Aplikasi Graf pada Kriteria Cakupan Graf Terstruktur untuk Pengujian Perangkat Lunak

Muhammad Akmal 13517028
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
akmalmuhammad51@gmail.com

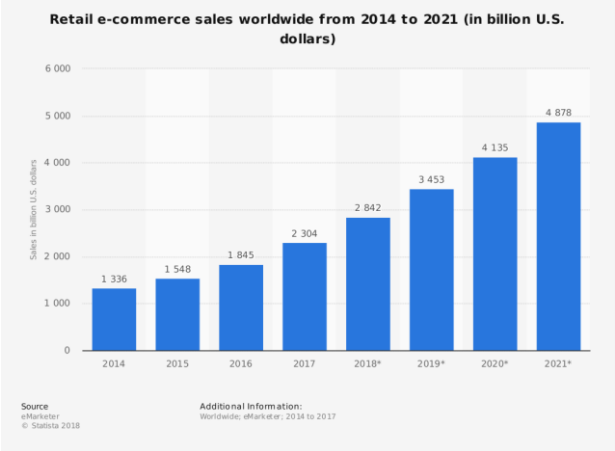
***Abstract*—Di era yang serba digital ini, perangkat lunak telah diimplementasikan secara luas dan besar-besaran yang membuat perangkat lunak hampir memiliki peran penting dalam setiap sektor kehidupan. Perangkat lunak juga telah diproduksi baik secara skala besar untuk perusahaan besar sampai skala kecil untuk kepentingan pribadi sekalipun. Kondisi ini mengimplikasikan semakin meningkatnya kebutuhan akan perangkat lunak yang berkualitas bagi dunia. Salah satu aspek penting yang perlu diperhatikan dalam pengembangan perangkat lunak adalah pengujian perangkat lunak. Makalah ini menyajikan aplikasi teori graf pada pengujian perangkat lunak yang yang memiliki peran penting dalam menjaga kualitas dan performa dari perangkat lunak khususnya pada konsep Kriteria Cakupan Graf Terstruktur atau disebut juga Kriteria Cakupan Aliran Kontrol.**

***Keywords*—Cakupan Graf, Graf, Kriteria Cakupan Graf Terstruktur, Pengujian Perangkat Lunak.**

I. PENDAHULUAN

Kata "perangkat lunak" seharusnya sudah tidak asing di telinga kita. Kehidupan kita sudah tidak bisa lepas dari satu kata tersebut. Semenjak bangun pagi sampai tidur lagi kita selalu berhubungan dengan perangkat lunak. Mulai dari gawai yang selalu kita bawa kemanapun kita pergi, televisi yang memberikan hiburan untuk kita dengan berbagai stasiun televisinya hingga benda-benda yang jarang kita sadari keberadaan perangkat lunak di dalamnya seperti rambu lalu lintas, mesin jaja, hingga kendaraan bermotor yang kita miliki. Semua hal tersebut memiliki perangkat lunak di dalamnya yang berfungsi untuk menjalankan fungsi dari benda-benda dan komponen-komponen yang menyusunnya tersebut secara baik dan benar.

Dengan berkembangnya ilmu pengetahuan dan teknologi, produksi hal-hal yang berbau elektronik semakin massal dilakukan dan semakin menarik untuk diperhatikan dikarenakan permintaan pasar yang tinggi pula. Dapat dilihat pada data statistik tentang pembelian barang online secara global (*e-commerce*) dari tahun 2014 sampai prediksi tahun 2021 yang masih didominasi oleh pembelian barang elektronik seperti komputer serta gawai pintar yang semakin meningkat tiap tahunnya pada Gambar 1. Hal ini menyebabkan dibutuhkannya perkembangan perangkat lunak yang dilakukan secara terus-menerus baik dari sistem secara skala besar hingga dalam skala kecil serta baik itu dalam perilaku perangkat lunak secara independen hingga interaksi perangkat lunak dengan perangkat keras yang ada agar bisa mendukung kelayakan pakai serta kualitas alat-alat elektronik yang akan dijual.



Gambar 1. Statistik Penjualan *e-commerce*
Sumber:
https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/
(Akses : 9 Desember 2018)

Tujuan diproduksinya alat-alat elektronik tidak lain adalah untuk memberikan kemudahan pada kehidupan pengguna atau manusia pada umumnya. Dengan berbagai masalah manusia yang muncul ke permukaan, muncul ide untuk bisa menyelesaikannya dengan perangkat lunak yang bisa jadi diintegrasikan dengan sistem perangkat keras atau digunakan secara independen seperti aplikasi pada gawai. Yang menjadi tantangan pada pengembangan perangkat lunak ini adalah pengguna yang dalam hal ini target pasar akan selalu mengasumsikan yang mereka pakai adalah selalu benar. Hal ini membuat tantangan tersendiri bagi para pengembang untuk dapat memastikan bahwa tidak akan terjadi kesalahan pada perangkat lunak agar asumsi tersebut dapat berjalan sebagaimana mestinya. Salah satu hal yang sangat penting untuk diperhatikan adalah pengujian perangkat lunak. Pengujian ini bertujuan untuk memenuhi persyaratan serta target pengguna yang telah ditentukan sebelumnya.

