Penerapan Graf Untuk Mengatur Dependensi Antar Aplikasi Dalam Sistem Operasi Berbasis Linux

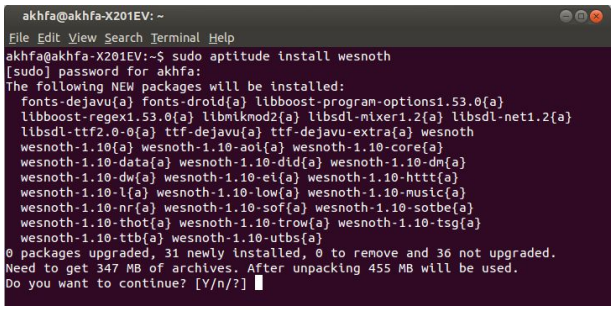
Akhmad Fakhoni Listiyan Dede[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*akhmadfakhoni@gmail.com*

Pemasangan aplikasi di sistem operasi berbasis linux menerapkan sistem ketergantungan antar aplikasi. Hal ini disebut juga sebagai dependensi. Setiap aplikasi yang dipasang biasanya memerlukan aplikasi yang lain agar bisa berjalan dengan baik. Permasalahan dalam masalah ini direpresentasikan menggunakan graf. Yang masih menjadi permasalahan saat ini adalah saat aplikasi sudah terpasang beserta dependensi-nya, saat aplikasi tersebut dihapus, dependensi-nya tidak ikut terhapus. Hal ini dikarenakan belum adanya keterikatan antara aplikasi dan dependensinya. Permasalahan ini akan coba diselesaikan menggunakan graf.

***Kata Kunci*—Dependensi, Linux, graf.**

I. PENDAHULUAN

Sistem operasi berbasis Linux adalah sistem operasi yang terkenal karena kestabilannya. Pengalaman penulis dalam memakai sistem operasi ini sangat luar biasa. Namun ada salah satu saat dimana penulis ingin memasang suatu aplikasi, dan aplikasi tersebut membutuhkan banyak dependensi lainnya agar bisa berjalan dengan baik.



Gambar 1-1 Contoh aplikasi yang membutuhkan banyak dependensi

Pada gambar di atas, terlihat penulis ingin memasang aplikasi bernama wesnoth, dan aplikasi tersebut memerlukan dependensi fonts-dejavu, fonts-droid, dan seterusnya hingga wesnoth-1.10. Permasalahan yang terjadi adalah saat penulis hendak menghapus aplikasi wesnoth tersebut, dependensi yang tadinya ikut terpasang terkadang tidak ikut terhapus. Hal ini akan menjadi masalah jika aplikasi yang telah terpasang sudah sangat banyak, kemudian saat dihapus akan meninggalkan aplikasi "sampah" yang tidak lagi dipakai oleh sistem. Oleh karena itu penulis mencoba menyelesaikan persoalan ini menggunakan representasi graf. Walaupun permasalahan ini bisa ditemui pada semua distribusi sistem operasi berbasis Linux, namun penulis hanya akan membahas sistem operasi Ubuntu.

II. TEORI GRAF

A. *Definisi Graf*

Secara matematis, graf G didefinisikan sebagai himpunan (V,E) yang dalam hal ini V adalah himpunan tidak kosong dari simpul - simpul (*vertices* atau *node*) dan E adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul.

Definisi tersebut menyatakan bahwa V tidak boleh kosong, sedangkan E boleh kosong. Jadi, sebuah graf dimungkinkan tidak mempunyai sisi satu buah pun, tetapi simpulnya harus harus ada, minimal satu. Graf yang hanya mempunyai satu buah simpul tanpa sebuah sisi pun dinamakan graf trivial.

B. *Jenis - jenis Graf*

Graf dapat dikelompokkan menjadi beberapa kategori bergantung pada sudut pandang pengelompokannya. Sudut pandang tersebut dapat berupa ada tidaknya sisi ganda, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi.

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka graf dapat dikelompokkan menadi dua jenis:

1. Graf Sederhana (*simple graph*)

   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana.
2. Graf tak sederhana (*unsimple graf*)

   Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana, yaitu graf ganda *(multigraph)* dan graf semu *(pseudograph)*. Graf ganda adalah graf yang mengandung sisi ganda. Sisi ganda yang menghubungkan dua simpul bisa lebih dari dua buah.

