Spesifikasi Komputer Rakitan Berdasarkan Kebutuhan dan Anggaran Menggunakan Algoritma Backtracking

Rahmat Haryadi Kiswanto
Teknik Informatika
STIMIK Sepuluh Nopember Jayapura
Jayapura, Indonesia
e-mail: kissonetwo74@gmail.com

Diajukan: 6 Februari 2020; Direvisi: 1 April 2020; Diterima: 27 April 2020

Abstrak

Pemilihan komputer yang baik adalah menyesuaikan dengan kebutuhan dan ketersediaan anggaran karena hal ini berhubungan dengan spesifikasi perangkat keras di dalamnya. Masih banyak masyarakat awam tidak mempunyai pengetahuan komponen komputer rakitan. Hal ini bias menjadi penyebab kurang optimalnya dalam memilih spesifikasi komputer rakitan berdasarkan kebutuhan dan ketersediaan anggaran. Sehingga perlu adanya suatu sistem yang dapat membantu masyarakat dalam memberikan informasi spesifikasi (komponen-komponen) komputer rakitan yang menyesuaikan dengan kebutuhan dan anggaran yang dimiliki calon pembeli. Pengembangan aplikasi dilakukan dengan mengamati langsung proses pembelian Komputer rakitan yang dilakukan di toko-toko komputer. Selanjutnya melakukan pengumpulan data perangkat keras komputer yang akan dijadikan spesifikasi komputer rakitan. Memilih Algoritma Backtracking dengan teknik Depth First Search dengan fungsi pembatas harga dalam proses penyusunan spesifikasi perangkat keras komputer. Pengujian dilakukan pada 4 komponen hardware untuk Paket Office dengan batas anggaran Rp 5.000.000, dan memperoleh output harga Rp 4.978.000 dengan performa indeks 5482 dan pada Paket Grafik/Game, pengujian dilakukan pada 5 komponen hardware dengan batas anggaran Rp 8.000.000 dan memperoleh 2 pilihan output harga Rp 7.973.000 dan Rp 8.000.000 dengan masing-masing indeks 12749 dan 12373. Kedua pengujian menghasilkan keluaran kombinasi hardware yang tidak melebihi batas anggaran yang dimasukkan.

Kata kunci: *Komputer, Backtracking, Depth First Search.*

Abstract

The selection of a computer is to adjust the need and availability of budget because this is related to the hardware specification. Many ordinary people do not have knowledge about computer components, It will not lead to optimal choices in choosing computer based on need and budget. It is necessary to have system that can help them to provide information of specification of assembling computer that adjust to the needs and budgets. The development of application was carried out by directly observing the process of purchasing at computer shops. Next, collected the data of hardware for the databases of system. Chose the Backtracking Algorithm that ran a depth-first search technique with the boundary function is the budget limit. The testing was performed for 4 hardware component that arranges of the Office Package with a budget limit of Rp 5,000,000, and output for cost is Rp 4,978,000 with index performance was 5482, and the Gaming Package was performed for 5 hardware components with a budget limit of Rp 8,000,000 and the output for cost were Rp 7,973,000 and Rp 8,000,000 with index was both 12749 and 12373. The testing has shown that the output of the application was not over a budget limit.

Keywords*: Computer, Backtracking, Depth First Search.*

1. Pendahuluan

Komputer rakitan saat ini masih menjadi pilihan bagi masyarakat kelas menengah ke bawah dalam membantu mereka dalam aktivitas sehari-hari seperti pembuatan laporan pekerjaan, tugas-tugas akademik, multimedia (music, film, desain), dan lain-lain. Banyak dari perusahaan komputer ternama memproduksi komputer *built up* dengan berbagai tipe dan spesifikasi dengan maksud untuk menyesuaikan dengan kebutuhan pengguna. Komputer *built up* yang diproduksi oleh produsen ternama dapat dikatakan lebih

mahal jika dibandingkan dengan komputer rakitan dengan spesifikasi yang sama. Di samping itu juga banyak konsumen yang menginginkan spesifikasi komputer yang sesuai dengan kebutuhan dan keinginannya. Dengan demikian alternatif komputer rakitan, dapat dijadikan pilihan untuk menentukan sendiri spesifikasi dari komputer yang ingin dibeli dengan harga lebih murah dibandingkan komputer *built-up* dengan spesifikasi yang hampir sama atau bahkan sama.

Untuk merakit komputer dari komponen-komponen utamanya seperti *mainboard*, CPU, *harddisk*, RAM, dan VGA *card*, membutuhkan pengetahuan terhadap kompatibilitas antar komponen, performa dan harga setiap komponen. Permasalahan yang terjadi tidak semua masyarakat awam mengetahui tentang komponen komputer rakitan yang baik. Selain dari itu juga banyaknya produk yang dikeluarkan oleh produsen-produsen *hardware*, sehingga membuat para konsumen mengalami kesulitan untuk memilih produk *hardware* yang sesuai dengan kebutuhan dan juga sesuai dengan anggaran. Hal ini membuat masyarakat awam lebih banyak langsung datang ke toko-toko komputer untuk menanyakan spesifikasi komputer yang sesuai dengan kebutuhannya, dan antara satu toko komputer dengan toko komputer yang lain belum tentu sama spesifikasi yang diberikan untuk kebutuhan yang sama dengan harga yang bervariasi di bawah anggaran ataupun di atas anggaran.