Salah satu model abstrak yang dapat digunakan untuk pengujian perangkat lunak adalah menggunakan graf. Boris Beizer di dalam bukunya yang berjudul *Software Testing Techniques* menyatakan bahwa pengujian itu sederhana, yang harus dilakukan penguji adalah temukan grafnya dan cakuplah

semuanya[2]. Graf membantu dalam memberikan visualisasi terhadap apa yang akan diuji serta memberikan kemudahan bagi para penguji untuk bisa mendefinisikan kriteria serta desain tes untuk tercapainya tujuan pengujian perangkat lunak yaitu agar perangkat lunak dapat berjalan sebagaimana mestinya.

II. TEORI DASAR GRAF

A. *Definisi Graf*

Teori graf merupakan pokok bahasan yang sudah tua usianya namun memiliki banyak terapan sampai saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graf adalah dengan menyatakan objek sebagai noktah, bulatan atatu titik, sedangkan hubungan antara objek dinyatakan dengan garis. Secara matematis, graf didefinisikan sebagai berikut :

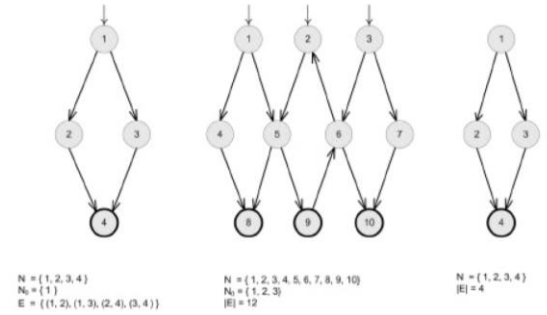
- Himpunan N yang berisi simpul-simpul
- Himpunan N_0 yang berisi simpul awal, di mana $N_0 \subseteq N$ serta simpul awal n_0 ditandai dengan adanya sisi yang muncul ke arah n_0 yang tidak memiliki predesesor.
- Himpunan N_f yang berisi simpul-simpul akhir di mana $N_f \subseteq N$ serta simpul akhir n_f ditandai dengan cetak tebal pada simpulnya.
- Himpunan E yang berisi sisi, di mana E adalah himpunan bagian dari $N \times N$

Simpul pada graf dapat dinomori dengan huruf, seperti *a,b,c*, ..., dengan bilangan asli 1,2,3, ..., atau gabungan keduanya. Sedangkan sisi yang menghubungkan simpul *u* dengan simpul *v* dinyatakan dengan pasangan (*u,v*) atau dinyatakan dengan lambang e_1,e_2,e_3, ... Dengan kata lain, jika e adalah sisi yang menghubungkan simpul *u* dengan simpul *v*, maka *e* dapat ditulis sebagai

$$e = (v_i, v_j)$$

Secara geometri, graf digambarkan sebagai sekumpulan noktah (simpul) di dalam bidang dwimatra yang dihubungkan dengan sekumpulan garis (sisi). Simpul v_i disebut sebagai predesessor sedangkan simpul v_j disebut suksesor.

Graf yang bisa digunakan untuk membuat tes adalah yang memiliki N,N_0, dan N_f paling sedikit satu simpul serta perhatikan bahwa N_0 dapat memiliki lebih dari 1 simpul.



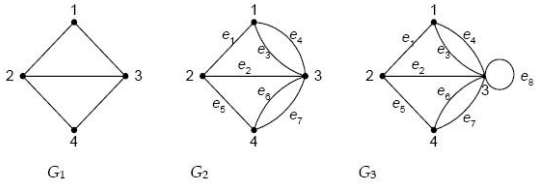
Gambar 2. Graf yang memiliki satu simpul awal (kiri), graf yang memiliki banyak simpul awal (tengah), graf yang tidak memiliki simpul awal (kanan).[1]

B. *Jenis-jenis Graf*

Graf dapat dikelompokkan menjadi beberapa kategori (jenis) bergantung pada sudut pandang pengelompokannya. Pengelompokan graf dapat dipandang berdasarkan ada tidaknya sisi ganda atau sisi kalang atau berdasarkan orientasi arah pada sisi.

