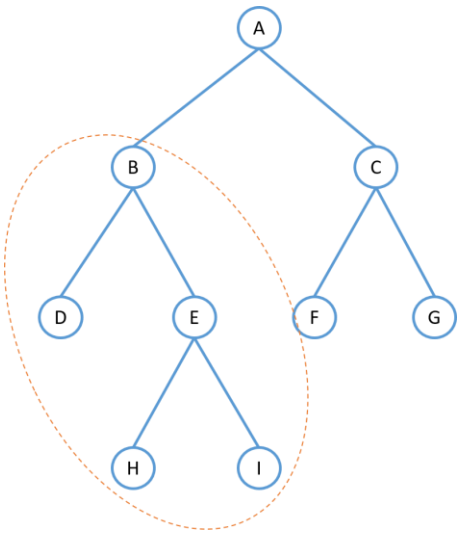
Simpul merupakan bagian dari pohon yang akan menyipan informasi, setiap simpul biasanya memiliki ID khusus untuk membedakan suatu simpul dengan simpul lainnya dlaam graf. Misal pada gambar 1, simpul memiliki ID berupa alfabet dari huruf A sampai dengan I. Jadi bisa disebut dengan simpul A, simpul B, dan seterusnya. Simpul utama, yaitu simpul yang tidak memiliki *parent* (biasanya terletak di posisi paling tinggi, contoh dalam gambar 1 simpul A) disebut simpul akar. Sementara simpul terluar dari pohon atau yang tidak memiliki anak disebut simpul daun.

2) Parent dan Child

Setiap simpul memiliki hubungan antar simpul yang direpresentasikan dengan sebuah garis, hubungan itu disebut dengan hubungan *parent-child*. Simpul *parent* adalah simpul yang derajatnya lebih tinggi dalam pohon. Misal pada gambar 1, simpul A adalah *parent* dari simpul B dan simpul C, sementara simpul B dan simpul C merupakan anak dari A.

3) Upapohon

Upapohon atau disebut juga subpohon adalah bagian dari pohon yang juga membentuk suatu pohon.



Gambar 2 Pohon dan upapohon

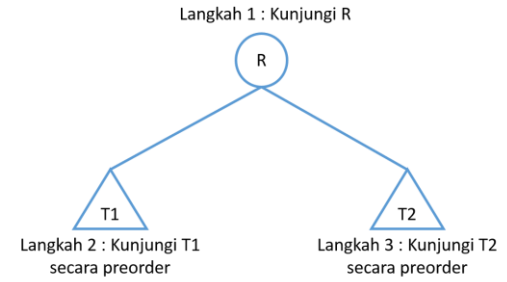
Pada gambar 2 dapat dilihat, bagian dari pohon yang diberi lingkaran dengan dengan garis putus-putus merupakan upapohon dari keseluruhan pohon. Upapohon itu sendiri juga merupakan sebuah pohon.

4) Representasi Struktur Data Pohon

Selain dengan representasi dengan graf, struktur data pohon bisa direpresentasikan dengan cara menuliskan ID atau ini dari semua simpul pohon menjadi sebuah string. Ada tiga cara penulisan simpul pohon menjadi string, yaitu dengan preorder, inorder, dan postorder

a) Preorder

Penulisan dengan cara preorder dimulai simpul *parent* kemudian ke simpul anak dengan posisi palin kiri, setelah itu ke anak lain (masih *parent* yang sama) secara terurut dari kiri ke kanan.

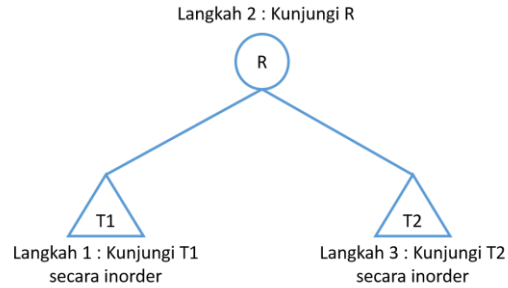


Gambar 3 Skema penelusuran pohon dengan preorder[2]

Contoh untuk pohon pada gambar 1, penulisan secara preordernya adalah A-B-D-E-H-I-C-F-G.

b) Inorder

Penulisan dengan cara preorder dimulai simpul anak terkiri kemudian ke simpul *parent*, setelah itu ke anak lain (masih *parent* yang sama) secara terurut dari kiri ke kanan.

