BAB II
TINJAUAN PUSTAKA

II.1 Profil Instansi

GITS Indonesia adalah perusahaan pengembang aplikasi Android. Berpengalaman selama hampir 3 tahun mengembangkan aplikasi, membuat GITS Indonesia dikenal sebagai developer Android paling produktif di Indonesia saat ini. Selain Android, hingga saat ini GITS Indonesia juga banyak mengembangkan aplikasi di berbagai platform mobile seperti iOS, Symbian, JavaME, Nokia Seri 40 dan Blackberry.

II.1.1 Sejarah Instansi

GITS Indonesia pertama kali berdiri pada September 2008 dengan nama Ganesha IT Solution (GITS). Akhir 2009 GITS memutuskan untuk memfokuskan pada pengembangan teknologi mobile terutama Android. Pada Juli 2011, GITS resmi menjadi badan usaha yang terdaftar dengan nama PT. GITS Indonesia. Sejak masa awal berdirinya, GITS telah menjalin kerja sama dengan berbagai instansi swasta maupun pemerintah di antaranya Departemen Pekerjaan Umum, Telkom Indonesia, Sarihusada, Nokia Indonesia serta Samsung Elektronik Indonesia. Beberapa produk mobile yang telah dikembangkan di antaranya adalah Toresto, Kosakata, Bahasa Dictionary, KoranDroid, dan Imsakiyah. Selain produk sendiri, terdapat juga beberapa aplikasi hasil kerja sama dengan kliennya seperti MindSocial for Mindtalk, Telkom H!Bandung, dan SGM Babies Doctor. Tagline GITS Indonesia adalah “get it simple”. Tagline tersebut menunjukan keinginan besar untuk menjadi *pioneer* teknologi mobile terutama Android yang selalu produktif mengembangkan aplikasi untuk membantu penggunanya menjalankan aktivitasnya sehari-hari.

II.1.2 Visi dan Misi Instansi

Visi yang dijalankan perusahaan adalah menjadi pioneer teknologi mobile di Indonesia dalam membantu penggunanya menjalankan aktivitasnya sehari-hari. Misi perusahaan memiliki tiga jenis yaitu :

1. Produktif

   Meningkatkan produktivitas perusahaan dan sumber daya manusianya untuk terus aktif dalam pengembangan aplikasi.

2. Estetis

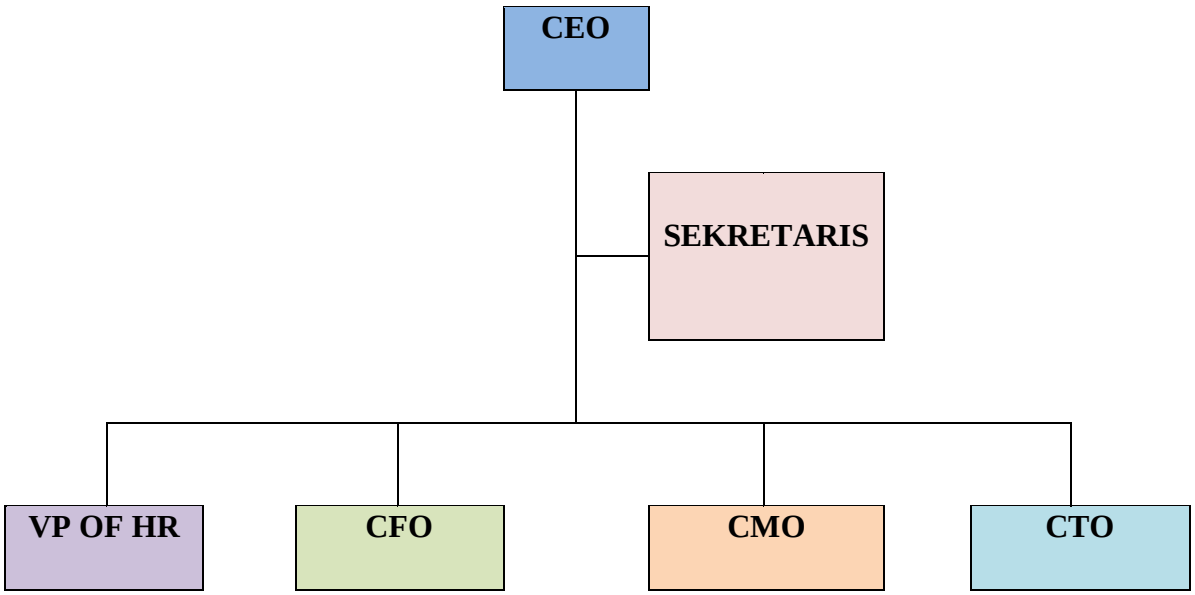
   Mengutamakan estetika dan ergonomi pada setiap aplikasi yang dikembangkan demi kemudahan pengguna.

3. Solutif

Tidak berhenti untuk mencari solusi terbaik menggunakan teknologi mobile yang memudahkan aktivitas penggunanya.

II.1.3 Struktur Organisasi Instansi

Struktur organisasi instansi adalah gambar bagan yang menjelaskan posisi dan hierarki struktur kerja pegawai di dalam perusahaan. Struktur organisasi perusahaan dapat dilihat padaGambar II-1 Struktur Organisasi PT. GITS Indonesia.



Gambar II-1 Struktur Organisasi PT. GITS Indonesia

II.1.4 Deskripsi Kerja Instansi

Deskripsi kerja Struktur Organisasi adalah penjelasan tentang bagian kerja. Deskripsi kerja dapat dilihat pada Tabel II-1 Deskripsi Kerja PT. GITS Indonesia.

Tabel II-1 Deskripsi Kerja PT. GITS Indonesia

No	Jabatan	Deskripsi Kerja
1	CFO (Chief Executive Officer)	Komunikator, pengambil keputusan, pemimpin, pengelola dan eksekutor
2	Sekertaris	Meliputi pengetikan, pembuatan surat menyurat, membuat laporan, pengaturan jadwal dan absensi.
3	VP (Vice President) of HR (Human Resources)	Manajemen sumber daya manusia (MSDM) dalam memberikan kontribusi pada pencapaian efektivitas organisasi. Melakukan pemilihan SDM untuk melaksanakan proyek.
4	CFO (Chief Financial Officer)	CFO bertanggung jawab untuk mengelola keuangan risiko perusahaan. Petugas ini juga bertanggung jawab untuk perencanaan keuangan dan pencatatan, serta pelaporan keuangan kepada manajemen yang lebih tinggi.
5	CMO (Chief Marketing Officer	CMO adalah seorang eksekutif perusahaan bertanggung jawab atas kegiatan pemasaran dalam suatu organisasi. Dengan tanggung jawab utamanya adalah pengembangan produk, manajemen saluran distribusi, komunikasi pemasaran (termasuk iklan dan promosi), harga, riset pasar dan layanan pelanggan
6	CTO (Chief Technology	CTO memiliki posisi tingkat eksekutif

	Officer)	dalam sebuah perusahaan atau badan lain yang penghuni difokuskan pada isu-isu ilmiah dan teknologi dalam sebuah organisasi.