Berdasarkan ada tidaknya gelang atau sisi ganda, graf dapat dikelompokkan menjadi 2 jenis :

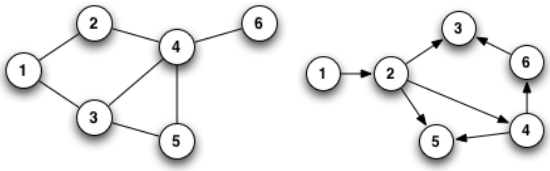
1. Graf sederhana : Graf yang tidak mengandung gelang atau sisi ganda.
2. Graf tak-sederhana : Graf yang mengandung sisi ganda atau gelang. Ada 2 macam graf tak-sederhana yaitu graf ganda dan graf semu yang masing-masing artinya graf yang mengandung sisi ganda dan graf yang mengandung gelang (termasuk bila memiliki sisi ganda sekalipun).



Gambar 3. Tiga buah graf, G1 adalah graf sederhana, G2 adalah graf ganda, G3 adalah graf semu.[4]

Berdasarkan orientasi arah, graf dapat dikelompokkan menjadi 2 jenis :

1. Graf tak-berarah : Graf yang sisinya tidak mempunyai orientasi arah.
2. Graf berarah : Graf yang setiap sisinya diberikan orientasi arah. Pada graf ini (*u,v*) tidak sama dengan (*v,u*). Sisi berarah disebut sebagai busur (*arc*).



Gambar 4. Graf tak-berarah (kiri) dan graf berarah (kanan).
Sumber: http://think-like-a-git.net/sections/graph-theory/directed-versus-undirected-graphs.html (akses : 9 Desember 2018)

C. *Terminologi Dasar*

Pada terminologi dasar ini, perlu digaribawahi bahwa hanya terminologi yang akan dipakai untuk pembahasan yang akan dicantumkan.

1. Lintasan
   Lintasan yang panjangnya *n* dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, ..., e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1), e_2 = (v_1, v_2), ..., e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G.
2. Siklus atau Sirkuit
   Lintasan yang berawal dan berakhir pada simpul yang sama.
3. Terhubung
   Dua buah simpul *u* dan *v* disebut terhubung jika terdapat lintasan dari *u* ke *v*.
4. Upagraf

Misalkan $G = (V, E)$ adalah sebuah graf. $G_1 = (V_1, E_1)$ adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$.

5. Upalintasan : Lintasan l_i disebut upalintasan l_j jika dan hanya jika setiap sisi serta simpul pada l_i juga termasuk sisi dan simpul pada l_j dengan urutan yang sama tanpa adanya tambahan simpul atau sisi di antaranya.

III. Teori Dasar Pengujian Perangkat Lunak

A. *Abstract Data Type* (*ADT*)

ADT adalah definisi *type* dan sekumpulan primitif (operasi dasar) terhadap *type* tersebut. Selain itu, dalam sebuah ADT yang lengkap, disertakan pula definisi *invariant* (persyaratan tertentu yang selalu berlaku) dan aksioma dari *type*.

Definisi type dari sebuah ADT dapat mengandung sebuah definisi ADT lain, misalnya ADT Waktu yang terdiri dari ADT Jam dan ADT Tanggal, ADT Garis yang terdiri dari dua buah ADT *Point*. *Type* sendiri diterjemahkan menjadi *type* terdefinisi dalam bahasa bersangkutan, misalnya menjadi *record* dalam bahasa Ada/Pascal, atau *struct* dalam bahasa C.

B. *Terminologi Dasar*

Pada makalah ini, dihindari ekspresi sejenis awakutu / *bug* dan didefinisikan 3 terminologi yang merupakan bagian dari *bug* :

Kesalahan Perangkat Lunak : kerusakan statik pada perangkat lunak.

Eror Perangkat Lunak : Keadaan internal yang tidak benar yang merupakan manifestasi dari kesalahan.

Kegagalan Perangkat Lunak : Perilaku yang tidak benar yang berhubungan dengan syarat atau deskripsi lain tentang perilaku yang diharapkan.

Sebagai contoh, perhatikan keadaan dokter mendiagnosa pasiennya. Pasien datang ke ruang periksa dengan berbagai *kegagalan* (yaitu penyakit). Dokter lalu harus menemukan *kesalahan*, atau akar masalah dari penyakit tersebut. Untuk menemukannya, dokter akan mengecek apakah ada kondisi internal pasien yang berbeda dari yang seharusnya, seperti darah tinggi, kadar gula darah tinggi, atau kadar kolesterol tinggi. Hal ini berkorespondensi dengan *eror*. Namun, yang harus digarisbawahi kesalahan pada perangkat lunak adalah kesalahan desain. Hal tersebut tidak muncul secara spontan, tetapi ada sebagai hasil dari keputusan yang diambil manusia.