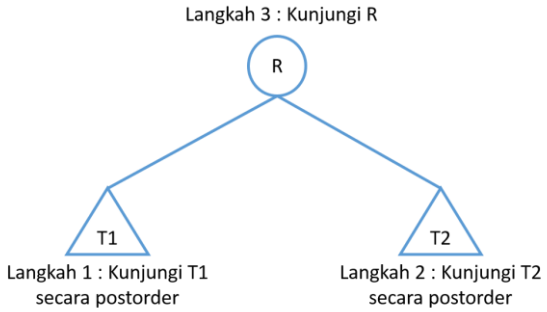


Gambar 4 Skema penelusuran pohon dengan inorder[2]

Contoh untuk pohon pada gambar 1, penulisan secara inordernya adalah D-B-H-E-I-A-F-C-G.

c) Postorder

Penulisan dengan cara postorder dimulai simpul anak terkiri, kemudian ke simpul anak dengan *parent* yang sama di sebelah kanannya, setelah ke simpul *parent*.



Gambar 5 Skema penelusuran pohon dengan postorder[3]

Contoh untuk pohon pada gambar 1, penulisan secara inordernya adalah D-H-I-E-B-F-G-C-A.

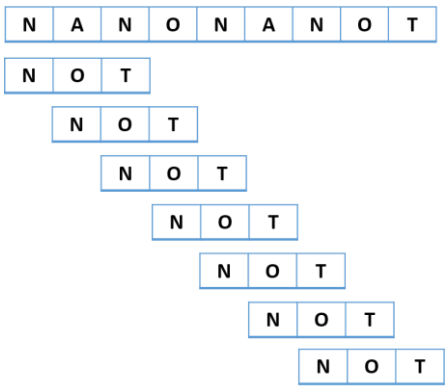
B. *Algortima Pencocokan String*

Algoritma pencocokan string merupakan algoritma sangat banyak kegunaannya, salah satunya adalah untuk pencarian informasi. Persoalan pencarian informasi merupakan salah satu persoalan yang krusial, dimana dituntut kecepatan dan keakuratan. Untuk itu algoritma pencocokan string juga dituntut memiliki kompleksitas sebaik mungkin. Telah banyak algoritma pencocokan string yang ditemukan untuk menjawab tuntutan, tersebut. Algoritma pencocokan string antara lain dengan algoritma brute force, KMP (Knuth-Morris-Pratt), dan Boyer-Moore. Algoritma KMP dan Boyer-moore merupakan algoritma pencocokan string yang banyak digunakan karena memiliki kompleksitas yang jauh lebih baik dibandingkan dengan algoritma brute force.

Pada algoritma pencocokan string ada dua masukan yang akan diperiksa, yaitu *text* berupa string yang ingin diperiksa dan *pattern* berupa string yang ingin dicari di dalam *text*.

1) Algoritma Brute Force

Algoritma brute force merupakan algoritma pencocokan string yang paling sederhana secara ide pencocokkannya dan yang paling mudah diimplementasikan. Namun algoritma brute force ini memiliki kekurangan, yaitu kompleksitas algoritmanya sangat besar. Kompleksitas algoritma brute force untuk kasus terburuk bisa mencapai O(nm) dengan n adalah banyaknya karakter pada *text* dan m adalah banyaknya karakter pada *pattern*[1]. Ide utama dari algoritma brute force adalah membandingkan seluruh *pattern* dengan *text* sampai ditemukan kecocokan. Jika ditemukan ketidak cocokan saat pembandingan, maka posisi *text* akan digeser ke kanan sebanyak satu karakter, kemudian dibandingkan lagi *pattern* dimulai dari karakter pertama.



