

Rivalitas Algoritme BFS dan DFS dalam Menentukan Arah Tur-Produk Sekuensial untuk Pengguna Baru

Ryan Samuel Chandra - 13521140
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): cadanganryansc@gmail.com

Abstrak—Selama ini, tur-produk (tutorial/panduan) pada banyak aplikasi disajikan secara *brute force*. Algoritme ini tidak praktis, sebab kode tur harus dirombak ulang apabila ada fitur baru yang ditambahkan. Penulis mengusulkan algoritme *traversal* graf: BFS dan DFS sebagai penggantinya. Dengan paradigma objek, aplikasi sederhana dibuat untuk diujicobakan kepada beberapa orang mahasiswa. Hasil yang didapat dianalisis sebagai pembandingan efektivitas kedua algoritme untuk tur produk.

Kata Kunci—Strategi Algoritma, Penelusuran Graf, Algoritme BFS dan DFS, Tur Produk, Bahasa Pemrograman C++

Abstract—Until now, product-tours (tutorials/guides) on many applications are written by using *brute force*. This algorithm is impractical, because the tour-code has to be reconstructed every time a new feature is added. The author proposes graph traversal algorithm: BFS and DFS as a replacement. With object-oriented paradigm, simple application is made to be tested on several students. The results obtained were analyzed to compare the effectiveness of these two algorithms in designing product tours.

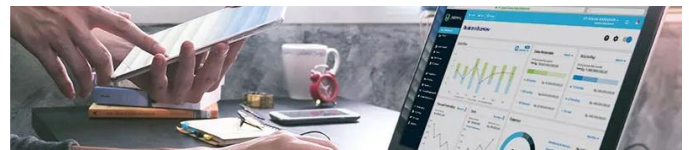
Keywords—Algorithm Strategies, Graph Traversal, BFS and DFS Algorithm, Product Tour, C++ Programming Language

I. PENDAHULUAN

Hampir seluruh karya tulis saya dimulai dengan satu-dua baris kalimat khas, “Berbeda dengan ciptaan lainnya, manusia dapat berpikir untuk mencari solusi, dan mengembangkan peradaban.” Tidak bisa dipungkiri lagi, dunia kita telah berubah menjadi panggung eksplorasi nirbatas. Setiap saat, manusia selalu berusaha memecahkan berbagai persoalan, demi mengembangkan ilmu pengetahuan yang kian canggih. Meski sadar akan ketidaksempurnaannya, akal budi beserta hati nurani—pemberian Tuhan Yang Mahakuasa—mendorong manusia merealisasikan kemajuan besar-besaran, termasuk dalam bidang teknologi dan informasi.

Komputer diartikan sebagai alat elektronik otomatis yang dapat menghitung atau mengolah data secara cermat menurut instruksi, dan memberikan hasil pengolahan, serta dapat menjalankan sistem multimedia. Di dunia modern, persentase ketergantungan kegiatan manusia terhadap komputer mencapai ketinggian di luar nalar. Sehari saja gawai tersebut dilarang digunakan, bisa dibayangkan seberapa macet aktivitas belajar dan mengajar, manajemen proyek, hingga perdagangan.

Salah satu produk nyata nonbenda yang sudah umum beredar di masyarakat adalah perangkat lunak (*software*). Menurut KBBI, perangkat lunak adalah program atau aplikasi yang diisikan ke dalam memori internal komputer. Dengan dukungan dari perangkat keras yang sesuai, *software* dapat membantu manusia dalam menyelesaikan banyak pekerjaan secara efisien.



Gambar 1 Ilustrasi Penggunaan Perangkat Lunak
(Sumber: <https://dianisa.com/pengertian-software/>)

Secanggih-canggihnya perangkat lunak, orang awam belum tentu mampu mendapatkan manfaat maksimal tanpa melihat tata cara penggunaannya terlebih dahulu. Tuntunan langkah demi langkah biasanya dikemas rapi menjadi sebuah video atau teks yang bernama tutorial. Dalam aplikasi, fitur ini kerap disebut tur produk (*product tour*). Nyatanya, sebagian besar pengguna baru, lebih memilih memahami sebuah aplikasi melalui pengalaman menggunakannya. Namun, tur produk setidaknya tetap memberikan gambaran keseluruhan aplikasi, serta jalur instan menuju fungsionalitas yang diinginkan.

Penelusuran secara daring telah membuktikan bahwa belum banyak algoritme tur-produk yang diekspos ke publik. Atas dasar inilah rasa penasaran penulis seketika bangkit. Pasalnya, masalah ini membutuhkan struktur kronologis yang paling mudah dikaitkan dengan *brute force*. Padahal, pembangunan kode dan kompleksitas algoritme tersebut tidak elegan.

Adapun pendekatan lain yang diusulkan oleh penulis adalah algoritme *traversal* graf, yakni BFS dan DFS. Di sini, masing-masing fitur tombol dianggap sebagai sebuah objek (sesuai konsep *object-oriented programming*) dan direpresentasikan oleh sebuah simpul dalam graf. Sisi yang menghubungkan dua buah simpul menyatakan bahwa fitur dengan aras yang lebih dalam, dapat dicapai dari fitur bersangkutan. Dimulai dari halaman utama, BFS dan DFS dipastikan mengunjungi setiap fitur dengan alur berbeda, untuk kemudian menyediakan layanan *guide* kepada pengguna.

Percobaan kali ini berfokus pada pertandingan BFS dan DFS, yang dinilai dari segi keefektifitasan waktu untuk membuat pengguna mengerti aplikasi sederhana yang diberikan dadakan.

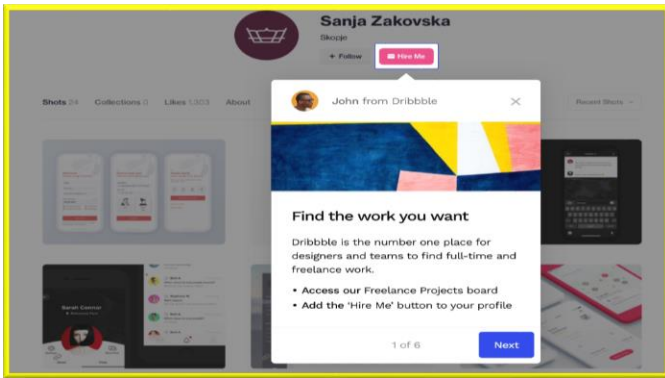
II. TINJAUAN PUSTAKA

A. Tur Produk

Seseorang yang ingin produknya selalu dikenang, harus menyediakan semua informasi yang diperlukan oleh para pengguna. Tur-produk adalah pola orientasi yang digunakan untuk memandu pengguna secara virtual terkait dasar-dasar dan fitur yang diperlukan dari suatu produk [1]. Tur produk seharusnya definitif, jelas, dan sederhana, agar pengguna dapat mengerti dengan cepat. Dengan panduan kontekstual saat berinteraksi dengan produk, pengguna langsung mengenali arti produk tersebut, dan mengambil tindakan yang tepat untuk menyelesaikan tujuan [3].

Tanpa tur produk (*product tour*), pengguna baru mungkin kesulitan karena rumitnya produk. Tur produk memiliki beberapa keuntungan sebagai berikut [4]:

- mengurangi tingkat “*user churn*”, yaitu ukuran banyaknya pengguna yang berhenti menggunakan produk setelah penggunaan pertama,
- meningkatkan keterlibatan serta kepuasan pengguna,
- cara yang lebih efisien dan efektif untuk memperkenalkan produk, daripada dokumentasi atau video tutorial.