II.2 Landasan Teori

Bab ini akan memaparkan tentang landasan teori yang digunakan untuk melakukan penelitian skripsi. Adapun landasan teori yang akan dibahas adalah tentang algoritma, bahasa pemrograman yang digunakan dan tools pembangunan perangkat lunak simulasi.

II.2.1 Optimalisasi

Optimasi adalah salah satu disiplin ilmu dalam matematika yang fokus untuk mendapatkan nilai minimum atau maksimum secara sistematis dari suatu fungsi, peluang, maupun pencarian nilai lainya dalam berbagai kasus. Optimasi global adalah algoritma optimasi yang menggunakan ukuran atau rentang yang luas daerah pencariannya tidak terbatas atau global. Kelebihan dari penggunaan optimasi global ini menurut Weise, Thomas (2008) adalah untuk mencegah tindakan fokus pada daerah masalah saja akan tetapi lebih meningkatkan kemungkinan untuk mencari sebuah global optimum pada daerah yang lebih luas. [**5**] Jadi, optimasi global adalah optimasi yang bertujuan untuk mencari solusi terbaik secara global atau menyeluruh pada sebuah model permasalahan. Pada optimasi global ini, batasannya adalah waktu, dimana rentang waktunya bersifat global, sehingga daerah penyelesaiannya lebih luas. Optimasi sangat berguna di hampir segala bidang dalam rangka melakukan usaha secara efektif efisien untuk mencapai target hasil yang ingin dicapai. Tentunya hal ini akan sangat sesuai dengan prinsip ekonomi yang berorientasikan untuk senantiasa menekan pengeluaran untuk menghasilkan outputan yang maksimal. Optimasi ini juga penting karena persaingan saat ini sudah benar benar sangat ketat.

Seperti yang dikatakan di awal, bahwasanya optimasi sangat berguna bagi hamper seluruh bidang yang ada, maka berikut ini adalah contoh contoh bidang yang sangat terbantu dengan adanya teknik optimasi tersebut. Bidang tersebut, antara lain : Arsitektur, Data Mining, Jaringan Komputer, Signal And Immage

Processing, Telekomunikasi, Ekonomi, Transportasi, Perdagangan, Pertanian, Perikanan, Perkebunan, Perhutanan, dan sebagainya.

Teknik optimasi secara umum dapat dibagi menjadi dua bagian, yang pertama adalah *Mathematical Programming*, dan yang kedua adalah *Combinatorial Optimatimization*. Dalam bidang *mathematical programming* dapat dibagi menjadi dua kembali, yaitu *support vector machines* dan *gradient descent*. Dan pada bidang Combinatorial Optimization kembali difokuskan lagi ke dalam dua bidang, yaitu *Graph Theory* dan *Genetic Algorithm*. Pemfokusan pemfokusan bidang tersebut dikarenakan beberapa parameter, diantaranya, *Restoration, Feature selection, Classification, Clustering, RF assignment, Compression*, dan sebagainya.

Adapun hal lain secara global yang penting untuk diperhatikan adalalh fokus terhadap model dan masalah serta cara berfikir yang analitis. Kita harus fokus terhadap model dan masalah agar tujuan utama dari kasus tersebut tercapai, jangan sampai terlalu konsen pada optimasi tetapi goalnya sendiri malah tidak tercapai. Sedangkan berfikir analitis dimaksudkan agar kita peka terhadap keadaan dan mampu berfikir secara bebas untuk menemukan solusi solusi yang diperlukan.

Sebagai contoh implementasi teknik optimasi ini, kita lihat ambil cara mudah untuk mengoptimalkan performance computer pada saat memakai suatu program agar berjalan lebih lancer. Caranya adalah dengan mematikan program program yang running namun sebenarnya tidak diperlukan. Jika computer kita tidak sedang membutuhkan koneksi dengan jaringan, sebaiknya semua service yang mendukung ataupun berhubungan dengan jaringan, ada baiknya dimatikan. Selain itu, jika kita pun tidak terkoneksi dengan ‘jalan masuknya virus’, ada baiknya sekalian mematikan anti virus yang sedang bekerja. Hal ini akan membuat performance computer kita lebih optimal, dengan mematikan program program yang tidak sedang dipakai dan memakan memory.

II.2.2 Algoritma

Algoritma adalah Urutan langkah-langkah untuk memecahkan suatu masalah. Terdapat beberapa definisi lain dari algoritma tetapi pada prinsipnya senada dengan definisi yang diungkapkan diatas yang kita kutip dari berbagai literatur, antara lain [**6**]:

1. Algoritma adalah deretan langkah-langkah komputasi yang mentransformasikan data masukan menjadi keluaran.
2. Algoritma adalah deretan instruksi yang jelas untuk memecahkan masalah, yaitu untuk memperoleh keluaran yang diinginkan dari suatu masukan dalam jumlah waktu yang terbatas.
3. Algoritma adalah Prosedur komputasi yang terdefinisi dengan baik yang menggunakan beberapa nilai sebagai masukan dan menghasilkan beberapa nilai yang disebut keluaran. Jadi, algoritma adalah deretan langkah komputasi yang mentransformasikan masukan menjadi keluaran.

Algortima adalah jantung ilmu computer atau informatika. Banyak cabang dari ilmu komputer yang diacu dalan terminologi algoritma, misalnya algoritma perutean (routing) pesan di dalam jaringan komputer, algoritma Brensenham untuk menggambar garis lurus (bidang grafik kumputer), algoritma Knuth-Morris-Pratt untuk mencari suatu pola di dalam teks (bidang information retrievel), dan sebagainya.

Ditinjau dari asal usul kata, kata “algoritma” sendiri mempunyai sejarah yang cukup aneh. Kata ini tidak muncul di dalam kamus Webster sampai akhir tahun 1957. Orang hanya menemukan kata algorism yang berarti proses menghitung dengan angka Arab. Anda dikatakan algorist jika Anda menggunakan angka Arab. Para ahli bahasa berusaha menemukan asal kata algorism ini, namun hasilnya kurang memuaskan. Akhirnya para ahli sejarah matematika menemukan asal mula kata tersebut. Kata algorism berasal dari nama penulis buku arab yang terkenal, yaitu Abu Ja’afarMuhammad Ibnu Musa al-Khuwarizmi (al-Khuwarizmi dibaca orang barat menjadi algorism). Al-Khuwarizmi menulis buku yang berjudul Kital al jabar wal-muqabala, yang artinya “Buku pemugaran dan pengurangan” (The book of restoration and reduction). Dari judul buku ini kita juga memperoleh akar kata “aljabar” (algebra). Perubahan dari kata algorism

menjadi algoritm muncul karena kata algorism sering dikelirukan dengan arithmetic, sehingga akhiran –sm beubah menjadi –thm. Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa/lumrah, maka lambat laun kata algorithm berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna aslinya. Dalam bahasa Indonesia, kata *algorithm* diserap menjadi "algoritma".