Gambar 6 Skema pencocokan string dengan algoritma brute force

2) Algoritma KMP

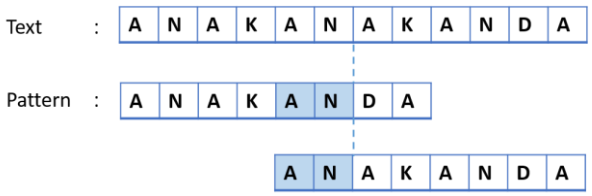
Algoritma KMP ditemukan oleh D.E. Knuth, J.H. Morris, dan V.R. Pratt[1]. Algoritma KMP melakukan pencocokan *pattern* dengan *text* dimulai dari posisi patter paling kanan ke kiri. Ide utama dari algoritma KMP untuk mengefisienkan pencarian string adalah dengan pergeseran untuk menghindari pencocokan yang tidak perlu. Pergeseran ditentukan oleh kecocokan prefiks dari *pattern* dengan suffiks dari *text* dengan jumlah kecocokan paling besar yang akan dihitung oleh sebuah fungsi pingiran.

a) Prefiks dan Suffiks

Prefiks adalah substring yang mungkin dibentuk dari awal ke akhir *pattern* hingga sebanyak i dengan i < jumlah *pattern*. Sementara suffiks adalah substring yang mungkin dibentuk dari akhir ke awal *pattern* hingga sebanyak i dengan i < jumlah *pattern*.

Contoh:

Pattern : POHON
Prefiks : ∅, P, PO, POH, POHO
Suffiks : ∅, N, ON, HON, OHON



Gambar 7 Skema pergeresan algoritma KMP dengan memanfaatkan prefiks dan suffiks

b) Fungsi Pinggiran

Fungsi pinggiran adalah fungsi yang menghitung jumlah kecocokan karakter prefiks dari *pattern* dengan suffiks dari *text* pada suatu posisi dimana ditemukan ketidakcocokan antara *pattern* dengan *text*.

j	1	2	3	4	5	6	7	8
P[j]	A	N	A	K	A	N	D	A
B[j]	0	0	1	0	1	2	0	1

Keterangan :
j : posisi karakter
P[j] : karakter dari pattern pada posisi j
B[j] : banyaknya kesamaan prefiks pattern dengan suffiks text pada posisi j

Gambar 8 Penentuan fungsi pinggiran

Misalnya pada gambar 7, terjadi ketidakcocokan pada posisi k = 7, maka fungsi pinggiran B[j] dengan j = k-1, akan menghasilkan B[6] = 2. Artinya pergeseran dilakukan hingga kecocokan antara prefiks *pattern* dengan suffiks *text* sebanyak 2 karakter. Atau jumlah pergeseran yaitu j – B[j] dalam kasus gambar 6 pergeserannya sebanyak 6-2=4 karakter.

Dengan menggunakan algoritma KMP, jumlah perbandingan karakter pada pencocokan string dapat dikurangi. Karena itulah algoritma KMP ini memiliki kompleksitas yang jauh lebih baik dari algoritma brute force, yaitu O(n+m) dengan n adalah banyaknya karakter pada *text* dan m adalah banyaknya karakter pada *pattern*[1].

3) Algoritma Boyer-Moore

Algoritma Boyer-Moore ditemukan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977[2]. Berbeda dengan algoritma KMP atau algoritma pencocokan string pada umumnya, algoritma Boyer-Moore melakukan pencocokan string dimulai dari kanan ke kiri *pattern*. Ide dari utama dari algoritma ini adalah melakukan jump pada suatu karakter apabila ditemkan ketidakcocokan saat proses perbandingan. Character jump ini dilakukan berdasarkan fungsi last-occurance. Fungsi last-occurrence ini adalah fungsi yang mencatat posisi setiap karakter yang ada di dalam *text* di dalam *pattern*. Fungsi ini mengembalikan posisi karakater di dalam *pattern* jika karakter memang ada di dalam *pattern*, namun jika karakter tidak ada di dalam *pattern*, fungsi akan mengembalikan nilai -1.