Didefinisikan pula istilah untuk tujuan dari pengujian perangkat lunak yang pada pembahasan ini menitikberatkan pada verifikasi dibanding validasi dikarenakan verifikasi lebih kepada aktivitas teknis yang mengarah pada pengetahuan perangkat lunak.

Verifikasi : Proses untuk menentukan apakah sebuah produk dari fase dari proses pengembangan perangkat lunak memenuhi syarat yang telah ditetapkan pada fase sebelumnya.

Validasi : Proses mengevaluasi perangkat lunak pada tahap akhir pengembangan perangkat lunak untuk memastikan pemenuhan kebutuhan pengguna.

Pengujian yang dibahas bukanlah termasuk pengawakutuan (*debugging*), berikut definisinya :

Pengujian (*Testing*) : Mengevaluasi perangkat lunak dengan mengamati eksekusinya.

Kegagalan pengujian (*Test Failure*) : Eksekusi dari pengujian yang menghasilkan kegagalan perangkat lunak.

Pengawakutuan (*Debugging*) : Proses untuk mencari kesalahan yang ditunjukkan oleh kegagalan.

C. *Desain Tes Model-Driven (DTMD)*

Sudah barang tentu pengujian perangkat lunak sangatlah rumit, dan tujuan utama kita untuk menjadikan perangkat lunak yang sepenuhnya benar tidaklah dapat tercapai oleh kita. Dibandingkan dengan mencari "kebenaran", perekayasa perangkat lunak yang bijak akan mencoba untuk mengevaluasi "perilaku" perangkat lunak itu untuk menentukan apakah perangkat lunak tersebut dapat diterima dengan berbagai pertimbangan mulai dari keamanan hingga efisiensi dengan menggunakan metode abstraksi.

Metode abstraksi yang cukup populer saat ini adalah Desain Tes Model-Driven (DTMD). Desain Tes Model-Driven memecah pengujian menjadi tugas-tugas yang lebih kecil untuk menyederhanakan pembuatan pengujian. Kunci utama dari DTMD adalah desain *test case* yang bisa menjadi kunci utama penentu apakah tes sudah sukses menemukan semua kegagalan dalam perangkat lunak. *Test Case* dapat didesain dengan pendekatan "*human-based"* yaitu dengan menggunakan pengetahuan pada tujuan perangkat lunak diciptakan dan pengetahuan manusia terhadap pengujian, juga bisa dengan pendekatan "*criteria-based"* yang menitik beratkan pada pemenuhan tujuan rekayasa salah satunya adalah kriteria cakupan. Sebagai contoh, jika software tertanam pada pesawat terbang, seorang desainer "*human-based*" harus mengerti penerbangan sedangkan seorang desainer "*criteria-based*" harus mengerti bagaimana program bekerja baik secara independen maupun secara umum dalam arti interaksi program dengan hal-hal dan komponen-komponen lain yang ada di dalam pesawat.

Kriteria cakupan dapat memberi kita struktur yang jelas dalam menemukan ruang masukan yang cocok hingga menentukan waktu yang tepat di mana kita tahu kita sudah cukup melakukan pengujian yang mana merupakan salah satu manfaat penting dari kriteria cakupan. Kriteria cakupan juga memiliki jaminan pada 2 tujuan yang penting bagi penguji : (1) penguji telah mengamati cukup banyak tempat di sudut ruang masukan yang ada, dan (2) pengujian memiliki tingkat tumpang tindih yang rendah. Secara umum, kriteria cakupan memiliki peran yang penting dalam pengembangan kualitas serta biaya untuk pembuatan data tes. Faktanya, semua kriteria cakupan dapat dikerucutkan menjadi beberapa kriteria dengan hanya 4 struktur matematika : domain masukan, graf, ekspresi logika dan deskripsi sintaks. Pada makalah ini, akan dibahas kriteria cakupan dengan menggunakan graf saja.

D. Cakupan Graf

Graf akan digunakan untuk mendefinisikan kriteria serta tes desain. Kita sebut sebuah simpul *n* (atau sisi *e*) secara sintaks dapat tercapai dari simpul n_i jika terdapat lintasan dari simpul n_i ke *n* (atau sisi *e*). Didefinisikan lintasan tes yang merupakan salah satu kunci dalam cakupan graf ini serta definisi dari cakupan graf itu sendiri.

Lintasan Tes : Sebuah lintasan *p*, mungkin panjangnya 0 yang dimulai dari simpul yang berada di N_0 dan berakhir pada simpul yang berada di N_f.