Dari permasalahan yang terjadi maka penelitian ini dilakukan untuk mengembangkan suatu sistem penyusunan spesifikasi komputer berdasarkan kebutuhan dan anggaran yang dimiliki konsumen. Sistem yang dibangun menggunakan Algoritma Backtracking yang merupakan algoritma *Depth First Search* (DFS) dengan tambahan fungsi pembatas, dengan tujuan penyusunan spesifikasi komputer tidak akan melebihi batas maksimal anggaran yang dimiliki konsumen.

Penelitian tentang penentuan spesifikasi komputer oleh [1], sistem dapat melakukan pencarian dengan menggunakan metode algoritma genetika untuk mendapatkan solusi yang optimal dengan *output* berupa spesifikasi PC, di mana total harga yang didapat mendekati dengan kemampuan finansial yang telah di-*input*-kan oleh konsumen. Penelitian yang dilakukan oleh [2] tentang penentuan spesifikasi komputer rakitan berdasarkan kebutuhan pemakai dan harga, penelitian ini menerapkan metode basis data *fuzzy* yang mengubah data numerik menjadi bahasa sehari-hari. Penelitian kedua ini menghasilkan suatu sistem yang dapat mempercepat proses penentuan spesifikasi yang digunakan dan menghemat anggaran yang dikeluarkan..

Penelitian terhadap penerapan terhadap metode DFS oleh [3], tentang penerapan metode DFS pada pencarian rute bus kota berbasis *web mobile* di Solo. Penelitian ini menghasilkan bahwa metode DFS dapat menampilkan semua solusi yang dapat terjadi sesuai dengan keinginan pemakai, baik dari segi jarak atau biaya yang paling optimal. Penelitian terhadap penerapan Algoritma Backtracking dilakukan oleh [4], yaitu perancangan aplikasi *game* labirin dengan menggunakan Algoritma Backtracking. Pada penelitian ini proses mencari jalan keluar permainan labirin dengan cara membentuk lintasan dari akar ke daun dan simpul-simpul yang sudah dilahirkan dinamakan simpul hidup, jika lintasan yang diperluas yang sedang dibentuk tidak mengarah ke solusi, maka simpul tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*). Simpul yang sudah mati ini tidak akan diperluas lagi. Kemudian penelitian oleh [5], implementasi Algoritma Backtracking pada aplikasi permainan tradisional dam-daman berbasis Java desktop. Pada penelitian ini Algoritma Backtracking adalah algoritma yang berbasis pada *Depth First Search* (DFS) untuk mencari solusi persoalan secara lebih efisien. Algoritma Backtracking tidak memeriksa semua kemungkinan solusi yang ada, hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan, akibatnya waktu pencarian dapat dihemat.

2. Metode Penelitian

2.1. Tahapan Penelitian

Penelitian ini dilakukan dengan melalui beberapa tahapan yang dijabarkan sebagai berikut:

1. Studi Kelayakan
   Pada tahap dilakukan studi pustaka dari hasil-hasil penelitian pada jurnal-jurnal ataupun artikel-artikel ilmiah yang dipublikasikan, khususnya yang relevan dengan permasalahan pemilihan spesifikasi komputer dan metode yang digunakan dalam penelitian ini. Selanjutnya dilakukan observasi terhadap kondisi di lapangan dengan mengunjungi beberapa toko komputer yang ada di Jayapura, terutama terkait dengan jual beli komputer rakitan. Dengan melihat proses interaksi antara *user* dan penjual dalam menentukan spesifikasi dan harga komputer yang akan dirakit.
2. Desain Sistem
   Melakukan perancangan sistem menggunakan UML (*use case diagram, class diagram, dan sequence diagram)*, Merancang skema *database* menggunakan *Entity Relationship Diagram* (ERD) dan merancang antarmuka pengguna untuk interaksi pengguna dengan sistem.

3. Implementasi
   ERD ditransformasikan ke dalam DBMS MySQL, mengkodekan hasil rancangan sistem dengan menerapkan Algoritma Backtracking di dalamnya menggunakan bahasa pemrograman Java.
4. *Testing*
   Pengujian dilakukan secara *white-box* untuk menguji kompleksitas dari program dan *black-box* untuk menguji fungsionalitas sistem berjalan dengan baik.

2.2. Teknik Pengumpulan Data

Data utama yang dibutuhkan adalah data komponen perangkat keras yang diambil dari beberapa *website* yang menyediakan harga komponen dan performa tiap komponen *hardware*. Data yang diperoleh ini kemudian dimasukkan ke dalam basis data yang digunakan sebagai data utama yang kemudian diolah dengan menggunakan Algoritma Backtracking untuk mendapatkan spesifikasi yang sesuai dengan anggaran dan kebutuhan konsumen.