   Graf semu adalah graf yang mengandung gelang. Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya

sendiri.

Berdasarkan jumlah simpul pada suatu graf, secara umum graf dapat digolongkan menjadi 2 jenis:

1. Graf berhingga (*limited graph*)
   Graf berhingga adalah graf yang jumlah simpulnya n, berhingga.
2. Graf tak-berhingga (*unlimited graph*)
   Graf tak-berhingga mempunyai jumlah simpul n, tak berhingga banyaknya.

Sisi pada graf dapat mempunyai orientasi arah. Berdasarkan orientasi arahnya, graf dibedakan menjadi dua jenis:

1. Graf tak-berarah
   Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi jika v adalah simpul (*vertex*), maka $(v_j, v_k) = (v_j, v_k)$.
2. Graf berarah
   Graf yang tiap sisinya diberikan orientasi arah disebut graf berarah. Sisi yang berarah disebut juga dengan busur (*arc*). Pada graf berarah, (v_j, v_k) dan (v_k, v_j) menyatakan dua busur yang berbeda, dengan kata lain $(v_j, v_k) \neq (v_k, v_j)$. Untuk busur (v_j, v_k) simpul vj dinamakan simpul asal (*initial vertex*) dan vk dinamakan simpul terminal (*terminal vertex*). Graf berarah sering dipakai untuk menggambarkan aliran proses, peta lalu lintas suatu kota, dan sebagainya. Pada graf berarah, gelang diperbolehkan, tetapi sisi ganda tidak.

C. Istilah - istilah Dalam Graf

Ada beberapa istilah dasar yang sering dipakai dalam pembahasan kali ini.

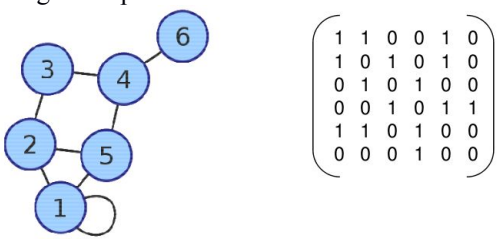
1. Bertetangga (*adjacent*)
   Dua buah simpul pada graf tak-berarah G dikatakan bertetangga jika keduanya terhubung langsung dengan sebuah sisi. Dengan kata lain, v_j bertetangga dengan v_k jika (v_j, v_k) adalah sebuah sisi pada graf G.
2. Bersisian (*incident*)
   Untuk sembarang sisi $e = (v_j, v_k)$, sisi e dikatakan berisian dengan simpul v_j dan v_k.
3. Simpul terpencil (*Isolated Vertex*)
   Simpul yang tidak mempunyai sisi yang bersisian dengannya atau dapat juga dinyatakan bahwa simpul terpencil adalah simpul yang tidak satupun bertetangga dengan simpul - simpul lainnya.
4. Graf kosong (*Null Graph* atau *Empty Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong.
5. Derajat (*Degree*)
   Derajat suatu simpul pada graf tak berarah adalah jumlah sisi yang bersisian dengan simpul tersebut.
6. Lintasan (*Path*)
   Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan vn di dalam graf G ialah barisan berselang - seling antara simpul dan sisi yang berbentuk v_1, e_1, v_1, e_2,..., v_{n-1}, e_n, v_n sedemikian sehingga $e_1 = (v_o, v_1)$, $e_2 = (v_1, v_2)$, $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G.
7. Siklus (*Cycle*) atau sirkuit(*Circuit*)
   Lintasan yang berawal dan berakhir pada simpul yang sama.
8. Terhubung (*Connected*)

Keterhubungan dua buah simpul sangat penting di dalam graf. Dua simpul v_j dan v_k dikatakan terhubung jika terhubung jika terdapat lintasan dari v_j ke v_k. Graf dikatakan sebagai graf terhubung jika setiap pasangan simpul di dalam graf terhubung.

D. Representasi Graf di dalam Abstract Data Type

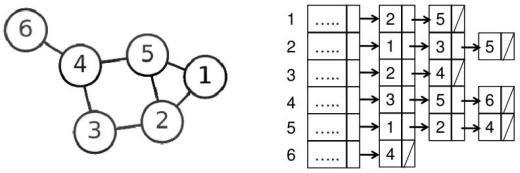
Ada beberapa cara merepresentasikan graf agar komputer bisa mengolahnya. Cara tersebut antara lain:

1. Adjacency Matriks
   Jika G terdiri dari n buah simpul, maka adjacency matriks adalah matriks n x n dengan $M_{j,k}$ menyatakan jumlah busur antara simpul j dengan simpul k.