Cakupan Graf : Diberikan himpunan syarat tes ST (untuk selanjutnya disebut ST saja) untuk kriteria graf K, himpunan tes T memenuhi K pada graf G jika dan hanya jika untuk setiap syarat tes *st* pada ST, terdapat paling tidak satu lintasan tes *l* pada *lintasan*(*T*) di mana *p* sama dengan *st*.

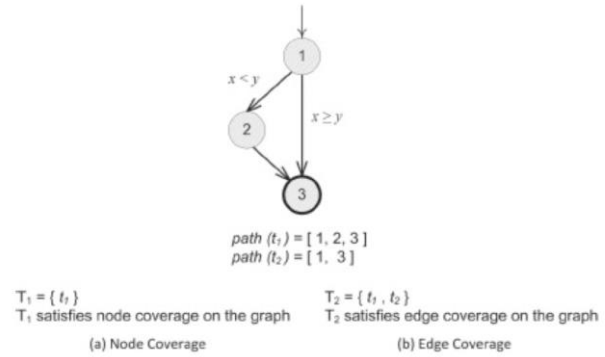
Kriteria dari cakupan graf terbagi menjadi 2 yaitu kirteria cakupan aliran kontrol atau biasa disebut juga kirteria cakupan graf terstruktur serta kriteria cakupan aliran data. Pada makalah ini hanya akan dibahas tentang kriteria cakupan aliran kontrol atau graf terstruktur.

E. Kriteria Cakupan Graf Terstruktur

Kriteria cakupan graf terstruktur bermula pada konsep untuk mengekstrak graf dari artefak perangkat lunak yang ada seperti kode, desain, spesifikasi, atau syarat sekalipun. Konsep ini lalu dikembangkan pada tujuan untuk mendapatkan *test case* yang mumpuni, yaitu dibutuhkannya kriteria untuk dapat mengunjungi setiap simpul yang ada dan setiap sisi yang ada pada graf tersebut agar dapat ditemukannya kesalahan perangkat lunak. Namun, pada perjalanannya ditemukan banyak istilah cakupan yang membuat kriteria yang satu ini cukup kompleks maupun istilah baru, istilah-istilah itu antara lain :

Cakupan Simpul : ST yang mengandung setiap simpul yang dapat tercapai secara sintaks pada graf G.

Cakupas Sisi : ST yang mengandung setiap lintasan yang dapat tercapai dengan panjang lebih dari 1 secara inklusif pada graf G.



Gambar 5. Perbedaan Cakupan Simpul dan Cakupan Sisi[1]

Cakupan Pasangan Sisi : ST yang mengandung setiap lintasan yang dapat tercapai dengan panjang lebih dari 2 secara inklusif pada graf G.

Lintasan sederhana : Sebuah lintasan dikatakan sebuah lintasan sederhana jika tidak ada simpul yang muncul lebih dari sekali pada suatu lintasan kecuali muncul pada awal dan akhir simpul di lintasan tersebut.

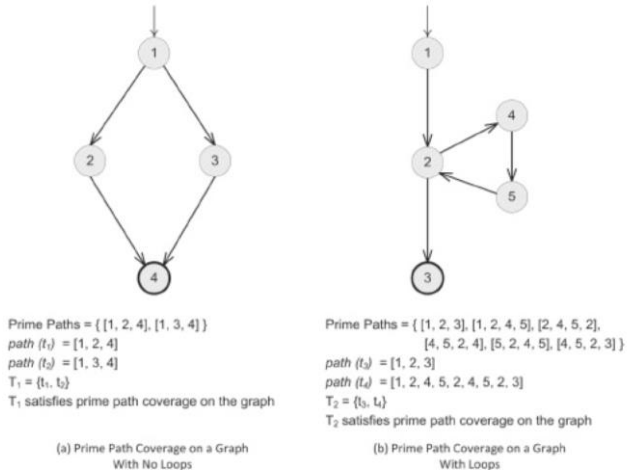
Lintasan prima : Sebuah lintasan dari n_i *ke* n_j disebut sebagai lintasan prima jika lintasan tersebut adalah lintasan sederhana dan tidak muncul sebagai upalintasan dari lintasan sederhana yang lain.

Cakupan Lintasan Prima : ST yang mengandung setiap lintasan prima pada graf G.

Cakupan lintasan prima memiliki 2 kasus khusus yaitu adanya gelang dengan lintasan perjalanan keliling yang didefinisikan sebagai lintasan prima yang memiliki simpul awal dan akhir yang sama. Sehingga cakupan ini terbagi menjadi 2 yaitu :

Cakupan Perjalanan Keliling Sederhana : ST yang mengandung paling tidak satu perjalanan keliling untuk setiap simpul yang dapat tercapai pada graf G.