Misal T adalah himpunan karakter yang ada pada string *text*, dan T = {A,D,F,K,N,Z} dan karakter x adalah anggota dari himpunan T. Maka untuk *pattern* ANAKANDA, fungsi last–occurrencenya (L(x)) adalah sebagai berikut.

1	2	3	4	5	6	7	8
A	N	A	K	A	N	D	A

x	A	D	F	K	N	Z
L(x)	8	7	-1	4	6	-1

Gambar 9 Fungsi last-occurrence pada algoritma Boyer-Moore

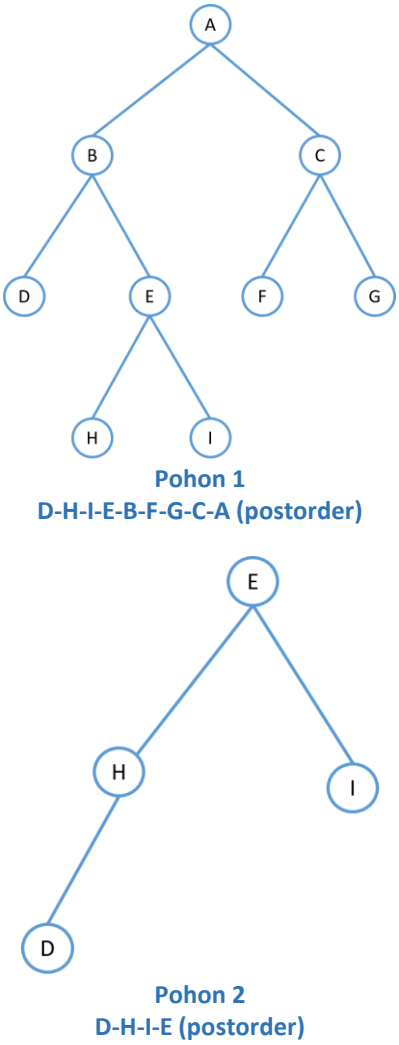
Dengan diketahui fungsi last-occurencenya, apabila terjadi ketidakcocokan saat proses pencocokan string, maka akan dicari berapakah nilai L(x) pada karakter yang tidak cocok di dalam string tersebut. Kembalian nilai L(x) tersebut akan menjadi dasar akan meloncat kemanakah proses selanjutnya.

Gambar 10 Skema pencaria pada algoritma Boyer-Moore

III. ANALISIS

A. *Cara Merepresentasikan Pohon*

Untuk menggunakan metode validasi upapohon dengan algoritma pencocokan string ini perlu dipastikan cara merepresentasikan pohon adalah tepat sehingga algoritma dapat berjalan dan menghasilkan hasil yang inginkan. Seperti yang sudah diketahui ada cara merepresentasikan menjadi string dengan tiga metode, yaitu preorder, inorder dan postorder. Misal metode yang digunakan adalah postorder. Namun itu saja tidak cukup, perlu juga aturan penulisan lain untuk memperjelas pohon yang dibuat agar tidak terjadi kekeliruan dalam menentukan apakah suatu pohon adalah upapohon dari suatu pohon lain atau bukan. Misal jika kita merepresentasikan pohon pada gambar 1 dengan representasi D-H-I-E-B-F-G-C-A hal yang mungkin terjadi adalah kekeliruan validasi karena pada cara representasi tersebut yang diperhatikan hanya sususna simpulnya.



Gambar 11 Pohon 2 bukan upapohon dari pohon 1

Seperti terlihat pada gambar 6, pohon 2 bukan upapohon dari pohon satu sebab memang tidak ada susunan simpul yang memebentuk pohon pada pohon 1 yang sesuai dengan pohon 2, namun apabila menggunakan algoritma pencocokan string dengan representasi pohon seperti tersebut diatas, pohon 2 dapat divalidasi sebagai upapohon dari pohon 1 karena *pattern* string pohon 2 cocok dengan substring dari string pohon 1.