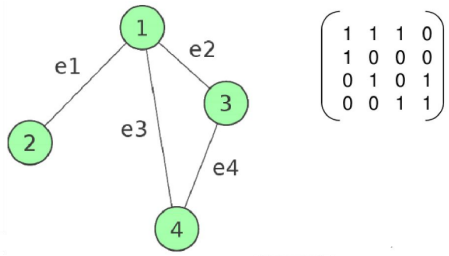
Gambar 2-1 Representasi graf menggunakan matrik ketetanggaan

2. Adjacency List
   Simpul disimpan sebagai objek / *record* yang selain memuat informasi tentang simpul tersebut juga mengandung list dari seluruh simpul yang terhubung dengan simpul tersebut.



Gambar 2-2 Representasi graf menggunakan list ketetanggaan

3. Incidence Matriks
   Matriks yang memperlihatkan hubungan antara dua kelompok objek pembangun graf, yaitu simpul sebagai elemen baris dan busur sebagai elemen kolom. Mj,k akan bernilai *true* jika ada hubungan antara simpul j dengan simpul k.



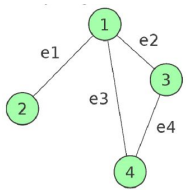
Gambar 2-3 Representasi list menggunakan matrik bersisian

4. Incidence List

   Simpul dan busur disimpan sebagai objek / *record* yang memuat informasi tentang simpul / busur tersebut. Setiap simpul menyimpan list dari busur yang terhubung dengannya. Setiap busur menyimpan list dari simpul yang terhubung dengannya.

5. List Sisi

   List sisi adalah suatu tabel pasangan simpul yang membentuk sisi.



Sisi	Simpul-1	Simpul-2
e1	1	2
e2	1	3
e3	1	4
e4	3	4

Gambar 2-4 Representasi list menggunakan list sisi

III. REPOSITORY PERANGKAT LUNAK

A. Sekilas Tentang Ubuntu

Ubuntu adalah distribusi Linux, berbasis Debian, dan memili fitur - fitur yang khas. Sistem operasi ini berjalan di atas kernel. Kernel ini menghubungkan sistem operasi dengan perangkat keras komputer. Pada awalnya, Linux dikenal sebagai sistem operasi *text-based*, artinya dijalankan menggunakan perintah-perintah yang diketik di komputer. Dalam perkembangannya, para pengembang mengembangkan sistem operasi ini, sehingga bisa dijalankan secara mudah oleh orang awam sekalipun, dengan cara membuat *user interface* yang berbasis GUI (*Graphical User Interface*). Sekarang sudah banyak variasi dari Linux yang sangat mudah untuk dijalankan.

Beberapa alasan untuk menggunakan Ubuntu:

1. Menginginkan komputer yang dapat dinyalakan (*boot*) dengan cepat.
2. Menginginkan sistem operasi yang bagus dan modern namun tidak ingin membeli Mac OS.
3. Memiliki pikiran bahwa perangkat lunak seharusnya bebas, bebas untuk dipakai dan dimodifikasi.
4. Tidak ingin mengeluarkan banyak uang hanya untuk membeli perangkat lunak.
5. Memiliki komputer yang sudah berumur, namun masih ingin memakainya.

B. Definisi Repositori

Dalam manajemen aplikasinya, Ubuntu memakai metode menyimpan semua aplikasi di dalam sebuah "gudang", dan baru memasang aplikasi yang hanya dibutuhkan oleh pengguna. Gudang inilah yang disebut sebagai repositori. Berkat sistem seperti ini, sebagian besar distribusi Linux tidak memerlukan space yang besar saat diunduh oleh pengguna. Kebanyakan berukuran antara 700 MB - 1 GB. Berbeda dengan Windows yang memerlukan space hingga 4 GB lebih untuk didistribusikan.