2.3. Instrumen Penelitian

Dalam penelitian ini menggunakan perangkat keras dan perangkat lunak untuk mengembangkan sistem dapat dilihat pada Tabel 1 dan Tabel 2:

Tabel 1. Perangkat keras.

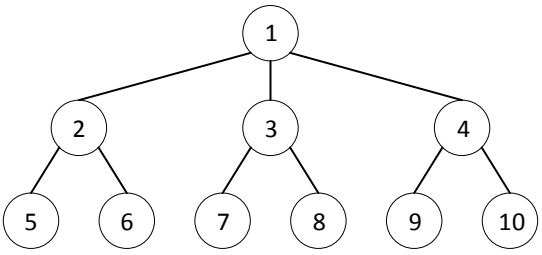
Item	Spesifikasi
Laptop MacBook Air	Processor Core i5, Memori 4 GB, SSD 128 GB
Printer	Epson L300

Tabel 2. Perangkat lunak.

Item	Keterangan
Sistem Operasi	Mac OS High Sierra
Editor Pemrogram	Apache NetBeans IDE
Bahasa Pemrograman Java	JDK 11
Basis data	MySQL

2.4. Algoritma Bactracking

Algoritma DFS merupakan algoritma pencarian dengan menelusuri *node* terdalam pada pohon pencarian. DFS memiliki kebutuhan memori yang sangat sederhana, hanya perlu menyimpan satu jalur dari akar ke simpul daun. Algoritma Backtracking Search merupakan varian dari DFS yang menggunakan sedikit memori. Perbedaannya pada Backtracking semua solusi dibuat dalam bentuk pohon solusi dan akan menelusuri pohon tersebut secara DFS dengan menggunakan fungsi pembatas sampai ditemukan solusi yang layak [6]. Jadi fungsi pembatas ini yang akan memberikan efisiensi pada saat penelusuran *node*. Misalnya pada Gambar 1 penelusuran untuk *node* (1,2,5) tidak akan dilakukan sepenuhnya sampai di *node* 5 maupun ke *node* 6. Jika pada *node* 2 sudah melewati nilai dari fungsi pembatas, maka penelusuran akan langsung berganti ke *node* 1, 3, dan seterusnya sampai solusi yang layak ditemukan atau bisa juga tidak ada solusi.



Gambar 1. Pohon solusi.

3. Hasil dan Pembahasan

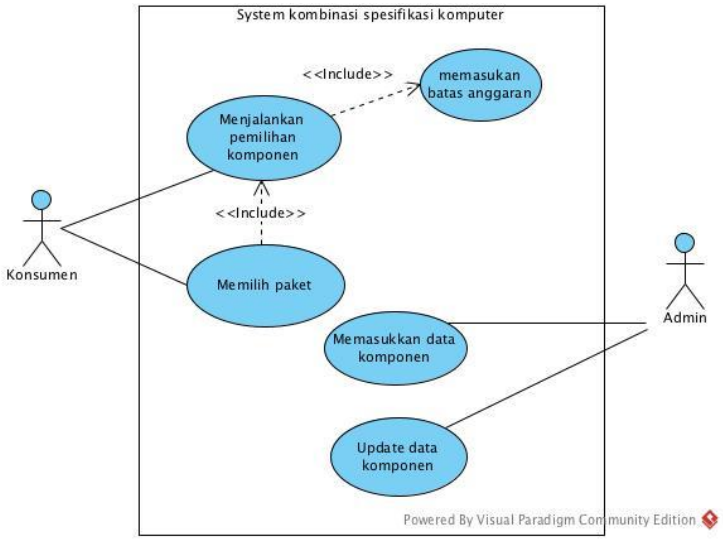
3.1. Desain Sistem

Sistem yang dibuat untuk menangani pencarian spesifikasi komputer yang sesuai dengan kebutuhan dan anggaran konsumen. Dari data-data *hardware* komputer yang ada digunakan untuk

dilakukannya proses kombinasi perangkat keras dengan menggunakan Algoritma Backtracking dengan batasan biaya yang dimiliki oleh konsumen dan perancangan sistem dibuat menggunakan UML.

3.1.1. *Use Case Diagram*

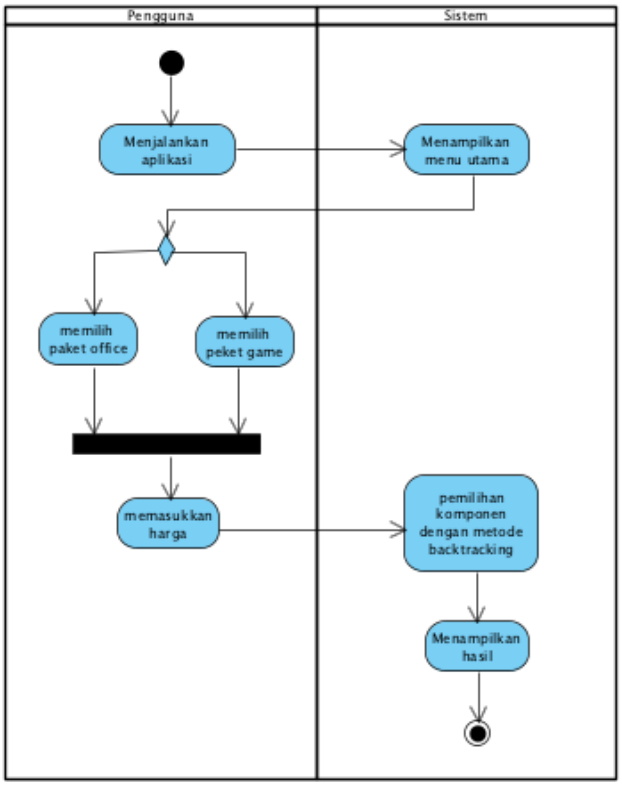
Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case diagram* ditunjukkan pada Gambar 2.