Cakupan Perjalanan Keliling Lengkap : ST yang mengandung semua lintasan perjalanan keliling untuk setiap simpul yang dapat tercapai pada graf G.



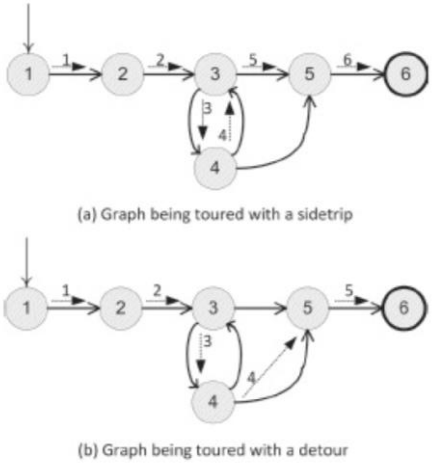
Gambar 6. Penerapan Cakupan Lintasan Prima pada (a) Graf yang tidak memiliki gelang serta (b) Graf yang memiliki gelang[1]

Salah satu hal yang cukup menantang bagi para penguji adalah adanya gelang pada graf yang telah diekstrak sebelumnya. Mengapa gelang memberikan tantangan tersendiri? Hal ini dikarenakan dengan terbentuknya lintasan sederhana, dapat menjadi hal yang belum cukup jelas apakah lintasan tersebut harus dilewati secara benar sesuai urutannya tanpa ada tambahan simpul atau sisi pada gelang, ataukah sebenarnya boleh saja untuk melewati gelang terlebih dahulu lalu tetap melanjutkan sesuai dengan urutan pada lintasan sederhana yang ada. Hal ini memunculkan istilah baru lagi yaitu tur (*tour*), perjalanan sampingan (*sidetrip*), dan perjalanan memutar (*detour*).

Tur : Lintasan tes *p* dikatakan melakukan tur pada upalintasan *q* jika dan hanya jika *q* adalah upalintasan dari *p*.

Tur dengan perjalanan sampingan : Lintasan tes *p* dikatakan melakukan tur pada upalintasan *q* dengan perjalanan sampingan jika dan hanya jika setiap **sisi** pada *q* juga merupakan sisi pada *p* dengan urutan yang sama.

Tur dengan perjalanan memutar : Lintasan tes *p* dikatakan melakukan tur pada upalintasan *q* dengan perjalanan memutar jika dan hanya jika setiap **simpul** pada *q* juga merupakan simpul pada *p* dengan urutan yang sama.



Gambar 7. (a) Graf yang lintasannya merupakan tur dengan perjalanan sampingan dan (b) Graf yang lintasannya merupakan tur dengan perjalanan memutar[1]

Walaupun perbedaannya bisa dikatakan sedikit, namun istilah ini cukup berguna. Upalintasan [3,4,3] merupakan perjalanan sampingan pada lintasan [2,3,5] dikarenakan memulai dari simpul 3 dan mengakhiri kembali pada simpul 3 dengan urutan sisi yang sama. Sedangkan upalintasan [3,4,5] adalah perjalanan memutar dari lintasan [2,3,5] dikarenakan memulai dan mengakiri lintasan dengan urutan simpul yang sama walaupun berbeda sisi.

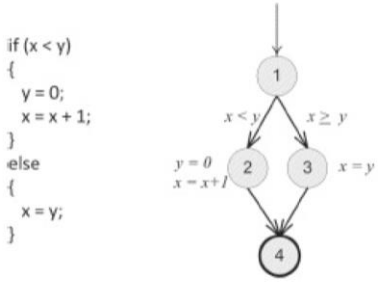
IV. Penerapan Cakupan Graf Terstruktur dalam Pengujian Perangkat Lunak

Penerapan Cakupan Graf Terstruktur pada pengujian perangkat lunak terbagi menjadi 4 yaitu untuk kode sumber, elemen desain, spesifikasi, dan kasus penggunaan.

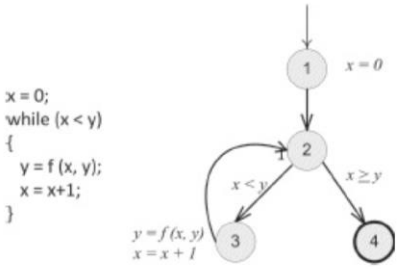
A. Kode Sumber

Kode sumber merupakan hal yang sangat lumrah di kalangan pengembang dan pemrogram. Hal ini dikarenakan kebanyakan pekerjaannya adalah mengenai kode sumber. Dengan menggunakan Cakupan Graf Terstruktur atau Cakupan Aliran Kontrol, kode sumber dapat divisualisasikan agar dapat mempermudah penguji dalam menghasilkan tes lintasan yang sesuai denga teori-teori cakupan yang telah dijelaskan sebelumnya.