C. Kategori Repositori

- *Main Distribution*

  Repositori jenis ini mengandung paket yang dibutuhkan untuk memasang Ubuntu.
- *Security Updates*

  Paket-paket yang berfungsi untuk menambal lubang keamanan pada Ubuntu tersedia di repositori ini.
- *Recommended Updates*

  Repository ini terdiri atas versi - versi yang lebih baru daripada versi di repositori Main Distribution. Contohnya seperti Security Updates. Pembaharuan untuk keamanan ini memperbaiki bug, namun sifatnya tidak mendesak. Jadi di repositori ini hanya terdiri atas pembaharuan pembaharuan yang sifatnya tidak mendesak.
- *Proposed Updates*

  Terdiri atas paket - paket yang baru, dan masih dalam bentuk *testing*. Jadi masih memungkinkan banyak *bug*. Oleh karena itu repositori ini tidak direkomendasikan untuk diaktifkan.
- Unsupported (Backports) Updates

  Terdiri atas paket perangkat lunak yang sebenarnya ditujukan untuk Ubuntu versi berikutnya, namun dipaketkan untuk bisa dipasang di Ubuntu versi yang sekarang. Perangkat lunak yang masuk di dalam kategori ini belum di uji secara mendalam, sehingga besar kemungkinan masih memiliki banyak bug. Meskipun begitu, perangkat lunak tersebut juga memiliki berbagai macam fitur baru yang pantas untuk dicoba.

D. *Komponen Repositori*

Komponen komponen dalam repositori adalah sebagai berikut:

- Main

  Terdapat hampir semua perangkat lunak yang merupakan fitur dasar dari Pemasangan Ubuntu.
- Universe

  Terdapat sebagian besar perangkat lunak bebas yang tersedia untuk saat ini. Sebagian besar berasal dari repositori Debian, meskipun ada beberapa yang dioptimasi agar bisa berjalan dengan baik di Ubuntu.
- Restricted

  Beberapa driver dari perangkat keras komputer hanya didistribusikan dalam bentuk binary, artinya

tidak mencantumkan *source code*. Driver ini juga memiliki lisensi yang tidak tepat dengan lisensi *free software*. Perangkat lunak yang memiliki karakteristik tersebut masuk di dalam komponen Restricted.

- Multiverse

  Hampir sama seperti perangkat lunak yang ada di dalam komponen restricted, hanya saja di multiverse ini, tidak ada satupun perangkat lunaknya yang penting dalam pemasangan Ubuntu.

- Source code

  Pusat dari segala kode sumber aplikasi. Sangat penting bagi pengembang perangkat lunak.

- Partner

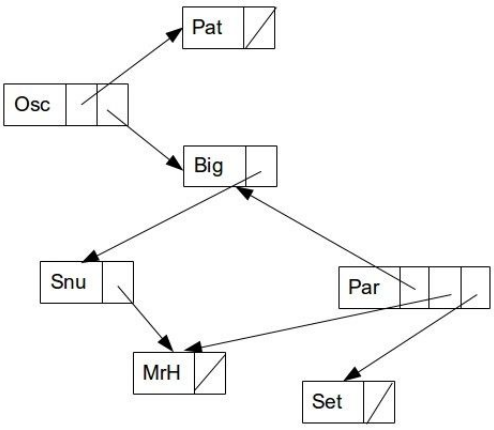
  Canonical sebagai perusahaan yang berada di belakang Ubuntu tentu saja memiliki partner. Perangkat lunak yang berasal dari partner Ubuntu ini tersedia di komponen Partner.

IV. REPRESENTASI DALAM GRAF

Berdasarkan 5 macam representasi graf yang telah dibahas sebelumnya, penulis menganggap bahwa representasi yang paling tepat adalah adjacency list dengan sedikit modifikasi.

Agar repositori bisa sepenuhnya diaplikasikan dalam bentuk graf, maka setiap aplikasi atau paket di dalam repositori diwakili oleh sebuah simpul. Setiap aplikasi yang telah berupa simpul tersebut akan menunjuk aplikasi lain yang merupakan dependensinya. Paket yang tidak memiliki dependensi tidak akan menunjuk paket manapun.

Misal ada repositori lengkap yang terdiri atas aplikasi bernama Oscar (Osc), Patrik (Pat), BigBird (Big), Snuffleupagus (Snu), Parch (Par), MrHooper (MrH), dan Setaru (Set). Oscar membutuhkan dependensi Patrik dan BigBird. Patrik tidak membutuhkan apapun. BigBird membutuhkan Snuffleupagus. Snuffleupagus membutuhkan MrHooper. MrHooper tidak membutuhkan apapun. Parch membutuhkan BigBird, MrHooper, dan Setaru, maka list yang terbentuk sebagai berikut.