Gambar 2 Contoh Tur-Produk Sebuah Aplikasi

(Sumber: <https://dribbble.com/shots/10846965-Intercom-Product-Tour-Step>)

Tur produk juga disajikan dalam banyak variasi. Beberapa jenis tur produk adalah [4]:

1. **Onboarding tour**: ditampilkan saat pertama kali pengguna masuk atau mulai menggunakan produk.
2. **Feature tour**: memperkenalkan fitur khusus produk kepada pengguna. Muncul ketika ada tindakan tertentu dari pengguna, atau dapat diakses melalui menu bantuan.
3. **Interactive tour**: menggunakan animasi, video, dan kuis yang melibatkan tanggapan pengguna.
4. **Tooltip tour**: menggunakan *pop-up* untuk menyorot bagian tertentu dari antarmuka dan menjelaskan cara kerjanya.
5. **Video tour**: menggunakan video untuk menampilkan produk dan fitur-fiturnya; digunakan sebagai tambahan.
6. **Micro tour**: singkat dan terfokus yang menyoroti satu fitur atau aspek spesifik dari produk.
7. **Guided tour**: dipimpin oleh pemandu langsung, baik melalui konferensi video atau obrolan, yang dapat menjawab pertanyaan dan memberikan bantuan personal.
8. **Self-paced tour**: memungkinkan pengguna menelusuri tur sesuai kecepatan masing-masing; tur ini dapat dijeda atau diputar ulang sewaktu-waktu sesuai kebutuhan pengguna.

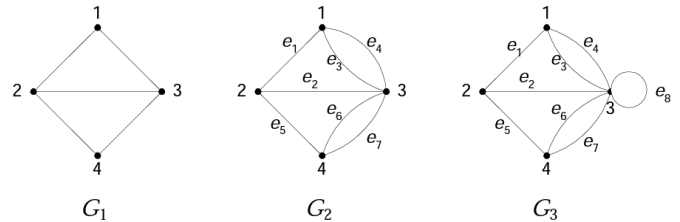
9. **Gamified tour**: menggunakan elemen seperti permainan.

10. **Virtual tour**: menggunakan *virtual reality* atau *augmented reality* untuk memberikan pengalaman yang mendalam.

Tur produk yang sukses membuat pengguna memperdalam pengetahuan produk, terdiri dari 25 kata per langkah, dan tidak boleh menggunakan gaya bahasa yang memerintah pengguna. Tur perlu ditunjukkan secara bertahap, menawarkan informasi yang tepat pada waktu yang tepat. Bahkan, pengguna diberikan opsi untuk melihat, atau tidak melihat tur. Tombol "tunda" disediakan untuk memungkinkan pengguna kembali dengan nyaman. Tur harus diungkapkan dengan pesan dan desain menarik, cocok dengan tema atau merek, serta memberikan nilai tertentu. Terakhir, pola desain yang digunakan tidak terlalu invasif (misalnya menutupi seluruh layar). Pengguna diberikan kesempatan untuk belajar sambil menerapkan. Untuk memenuhi kebutuhan pengguna yang terus berkembang, tur produk selalu diuji, dan umpan balik yang diterima menjadi pertimbangan saat perbaikan dilaksanakan.

B. Teori Graf

Graf adalah sebuah struktur yang digunakan sebagai representasi objek-objek diskrit dan hubungan antara objek tersebut [5]. Definisi graf $G = (V, E)$, yang dalam hal ini $V = \{v_1, v_2, \dots, v_n\}$ adalah himpunan tidak-kosong dari simpul (*vertices*) dan $E = \{e_1, e_2, \dots, e_n\}$ adalah himpunan sisi (*edges*) yang menghubungkan sepasang simpul.

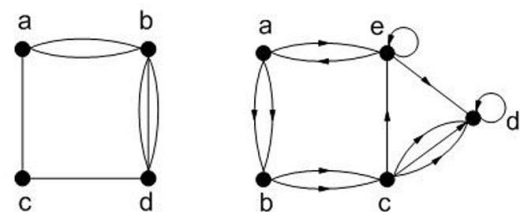


Gambar 3 Contoh Graf Tidak Berarah

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis2020-2021/Graf-2020-Bagian1.pdf>)

Pada gambar di atas, G_3 adalah graf dengan $V = \{1, 2, 3, 4\}$ dan $E = \{(1, 2), (2, 3), (1, 3), (1, 3), (2, 4), (3, 4), (3, 4), (3, 3)\}$ atau bisa langsung dituliskan $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$. Berdasarkan orientasi arah pada sisi, terdapat graf tak-berarah (*undirected graph*), yaitu graf yang tidak mempunyai orientasi arah, dan graf berarah (*directed graph*), yaitu graf yang setiap sisinya diberikan orientasi arah.

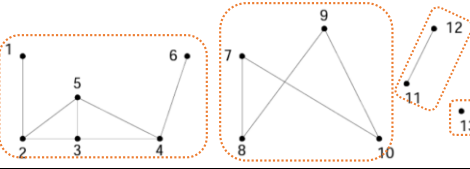
Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana (*simple graph*, contohnya G_1), sedangkan lawannya adalah graf tak-sederhana (*unsimple graph*, contoh: G_2, G_3). Graf tak-sederhana dibedakan lagi menjadi graf ganda (ada sisi ganda; *multigraph*) serta graf semu (ada sisi gelang; *pseudograph*).



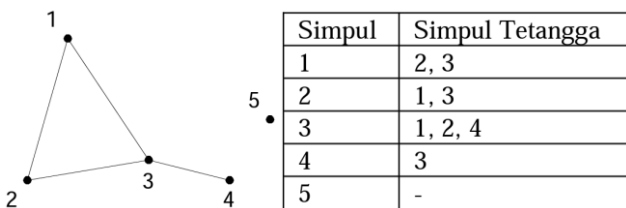
Gambar 4 Graf Ganda Tak-Berarah dan Graf Semu Berarah

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis2020-2021/Graf-2020-Bagian1.pdf>)

Tabel 1 Terminologi pada Graf [51]

#	Istilah	Penjelasan
1	Bertetangaan	Untuk setiap $e = (v_1, v_2) \rightarrow v_1$ bertetangga dengan v_2
2	Bersisian	Untuk setiap $e = (v_1, v_2) \rightarrow e$ bersisian dengan v_1 dan v_2
3	Simpul Terpencil	Simpul yang tidak punya sisi yang bersisian dengannya
4	Graf Kosong	Graf yang E -nya merupakan himpunan kosong ($E = \{ \}$)
5	Derajat (<i>Degree</i>)	Jumlah sisi yang bersisian dengan sebuah simpul; $d(v)$
6	Lintasan (<i>Path</i>)	Barisan berselang-seling simpul dan sisi: $v_0, e_1, v_1, e_2, v_2, \dots, v_{n-1}, e_n, v_n$ yang membentuk jalur dengan panjang $= n$
7	Siklus / Sirkuit	Lintasan berawal dan berakhir pada simpul yang sama
8	Kerterhubungan	v_1 dan v_2 terhubung jika terdapat lintasan dari v_1 ke v_2 .
9	Upagraf (<i>Subgraph</i>)	$G = (V, E), G_1 = (V_1, E_1)$ adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$. Komplemen $= G - G_1$
10	Komponen Graf (<i>Connected Component</i>)	Jumlah maksimum upagraf terhubung dalam suatu graf. Contoh sebuah graf yang mempunyai 4 komponen: 
11	Upagraf Rentang	$G = (V, E), G_1 = (V_1, E_1)$ upagraf rentang jika $V_1=V$
12	Graf Berbobot	Graf yang setiap sisinya diberi sebuah harga (bobot)
13	Graf Lengkap	<i>Simple</i> , setiap simpul punya sisi ke semua simpul lain
14	Graf Lingkaran	Graf sederhana yang setiap simpulnya berderajat dua
15	Graf Teratur	Graf yang setiap simpulnya mempunyai derajat sama
16	Graf <i>Bipartite</i>	Himpunan simpulnya dapat dipisah menjadi dua bagian, setiap sisi menghubungkan simpul di V_1 ke simpul di V_2

Graf dapat direpresentasikan sebagai matriks ketetangaan (*adjacency matrix*), yaitu $A = [a_{ij}]$; a_{ij} bernilai jumlah sisi yang menghubungkan simpul i dan j ⁽¹⁾, dan bernilai 0 jika kedua simpul tidak bertetangga. Selain itu, ada representasi berupa matriks bersisian (*incidency matrix*); a_{ij} bernilai 1 (atau bobot sisi) jika simpul i bersisian dengan simpul j , dan bernilai 0 untuk sebaliknya. Graf juga dapat ditulis sebagai *adjacency list* (senarai ketetangaan).