Gambar 2. *Use case diagram*.

3.1.2. *Activity Diagram*

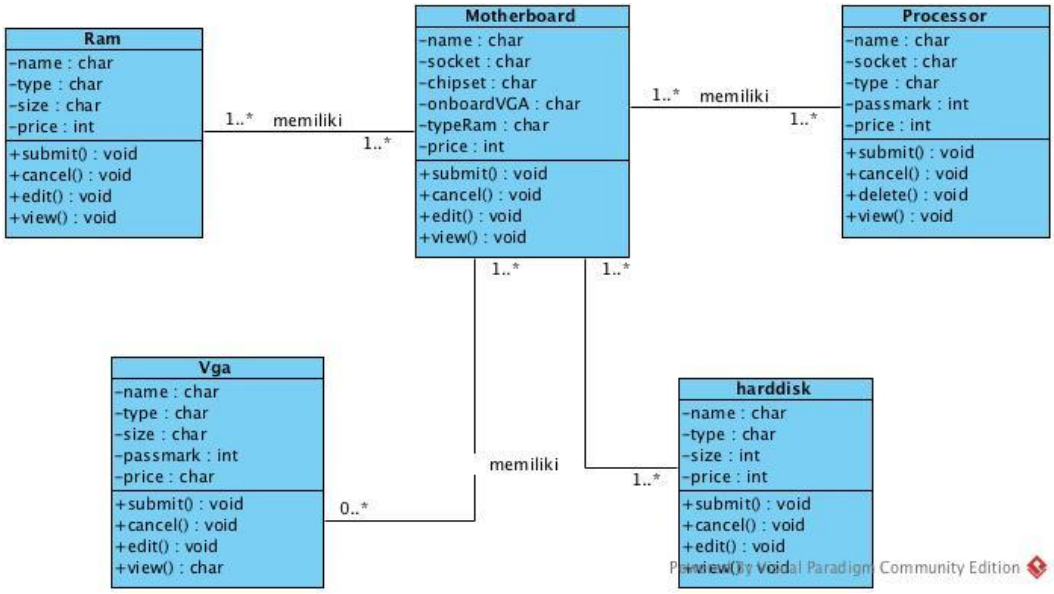
Activity diagram (diagram aktivitas) menggambarkan proses yang terjadi pada pemilihan kebutuhan komputer, seperti tampak pada Gambar 3.



Gambar 3. *Activity diagram*.

3.1.3. *Class Diagram*

Class diagram merupakan sebuah diagram statis, yang menampilkan bentuk statis dari sebuah aplikasi. Mendeskripsikan atribut dan operasi/fungsi dari suatu *class*. Diagram ini menunjukkan sekumpulan *class, interface*, asosiasi dan kolaborasi yang dikenal juga dengan sebuah diagram struktural. Gambar 4 menunjukkan *class diagram* pada sistem yang dibangun.



Gambar 4. *Class diagram*.

3.1.4. *Sequence Diagram*

Sequence diagram menggambarkan perilaku sebagai urutan pesan yang terjadi di antara sekumpulan objek [7][8]. Diagram ini juga dikatakan sebagai *interaction diagram* yang digunakan untuk memodelkan aspek dinamik dari sistem perangkat lunak yang dibangun. Gambar 5 merupakan *sequence diagram* dari sistem yang menunjukkan interaksi antar objek dalam proses pencarian kombinasi komponen komputer.

Gambar 5. *Sequence diagram*.

3.2. Implementasi

Hasil dari perancangan kemudian diimplementasikan ke dalam *database* dan pengodean menggunakan bahasa pemrograman Java untuk mendapatkan tampilan antarmuka yang dapat digunakan oleh admin dan pengguna, di dalam pengodean ini juga membutuhkan API *database* MySQL untuk dapat menghubungkan sistem ke *database* sehingga nantinya *admin* dan *user* dapat berinteraksi dengan sistem dan *database* melalui antarmuka grafis, serta menerapkan algoritma *backtracking* untuk proses penyusunan spesifikasinya.

3.2.1. Implementasi *Database*

Objek-objek yang telah diidentifikasi pada perancangan sistem diimplementasikan ke dalam *database* dengan menggunakan DBMS MySQL. *Database* ini digunakan untuk penyimpanan data *motherboard*, *processor*, kartu grafis, *harddisk*s, dan memori RAM. Tabel 3 – 7, merupakan struktur tabel di dalam *database*.