Kata algoritma perama kali digunakan pada tahun 1950 dalam kata "algoritma Euclidean" (Euclid's algorithm). Euclid, seorang matematikawan Yunani (lahir pada tahun 350 M), dalam bukunya yang berjudul Element menuliskan langkah-langkah untuk menemukan pembagi bersama terbesar (common greatest divisor atau gcd), dari dua buah bilangan bulat, m dan n (tapi Euclid tidak menyebut metodenya itu sebagai algoritma, baru abad modernlah ornag-orang menybut metodenya itu sebagai "algoritma Euclidean"), Pembagi terbesar dari dua buah bilangan bulat tak-negatif adalah bilangan bulat positif terbesar yang habis membagi kedua bilangan tersebut. [**6**]

II.2.2.1. Analisis Algoritma

Analisis algoritma dilakukan untuk mengetahui seberapa baik sebuah algoritma diimplementasikan untuk menyelesaikan suatu kasus. Penilaian terhadap algoritma melibatkan analisis algoritma sehingga terdapat 2 (dua) tipe analisis algoritma yaitu:

1. **Aspek kualitatif**

Aspek kualitatif yaitu analisis untuk memeriksa kebenaran algoritma. Analisis dengan aspek kualitatif dilakukan dengan penelusuran algoritma, dilakukan penelusuran logik menggunakan teknik matematika untuk membuktikan kebenaran atau implementasi algoritma atau mengujinya dengan data. Sebagai contoh adalah algoritma pengurutan data (*sorting*) tidak dapat disebut sebagai algoritma pengurutan jika algoritma tidak dapat mengurutkan sembarang masukan barisan data.

2. **Aspek kuantitatif**

Aspek kuantitatif yaitu analisis terhadap efisiensi algoritma. Aspek kuantitatif dilakukan dengan melakukan perhitungan kompleksitas komputasi (waktu) dan ruang. Aspek kuantitatif mencoba mengukur seberapa besar sumber

daya yang diperlukan suatu algoritma 2 (dua) sumber daya yang diukur adalah kecepatan bekerja algoritma dan ruang yang diperlukan untuk bekerja. Notasi untuk menyatakan kinerja antara lain adalah big-oh (O) yaitu waktu komputasi algoritma berbanding terhadap satu fungsi tertentu. Big-O biasanya hanya dinyatakan dengan suku yang paling berarti dan menghilangkan konstanta pengalinya. Jadi O((n2-n)/2) hanya dinyatakan dengan O(n2).

Terdapat tiga cara yang dapat dilakukan dalam melakukan analisis algoritma yaitu :

1. Analisis untuk memeriksa kebenaran algoritma.
2. Analisis efisiensi algoritma (kompleksitas komputasi dan ruang).
3. Analisis optimalitas algoritma.

1. Analisis Kebenaran Algoritma

Beberapa hal yang dapat dilakukan dalam menguji kebenaran suatu algoritma antara lain adalah:

1. Penelusuran algoritma
2. Penelusuran logik (*assertion*)
3. Implementasi algoritma
4. Pengujian dengan data
5. Teknik pengujian matematis untuk pembuktian kebenaran

Analisis kebenaran algoritma juga sering dimasukkan sebagai proses validasi algoritma.

2. Analisis Efisiensi Algoritma

Fase dalam analisis efisiensi algoritma terdapat dua hal yaitu *a priori analysis* dan *a priori testing*.

1. A priori analysis

Fase *a priori analysis* bertujuan untuk menemukan fungsi beserta parameter-parameter yang relevan yang membatasi waktu komputasi algoritma. Adapun notasi matematika yang digunakan untuk menunjukkan hasil adalah O-notation (Big-O), Ω-notation dan Θ-notation.

2. *A posteriori testing*

Pada fase ini dikumpulkan statistik nyata konsumsi waktu dan ruang suatu algoritma pada mesin dan bahasa pemrograman tertentu.

Tujuan dari dilakukannya fase ini dalah untuk :

Menentukan jumlah waktu dan ruang penyimpanan yang diperlukan program. Kegunaan dari fase ini adalah untuk memvalidasi *a priori analysis*.

3. Analisis Optimalitas Algoritma

Algoritma disebut optimal jika tidak terdapat algoritma lain di kelas persoalan itu yang mempunyai jumlah operasi yang lebih sedikit dibanding algoritma itu. Harus ditemukan *lower bound* jumlah operasi minimum yang perlu dilakukan untuk penyelesaian masalah. Jika algoritma menyelesaikan masalah dengan jumlah operasi yang sama dengan *lower bound* maka algoritma disebut optimal.

4. Kompleksitas Waktu Asimptotik

Kompleksitas waktu asimptotik merupakan waktu yang dibutuhkan suatu Algoritma menyelesaikan tiap langkahnya. Setiap Algoritma memiliki kompleksitas waktu yang berbeda-beda. Komplesitas waktu asimptotik diperlukan untuk menghitung performansi suatu Algoritma. Untuk menghitung kompleksitas waktu asimptotik suatu Algoritma digunakanlah notasi "O-Besar" (*Big-O*) yang merupakan notasi kompleksitas waktu asimptotik.

Perhitungan kompleksitas waktu asimptotik dilakukan dengan menghitung nilai O-besar dari setiap instruksi di dalam Algoritma. Aturan dalam perhitungan teorema O-Besar adalah sebagai berikut [**6**] :

1. Pengisian nilai (*assignment*), perbandingan, operasi aritmetik, *read*, *write* membutuhkan waktu *O*(1).
2. Pengaksesan elemen larik atau memilih *field* tertentu dari sebuah *record* membutuhkan waktu *O*(1).

```
read(x);          O(1)
x:=x+a[k];        O(1)+ O(1)+ O(1)= O(1)
writeln(x);       O(1)
```

Kompleksitas waktu di atas adalah *O*(1), didapat dari

= $O(1)+ O(1)+ O(1)$

= $O(\max(1,1))+ O(1)$

= $O(1)+ O(1)$

= $O(\max(1,1))$

= $O(1)$

3. If c then s1 else s2. Membutuhkan waktu Tc + max(Ts1,Ts2).

```
read(x);                  O(1)
if x mod 2=0  then        O(1)
  begin
    x:=x+1;               O(1)
  writeln(x);             O(1)
  end
else
  writeln(x);             O(1)
```

Kompleksitas waktu di atas adalah $O(1)$, didapat dari

= $O(1) + O(1)$ max ($O(1)+ O(1)$, $O(1)$)

= $O(1) + \max(O(1), O(1))$

= $O(1)$

4. Kalang for. Kompleksitas waktu kalang for adalah jumlah pengulangan dikali dengan kompleksitas waktu badan kalang.

```
For i=1 to n do           O(n)
  Jumlah:= jumlah +a[i];  O(1)
```

Kompleksitas waktu di atas adalah $O(n)$, didapat dari

= $O(n) . O(1)$

= $O(n.1)$

= $O(n)$

5. While c do s; dan repeat s until c; untuk kedua buah kalang, kompleksitas waktunya adalah jumlah pengulangan dikali dengan waktu badan c dan s.

```
 i:=2;                          O(1)
while i<=n do                   O(n)
 begin
   jumlah:=jumlah+a[i];         O(1)
   i:= i +1;                    O(1)
end;
```

Kompleksitas waktu di atas adalah $O(1)$, didapat dari

$= O(1) + O(n)\{O(1)+O(1)\}$

$= O(1) + O(n)\ O(1)$

$= O(1) + O(n.1)$

$= O(1) + O(n)$

$= O(n)$

II.2.2.2. Strategi Algoritma

Strategi algoritmik adalah kumpulan metode atau teknik untuk memecahkan masalah guna mencapai tujuan yang ditentukan, yang dalam hal ini deskripsi metode atau teknik tersebut dinyatakan dalam suatu urutan langkah-langkah penyelesaian.