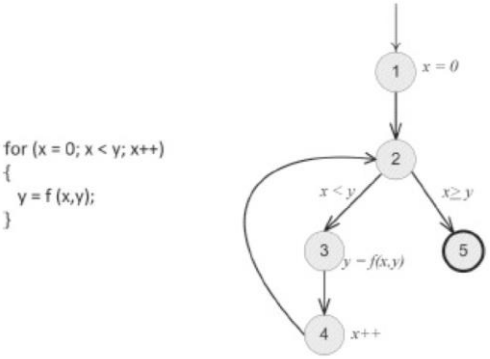
Berikut contol aliran kontrol dari struktur *if-else, for, while,* dan *do-while*



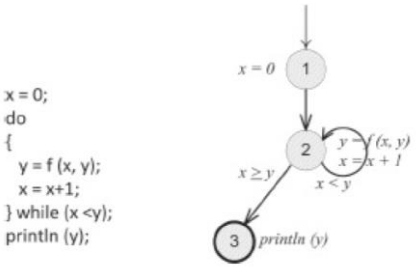
Gambar 8. Graf Aliran Kontrol untuk struktur *if-else*[1]



Gambar 9. Graf Aliran Kontrol untuk struktur *while*[1]



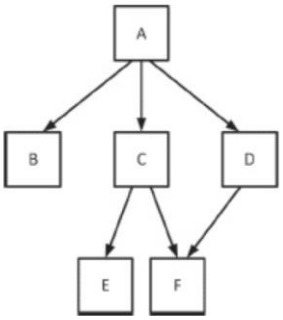
Gambar 10. Graf Aliran Kontrol untuk struktur gelang *for*



Gambar 11. Graf Aliran Kontrol untuk struktur *do-while*[1]

B. Elemen Desain

Berbeda halnya dengan kode sumber, elemen desain lebih berfokus pada komponen-komponen yang terdapat pada perangkat lunak itu seperti *method* atau fungsi-fungsi yang didefinisikan. Salah satu cakupan graf untuk elemen desain yang cukup populer adalah graf pemanggilan. Kita tahu bahwa dalam sebuah kode seringkali dilakukan sebuah abstraksi agar semakin mudah dipahami dengan membuat fungsi-fungsi atau *method*, dan seringkali pula setiap *method* membutuhkan *method* lain untuk menjalankan fungsinya. Pada Gambar 12, diperlihatkan salah satu penerapan graf aliran kontrol yaitu graf pemanggilan yang berisi 6 *method*.

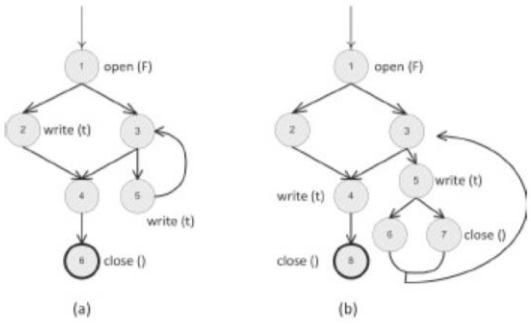


Gambar 12. Graf pemanggilan sederhana[1]

Cakupan kriteria dapat diterapkan pada graf ini. Misalnya dengan menggunakan cakupan sisi, dapat terlihat bahwa akan terjadi 2 kali pemanggilan pada *method* F, sedangkan dengan menggunakan cakupan simpul dapat terlihat bahwa akan ditelusurinya masing-masing simpul dengan *test case* yang dibuat. Cakupan simpul pada bagian ini dapat disebut juga sebagai Cakupan *Method*. Adapun dengan cakupan sisi disebut juga sebagai Cakupan Pemanggilan.

C. Spesifikasi

Untuk spesifikasi, cakupan graf aliran kontrol dapat membantu pengujian dalam memvisualisasikan dan membuat tes pada aturan-aturan secara sekuensial yang berlaku. Aturan-aturan tersebut berkorespondensi dengan *invariant*. Misalnya, kita tidak bisa melakukan *pop* pada suatu *stack* tanpa adanya pemanggilan fungsi *push*. *Invariant* seperti ini yang dapat direpresentasikan agar memudahkan pembuatan data tes. Sebagai contoh, Gambar 13 menunjukkan contoh penggunaan ADT terkait File Eksternal.