Gambar 5 Representasi Graf: Senarai Ketetangaan

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf>)

Istilah “*exhaustive search*” pada algoritme *brute force* juga berlaku pada dua algoritme yang sangat penting, yang secara sistematis memproses semua simpul dan sisi dalam graf ^[2]. Algoritme *traversal* graf, *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) terbukti berguna untuk berbagai aplikasi yang melibatkan graf dalam kecerdasan buatan, atau sekadar pemeriksaan sifat dasar graf seperti keterhubungan dan keberadaan sirkuit. Dalam proses pencarian solusi, terdapat pendekatan graf statis (terbentuk sebelum pencarian dilakukan), dan graf dinamis (terbentuk saat proses pencarian) ^[7].

⁽¹⁾ Pada graf berarah, nilai a_{ij} adalah sisi yang mengarah ke simpul j dari i .

C. Algoritme DFS

Depth-First Search atau pencarian mendalam, dimulai dari pada sembarang simpul dengan mengubah statusnya menjadi “dikunjungi”. Pada setiap iterasi, algoritme mengunjungi simpul yang belum dikunjungi dan bersebelahan dengan simpul saat ini. Proses terus berlanjut sampai bertemu jalan buntu—sebuah simpul tanpa simpul-tetangga yang belum dikunjungi. Lalu, algoritme mem-*backtrack* (kembali) sepanjang satu sisi ke simpul sebelumnya, dan mencoba untuk terus mengunjungi simpul yang belum dikunjungi dari sana ^[2].

Algoritme DFS dapat berhenti setelah *backtrack* ke simpul awal, dan bertemu jalan buntu. Pada saat itu, semua simpul dalam komponen-graf yang sama dengan simpul awal telah dikunjungi. Jika masih ada simpul yang belum dikunjungi, pencarian harus dimulai kembali dari salah satu simpul di komponen-graf yang lain ^[2].

Struktur data *stack* (tumpukan) dengan prinsip LIFO (*Last-In First-Out*) digunakan untuk melacak setiap langkah dari operasi DFS. Selain itu, dibuat juga sebuah tabel *boolean* yang bernama “dikunjungi”; $\text{dikunjungi}[x]=\text{true}$ jika simpul x sudah dikunjungi, dan sebaliknya bernilai *false*. Tabel tersebut harus dibuat di luar fungsi/prosedur (sebagai peubah global) sebab algoritme DFS bekerja secara rekursif.

```

procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS}

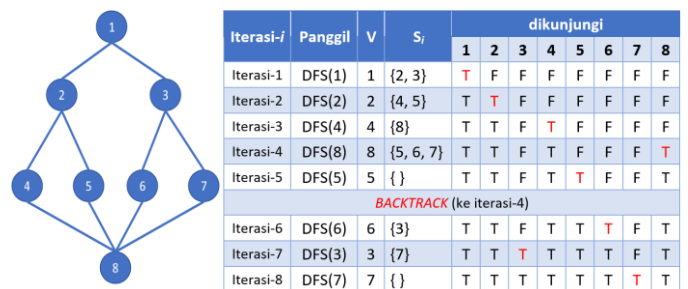
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi ditulis ke layar
}
Deklarasi
w : integer

Algoritma:
write(v)
dikunjungi[v] ← true
for w ← 1 to n do
  if A[v,w]=1 then {simpul v dan simpul w bertetangga }
    if not dikunjungi[w] then
      DFS(w)
    endif
  endif
endfor

```

Gambar 6 Pseudocode Algoritme DFS Secara Umum

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>)



Gambar 7 Proses Penelusuran Graf dengan DFS

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf> dan tabel dari dokumentasi penulis [dibuat menggunakan Microsoft Word])

Pada gambar di atas, urutan simpul yang dikunjungi adalah 1-2-4-8-5-6-3-7. Setiap simpul yang masuk ke kolom “V” ditandai sebagai “*true*” (dikunjungi). Proses melakukan satu kali *backtrack* karena menemukan jalan buntu di simpul 5.

Jadi, algoritme DFS termasuk ke dalam kategori *uninformed search* atau *blind search* karena tidak ada informasi tambahan

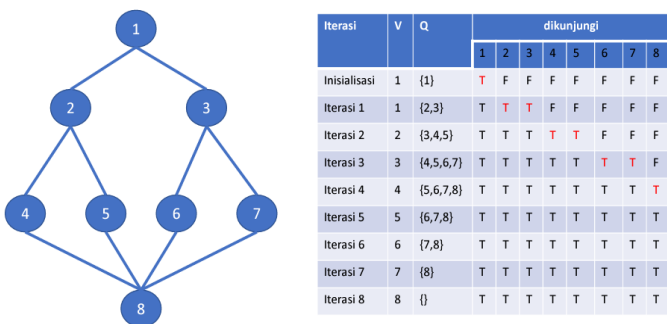
yang diberikan selain struktur graf itu sendiri. Kompleksitas algoritmenya adalah $O(|V|^2)$ jika graf direpresentasikan sebagai matriks ketetangaan, atau $O(|V|+|E|)$ jika representasi graf berupa senarai ketetangaan [2].

D. Algoritme BFS

Jika DFS adalah algoritme pencarian yang “berani” (berjalan sejauh mungkin dari simpul akar), *Breadth-First Search* atau pencarian melebar adalah algoritme pencarian yang bersifat “hati-hati”. Algoritme ini secara konsentris mengunjungi semua simpul yang bertetangga dengan simpul awal terlebih dahulu, lalu semua simpul yang berjarak dua sisi dari simpul awal, dan seterusnya, sampai semua simpul yang terhubung dengan simpul awal dikunjungi [2]. Jika masih ada simpul yang belum dikunjungi, algoritme dimulai kembali dari suatu titik pada komponen-graf yang lain.

Struktur data *queue* (antrean) dengan prinsip FIFO (*First-In First-Out*) digunakan untuk melacak setiap langkah dari operasi BFS. Selain itu, dibuat juga sebuah tabel *boolean* yang bernama “dikunjungi”; $dikunjungi[x]=true$ jika simpul x sudah dikunjungi, dan sebaliknya bernilai *false*.

Asumsikan *traversal* dimulai dari simpul v . Pertama-tama, kunjungi simpul v . Kemudian untuk setiap iterasi, kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu, lalu kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi [7].



Gambar 8 Proses Penelusuran Graf dengan BFS

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>)

Gambar di atas menunjukkan, setiap simpul yang masuk ke antrean langsung ditandai sebagai simpul yang sudah pernah dikunjungi, sehingga tidak akan masuk kembali ke antrean. Urutan simpul yang dikunjungi adalah: 1-2-3-4-5-6-7-8.

Sama seperti DFS, algoritme BFS termasuk ke dalam kategori *uninformed search* atau *blind search* karena tidak ada informasi tambahan yang diberikan selain struktur graf itu sendiri. Kompleksitas algoritmenya adalah $O(|V|^2)$ jika graf direpresentasikan sebagai matriks ketetangaan, atau $O(|V|+|E|)$ jika representasi graf berupa senarai ketetangaan [2].