Secara umum, strategi pemecahan masalah dapat dikelompokan menjadi:

1. Strategi solusi langsung, metode yang termasuk ke dalam strategi ini adalah Algoritma *Brute Force* dan Algoritma *Greedy*
2. Strategi berbasis pencarian pada ruang status, metode yang termasuk ke dalam strategi ini adalah Algoritma *Backtracking* dan Algoritma *Brach and Bound.*
3. Strategi solusi atas-bawah, metode yang termasuk ke dalam strategi ini adalah Algoritma *Divide and Conquer*.
4. Strategi solusi bawah-atas, metode yang termasuk ke dalam strategi ini adalah *Dynamic Programming*.

Algoritma *Greedy* merupakan salah satu metode yang paling populer untuk menyelesaikan persoalan optimasi. Yang dimaksud dengan persoalan optimasi adalah persoalan yang mencari solusi optimal, baik yang bersifat maksimal (maksimalisasi), atau minimal (minimalisasi). Dalam bahasa inggris,

Greedy berarti rakus atau tamak. Hal ini mencerminkan prinsip dari algoritma ini, yaitu "*take what you can get now!".* Algoritma ini akan membentuk solusi langkah per langkah. Pada setiap langkah, algoritma ini akan mengeksplorasi segala kemungkinan pilihan yang ada. Dari seluruh kemungkinan pilihan tersebut, akan diambil pilihan yang paling baik untuk setiap langkahnya. Pilihan terbaik ini disebut solusi optimal lokal.

Harapannya, dengan terus menerus mengambil pilihan optimal terbaik untuk tiap langkahnya, akan dihasilkan solusi global (solusi dari keseluruhan langkah) yang juga optimal. Kekurangan dari algoritma ini adalah tidak adanya perhatian terhadap konsekuensi dari tiap langkah yang diambil, sehingga bisa saja rangkaian langkah yang dihasilkan bukan merupakan solusi optimal global.

Algoritma greedy banyak digunakan dalam berbagai penyelesaian masalah, antara lain adalah :

1. *Optimal Storage on Tapes Problem.*
2. *Knapsack Problem.*
3. *Minimum Spanning Tree Problem*
4. *Shortest Path Problem*

Ada tiga pendekatan algoritma *greedy* dalam menyelesaikan persoalan *Integer Knapsack,* dimana salah satu pendekatan ini akan digunakan untuk cara memasukkan objek ke dalam knapsack [**2**]. Adapun tiga jenis pendekatannya yaitu :

1. ***Greedy by profit* (*Pi*)**

   Greedy by profit memprioritaskan objek dengan keuntungan paling besar. Cara ini digunakan untuk mengoptimalkan keuntungan dengan memilih objek yang memiliki keuntungan terbesar terlebih dahulu. Diketahui sebuah kapasitas *knapsack* W = 41, kemudian akan dimasukkan barang dengan bobot dan profit sebagai berikut :

   w1 = 5; p1 = 30;
   w2 = 15; p2 = 36;
   w3 = 25; p3 = 125;
   w4 = 15; p4 = 27;

Dari properti objek diatas maka perhitungan dengan menggunakan *greedy by profit* akan ditunjukkan pada Tabel II-2 Contoh *Kasus dan Pemilihan* Objek Dengan *Greedy By Profit*.

Tabel II-2 Contoh Kasus dan Pemilihan Objek Dengan *Greedy By Profit*

Properti Proyek			***Greedy by profit***	
I	w_i	p_i	***Knapsack 0/1***	***Fractional Knapsack***
1	5	30	0	0,2 atau 20%
2	15	36	1	1
3	25	125	1	1
4	15	27	0	0
Total bobot			**40**	**41**
Total keuntungan			**161**	**167**

Pada tabel baris *greedy by profit*, angka 1 (satu) menunjukkan bahwa kolom tersebut terpilih sebagai nilai paling optimal sedangkan angka 0 (nol) menunjukkan bahwa kolom tidak dipilih sebagai objek yang optimal dengan menggunakan cara *greedy by profit*.

2. *Greedy by weight* (*Wi*)

Greedy by weight memprioritaskan objek dengan berat paling ringan. Cara ini dimaksudkan untuk memaksimumkan keuntungan dengan memasukkan objek sebanyak mungkin ke dalam *knapsack*. Contoh kasus dengan menggunakan *greedy by weight* akan ditunjukkan dalam Tabel II-3 Contoh Kasus dan Pemilihan Objek *Dengan Greedy By Weight*. Properti objek yang digunakan masih dari contoh kasus sebelumnya.

Tabel II-3 Contoh Kasus dan Pemilihan Objek Dengan *Greedy By Weight*

Properti Proyek			***Greedy By Weight***	
I	w_i	p_i	***Knapsack 0/1***	***Fracrtional Knapsack***
1	5	30	1	1
2	15	36	1	1
3	25	125	0	0,24 atau 24%
4	15	27	1	1
Total bobot			**35**	**41**
Total keuntungan			**93**	**123**

Pada Tabel II-3 untuk pemilihan proyek dengan *knapsack* 0/1 solusi optimalnya adalah (1, 1, 0, 1) dengan total keuntungan 93 dimana nilai 1 (satu) adalah terpilih dan nilai 0 (nol) adalah tidak terpilih sebagai solusi optimal. Pemilihan proyek dengan *fractional knapsack* solusi optimalnya adalah (1, 1, 24%, 1) dengan total keuntungannya 123.

3. *Greedy by density* (*Pi/Wi*)

Greedy by density memprioritaskan objek dengan densitas (*pi / wi*) terbesar. Cara ini digunakan untuk memaksimumkan keuntungan dengan memilih objek yang memiliki keuntungan per unit berat terbesar. Contoh kasus dengan menggunakan *greedy* by density akan ditunjukkan dalam Tabel II-4 Contoh Kasus dan Pemilihan Objek *Dengan Greedy By Density*. Properti objek yang digunakan masih dari contoh kasus sebelumnya.

Tabel II-4 Contoh Kasus dan Pemilihan Objek Dengan *Greedy By Density*

Properti Proyek				***Greedy By Density***	
I	w_i	p_i	p_i/w_i	***Knapsack 0/1***	***Fractional Knapsack***
1	5	30	6	0	0,2 atau 20%
2	15	36	2,4	1	1
3	25	125	5	1	1
4	15	27	1,8	0	0
Total bobot				**40**	**41**
Total keuntungan				**161**	**167**

Tabel II-4 menunjukkan solusi optimal untuk pemilihan dengan *greedy by density* dengan *knapsack* 0/1 adalah (0, 1, 1, 0) dengan total keuntungan adalah 161. Nilai 0 (nol) berarti proyek ke-i tidak terpilih sebagai solusi optimal dan nilai 1 (satu) berarti proyek ke-i dinyatakan terpilih sebagai solusi optimal.