Tabel 3. Struktur tabel *motherboard*.

No	Nama Kolom	Tipe Data	Size	Keterangan
1	id	int	10	Primary Key
2	nama	varchar	255	
3	socket	varchar	255	
4	chipset	varchar	255	
5	onboardVGA	varchar	255	
6	type RAM	varchar	255	
7	harga	int	10	

Tabel 4. Struktur tabel *processor*.

No	Nama Kolom	Tipe Data	Size	Keterangan
1	id	int	10	Primary Key
2	nama	varchar	255	
3	socket	varchar	255	
4	type	varchar	255	
5	passmark	int	10	
6	harga	int	10	

Tabel 5. Struktur tabel kartu grafis.

No	Nama Kolom	Tipe Data	Size	Keterangan
1	id	int	10	Primary Key
2	nama	varchar	255	
3	type	varchar	255	
4	size	varchar	255	
5	passmark	int	10	
6	harga	int	10	

Tabel 6. Struktur tabel *harddisk*.

No	Nama Kolom	Tipe Data	Size	Keterangan
1	id	int	10	Primary Key
2	nama	varchar	255	
3	type	varchar	255	
4	size	varchar	255	
5	harga	int	11	

Tabel 7. Struktur tabel *memory*.

No	Nama Kolom	Tipe Data	Size	Keterangan
1	id	int	10	Primary Key
2	nama	varchar	255	
3	type	varchar	255	
4	size	varchar	255	
5	harga	int	11	

3.2.2. Implementasi Algoritma Backtracking

Algoritma Backtracking merupakan algoritma DFS yang menambahkan fungsi pembatas, dalam sistem penyusunan spesifikasi perangkat keras komputer ini, yang dilakukan dengan beberapa tahapan:

1. Menyusun Urutan Kombinasi Perangkat Keras.

Perangkat keras yang ada di dalam *database* disusun dengan ketentuan untuk *mainboard* berdasarkan harga tertinggi sampai yang terendah, untuk *processor* dan VGA berdasarkan *passmark* dari yang tertinggi sampai yang terendah, untuk *memory* dan *harddisk* berdasarkan *size* dari yang tertinggi sampai yang terendah. Kemudian dibuat kombinasi dari masing-masing komponen perangkat keras sesuai dengan paket kebutuhan, yaitu:

a. Paket Office mempunyai urutan kombinasi *mainboard*, *processor*, *memory*, dan *harddisk*.

b. Paket Game mempunyai 2 pilihan urutan kombinasi: (*mainboard*, *processor*, VGA, *memory*, *harddisk*) dan (*mainboard*, VGA, *processor*, *memory*, *harddisk*).

2. Proses *Backtracking*

Membangun pohon solusi untuk menyusun spesifikasi perangkat keras yang sesuai dengan *budget* untuk mendapatkan hasil yang optimal tidak melebihi *budget*. Diambil contoh untuk kebutuhan *game* dengan dana Rp 8.000.000, di mana dana ini menjadi fungsi pembatas. Pertama akan dilakukan pengurutan komponen pada tiap tabel perangkat keras, dengan data tiap tabel dapat dilihat pada Tabel 8 - 12:

Tabel 8. *Mainboard*.

Nama	Socket	Chipset	VGA_onboard	Type_RAM	Harga
Asrock Taichi	LGA1151	Intel Z370	yes	DDR4	3785000
Asrock Killer SLI	LGA1151	Intel Z370	yes	DDR4	2485000
Asrock Gaming K4	LGA1151	Intel B250	yes	DDR4	1740000

Tabel 9. *Processor*.

Nama	Socket	Chipset	Passmark	Harga
Intel Core i3-8100 3,6 Ghz Cache 6 mb	LGA1151	Intel	8033	2050000
Intel Core i5-7400 3,0 GHz Cache 6 mb	LGA1151	Intel	7327	3045000
Intel Core 13-6100 3,7 GHz Cache 3 mb	LGA1151	Intel	5482	1750000
Intel Pentium Gold 3,7 GHz Cache 3 mb	LGA1151	Intel	5199	1005000

Tabel 10. VGA.

Nama	Type	Size	Passmark	Harga
Asus AMD RX 570	GDDR5	4 GB	6891	1950000
Nvidia GTX 1050	GDDR 5	2 GB	4719	1200000
Asus Geforde GT 730 Kepler	GDDR5	2 GB	921	950000

Tabel 11. *Memory*.

Nama	Type	Size	Harga
V-gen PC17000	DDR4	4 GB	698000
Kingston PC12800	DDR3	2 GB	323000
V-Gen Rescue PC12800	DDR3	2 GB	275000

Tabel 12. *Harddisk*.