E. Pemrograman Berorientasi Objek

Menurut Meyer (1998), sebuah sistem yang dibangun berdasarkan metode berorientasi-objek adalah sebuah sistem yang komponennya dienkapsulasi menjadi kelompok data dan fungsi, yang dapat mewarisi atribut dan sifat dari komponen lainnya, dan komponen-komponen tersebut saling berinteraksi satu sama lain [8]. Seorang pemrogram tidak lagi membagi suatu persoalan menjadi data dan fungsi/prosedur secara terpisah.

Objek adalah benda (*thing*) atau sebuah entitas yang bersifat fisik seperti mobil, buku, gajah, atau bersifat *intangible* seperti lagu, akun bank, dan film. Dalam konsep *object-oriented programming*, “semua” dianggap sebagai objek—termasuk *integer*, *character*, dan sebagainya [8]. Objek dapat terbentuk atas objek-objek lain. Misalnya, objek mobil terdiri atas objek mesin, sasis, *body*. Mesin terdiri objek blok silinder, busi, piston, dan lain-lain.

Setiap objek memiliki peran dan tanggung jawab (layanan) yang telah disepakati, serta memiliki akses terhadap informasi yang diperlukan untuk memenuhinya. Informasi tersebut bisa dimiliki sendiri, ataupun ditanyakan ke objek lain. Tanggung jawab objek dapat berupa:

1. melakukan perhitungan (komputasi),
2. memberi tahu perubahan *state* dirinya,
3. mengkoordinir objek-objek lain,
4. memberi informasi tertentu bagi objek lain yang meminta.

Objek dan Kelas

Pada implementasi di dalam bahasa pemrograman, objek didefinisikan dengan sekelompok atribut dan *method*. Atribut adalah informasi milik objek yang dimaksud, juga menentukan karakteristik dan *state* suatu objek. Sedangkan *method* adalah aksi atau perilaku objek yang bisa dieksekusi untuk mendapat layanan dari objek tersebut.

Kelas adalah *blueprint* yang mendeskripsikan objek-objek, berisi definisi dari atribut dan *method*. Dari sebuah kelas bisa diciptakan banyak objek. Misalkan kelas lampu lalu lintas dengan *timer* sederhana, mendefinisikan bahwa setiap objek dari kelas ini memiliki *timer* di dalamnya. Setiap *instance* dari kelas ini adalah objek lampu lalu lintas yang memiliki *timer*, namun konfigurasi *timer* setiap *instance* dapat berbeda [8].

Objek diciptakan dengan mengirim *message* kepada kelas untuk menginvokasi suatu *method* khusus yang disebut *constructor (ctor)*. Dalam bahasa berorientasi-objek, nama *ctor* biasanya sama dengan nama kelas. Konstruktork dipakai untuk menginisialisasi atribut-atribut suatu objek [8].

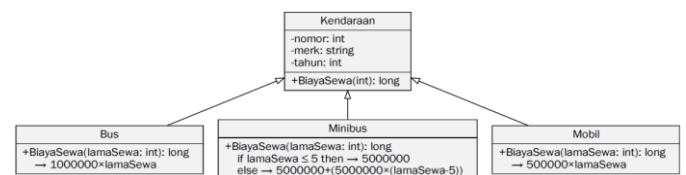


Gambar 9 Ilustrasi Kelas dan Objek

(Sumber: <https://www.oreilly.com/library/view/head-first-kotlin/9781491996683/ch04.html>)

Inheritance dan Polymorphism

Kemampuan kelas untuk menurunkan atribut dan *method* dari kelas lain disebut dengan *inheritance*. *Subclass / derived class* adalah sebutan bagi kelas “anak”, yang menurunkan atribut dan *method* dari kelas “induk” (*superclass / base class*).



Gambar 10 Contoh *Inheritance*

(Sumber: Salindia Kuliah IF2210 Pemrograman Berorientasi Objek, STEI-ITB)

Pada Gambar 10, tidak diperlukan atribut kategori pada kelas induk, karena fungsi `BiayaSewa()` diimplementasikan di masing-masing *subclass* sesuai dengan keperluan. Ketika ada jenis kendaraan baru, pemrogram dapat membuat *subclass* baru tanpa harus mengubah kelas “Kendaraan”.

Kelas “Kendaraan” adalah kelas abstrak karena ada *method* yang tidak diimplementasikan. Kelas abstrak tidak dapat langsung diinstansiasi. Objek yang diciptakan adalah instansiasi dari *subclass*-nya yang tidak abstrak [9].

Objek-objek dari kelas turunan memiliki sifat sebagai kelas tersebut dan sekaligus kelas dasarnya. Inilah yang disebut *polymorphism* (*poly* = banyak, *morph* = bentuk). Sebuah “variabel” dapat dideklarasikan bertipe “Kendaraan”, namun pada *run time* dapat diisi instansiasi dari kelas “Mobil” atau “Bus” (*subclass* dari “Kendaraan”). Sebaliknya, sebuah objek dari kelas “Mobil” dapat diperlakukan sebagai objek dari kelas “Kendaraan”. Jadi, *message* yang dapat diterima “Kendaraan” juga dapat diterima oleh “Mobil” [9].

F. Statistika Dasar

Populasi variabel acak diartikan sebagai data yang didapat dari keseluruhan pengamatan. Sampel adalah suatu himpunan bagian dari populasi. Variabel *n* menyatakan banyaknya sampel. Bias adalah inferensi *overestimate* / *underestimate* untuk hasil perhitungan statistik sampel [10].

Berikut adalah beberapa rumus statistik yang penting terkait properti sampel (X_1, X_2, \dots, X_n), dan uji hipotesis dua rata-ran:

Tabel 2 Rumus Statistik

Statistik	Rumus	Keterangan
Rata-Rata	$\bar{X} = \frac{\sum_{i=1}^n X_i}{n}$	-
Modus	Hitung nilai X_i yang paling banyak muncul	-
Median	$X_{med} = \begin{cases} \frac{X_{\frac{n+1}{2}}, & n \text{ ganjil} \\ \frac{X_{\frac{n}{2}} + X_{\frac{n}{2}+1}}{2}, & n \text{ genap} \end{cases}$	-
Variansi	$S^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n - 1}$	-
Simpangan Baku	$S = \sqrt{S^2}$	-
Kemiringan / Skewness	$Sk = \frac{\sum_{i=1}^n (X_i - \bar{X})^3}{n \times (S^2)^{\frac{3}{2}}}$	$Sk = 0 \rightarrow$ data distribusi normal
Uji Hipotesis Dua Rataan (menggunakan tabel distribusi-T)	$t' = \frac{(\bar{X}_1 - \bar{X}_2) - d_0}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$ $v = \frac{(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2})^2}{\frac{(S_1^2/n_1)^2}{n_1 - 1} + \frac{(S_2^2/n_2)^2}{n_2 - 1}}$	$H_0: \mu_1 - \mu_2 = d_0$ $H_1: \mu_1 - \mu_2 \neq d_0$ Daerah kritis: $t' < -t_{\alpha/2}$ atau $t' > t_{\alpha/2}$ v adalah derajat kebebasan

df/p	0.40	0.25	0.10	0.05	0.025	0.01	0.005	0.0005
1	0.324920	1.000000	3.077684	6.313752	12.70620	31.82052	63.65674	636.6192
2	0.288675	0.816497	1.885618	2.919986	4.30265	6.96456	9.92484	31.5991
3	0.276671	0.764892	1.637744	2.353363	3.18245	4.54070	5.84091	12.9240
4	0.270722	0.740697	1.533206	2.131847	2.77645	3.74695	4.60409	8.6103
5	0.267181	0.726687	1.475884	2.015048	2.57058	3.36493	4.03214	6.8688

Gambar 11 Potongan Tabel Distribusi-T

(Sumber: <https://www.dummies.com/article/academics-the-arts/math/statistics/how-to-use-the-t-table-to-solve-statistics-problems-147282/>)

III. IMPLEMENTASI

Implementasi algoritme BFS dan DFS untuk *product tour* kali ini bertema aplikasi belanja sederhana. Penulisan kode dilakukan dalam bahasa pemrograman C++, salah satu bahasa yang mendukung pemrograman berorientasi objek. Berikut disajikan isi dari semua fail *header* saja, sedangkan program lengkap dapat dilihat pada pranala di bagian **Pranala**.