Pemilihan dengan *fractional knapsack* menunjukkan solusi optimalnya adalah (20%, 1, 1, 0) dengan total keuntungan 167. Nilai 20% atau 0,2 mengindikasikan bahwa proyek ke-*i* memenuhi solusi optimal dengan besar pertimbangan sebesar 20%.

Ketiga contoh pendekatan algoritma pada Tabel II-2, Tabel II-3 dan Tabel II-4 mengindikasikan bahwa yang selalu menghasilkan solusi yang paling optimum adalah algoritma *fractional knapsack* dengan menggunakan pendekatan *greedy by density*. Diperkuat dengan teori dari Rinaldi Munir bahwa Algoritma *greedy* untuk persoalan *fractional knapsack* dengan strategi pemilihan objek berdasarkan p_i/w_i terbesar akan selalu memberikan solusi optimal.

Berdasarkan atas kesimpulan tersebut maka peneliti akan menggunakan algoritma *fractional knapsack* dengan pendekatan *greedy by density* untuk menyelesaikan pemilihan proyek di PT. GITS Indonesia. Selain itu dengan

menggunakan fractional *knapsack*, objek yang tidak muat ke dalam pilihan (dikarenakan melebihi total constraint) masih dapat dipertimbangkan dengan satuan persentase sehingga keuntungan dapat dioptimalkan.

Pencarian solusi algoritma *greedy* melibatkan pencarian sebuah himpunan bagian *S*, dari himpunan kandidat, *C*; yang dalam hal ini, *S* harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan *S* dioptimisasi oleh fungsi obyektif.

Berikut akan dijelaskan apa itu himpunan-himpunan atau elemen yang terdapat dalam algoritma *greedy* :

1. Himpunan kandidat

   Himpunan yang berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi

   Himpunan yang berisi kandidat-kandidat yang terpilih sebagai solusi.
3. Fungsi seleksi

   Memilih kandidat yang paling memungkinkan mencapai solusi optimal.
4. Fungsi kelayakan

   Memeriksa apakah himpunan kandidat yang telah terpilih dapat menjadi solusi yang layak, yaitu tidak melanggar *constraint* yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.
5. Fungsi objektif

Fungsi yang memaksimumkan atau meminimumkan nilai soulsi (misalkan keuntungan, panjang lintasan, dan lain-lain).

II.2.2.3. Algoritma Knapsack

Knapsack telah dipelajari secara intensif sejak karya perintis Dantzig yaitu abad ke pada 50-an. Pada saat itu algoritma *Knapsack* diimplementasikan baik untuk aplikasi langsung dalam industri maupun manajemen keuangan, tetapi lebih sering digunakan untuk penelitian dan hal yang bersifat teoritis. Sebagai sebuah masalah, *knapsack* yang sering terjadi dengan relaksasi berbagai masalah integer programming [**3**].

Knapsack problem atau *rucksack problem* adalah masalah optimasi kombinatorial. Namanya berasal dari masalah maksimasi untuk pilihan paling

tepat dari barang-barang yang akan dibawa dalam sebuah tas pada sebuah perjalanan. Sejumlah barang yang tersedia ini, masing-masing memiliki berat dan nilai, yang menentukan jumlah barang yang dapat dibawa sehingga total berat tidak melebihi kapasitas tas dan dengan total nilai yang sebesar mungkin.

Terdapat beberapa jenis *knapsack problem* antara lain adalah :

1. **0/1 *Knapsack***

   Pada knapsack 0/1 setiap barang hanya tersedia 1 unit, *take it or leave it*. Cara pemilihan solusi optimumnya hanya tidak atau ya (0 atau 1) atau dengan kata lain objek lain yang tidak memenuhi aturan *constraint* tidak dapat dipertimbangkan lagi. Contoh kasus *knapsack* 0/1 dapat dilihat dalam Tabel II-2, Tabel II-3 dan Tabel II-4 pada sub bab sebelumnya.

2. ***Fractional knapsack***

   Barang boleh dibawa sebagian saja (unit dalam pecahan). Versi problem ini menjadi masuk akal apabila barang yang tersedia dapat dibagi-bagi misalnya gula, tepung, dan sebagainya. Pada algoritma *fractional knapsack* cara pemilihan solusi optimumnya bisa tidak, ya, dan dipertimbangkan dalam pecahan (0 atau 1 atau 1/x) sehingga bobot totalnya dapat dioptimalkan dengan total nilai *constraint*nya. Contoh pemilihan objek yang optimal fractional *knapsack* dapat dilihat dalam Tabel II-2, Tabel II-3, dan Tabel II-4 pada subbab sebelumnya.

II.2.3 *Tools* Pemodelan

Tools pemodelan adalah alat atau pernagkat dan metode yang digunakan untuk memodelkan perancangan perangkat lunak simulasi. Adapun tools pemodelan yang digunakan dalam penelitian ini adalah :

1. Basis Data
2. *Flow Chart*
3. Diagram Konteks
4. DFD (*Data Flow Diagram*)

II.2.3.1. *Basis Data*

Data merupakan fakta mengenai suatu objek seperti manusia, benda, peristiwa, konsep, keadaan dan sebagainya yang dapat dicatat dan mempunyai arti

secara implisit. Data dapat dinyatakan dalam bentuk angka, karakter atau simbol, sehingga bila data dikumpulkan dan saling berhubungan maka dikenal dengan istilah basis data (database). Sedangkan menurut George Tsu-der Chou basis data merupakan kumpulan informasi bermanfaat yang diorganisasikan ke dalam aturan yang khusus. Informasi ini adalah data yang telah diorganisasikan ke dalam bentuk yang sesuai dengan kebutuhan seseorang. Menurut *Encyclopedia of Computer Science and Engineer*, para ilmuwan di bidang informasi menerima definisi standar informasi yaitu data yang digunakan dalam pengambilan keputusan.

Definisi lain dari basis data menurut Fabbri dan Schwab adalah sistem berkas terpadu yang dirancang terutama untuk meminimalkan duplikasi data. Menurut Ramez Elmasri mendefinisikan basis data lebih dibatasi pada arti implisit yang khusus, yaitu:

1. Basis data merupakan penyajian suatu aspek dari dunia nyata (*real world*).
2. Basis data merupakan kumpulan data dari berbagai sumber yang secara logika mempunyai arti implisit. Sehingga data yang terkumpul secara acak dan tanpa mempunyai arti, tidak dapat disebut basis data.
3. Basis data perlu dirancang, dibangun dan data dikumpulkan untuk suatu tujuan. Basis data dapat digunakan oleh beberapa *user* dan beberapa aplikasi yang sesuai dengan kepentingan *user*.