Nama	Type	Size	Harga
Toshiba	SATA	1 TB	625000
WDC-Blue	SATA	500 GB	640000
WDC	SATA	320 GB	240000

Bila dibuat dalam bentuk *n-tuple* untuk masing-masing komponen adalah sebagai berikut:

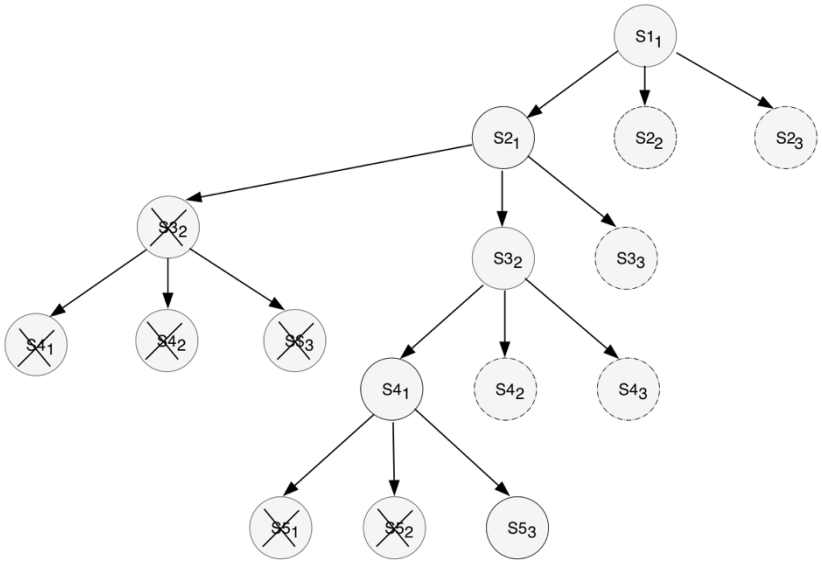
$MB = (MB1, MB2, \ldots, MBn)\ MBi \in S1i$
$PS = (PS1, PS2, \ldots, PSn),\ PSj \in S2j$
$VG = (VG1, VG2, \ldots, VGn),\ VGk \in S3k$
$MM = (MM1, MM2 \ldots, MMn),\ MMl \in S4l$
$HD = (HD1, HD2, \ldots, HDn),\ HDm \in S5m$

Kemudian solusi dinyatakan sebagai $S = (S1_i, S2_j, S3_k, S4_l, S5_m)$. Ruang solusinya didapat dengan mengombinasikan semua kemungkinan dengan melakukan penelusuran secara *DFS* dan hal ini akan memakan waktu cukup lama. Di sinilah metode Backtracking digunakan untuk mengurangi penelusuran secara menyeluruh pada kombinasi perangkat keras dengan menggunakan fungsi pembatas yaitu harga. Solusi dibuat dalam bentuk pohon abstrak dengan membentuk lingkaran dari akar ke daun. Aturan penelusuran secara DFS.

Simpul-simpul yang dilahirkan dinamakan simpul hidup (*live node*), kemudian simpul hidup diperluas dinamakan simpul *E* (*Expansion Node*), jika lintasan tidak mengarah ke solusi, maka simpul *E* dibunuh dan simpul mati ini tidak diperluas lagi. Fungsi yang digunakan untuk membunuh simpul *E* adalah fungsi pembatas yaitu nilai anggaran yang dimasukkan. Jika lintasan berakhir dengan simpul mati, maka

proses diteruskan dengan membangkitkan simpul anak lainnya, bila tidak ada lagi simpul anak yang dapat dibangkitkan maka pencarian solusi lanjutkan dengan runut mundur (*backtrack*) ke simpul hidup terdekat (simpul orang tua). Selanjutnya simpul ini menjadi simpul *E* yang baru.

Pencarian dimulai dari simpul $S1_1$ (3.750.000) sebagai simpul orang tua, simpul anak yang pertama kali dibangkitkan adalah $S2_1$ (2.050.000). Kemudian $S1_1$ dan $S2_1$ dijumlahkan karena masih kurang dari batas anggaran (3.750.000 + 2.050.000 < 8.000.000), maka penelusuran dilanjutkan dengan membangkitkan simpul $S3_1$ (1.950.000), kemudian $S3_1$ dijumlahkan (3.750.000 + 2.050.000 + 1.950.000 < 8.000.000). Penelusuran dilanjutkan kembali dengan membangkitkan simpul $S4_1$ (503.000) kemudian dijumlahkan lagi (3.750.000 + 2.050.000 + 1.950.000 + 698.000 > 8.000.000). Karena jumlah melebihi batas anggaran maka simpul $S4_1$ dibunuh dan *backtrack* ke simpul $S3_1$, kemudian membangkitkan simpul $S4_2$ (323.000) dan dijumlahkan (3.750.000 + 2.050.000 + 1.950.000 + 323.000 > 8.000.000). Di sini jumlah masih melebihi batas anggaran, maka simpul $S3_1$ membunuh simpul $S4_2$ dan membangkitkan simpul $S4_3$ (275.000), kemudian menjumlahkan lagi (3.750.000 + 2.050.000 + 1.950.000 + 275.000 > 8.000.000). Karena masih melebihi maka kembali ke simpul $S3_1$ dan simpul anak untuk $S3_1$ sudah habis maka simpul $S3_1$ dibunuh dan kembali ke simpul $S2_1$ dan membangkitkan simpul $S3_2$ (1.200.000). Sekarang penelusurannya ($S1_1$, $S2_1$, $S3_2$) jumlahnya (3.750.000 + 2.050.000 + 1.200.000 < 8.000.000) belum melebihi batas anggaran maka simpul anaknya dibangkitkan $S4_1$ (6.980.000) lalu dijumlahkan (3.750.000 + 2.050.000 + 1.200.000 + 698.000 < 8.000.000) maka simpul $S5_1$ (625.000) dibangkitkan dan dijumlah (3.750.000 + 2.050.000 + 1.200.000 + 698.000 + 625.000 > 8.000.000) melebihi maka simpul $S5_1$ dibunuh dan bangkitkan simpul $S5_2$ dan jumlahkan (3.750.000 + 2.050.000 + 1.200.000 + 698.000 + 640.000 > 8.000.000) melebihi maka simpul $S5_2$ dibunuh juga dan bangkitkan simpul $S5_3$ dan jumlahkan (3.750.000 + 2.050.000 + 1.200.000 + 698.000 + 240.000 < 8.000.000), sehingga ruang solusi yang didapat adalah ($S1_1$, $S2_1$, $S3_2$, $S4_1$, $S5_3$) dengan total harga 7.938.000 tidak melebihi batas anggaran. Proses penelusuran ruang solusi secara *backtracking* dengan pohon abstrak dapat dilihat pada gambar 6.