Gambar 13. Graf Aliran Kontrol pada ADT File Eksternal[1]

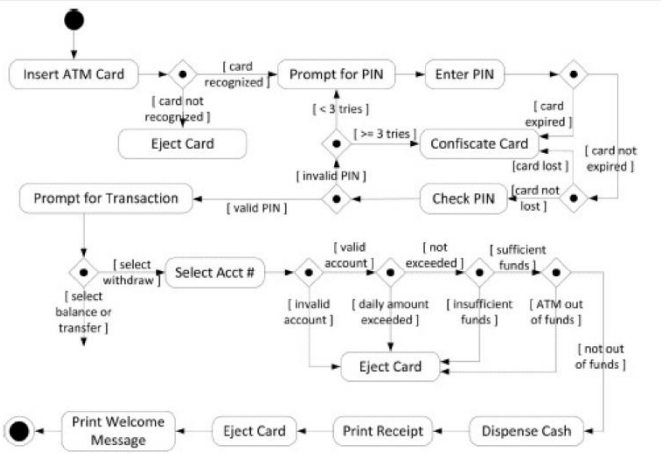
Pada kasus ini, ADT memiliki beberapa *invariant* beberapa diantaranya :

1. Pemanggilan fungsi *open( )* harus dilakukan sebelum pemanggilan fungsi *write( ).*
2. Pemanggilan fungsi *open( )* harus dilakukan sebelum setiap pemanggilan fungsi *close( ).*
3. Pemanggilan fungsi *close( )* tidak boleh dilakukan setelah pemanggilan fungsi *close( )* kecuali terdapat pemanggilan fungsi *open( )* diantara keduanya.

Dengan adanya *invariant* tersebut kita bisa melakukan pengecekan dengan menggunakan cakupan-cakupan seperti lintasan prima atau lintasan sederhana untuk dapat mencari dan memastikan bahwa *invariant* masih dapat terpenuhi semuanya.

D. Kasus Penggunaan

Pada kasus penggunaan, observasi yang dilakukan lebih kepada runtutan aksi perangkat lunak pada setiap masukan yang diberikan oleh pengguna. Maka dari itu dinamakan kasus penggunaan. Coba kita perhatikan kasus pengambilan uang tunai pada ATM (Anjungan Tunai Mandiri). Dengan menggunakan cakupan graf, kita perlu mengenumerasi kondisi-kondisi yang dihasilkan sehingga dihasilkan graf aliran kontrol seperti pada Gambar 14.



Gambar 14. Graf Aliran Kontrol pada kasus pengambilan uang tunai di ATM[1]

Dari graf yang telah diekstrak, dapat dilakukan berbagai macam cakupan untuk mendapatkan data tes yaitu mulai dari cakupan simpul dan cakupan sisi, sampai cakupan tur, baik dengan perjalanan sampingan, perjalanan memutar ataupun tidak sama sekali.

V. KESIMPULAN

Ilmu pengujian perangkat lunak merupakan ilmu yang cukup penting dalam bingkai keilmuan pengembangan perangkat lunak. Hal ini dikarenakan dibutuhkannya suatu sistem yang dapat memastikan keamanan, kualitas, kebertahanan, hingga efisiensi dari perangkat lunak yang dikembangkan dan ilmu pengujian ini sangat bermanfaat untuk hal tersebut. Salah satu ilmu yang dipakai dalam pengembangan ilmu pengujian perangkat lunak adalah graf yang salah satunya memunculkan konsep graf aliran kontrol atau graf terstruktur. Konsep ini membawa penguji agar bisa memvisualisasikan serta mendapatkan data tes yang sangat mumpuni dalam pengujian baik itu dalam pengujian kode sumber, elemen desain, spesifikasi, hingga kasus penggunaan.

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepad Allah SWT yang telah memberikan kemampuan kepada penulis untuk bisa menyelesaikan makalah ini tepat pada waktunya. Penulis juga mengucapkan terima kasih terhadap orangtua yang telah mendukung saya dalam berkembang dan berkuliah di Teknik Informatika ITB serta Bapak Rinaldi Munir yang telah memberikan pengajaran terhadap perkuliahan IF 2220 Matematika Diskrit terutama pada pengajaran materi graf yang saya angkat pada tema makalah kali ini.

REFERENSI

[1] Ammann, P., & Offutt, J. (2017). *Introduction to Software Testing.* New York: Cambridge University Press

[2] Beizer, B. (1990). *Software Testing Techniques.* New York: Van Nostrand Reinhold, Inc.

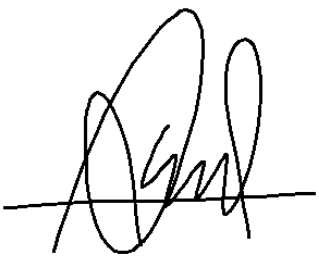
[3] Liem, I. (2008). *Diktat Struktur Data.* Bandung.

[4] Munir, R. (2012). *Matematika Diskrit.* Bandung: Informatika Bandung.

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, 9 Desember 2018



Muhammad Akmal 13517028