Untuk itu diperlukan cara representasi lain yang dapat megnhindari kekeliruan semacam ini.

Cara merepresentasikan pohon yang lebih tepat adalah dengan memperjelas status *parent-child* dengan menggunakan suatu string tanda. Seperti yang terlihan pada representasi sebelumnya status parrent-*child* dari pohon sama sekali tidak terlihat. Alternatif cara yang dapat digunakan adalah dengan menambahkan tanda kurung yang menyatakan bahwa suatu simpul adalah anak dari simpul lain. Contoh : (BC)A artinya simul B dan simpul C adalah anak dari A, sebaliknya A adalah parret dari simpul B dan simpul C. dengan representasi yang baru, kekeliruan validasi pada gambar 6 tidak akan terjadi. Pohon 1 dengan representasi baru menjadi ((D(HI)E)B(FG)C)A sementara pohon 2 menjadi ((D)HI)E. Dengan representasi yang baru, validasi dengan algoritma pencocokan string akan menghasilkan jawaban yang benar yaitu pohon 2 bukan merupakan upapohon dari pohon 1 karena string *pattern* pohon 2 tidak ditemukan pada pohon 1.

B. Algoritma yang Digunakan dan Proses Validasi

Untuk menginmplementasikan program validasi upapohon ini algoritma yang akan digunakan adalah algoritma KMP dana Boyer-Moore.

Proses validasi dilakukan dengan membandingkan *pattern* suatu pohon (pohon 2) dengan suatu pohon lain (pohon 1) yang ingin diperiksa dengan menggunakan algoritma. Program akan menyatakan bahwa pohon 2 merupakan upapohon dari pohon 1 jika string *pattern* pada pohon 2 ditemukan atau merupakan substring dari pohon 1.