Dari beberapa definisi-definisi tersebut, dapat dikatakan bahwa basis data memounyai berbagai sumber data dalam pengumpulan data, bervariasi derajat interaksi kejadian dari dunia nyata, dirancang dan dibangun agar dapat digunakan oleh beberapa user untuk berbagai kepentingan.

II.2.3.2. *Flowchart*

Flowchart adalah penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program. Flowchart menolong analis dan programmer untuk memecahkan masalah kedalam segmen-segmen yang lebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoperasian. Ada beberapa petunjuk yang harus diperhatikan bagi seseorang analis atau programmer dalam membuat flowchart yaitu :

1. Flowchart digambarkan dari halaman atas ke bawah dan dari kiri ke kanan.
2. Aktivitas yang digambarkan harus didefinisikan secara hati-hati dan definisi ini harus dapat dimengerti oleh pembacanya.
3. Kapan aktivitas dimulai dan berakhir harus ditentukan secara jelas.
4. Setiap langkah dari aktivitas harus diuraikan dengan menggunakan deskripsi kata kerja, misalkan "Menghitung Pajak Penjualan".
5. Setiap langkah dari aktivitas harus berada pada urutan yang benar.
6. Lingkup dan range dari aktifitas yang sedang digambarkan harus ditelusuri dengan hati-hati. Percabangan-percabangan yang memotong aktivitas yang sedang digambarkan tidak perlu digambarkan pada flowchart yang sama. Simbol konektor harus digunakan dan percabangannya diletakan pada halaman yang terpisah atau hilangkan seluruhnya bila percabangannya tidak berkaitan dengan sistem.
7. Gunakan simbol-simbol flowchart yang standar.

Flowchart terbagi dalam lima jenis yaitu :

1. Flowchart Sistem (*System Flowchart*)
2. Flowchart Paperwork / Flowchart Dokumen (*Document Flowchart*)
3. Flowchart Skematik (*Schematic Flowchart*)
4. Flowchart Program (*Program Flowchart*)
5. Flowchart Proses (*Process Flowchart*)

Simbol-simbol flowchart yang biasanya dipakai adalah simbol-simbol flowchart standar yang dikeluarkan oleh ANSI dan ISO. Simbol-simbol ini dapat dilihat pada Gambar simbol standar flowchart**.**

II.2.3.3. *Diagram Konteks*

Jenis pertama *Context* Diagram, adalah data flow diagram tingkat atas (DFD Top Level), yaitu diagram yang paling tidak detail, dari sebuah sistem informasi yang menggambarkan aliran-aliran data ke dalam dan ke luar sistem dan ke dalam dan ke luar entitas-entitas eksternal. (CD menggambarkan sistem dalam satu lingkaran dan hubungan dengan entitas luar. Lingkaran tersebut menggambarkan keseluruhan proses dalam sistem). Diagram konteks adalah arus data yang berfungsi untuk menggambarkan keterkaitan aliran-aliran data antara

sistem dengan bagian-bagian luar (kesatuan luar). Kesatuan luar ini merupakan sumber arus data atau tujuan data yang berhubungan dengan sistem informasi tersebut. Diagram konteks bisa disebut dengan *"Model sistem pokok (fundamental system model) mewakili keseluruhan elemen software dengan masukan (input) dan keluaran (output) yang diidentifikasi dengan anak panah masuk dan keluar memperlihatkan sumber data".*

II.2.3.4. *Data Flow Diagram (DFD)*

Data Flow Diagram (DFD) adalah representasi grafik dari sebuah sistem. DFD menggambarkan komponen-komponen sebuah sistem, aliran-aliran data di mana komponen-komponen tersebut, dan asal, tujuan, dan penyimpanan dari data tersebut [**8**]. *Data Flow Diagram* digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau lingkungan fisik dimana data tersebut akan disimpan. *Data Flow Diagram* juga digunakan pada metodologi pengembangan sistem yang terstruktur.

II.2.4 *Tools* Implementasi

Tools implementasi adalah alat-alat atau metode yang digunakan untuk mengimplementasikan perangkat lunak simulasi yang dibangun. Tools implementasi perangkat lunak yang dibangun antara lain adalah :

1. Bahasa Pemrograman
2. Borland Delphi

II.2.4.1. Bahasa Pemrograman

Pascal merupakan salah satu bahasa pemrograman tingkat tinggi. Pemrograman tingkat tinggi menandakan bahwa Pascal banyak menggunakan bahasa manusia dalam penulisan sintaksnya. Beberapa bahasa pemrograman tingkat tinggi lainnya yang ada yaitu BASIC dan DELPHI. Sedangkan bahasa pemrograman tingkat rendah yaitu bahasa pemrograman yang masih banyak menggunakan tanda-tanda sehingga bahasa ini cenderung lebih "kotor". Beberapa bahasa pemrograman tingkat rendah yang sering digunakan adalah C, C++ dan Java.

Turbo Pascal dan C merupakan program compiler. Compiler sendiri berarti program yang menerjemahkan tulisan berupa kode program menjadi bahasa mesin yang dapat dimengerti oleh komputer. Proses yang dilakukan disebut *compile* atau *compiling.* Suatu program dapat di-compile dengan menggunakan kombinasi tombol Alt + F9. Sedangkan untuk menjalankan program, tombol yang dapat digunakan adalah Ctrl + F9. Bila selama proses compile terdapat error atau kesalahan dalam penulisan program, maka Turbo Pascal atau C akan memberitahukan letak kesalahan tersebut sehingga pengguna dapat memperbaikinya di halaman editor. Turbo Pascal merupakan bahasa yang case insensitive yang berarti penulisan dalam huruf kapital maupun huruf kecil tidak dipermasalahkan. Akan tetapi pada C yang case sensitive, maka penulisan huruf kapital atau kecil harus benar-benar diperhatikan.

Pada setiap kode pemrograman, terdapat aturan yang harus dipatuhi agar program tersebut dapat menjalankan (*compile*) suatu program dengan baik tanpa error. Struktur utama pada Turbo Pascal adalah sebagai berikut :

```
program     ... ; {deklarasi nama program}
uses        ... ; {deklarasi penggunaan unit}
label       ... ; {deklarasi label}
const       ... ; {deklarasi konstant}
type        ... ; {deklarasi tipe data}
var         ... ; {deklarasi variabel}
procedure    ... ; {deklarasi procedure}
function    ... ; {deklarasi function}
begin
statement; {program ditulis di sini}
...
end.
```

II.2.4.2. Borland Delphi

Borland Delphi 7 adalah paket bahasa pemrograman yang bekerja dalam sistem operasi windows. Delphi merupakan bahasa pemrograman yang mempunyai cakupan kemampuan yang luas dan sangat canggih. Berbagai jenis

aplikasi dapat dibuat dengan Delphi, termasuk aplikasi untuk mengolah teks, grafik, angka, database dan aplikasi web.

Secara umum, kemampuan Delphi adalah menyediakan komponen-komponen dan bahasa pemrograman yang andal, sehingga memungkinkan untuk dapat membuat program aplikasi sesuai dengan keinginan, dengan tampilan dan kemampuan yang canggih.