A. Struktur Kelas Graf

```
main.c Graph.hpp Things.hpp Game.hpp Feature.hpp Button.hpp
1 #ifndef _GRAPH_HPP_
2 #define _GRAPH_HPP_
3
4 #include <iostream>
5 #include <list>
6 #include <vector>
7 using namespace std;
8
9 class Graph {
10 public:
11     list<int> *adjlist;
12     int n;
13     bool *DFSvisited;
14     vector<int> DFSnodes;
15     vector<int> BFSnodes;
16
17     Graph();
18     Graph(int n);
19
20     void addEdge(int u, int v, bool bi);
21     void printGraph();
22     void BFS(int s);
23     void DFS(int s);
24
25     vector<int> getBFS();
26     vector<int> getDFS();
27 };
28
29 #endif
```

Gambar 12 Struktur Kelas Graf dalam File “Graph.hpp”
(Sumber: dokumentasi penulis)

Kelas ini memiliki atribut `list<int> *adjlist` yang berfungsi untuk merepresentasikan graf dalam bentuk senarai ketetanggaan. *List*, yang merupakan salah satu bagian dari C++ STL (*Standard Template Library*) ini, dapat dipakai untuk alokasi memori nonkontinu dan dinamis. *Pointer* digunakan untuk menunjuk ke alamat memori dari objek *list*, sehingga pengaksesan, manipulasi, dan modifikasi isi dari objek tersebut berlangsung secara efisien.

Jadi, isi dari graf dalam aplikasi ini adalah *integer*, bukan objek fitur/tombol aplikasi secara langsung (padahal seharusnya bisa). Hambatan ini terjadi karena keterbatasan waktu serta pengetahuan akan bahasa pemrograman yang dipakai. Ketika ada perintah `addEdge(1, 2, true)`, yang sebenarnya dilakukan program adalah `adjlist[1].push_back(2)` dan `adjlist[2].push_back(1)`, atau dalam bahasa sehari-hari: *list* ditambahkan elemen “2” pada indeks 1 dan elemen “1” pada indeks 2. Meskipun demikian, graf tetap dapat berkontribusi secara optimal, hanya dengan menambahkan sebuah senarai *L* pada kelas pengontrol, yang mencocokkan urutan objek tertentu di dalamnya dengan nomor simpul. Misalnya, objek A bertetangga dengan objek B, maka A diletakkan pada `L[1]` dan B diletakkan pada `L[2]`.

Atribut *n* akan diisi oleh jumlah simpul dalam graf. Terdapat juga boolean `*DFSvisited` untuk menyimpan simpul mana saja yang sudah dikunjungi, apabila penelusuran menggunakan algoritme DFS. Bagian lain dari STL, *vector*, digunakan untuk menyimpan urutan simpul hasil *traversal*

algoritme BFS maupun DFS, yang dibangkitkan melalui pemanggilan *method* kelas graf, void BFS(int s) atau void DFS(int s) dengan parameter “s” adalah nomor dari simpul (indeks) pertama penelusuran.

B. Fitur dan Tombol

```

main.c Graph.hpp Things.hpp Game.hpp Feature.hpp Button.hpp
1 #ifndef FEATURE_HPP
2 #define FEATURE_HPP
3
4 #include <iostream>
5 using namespace std;
6
7 class Feature {
8     protected:
9         int state;
10    public:
11        Feature(int);
12        void virtual showTour() = 0;
13        void printDisplay();
14        int getState();
15    };
16 #endif
17
main.cpp Graph.hpp Things.hpp Game.hpp Feature.hpp Button.hpp
1 #ifndef BUTTON_HPP
2 #define BUTTON_HPP
3
4 #include <iostream>
5 #include "Feature.hpp"
6 using namespace std;
7
8 class Button : public Feature {
9     private:
10
11    public:
12        Button(int);
13        void virtual showTour() = 0;
14        Button& operator= (const Button&);
15    };
16
17 class Button00 : public Button { //MAIN
18     private:
19
20    public:
21        Button00();
22        void showTour();
23    };
24
25 class Button01 : public Button { //REN
26     private:
27
28    public:
29        Button01();
30        void showTour();
31    };
32
33 class Button02 : public Button { //YANM
34     private:
35
36    public:
37        Button02();
38        void showTour();
39    };
40

```

Gambar 13 Struktur Kelas Fitur dan Tombol dalam File “Feature.hpp” dan “Button.hpp” (tidak lengkap)
(Sumber: dokumentasi penulis)

Fitur dari aplikasi adalah antarmuka berisi fungsionalitas tertentu yang disajikan kepada pengguna pada satu waktu. Kelasnya memiliki atribut state untuk membedakan fitur mana yang sedang menjadi sorotan saat ini. Selain itu, terdapat sebuah *method* dengan keyword virtual (*pure virtual*) menjadikan “Feature” sebagai kelas abstrak.

Kelas “Button” adalah turunan dari “Feature”, mewarisi *method* printDisplay() untuk mencetak tampilan ke layar sesuai dengan state-nya, serta mengimplementasikan *method* showTour(), untuk mencetak kegunaan dari dirinya.

C. Barang yang Dijual

```

main.cpp Graph.hpp Things.hpp Game.hpp Feature.hpp Button.hpp
1 #ifndef THINGS_HPP
2 #define THINGS_HPP
3
4 #include <iostream>
5 #include <cstring>
6 using namespace std;
7
8 class Things {
9     private:
10        int ID;
11        int price;
12        char name[10];
13    public:
14        Things();
15        Things(int ID, int price, char name[]);
16        int getID();
17        int getPrice();
18        char* getName();
19        Things& operator= (const Things&);
20    };
21
22 #endif

```

Gambar 14 Struktur Kelas Barang dalam File “Things.hpp”
(Sumber: dokumentasi penulis)

D. Kelas Pengontrol

```

main.cpp Graph.hpp Things.hpp Game.hpp Feature.hpp Button.hpp
1 #ifndef GAME_HPP
2 #define GAME_HPP
3
4 #include <cstring>
5 #include "Feature.hpp"
6 #include "Button.hpp"
7 #include "Things.hpp"
8 #include "Graph.hpp"
9
10 class Game {
11     private:
12        int money;
13        int change;
14        vector<Things> inventory;
15        Things buyable[6];
16
17        vector<Button*> list_button;
18        vector<Things> cart;
19        int priceCart = 0;
20        Graph app_structure;
21        int curr_state;
22
23    public:
24        Game(vector<Button*> list_button);
25
26        Graph getGraph();
27        vector<Button*> getListButton();
28        int getMoney();
29        vector<Things> getInventory();
30        int getCurrentState();
31        void setCurrentState(int state);
32
33        int countTotalPrice();
34        void cartToInventory();
35
36        void doBFS();
37        void doDFS();
38        void inputCommand();
39    };
40
41 #endif

```

Gambar 15 Struktur Kelas Pengontrol dalam “Game.hpp”
(Sumber: dokumentasi penulis)

Senarai tambahan vector<Button*> list_button, isinya harus dijaga agar memiliki keterurutan yang sesuai dengan indeks senarai ketetangaan graf. Graf sendiri disimpan dalam sebuah variabel yang diberi nama “app_structure”.

Selain mengontrol keuangan dan inventarisasi, kelas ini memiliki *method* doBFS() dan doDFS() yang memanggil *method* “BFS” dan “DFS” dengan s=0 pada atribut grafnya.