IV. IMPLEMENTASI DAN PENGUJIAN

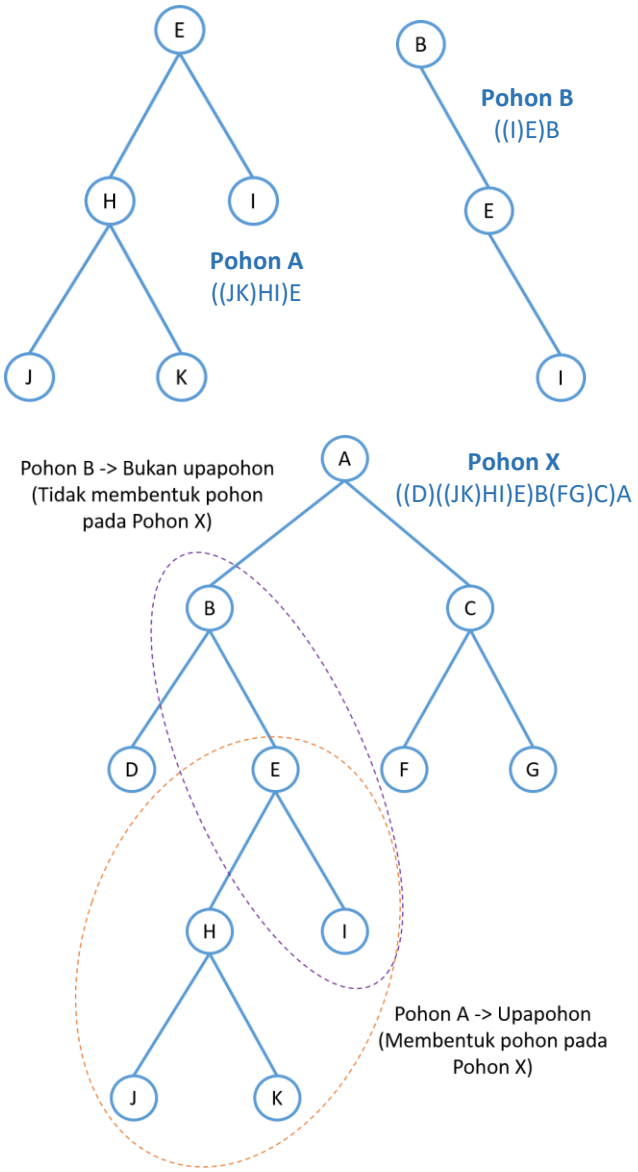
A. Impementasi Program

Implementasi program validasi upapohon ini dilakukan dengan bahasa pemrograman Java. Akan ada dua program yang dibuat, program pertama ValidatorUpapohon.java diimplementasikan menggunakan alogritma KMP. Sementara program ValidatorUpapohon2.java kedua diimplementasikan dengan algoritma Boyer-Moore.

Program yang akan menerima masukan berupa 2 buah pohon yaitu pohon 1 dan pohon 2. Pohon 1 akan diperiksa apakah pohon 2 merupakan upapohon dari pohon 1. Keluaran program adalah status apakah pohon 2 merupakan upapohon dari pohon 1. Jika pohon 2 merupakan upapohon dari pohon 1, maka keluaran program adalah VALID sementara jika pohon 2 bukan upapohon ari pohon 1, keluaran program adalah TIDAK VALID. Selain itu program juga akan menghitung banyaknya operasi perbandingan yang dilakukan untuk mengetahui kompleksitas dari program dan menampilkan hasilnya. Program juga akan mencatat waktu operasi dari program sebagai acuan kecepatan berjalannya program.

B. Pengujian Program

Pengujian program akan dilakukan dengan program baik yang menggunakan algoritma KMP maupun yang menggunakan algoritma Boyer-Moore. Masing-masing program akan diuji dengam masukan yang diharapkan akan menghasilkan keluaran VALID dan TIDAK VALID. Persoalan yang akan diujikan adalah seperti pada gambar berikut.

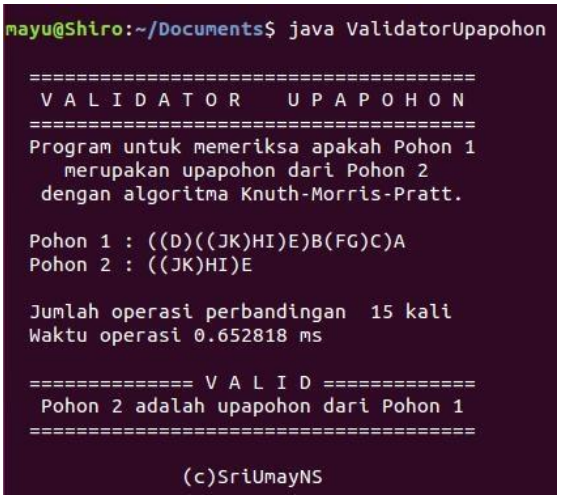


Gambar 12 Skema validasi pohon A dan pohon B pada pohon X

Persoalan pada gambar 12 akan diujikan dengan program yang telah dibuat untuk mnengetahui apakah program yang dibuat dapat bekerja dengan tepat sesuai dengan hasil yang diinginkan.

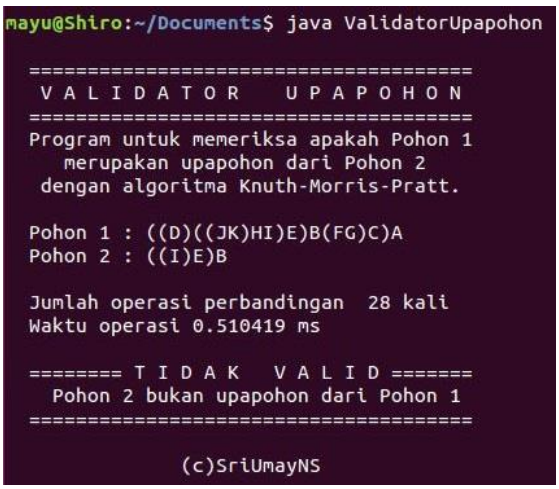
1) Pengujian program dengan Algoritma KMP