Untuk mempermudah *programer* dalam membuat program aplikasi, Delphi menyediakan fasilitas pemrograman yang sangat lengkap. Fasilitas pemrograman tersebut dibagi dalam 2 kelompok, yaitu *object* dan bahasa pemrograman. Secara ringkas, *object* adalah suatu komponen yang mempunyai bentuk fisik dan biasanya dapat dilihat (*visual*). *Object* biasanya dipakai untuk melakukan tugas tertentu dan mempunyai batasan-batasan tertentu. Sedangkan bahasa pemrograman secara singkat dapat disebut sebagai sekumpulan teks yang mempunyai arti tertentu dan disusun dengan aturan tertentu serta untuk menjalankan tugas tertentu. Delphi menggunakan struktur bahasa pemrograman *Object* Pascal yang sudah sangat dikenal di kalangan pemrogram profesional. Gabungan dari *object* dan bahasa pemrograman ini sering disebut sebagai bahasa pemrograman berorientasi *object* atau *Object Oriented Programing* (OOP).

Delphi telah memberikan komponen-komponen yang mudah dalam membangun aplikasi. Dalam grafis pun delphi dapat mengolah data yang membuat tampilan gambar seperti grafik equalizer dengan menggunakan komponen khsus. Secara matematis delphi 7 telah menyediakan operator untuk mengolah data sehingga dalam membuat aplikasinya pun lebih mudah, adapun tingkatan operator dalam delphi 7 sehingga jika muncul fungsi matematis maka akan diolah secara otomatis.

II.2.5 Pengujian Perangkat Lunak

Pengujian adalah Proses menelusuri dan mempelajari sebuah program dalam rangka menemukan kesalahan pada perangkat lunak sebelum diserahkan kepada pengguna [**9**]. Pengujian perangkat lunak dalam penelitian ini dibagi menjadi dua metode pengujian yaitu metode pengujian white-box dan metode pengujian black-box.

.

II.2.5.1. Metode Pengujian White-box

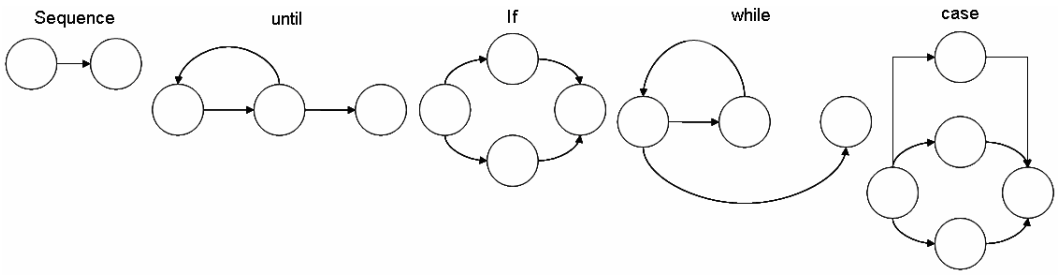
Merupakan metode perancangan test case yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan test case. Dengan menggunakan metode white box, analis sistem akan dapat memperoleh test case yang:

1. Menjamin seluruh independent path di dalam modul yang dikerjakan sekurang-kurangnya sekali
2. Mengerjakan seluruh keputusan logikal
3. Mengerjakan seluruh loop yang sesuai dengan batasannya
4. Mengerjakan seluruh struktur data internal yang menjamin validitas

Dua teknik dalam metode pengujian white-box adalah dengan uji coba basis path dan pengujian loop.

II.2.5.1.1. Basis Path

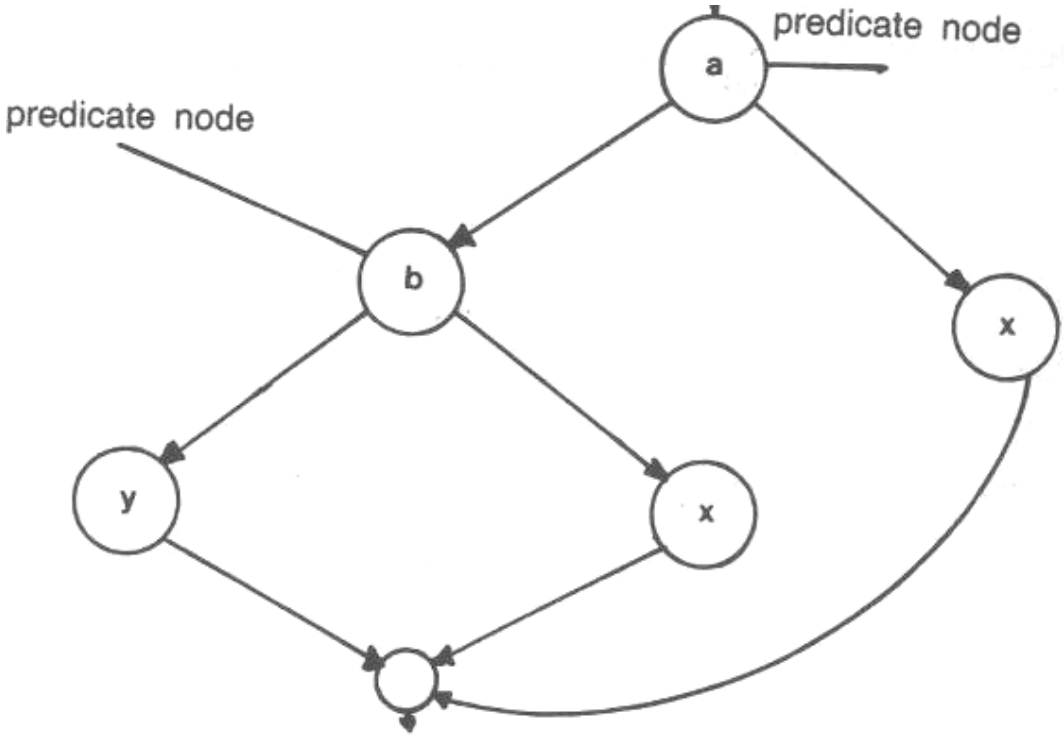
Merupakan teknik uji coba white box yang diusulkan Tom McCabe. Metode ini memungkinkan perancang test case mendapatkan ukuran kekompleksan logical dari perancangan prosedural dan menggunakan ukuran ini sebagai petunjuk untuk mendefinisikan basis set dari jalur pengerjaan. *Test case* yang didapat digunakan untuk mengerjakan basis set yang menjamin pengerjaan setiap perintah minimal satu kali selama uji coba. *Test case* dapat dilakukan dengan membuat diagram alir dari suatu algortima. Jenis notasi diagram alir dapat dilihat pada Gambar II-2.