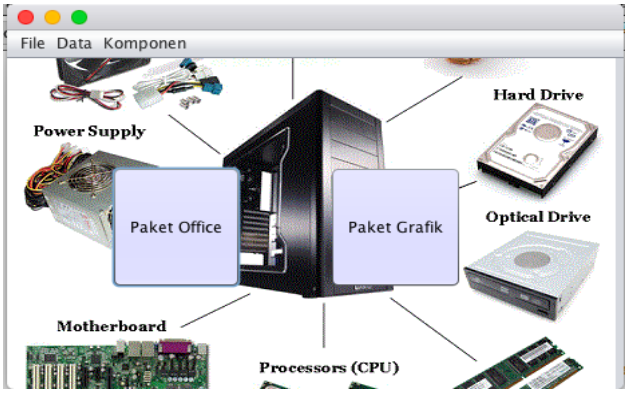


Gambar 6. Pohon abstrak penelusuran secara Bactracking.

Dengan cara yang sama juga dilakukan untuk Paket Game dengan kombinasi yang kedua dan juga untuk Paket Office.

3.2.3. Implementasi Antarmuka Grafis

Gambar 7 memperlihatkan tampilan menu utama dari aplikasi. Tampilan menu utama ini terdiri dari menu *file* dan tombol Paket Office dan Paket Grafik yang akan dijalankan jika pengguna ingin menampilkan antar muka paket.



Gambar 7. Menu utama.

Gambar 8 menampilkan antarmuka Paket Office. Antarmuka grafis ini digunakan pengguna untuk mendapatkan informasi komponen utama apa saja yang optimal untuk Paket Office berdasarkan besar anggaran yang dimiliki pengguna. Antarmuka akan menampilkan komponen-komponen dengan total harga yang tidak akan melebihi anggaran yang dimasukkan. Pada Paket Office ini meniadakan komponen kartu grafis (VGA) *non onboard*, jadi di sini hanya akan memilih komponen *motherboard* yang sudah memiliki VGA *onboard*. Performa yang ditampilkan hanya performa dari *processor*.

Gambar 8. Kombinasi Paket Office.

Gambar 9 Menampilkan antarmuka grafis Paket Game. Digunakan untuk mendapatkan informasi kombinasi komponen untuk Paket Game atau Grafik, susunan komponen dari atas ke bawah pada tampilan menunjukkan prioritas pilihan yang ditelusuri oleh Algoritma Backtracking. Pada paket ini menampilkan dua alternatif pilihan kombinasi. Pilihan pertama mengutamakan *processor* dibandingkan VGA *card*, sedangkan pilihan kedua mengutamakan VGA *card* dibandingkan *processor*. Performa yang ditampilkan merupakan total jumlah dari performa *processor* dan VGA *card*.

Gambar 9. Kombinasi Paket Games/Grafik.

3.3. Pengujian

Untuk memastikan bahwa aplikasi yang dibangun sudah berjalan dengan baik, maka perlu dilakukan pengujian terhadap aplikasi. Pengujian yang dilakukan pada penelitian ini menggunakan dua pengujian, yaitu pengujian *white-box* dan *black-box*.