Berikut adalah pengujian program validasi upapohon dengan algoritma KMP pada gambar 12. Pohon A dan Pohon B menjadi pohon yang akan diperiksa pada pohon X, pada program menjadi masukan pada Pohon 2. Sememtara masukan pada Pohon 1, yaitu pohon yang diperiksa, adalah Pohon X pada gambar 12.



Gambar 13 Hasil keluaran program valiadasi pohon A pada pohon X dengan algoritma KMP

Hasil keluaran program pada gambar 13 menunjukkan bahwa pohon 2 bukan merupakan upa pohon dari pohon 1. Hasil ini sesuai dengan skema pada gambar 12.



Gambar 14 Hasil keluaran program valiadasi pohon B pada pohon X dengan algoritma KMP

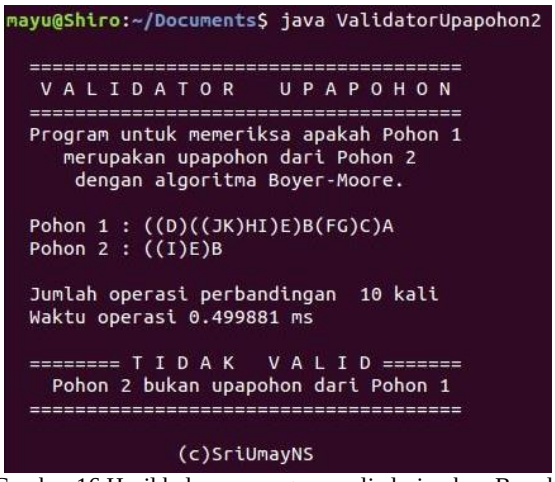
Hasil keluaran program pada gambar 14 menunjukkan bahwa pohon 2 bukan merupakan upapohon dari pohon 1. Hasil ini sesuai pada gambar 12.

2) Pengujian Program dengan Algoritma Boyer-Moore

Seperti halnya pegujian dengan algoritma KMP, berikut adalah pengujian program untuk kasus yang sama yaitu kasus pada gambar 12.

Gambar 15 Hasil keluaran program valiadasi pohon A pada pohon X dengan algoritma Boyer-Moore

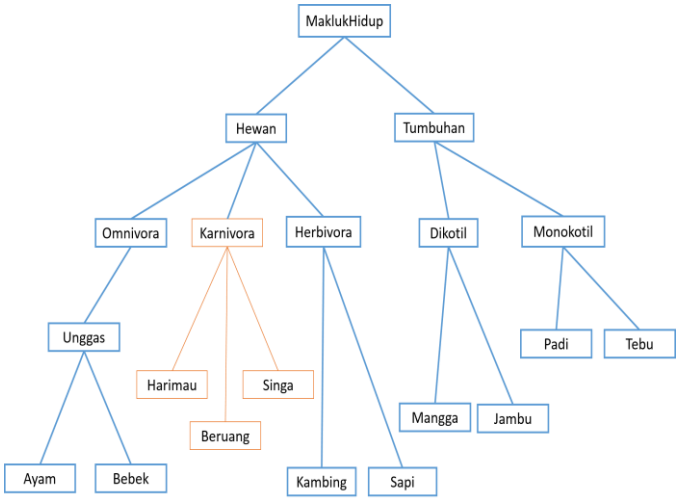
Hasil keluaran program pada gambar 15 menunjukkan bahwa pohon 2 bukan merupakan upa pohon dari pohon 1. Hasil ini sesuai dengan skema pada gambar 12.