Gambar II-2 Notasi Diagram Alir Pengujian

Lingkaran *(node)*, menggambarkan satu/lebih perintah prosedural. Urutan proses dan keputusan dapat dipetakan dalam satu node. *Tanda panah (edge)*,

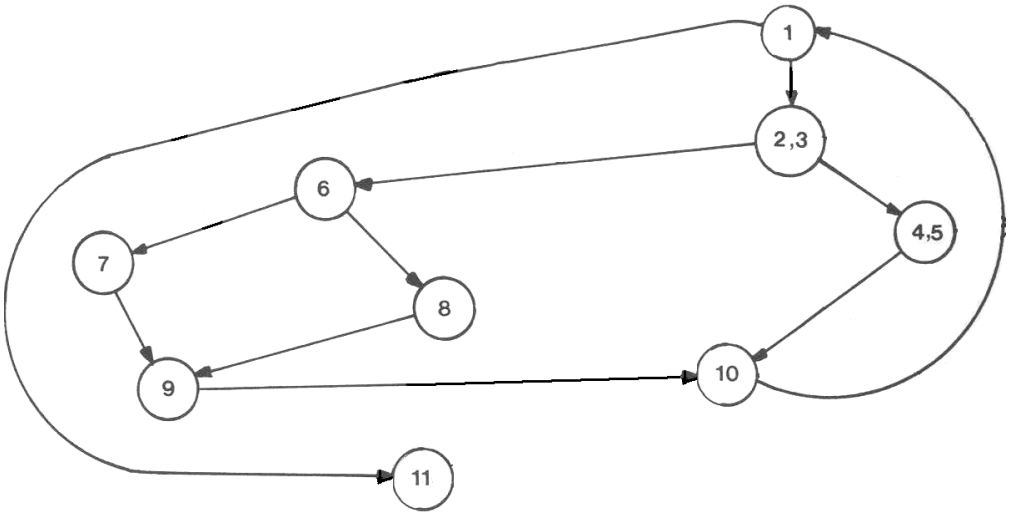
menggambarkan aliran kontrol. Setiap node harus mempunyai tujuan node. *Region* adalah daerah yang dibatasi oleh edge dan node. Termasuk daerah diluar grafik alir.



Gambar II-3 Contoh Diagram Alir (*Flow Graph*) dan *Predicate Node*

Node dibuat terpisah untuk masing-masing kondisi A dan B dari pernyataan IF A OR B. Masing-masing node berisi kondisi yang disebut *pridicate node* dan mempunyai karakteristik dua atau lebih edge darinya.

Cyclomatic complexity adalah metrik software yang menyediakan ukuran kuantitatif dari kekompleksan logikal program. Apabila digunakan dalam konteks metode uji coba basis path, nilai yang dihitung untuk *cyclomatic complexity* menentukan jumlah jalur independen dalam basis set suatu program dan memberi batas atas untuk jumlah uji coba yang harus dikerjakan untuk menjamin bahwa seluruh perintah sekurang-kurangnya telah dikerjakan sekali. Jalur *independent* adalah jalur yang melintasi atau melalui program dimana sekurang-kurangnya terdapat proses perintah yang baru atau kondisi yang baru.



Gambar II-4 Contoh Diagram Alir

Dari gambar Gambar II-4 Contoh Diagram Alir maka langkah perhitungannya adalah sebagai berikut :

Path 1 = 1 - 11

Path 2 = 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11

Path 3 = 1 - 2 - 3 - 6 - 8 - 9 ...: 10 - 1 - 11

Path 4 = 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11

Path 1,2,3,4 yang telah didefinisikan diatas merupakan basis set untuk diagram alir.

1. Cyclomatic complexity

Cyclomatic complexity digunakan untuk mencari jumlah path dalam satu flowgraph. Dapat dipergunakan rumusan sebagai berikut :

1. Jumlah region grafik alir sesuai dengan cyclomatic complexity.
2. Cyclomatix complexity V(G) untuk grafik alir dihitung dengan rumus:
   $V(G) = E - N + 2$
   Dimana:
   E = jumlah edge pada grafik alir
   N = jumlah node pada grafik alir
3. Cyclomatix complexity V(G) juga dapat dihitung dengan rumus:
   $V(G) = P + 1$
   Dimana P = jumlah predicate node pada grafik alir

Pada Gambar dapat dihitung cyclomatic complexity:

1. Flowgraph mempunyai 4 region
2. V(G) = 11 edge - 9 node + 2 = 4
3. V(G) = 3 predicate node + 1 = 4

Jadi cyclomatic complexity untuk flowgraph adalah 4.

2. Graph Metrik

Graph metrik merupakan software yang dikembangkan untuk membantu uji coba basis path atau struktur data. Graph metrik adalah matrik empat persegi yang mempunyai ukuran yang sama dengan jumlah node pada flowgraph. Masing-masing baris dan kolom mempunyai hubungan dengan node yang telah ditentukan dan pemasukan data matrik berhubungan dengan hubungan (edge) antar node.

Hubungan bobot menyediakan tambahan informasi tentang aliran kontrol. Secara simpel hubungan bobot dapat diberi nilai 1 jika ada hubungan antara node atau nilai 0 jika tidak ada hubungan. Dapat juga hubungan bobot diberi tanda dengan:

1. Kemungkinan link (edge) dikerjakan
2. Waktu yang digunakan untuk proses selama traversal dari link
3. Memori yang diperlukan selama traversal link
4. Sumber daya yang diperlukan selama traversal link

II.2.5.1.2. Pengujian Loop

Loop merupakan kendala yang sering muncul untuk menerapkan algoritma dengan tepat. Uji coba loop merupakan teknik pengujian white box yang fokusnya pada validitas dari loop. Kelas loop yaitu : loop sederhana, loop tersarang, loop terangkai, loop tidak terstruktur.

II.2.5.2. *Metode Pengujian Black-Box*

Metode ujicoba *blackbox* memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba blackbox memungkinkan pengembang software untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba blackbox bukan merupakan alternatif dari

ujicoba whitebox, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode whitebox.

Ujicoba blackbox berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi

Tidak seperti metode whitebox yang dilaksanakan diawal proses, ujicoba blackbox diaplikasikan dibeberapa tahapan berikutnya. Karena ujicoba blackbox dengan sengaja mengabaikan struktur kontrol, sehingga perhatiannya difokuskan pada informasi *domain*. Ujicoba didesain untuk dapat menjawab pertanyaanpertanyaan berikut :

1. Bagaimana validitas fungsionalnya diuji.
2. Jenis input seperti apa yang akan menghasilkan kasus uji yang baik.
3. Apakah sistem secara khusus sensitif terhadap nilai input tertentu.
4. Bagaimana batasan-batasan kelas data diisolasi.
5. Berapa rasio data dan jumlah data yang dapat ditoleransi oleh sistem.
6. Apa akibat yang akan timbul dari kombinasi spesifik data pada operasi sistem.

Dengan mengaplikasikan ujicoba blackbox, diharapkan dapat menghasilkan sekumpulan kasus uji yang memenuhi kriteria berikut :

1. Kasus uji yang berkurang, jika jumlahnya lebih dari 1, maka jumlah dari ujikasus tambahan harus didesain untuk mencapai ujicoba yang cukup beralasan.

2. Kasus uji yang memberitahukan sesuatu tentang keberadaan atau tidaknya suatu jenis kesalahan, daripada kesalahan yang terhubung hanya dengan suatu ujicoba yang spesifik.