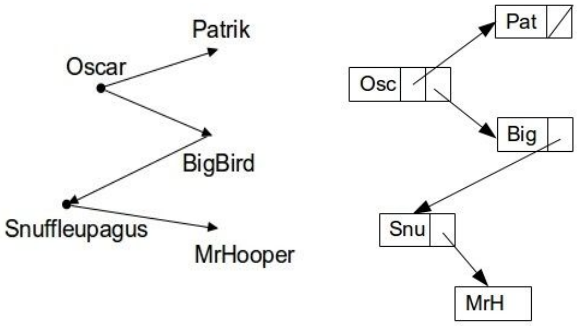


Gambar 4-1 Representasi repository dalam graf menggunakan list ketetanggaan yang dimodifikasi

Penulis akan memvisualisasikan proses pemasangan aplikasi, penghapusan aplikasi dan dependensi, penghapusan jika dependensi aplikasi juga dibutuhkan oleh aplikasi yang lain,

1. Pemasangan aplikasi yang dependensinya belum ada di sistem

   Misalkan penulis ingin memasang aplikasi yang bernama Oscar. Tetapi Oscar ini memiliki dependensi program yang bernama BigBird dan Patrik yang belum ada di sistem. Namun sayangnya, BigBird ini juga membutuhkan dependensi lain yang bernama Snuffleupagus. Dan Snuffleupagus ini juga mempunyai dependensi sendiri yang bernama MrHooper.

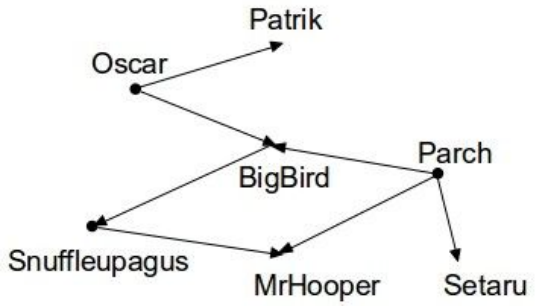


Gambar 4-2 Graf (kiri) dan List (kanan) yang terbentuk saat memasang aplikasi Oscar

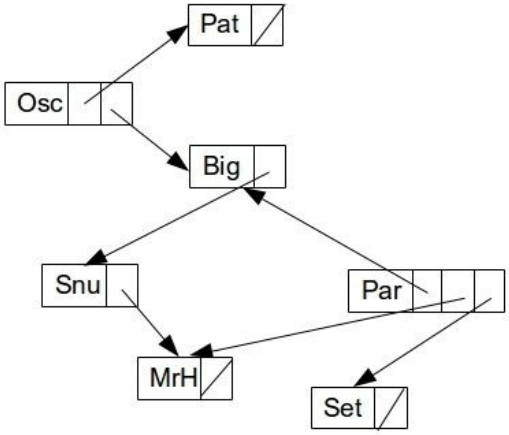
Gambar di atas merepresentasikan pemasangan aplikasi Oscar. Saat Oscar dipasang, maka Patrik, BigBird, Snuffleupagus dan MrHooper juga ikut terpasang. Terlihat pula bahwa Oscar menunjuk Patrik dan BigBird, artinya Oscar memerlukan Patrik dan BigBird agar terpasang dengan baik. Demikian pula BigBird membutuhkan Snuffleupagus dan Snuffleupagus membutuhkan MrHooper.

2. Memasang aplikasi yang sebagian dependensinya sudah ada di sistem

   Misal penulis ingin memasang aplikasi yang bernama Parch. Parch ini membutuhkan dependensi BigBird dan MrHooper yang sudah ada di dalam sistem. Selain itu, dia juga membutuhkan Setaru yang belum ada di sistem. Graf yang terbentuk sebagai berikut