Gambar 16 Hasil keluaran program valiadasi pohon B pada pohon X dengan algoritma Boyer-Moore

Hasil keluaran program pada gambar 16 menunjukkan bahwa pohon 2 bukan merupakan upapohon dari pohon 1. Hasil ini sesuai dengan skema pada gambar 12.

Dari hasil keluaran kedua program tersebut (program dengan algoritma KMP dan Boyer-Moore), untuk kasus ini program dengan algoritma Boyer-Moore jumlah operasi yang lebih sedikit dari pada Algoriyma KMP. Namun kedua algoritma tersebut sudah cukup mangkus unutk menyelesaikan persoalan validasi ini.



Gambar 17 Contoh ohon yang menyimpan informasi inheritance kelas pada suatu program simulasi makhluk hidup

Selain seperti contoh pengujian sebelumnya, program ini bisa digunakan untuk mencari berbagai macam pohon dengan informasi simpul tertentu. Misal seperti pada gambar 17, pohon menyimpan informasi mengenai *inheritance* dalam suatu program simulasi makhluk hidup. Node berupa nama kelas-kelas yang ada dalam program tersebut, maka untuk mengetahui suatu unit test atau suatu kelah dan seluruh kelas turunannya sama halnya dengan mencari upapohon dari pohon kelas tersebut. Misal pada program tersebut ingin dicari unit test kelas Karnivora beserta seluruh turunannya yaitu kelas Harimau, Singa, dan Beruang apakah ada atau tidak di dalam program maka, seperti pada gambar 17, untik test tersebut ada di dalam program. Dengan menggunakan algoritma pencocokan string seperti yang telah dijelaskan dalam makalah ini, validasi dapat dimasukkan dengan masukan berupa pohon dan upapohon yang ingin dicari. Dalam hal ini upa pohon unit test Karnivora dapat ditulis sebagai string seperti berikut ((Harimau Beruang Singa)Karnivora). Jadi program validasi ini sangat fleksibel tergantung dengan kebutuhan apa yang ingin divalidasi.

V. KESIMPULAN

Algoritma pencocokan string adalah algoritma yang memiliki banyak kegunaan dan bisa diekplorasi untuk berbagai macam kebutuhan. Salah satunya dapar digunakan untuk melakukan validasi apah suatu pohon merupakan upapohon dari suatu pohon lain. Dengan menggunakan algoritma pencocokan string persoalan validasi tersebut dapat diselesaikan dengan kompleksitas cukup baik dan memori yang dibutuhkan tidak banyak sebab pohon disimpan dalam bentuk string. Program juga sangat fleksibel dan bisa disesuaikan dengan kebutuhan.

KATA PENUTUP

Syukur *alhamdulillah* penulis ucapkan atas nikmat Allah SWT sehingga saya dapat menyelesaikan pembuatan makalah dalam pemenuhan tugas kuliah IF2211 Strategi Algoritma ini. Terima kasih kepada Bapak Rinaldi Munir dan Ibu Nur Ulfa Maulidevi yang telah membekali ilmu dalam kuliah IF2211 Strategi Algoritma yang menjadi pengetahuan untuk menuliskan makalah ini. Selain itu saya juga mengucapkan terima kasi kepada orang tua dan teman-teman yang telah mendukung saya dalam menyelesaikan makalah ini.

Kritik dan saran sangat disilakan sebagai instropeksi bagi penulis kedepannya. Semoga makalah ini dapat bermanfaat.

REFERENCES

[1] Munir, Rinaldi, "Diktat Kuliah IF2211 Strategi Algoritma," Bandung : Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2009.

[2] Munir, Rinaldi, "Diktat Kuliah IF2120 Matematika Diskrit," Bandung : Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2006.

[3] Boyer, Robert S.; Moore, J Strother (October 1977). "A Fast String Searching Algorithm.". Comm. ACM (New York, NY, USA: Association for Computing Machinery) **20** (10):762–772. doi:10.1145/359842.359859. ISSN 0001-0782

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, 6 Mei 2016

Sri Umay Nur'aini Sholihah
(13514007)