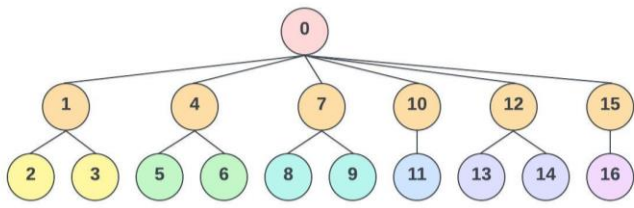
IV. EKSPERIMEN

Setelah aplikasi belanja sederhana berbasis *command-line interface* selesai dibangun, eksperimen kecil diperuntukkan bagi 10 orang yang semuanya berasal dari semester 4 jurusan teknik informatika. 5 orang di antaranya berasal dari ITB, dan 5 orang berasal dari institusi lain, berikut adalah daftarnya:

Tabel 3 Daftar Sampel Mahasiswa

Nama	Institusi	Nama	Institusi
Haziq Abiyu Mahdy	ITB	Aurelius Marvel I.	Binus
Sulthan Dzaky A.	ITB	Joan Arc	UNPAR
Michael Leon Putra W.	ITB	Thomas Teguh L.	ITHB
Jericho Russel S.	ITB	Samuel Oliver L.	ITHB
Nigel Sahl	ITB	Dave Russell	Binus

Di bawah ini adalah struktur graf tak-berarah yang diujikan. Pada perintah `addEdge(u, v, bi)` untuk graf, “bi” diberi nilai *true*. Ini berarti semua sisi yang ditambahkan tidak berarah. Jika “bi” diisi *false*, maka sisi yang ditambahkan mengarah dari simpul *u* ke simpul *v*. Setiap fitur/tombol dimasukkan ke dalam `list_button` secara berurutan agar berada tepat pada posisi yang semestinya (indeksnya mengikuti angka pada gambar). Ketika penelusuran menemukan angka “5”, program dapat langsung melihat `list_button[5]` dan mengeksekusi *method*-nya.



Gambar 16 Struktur Graf Aplikasi

(Sumber: dokumentasi penulis [dibuat dengan Lucidchart])

Percobaan dimulai dengan memberi tahu relawan bahwa aplikasi sederhana yang akan dihadapinya adalah aplikasi belanja ber-*puzzle*. Setelah itu relawan mengikuti *self-paced sequential tour* menggunakan salah satu algoritme. Relawan sebagai pengguna baru, dibebaskan untuk maju perlahan-lahan dalam tur, atau langsung melompati beberapa langkah.

Setelah semua fitur selesai dijelaskan, pengguna diberi tahu tujuannya, yaitu harus memiliki saldo sebesar 21.000 dengan minimal 3 barang sudah dibeli. Selain itu, pengguna juga bisa memeriksa kondisi keuangan, keranjang belanja, kembalian, dan inventaris, dengan cara memberi masukan “STATUS”. Jika diperlukan, pengguna diinformasikan bahwa aplikasi dapat dioperasikan dengan cara mengetikkan teks tertentu ke terminal lalu menekan tombol “Enter”, bukan mengklik tombol.

Penulis kemudian meminta pengguna melakukan beberapa langkah awal (tanpa menyebutkan bagaimana caranya) agar nominal 21.000 benar-benar dapat tercapai dari nominal awal. Lalu, pengguna harus melanjutkan sendiri. Selama pengguna berusaha mencapai tujuan, *guided tour* dari penulis tetap disediakan. Penulis menjawab pertanyaan yang terkait dengan masalah teknis aplikasi/program, tetapi tidak menjawab perihal bagaimana mencapai suatu fitur di aplikasi. Pengguna juga diizinkan untuk melihat kembali tur yang disediakan di awal.

Bingkai-bingkai berikut sekilas memperlihatkan perbedaan yang timbul akibat penggunaan algoritme BFS dan DFS. Ada 7 jenis tampilan yang berbeda. Untuk masing-masing tampilan, disertakan pula nomor yang cocok dengan **Gambar 16**.

Tur-Produk BFS	Tur-Produk DFS
Nomor: 0, 1, 4, 7, 10, 12, 15	Nomor: 0, 1
<p>Layar Utama</p> <p>Selamat datang, pelanggan yang kami hormati. Berikut adalah langkah demi langkah menggunakan aplikasi ini: Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'REN' untuk mencari barang. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'YANN' untuk klaim kembalian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'EAT' untuk konfirmasi pembelian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk melihat nomor barang, Anda dapat mengetik angka di bawah 'REN'. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk isi ulang saldo, Anda dapat mengetik angka di bawah 'YANN'. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk melihat kode pembelian, Anda dapat mengetik angka di bawah 'EAT'. Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar Utama</p> <p>Selamat datang, pelanggan yang kami hormati. Berikut adalah langkah demi langkah menggunakan aplikasi ini: Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'REN' untuk mencari barang. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'YANN' untuk klaim kembalian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'EAT' untuk konfirmasi pembelian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk melihat nomor barang, Anda dapat mengetik angka di bawah 'REN'. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk isi ulang saldo, Anda dapat mengetik angka di bawah 'YANN'. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Untuk melihat kode pembelian, Anda dapat mengetik angka di bawah 'EAT'. Tekan 'ENTER' jika Anda sudah mengerti...</p>
Nomor: 2, 3	Nomor: 2, 3
<p>Layar REN</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input nomor barang untuk menambahkannya ke keranjang Anda. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar REN</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input nomor barang untuk menambahkannya ke keranjang Anda. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>
Nomor: 2, 3	Nomor: 4
<p>Layar REN</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input nomor barang untuk menambahkannya ke keranjang Anda. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar Utama</p> <p>Anda dapat memberi input 'YANN' untuk klaim kembalian. Tekan 'ENTER' jika Anda sudah mengerti...</p>
Nomor: 5, 6	Nomor: 5, 6
<p>Layar YANN</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input '1' untuk mendapatkan kembalian.</p> <p>Anda dapat memberi input '2' untuk menyubangkan kembalian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar YANN</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input '1' untuk mendapatkan kembalian.</p> <p>Anda dapat memberi input '2' untuk menyubangkan kembalian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>
Nomor: 8, 9	Nomor: 7
<p>Layar EAT</p> <p>Pada halaman ini,</p> <p>Anda dapat memberi input '1' untuk konfirmasi pembelian.</p> <p>Anda dapat memberi input '2' untuk membatalkan pembelian. Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama. Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar Utama</p> <p>Anda dapat memberi input 'EAT' untuk konfirmasi pembelian. Tekan 'ENTER' jika Anda sudah mengerti...</p>

V. ANALISIS

Eksperimen yang dilaksanakan menghasilkan data berupa waktu yang dibutuhkan pengguna memainkan aplikasi untuk menyelesaikan tujuan yang diminta. Berikut adalah daftarnya:

Tabel 4 Hasil Eksperimen

Nama	Institusi	Algoritme	Durasi	Detik
Haziq Abiyuu Mahdy	ITB	BFS	11 menit 35 detik	695
Sulthan Dzaky Alvaro	ITB	BFS	11 menit 1 detik	661
Michael Leon Putra W.	ITB	BFS	14 menit 19 detik	859
Jericho Russel Sebastian	ITB	DFS	7 menit 54 detik	474
Nigel Sahl	ITB	DFS	13 menit 1 detik	781
Aurelius Marvel I.	Binus	DFS	19 menit 28 detik	1168
Joan Arc	UNPAR	DFS	13 menit 37 detik	817
Thomas Teguh Laksana	ITHB	BFS	11 menit 49 detik	709
Samuel Oliver L.	ITHB	BFS	11 menit 45 detik	705
Dave Russell	Binus	DFS	13 menit 35 detik	815

Dari data tersebut, dihitung nilai rata-rata durasi pengguna menamatkan *puzzle* dengan BFS sebagai tipe tutorial adalah 725,8 detik atau 12,0967 menit. Berbeda dengan rata-rata dari tipe lawannya, DFS, yang berada di angka 811 detik atau 13,5167 menit. Di sini kita bisa mengatakan dengan jelas bahwa algoritme BFS lebih baik daripada DFS dalam memberikan pemahaman kepada pengguna.