Gambar 4-3 Graf yang terbentuk saat menambahkan aplikasi Parch

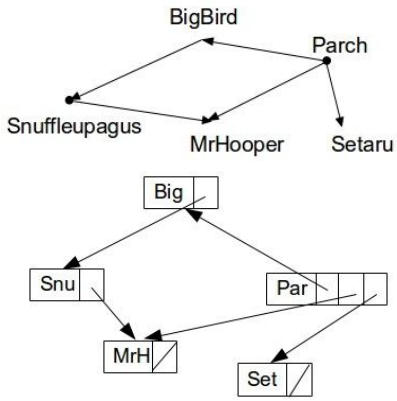


Gambar 4-3 List yang terbentuk saat menambahkan aplikasi Parch

Seperti terlihat pada gambar, saat sistem hanya perlu memasang Parch dan Setaru, maka dalam graf juga hanya ditambah dengan vertex Parch dan Setaru, kemudian ada busur dari Parch yang mengarah ke vertex dependensinya, yaitu BigBird, MrHooper, dan tentu saja Setaru. Untuk List hanya perlu menambahkan elemen Parch dan Setaru, dengan elemen Parch menunjuk ke BigBird, MrHooper, dan Setaru.

3. Menghapus Aplikasi yang dependensinya juga dibutuhkan oleh aplikasi lain.

   Misal penulis akan menghapus aplikasi Oscar, dimana dependensi Oscar adalah Patrik dan BigBird. Penampakan dari graf yang terbentuk adalah sebagai berikut.



Gambar 4-4 Graf (atas) dan List (bawah) yang terbentuk saat aplikasi Oscar dihapus dari sistem

Walaupun yang dihapus hanyalah Oscar, namun Patrik pun ikut terhapus. BigBird yang merupakan dependensi dari Parch tidak ikut terhapus. Begitu juga dengan Snuffleupagus tidak terhapus karena dia adalah dependensi dari BigBird. Pada list, elemen Oscar dan Patrik dihapus karena tidak ada lagi yang menunjuk mereka.

4. Menghapus aplikasi yang dependensinya tidak dibutuhkan oleh aplikasi lain.

   Misalkan penulis ingin menghapus aplikasi Parch, maka BigBird, MrHooper, dan Setaru akan ikut terhapus karena mereka tidak lagi dibutuhkan oleh aplikasi yang lain. Begitu juga dengan Snuffleupagus akan ikut terhapus karena BigBird juga terhapus. Sehingga keseluruhan graf akan terhapus.

Meskipun pada contoh di atas keseluruhan graf terhapus, namun pada kenyataannya tidaklah mungkin semua aplikasi dapat dihapus, karena pasti ada aplikasi yang dibutuhkan oleh sistem agar bisa berjalan dengan baik.

V. HASIL DAN ANALISIS

Dari beberapa contoh yang telah dibahas sebelumnya, sangatlah mungkin untuk merepresentasikan aplikasi atau paket yang ada di dalam repositori menjadi dalam bentuk graf. Hal yang perlu diperhatikan adalah bahwa database graf yang berupa simpul dari semua aplikasi bisa disimpan di dalam server repositori. Sistem operasi hanya perlu membuka database saat pengguna ingin memasang suatu aplikasi. Dengan begitu, dependensi antar aplikasi akan tersimpan datanya secara jelas, dan tidak ada alasan lagi ada aplikasi yang tertinggal saat ingin menghapus aplikasi tertentu. Satu hal yang belum terpikirkan oleh penulis adalah bagaimana representasi fisik dari graf tersebut yang disimpan baik di server repositori maupun di komputer pengguna agar dalam prakteknya tercipta sebuah sistem pembacaan file konfigurasi dependensi yang efisien sehingga permasalahan dependensi ini bisa terselesaikan secara tuntas.

VI. KESIMPULAN

Dalam makalah ini telah dijelaskan bagaimana merepresentasikan repositori Linux ke dalam graf sehingga pada akhirnya masalah dependensi dapat diselesaikan. Hasil dari makalah ini bisa diimplementasikan ke dalam sistem repositori Linux

Penulis berharap ada seseorang yang bisa memikirkan bagaimana representasi fisik dari konsep yang telah diajukan oleh penulis, dengan begitu konsep ini bisa benar - benar diaplikasikan.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. *Matematika Diskrit* (Revisi Keempat). Bandung: Informatika Bandung, 2010, bab. 8

[2] Raggi, Emilio dkk. *Beginning Ubuntu Linux Fifth Edition.* New York: Apress, 2010.

[3] http://kuliah.itb.ac.id/mod/resource/view.php?id=9258. Tanggal akses 15 Desember 2013, pukul 14.30

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, 16 Desember 2013

Akhmad Fakhoni L D 13513601