Setidaknya, ada tiga buah hal yang dapat diperhatikan. Pertama, algoritme BFS memberikan tur yang lebih singkat, karena setiap “tombol” yang ada pada layar yang sama dijelaskan terlebih dahulu. Pada tur DFS, pengguna dipaksa melihat tampilan yang sama berulang kali. Ketika kegunaan sebuah tombol di halaman “H” diterangkan, program berganti tampilan untuk menerangkan apa yang terjadi saat tombol tersebut ditekan, sebelum kembali dan menerangkan tombol lain di halaman “H”. Selain menyebabkan kebingungan, duplikasi semacam ini juga berdampak pada saat pengguna ingin melihat ulang tur yang diberikan. Proses *scrolling* ke atas dan ke bawah tentu menjadi lebih panjang.

Hal kedua yang menjadi perhatian, pengguna tidak mampu mengadaptasikan diri dengan cepat kepada sebuah aplikasi baru, apalagi berbasis *command-line* dan dibungkus menjadi sebuah *puzzle*. Beberapa orang terbukti bergantung pada tur secara ekstrim, hingga berujung bolak-balik melakukan pergeseran layar demi mencari petunjuk yang diinginkan.

Terakhir, kebanyakan dari relawan malah meremehkan tur, yang disangka hanya sebatas kata sambutan atau dapat diakses lagi dengan cepat. Padahal, penjelasan di dalamnya sangat dibutuhkan untuk penyelesaian *puzzle* secara efisien. Relawan yang berhasil dalam waktu tersingkat, yaitu 7 menit 54 detik, menciptakan anomali yang lumayan dahsyat. Meski berhadapan dengan algoritme DFS, penulis memerhatikan ketelitiannya menghafal dua-tiga bagian esensial sepanjang tur. Bahkan relawan tersebut mencatat harga-harga barang agar tidak perlu kembali ke halaman terkait. Hal ini menjadi dasar pertimbangan, bahwa tur yang tersedia tidak mengandeng kesalahan, tetapi keseriusan pengunjal yang kurang ditonjolkan.

Secara algoritme, masing-masing BFS dan DFS memiliki kompleksitas $O(|V|)$, atau $O(17)$ karena keseluruhan struktur graf

<p align="center">Nomor: 11</p>	<p align="center">Nomor: 8, 9</p>
<p>Layar 112</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE [INFORMASI] [HARGA] SHEKORONG = (5*4)-15 30000 RESHA = (10/4)+(1/2) 17500 GALIBER = (7*3)-8 24000 SARIKEN = (4*3)/16 10000 ZECTER = (11-12)*(-1) 45000 KUNAI = y (2y+3)-13 12500 </pre> <p>Pada halaman ini, Anda dapat memberi input '1' untuk konfirmasi pembelian Anda dapat memberi input '2' untuk membatalkan pembelian Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar 81</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE 1. KONFIRMASI 2. BATALKAN </pre> <p>Pada halaman ini, Anda dapat memberi input '1' untuk konfirmasi pembelian Anda dapat memberi input '2' untuk membatalkan pembelian Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>
<p align="center">Nomor: 13, 14</p>	<p align="center">Nomor: 10</p>
<p>Layar 24</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE 1. 30000 2. 50000 </pre> <p>Pada halaman ini, Anda dapat memberi input '1' untuk isi ulang saldo sebesar 30000 Anda dapat memberi input '2' untuk isi ulang saldo sebesar 50000 Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar Utama</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE </pre> <p>Untuk melihat kode pembelian, Anda dapat mengklik angka di bawah 'HENT' Tekan 'ENTER' jika Anda sudah mengerti...</p>
	<p align="center">Nomor: 11</p>
	<p>Layar 112</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE [INFORMASI] [HARGA] SHEKORONG = (5*4)-15 30000 RESHA = (10/4)+(1/2) 17500 GALIBER = (7*3)-8 24000 SARIKEN = (4*3)/16 10000 ZECTER = (11-12)*(-1) 45000 KUNAI = y (2y+3)-13 12500 </pre> <p>Pada halaman ini, Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>
<p align="center">Nomor: 16</p>	<p align="center">Nomor: 12</p>
<p>Layar 8</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE Kode Pembelian Anda: SC00 </pre> <p>Pada halaman ini, Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>	<p>Layar Utama</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE </pre> <p>Untuk isi ulang saldo, Anda dapat mengklik angka di bawah 'YMM' Tekan 'ENTER' jika Anda sudah mengerti...</p>
	<p align="center">Nomor: 13, 14</p>
	<p>Layar 24</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE 1. 30000 2. 50000 </pre> <p>Pada halaman ini, Anda dapat memberi input '1' untuk isi ulang saldo sebesar 30000 Anda dapat memberi input '2' untuk isi ulang saldo sebesar 50000 Tekan 'ENTER' jika Anda sudah mengerti...</p> <p>Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>
	<p align="center">Nomor: 15</p>
	<p>Layar Utama</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE </pre> <p>Untuk melihat kode pembelian, Anda dapat mengklik angka di bawah 'EAT' Tekan 'ENTER' jika Anda sudah mengerti...</p>
	<p align="center">Nomor: 16</p>
	<p>Layar 8</p> <pre> Ryan Samuel Chandra (13521140) BFS-DFS ALGORITHM STRATEGIES PUZZLE Kode Pembelian Anda: SC00 </pre> <p>Pada halaman ini, Anda dapat memberi input 'BACK' untuk kembali ke halaman utama Tekan 'ENTER' jika Anda sudah mengerti...</p>

memiliki 17 simpul. Tidak berbeda dengan algoritme *brute force* yang akan menjalankan tur dengan kompleksitas $O(n)$, dengan n adalah banyaknya penjelasan yang ditampilkan. Meskipun begitu, konstruksi dari BFS dan DFS jauh lebih rapi. Dengan graf (walau pemanfaatannya belum maksimal), penambahan fitur baru yang memiliki *method showTour()* tidak menyebabkan perubahan apa pun pada kode tur. Penambahan hanya dilakukan pada graf dengan fungsi *addEdge()* untuk memposisikan fitur baru di tempat yang benar.

Dari segi alokasi memori, kekalahan algoritme BFS dan DFS terhadap *brute force* harus diakui. Terdapat ruang cukup besar yang harus disediakan untuk menyimpan graf. Bagaimanapun, tidak bisa dipungkiri bahwa ini adalah sebuah *trade-off* yang baik, antara ruang dan kemangkusan struktur.

Dilihat dari rata-rata durasi, tur produk dengan penelusuran secara BFS diasumsikan menang untuk pengujian sampel yang penulis buat. Alurnya lebih sederhana untuk dipahami, dan meminimalisir repetisi selama tur sekaligus menghemat waktu.

Untuk memeriksa secara matematis, apakah rata-rata kedua populasi dapat dianggap sama, bisa digunakan uji hipotesis dengan $H_0: \mu_1 - \mu_2 = 0$ dan $H_1: \mu_1 - \mu_2 \neq 0$. Pertama-tama, dihitung variansi dari sampel. yaitu $s_1^2 = \frac{23604,8}{5-1} = 5901,2$ dan $s_2^2 = \frac{241970}{5-1} = 60492,5$.

Maka, nilai statistik $t' = \frac{(725,8-811)-0}{\sqrt{\frac{5901,2}{5} + \frac{60492,5}{5}}} \approx -0.739$ dan

derajat kebebasan $v = \frac{(\frac{5901,2}{5} + \frac{60492,5}{5})^2}{\frac{(5901,2/5)^2}{5-1} + \frac{(60492,5/5)^2}{5-1}} \approx 4,773 \approx 5$.

Jika kita menggunakan interval kepercayaan 90%, berarti peluang eror $\alpha = 1 - 0,9 = 0,1$. Mengacu ke tabel distribusi-T, $t_{\frac{\alpha}{2}}$ bernilai kurang lebih 2,015. Dengan demikian, karena pernyataan $-t_{\frac{\alpha}{2}} < t' < t_{\frac{\alpha}{2}}$ adalah benar, H_0 diterima, sehingga dapat disimpulkan bahwa rata-rata kedua populasi bernilai sama.

VI. KESIMPULAN

Tur-produk (*product-tour*) adalah tampilan yang digunakan untuk memandu pengguna secara virtual mengenai tata cara memakai fitur yang diperlukan dari suatu produk (aplikasi). Ada berbagai macam cara menyajikan tur produk, misalnya *self-paced*, *guided*, dan *onboarding*. Kebanyakan tur produk dibuat dengan *brute force*, yaitu mencetak satu per satu *guide* ke layar pengguna. Meskipun dapat bekerja, konstruksi dari algoritme tersebut sangat tidak elegan.

Algoritme BFS (*Breadth-First Search*) dan DFS (*Depth-First Search*) merupakan alternatif yang dapat digunakan untuk mengatasi masalah tersebut. Program dibuat dalam paradigma *object-oriented*, sehingga masing-masing fitur/tombol memiliki *method showTour()* untuk menampilkan kegunaan dirinya sendiri. Setiap fitur/tombol tersebut diposisikan secara tepat dalam sebuah graf yang akan ditelusuri dari simpul awal.

Pada percobaan kali ini, BFS lebih unggul karena rata-rata durasi yang diperlukan sampel pengguna untuk merampungkan

puzzle jauh lebih singkat. Tetapi, uji hipotesis dengan interval kepercayaan 90%, membuktikan bahwa rata-rata populasi kemungkinan besar sama. Maka, kesimpulan final yang ditarik adalah, algoritme BFS dan DFS sama-sama berpotensi menjadi algoritme yang efektif dalam hal memberikan pemahaman mengenai produk aplikasi melalui sebuah tur produk.

VII. PROSPEK PENGEMBANGAN

Terdapat beberapa poin dari penelitian penulis yang dapat diperbaiki lagi. Penerapan *object-oriented programming* dalam aplikasi belanja sederhana pada makalah ini, masih jauh dari *best practice*. Seperti yang sudah disampaikan, graf sebaiknya langsung diisi dengan objek tombol, bukan sekadar angka yang mengacu pada objek tombol. Selanjutnya, penanganan berbagai kejadian dalam aplikasi juga masih menggunakan *brute force*. Dalam file "Game.cpp", terlihat bahwa masih terdapat banyak sekali blok *if-else* untuk memproses masukan pengguna. Perbaikan yang diusulkan adalah dengan menambah *method execute()* pada setiap objek *button*, sehingga konsepnya seperti tombol sungguhan, yang ketika ditekan memanggil *method* tersebut untuk menjalankan aksi.

Penulis mendapatkan masukan dari relawan: kondisi yang wajib dipenuhi seharusnya disampaikan sebelum tur produk dimulai. Dengan demikian, selama tur berlangsung, pengguna lebih terarah untuk mengingat-ingat langkah tertentu yang dirasa diperlukan. Selain itu, tur dinilai masih terlalu membingungkan, karena para pengguna diminta untuk mempelajari segala fitur secara bersamaan. Desain ini memang belum sejalan dengan ketentuan tur produk sukses, yaitu harus muncul pada saat yang tepat dan dapat ditunda. Namun, berhubung topik dari makalah adalah "tur produk sekuensial", permasalahan tadi memang tidak bisa dihindari. Perbaikan yang diusulkan adalah menyela tur sebelum menuju aras yang lebih dalam, memberikan kesempatan bagi pengguna melakukan apa yang baru saja disaksikan, lalu menyambung kembali tur.

Menelaah lebih dalam, algoritme DFS digabungkan dengan *backtracking*, dapat menjadi *tool* untuk membuat tur produk yang lebih canggih, yaitu *preference-based*. Tur jenis ini memungkinkan pengguna mendapatkan panduan terurut yang sesuai dengan keinginan. Misalnya, pengguna berada di halaman isi saldo, dan ingin melihat barang yang dijual, tur dapat menunjukkan caranya, mulai dari kembali ke halaman utama, dilanjutkan dengan memilih fitur pencarian barang.

PRANALA

- Berikut adalah pranala menuju video Youtube: <https://youtu.be/qcT9hn2vGc>
- Berikut adalah pranala Github yang berisi kode program: https://github.com/RyanSC06/BFS_DFS-Product-Tour.git

UCAPAN TERIMA KASIH

Pertama-tama, penulis memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa, atas berkat dan rahmat yang begitu melimpah dalam hidup penulis, sehingga akhirnya makalah ini dapat selesai tepat waktu. Penulis juga mengucapkan terima kasih kepada kedua orang tua yang senantiasa mendukung, memberikan motivasi selama pengerjaan makalah ini. Tidak lupa teruntuk teman-teman seperjuangan yang terkasih, sahabat, yang sudah meluangkan waktunya untuk berkontribusi dalam pengambilan data percobaan.

Penulis secara khusus berterima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc., selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma di kelas K2 tahun ajaran 2022/2023. Beliau yang selama satu semester ini telah membagikan ilmu dan pengalamannya, serta kesempatan bagi penulis untuk belajar melalui penulisan makalah ini.

DAFTAR REFERENSI

- [1] Yıldırım, H. 26 Juli 2022. *Introducing: Product Tours – The Definitive Guide*. (Diakses dari <https://userguiding.com/blog/product-tour/> pada tanggal 2 April 2023)
- [2] Levitin, A. 2012. *Introduction to the Design & Analysis of Algorithms, 3rd Edition*. United States of America: Addison-Wesley.
- [3] Agrawal, P. 2023. *Product Tours 101: Guidelines, Inspirations, and Tools for 2023*. (Diakses dari <https://www.chameleon.io/blog/how-to-build-effective-product-tours> tanggal 19 Mei 2023)
- [4] Anonim. Tanpa tahun. *What is a Product Tour? Types of Product Tours? 5 examples of Product Tours*. (Diakses tanggal 19 Mei 2023 dari <https://activedemo.io/what-is-a-product-tour-types-examples/>)
- [5] Munir, Rinaldi. 2020. *Graf (Bag.1) (Bahan Kuliah IF2120 Matematika Diskrit STEI-ITB)*. (Diakses pada tanggal 20 Mei 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)
- [6] Munir, Rinaldi. 2020. *Graf (Bag.2) (Bahan Kuliah IF2120 Matematika Diskrit STEI-ITB)*. (Diakses pada tanggal 20 Mei 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf>)
- [7] Munir, Rinaldi dan Nur Ulfa Maulidevi. *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. (Diakses pada tanggal 20 Mei 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>)
- [8] Tim Pengajar IF2210. 27 Januari 2023. *Konsep OOP*. (Salindia Kuliah IF2210 Pemrograman Berorientasi Objek, STEI-ITB)
- [9] Tim Pengajar IF2210. 18 Februari 2021. *Konsep Inheritance*. (Salindia Kuliah IF2210 Pemrograman Berorientasi Objek, STEI-ITB)
- [10] Santoso, Judhi; Harlili dan Dwi H. Widyantoro. Tanpa tahun. *Distribusi Sampel*. (Salindia Kuliah IF2220 Probabilitas dan Statistika, STEI-ITB)
- [11] Santoso, Judhi; Harlili dan Dwi H. Widyantoro. Tanpa tahun. *Pengujian Hipotesis Statistik*. (Salindia Kuliah IF2220 Probabilitas dan Statistika, STEI-ITB)

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, 22 Mei 2023



Ryan Samuel Chandra
(13521140)