3.3.1. Pengujian *White-box*

Pengujian *white-box* dilakukan dengan tujuan untuk mengetahui kesalahan logika dalam kode program. Digunakan untuk *debugging code*, menemukan kesalahan tipografi acak, dan mengungkap asumsi pemrograman yang salah [9]. Pengujian *white-box* pada penelitian ini memastikan bahwa semua jalur independen dijalankan untuk memastikan semua *statement* pada program dijalankan minimal 1 kali. Pengujian ini dilakukan dengan membuat *flowgraph* dari logika yang diterapkan pada kode program dan menghitung *cyclomatic complexity* dari *flowgraph* dengan rumus $V(G) = E - N + 2$ [10], dan diperoleh hasil $V(G) = 9$. Maka dapat dikatakan program dengan Algoritma Backtracking ini mempunyai prosedur terstruktur dengan baik dan stabil, sebagaimana dapat dilihat pada Tabel 13 yang merupakan nilai *cyclomatic complexity* menggunakan teori Avioso menetapkan pada mulanya standar nilai maksimum untuk *cyclomatic complexity* adalah 10. Namun standar nilai lain seperti 15 atau 20 juga sudah disarankan. Terlepas dari standar tersebut, jika nilai *cyclomatic* melebihi angka 20 maka harus dipertimbangkan bahwa hasil tersebut mengkhawatirkan untuk risiko terjadinya kecacatan [11][12].

Tabel 13. Nilai *cyclomatic complexity.*

Nilai CC	Tipe prosedur	Tingkat risiko
1 - 4	Prosedur Sederhana	Rendah
5 - 10	Prosedur yang terstruktur dan baik	Rendah
11 - 20	Prosedur yang lebih kompleks	Menengah
21 - 50	Prosedur yang kompleks dan kritis	Tinggi
> 50	Rentan Kesalahan	Sangat Tinggi

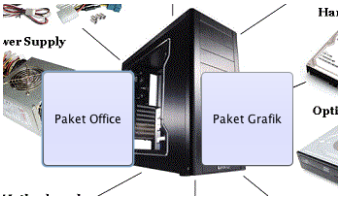
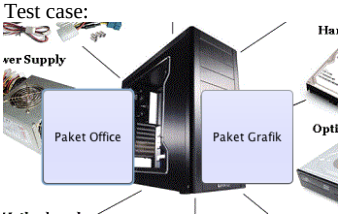
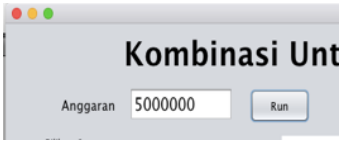
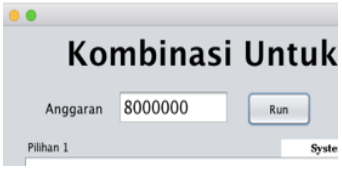
3.3.2. Pengujian *Black-box*

Black-box merupakan pengujian berdasarkan pada spesifikasi kebutuhan dan tidak perlu untuk menguji langsung pada kode program [9]. Teknik pengujian ini mengabaikan mekanisme internal atau struktur sistem dan berfokus pada *output* yang dihasilkan sebagai respons terhadap *input* yang dipilih dan kondisi eksekusi [13]. Pengujian ini murni hanya menguji pada fungsionalitas kebutuhan dari pengguna, dan untuk mengetahui apakah *input* dan *output* dari fungsionalitas berjalan dengan baik dan benar. Pengujian ini juga untuk menemukan *error* antara lain:

- Fungsi-fungsi yang tidak benar
- Kesalahan *interface*
- Kesalahan dalam struktur data
- Kesalahan kinerja

Tabel 14 merupakan pengujian *black-box* pada aplikasi yang dibangun.

Tabel 14. Pengujian *blackbox*.

Skenario Pengujian	Hasil yang Diharapkan	Hasil Pengujian
Klik tombol "Paket Office" pada menu utama Test case: 	Sistem akan menampilkan halaman pencarian kombinasi untuk Paket Office	 Sistem berhasil menampilkan halaman pencarian kombinasi Paket Office
Klik tombol "Paket Grafik" pada menu utama Test case: 	Sistem akan menampilkan halaman pencarian kombinasi untuk Paket Grafik/Game	 Sistem berhasil menampilkan halaman pencarian kombinasi untuk Game/Grafik
Masukkan anggaran pada kolom anggaran Paket Office kemudian klik tombol "*Run*". Test case: 	Sistem akan menampilkan kombinasi komponen komputer paling optimal dan total harga pencarian tidak melebihi batas anggaran	 Berhasil menampilkan kombinasi komputer yang tidak melebihi anggaran
Masukkan anggaran pada kolom anggaran paket grafik kemudian klik tombol "*Run*" Test case: 	Sistem akan menampilkan 2 pilihan kombinasi komponen komputer yang optimal dan tidak melebihi anggaran	 Berhasil menampilkan 2 pilihan kombinasi yang tidak melebihi batas anggaran.

Berdasarkan hasil pengujian semua fungsionalitas berjalan dengan baik dan benar serta sistem dapat memberikan hasil kombinasi spesifikasi komputer yang sesuai dengan kebutuhan dengan tidak melebihi batas anggaran yang dimasukkan dan kompatibilitas antara komponen tetap terjaga. Hasil yang diberikan tergantung dari seberapa banyak data komponen yang dimasukkan ke dalam basis data. Semakin banyak data komponen yang dimasukkan maka semakin baik spesifikasi komputer yang diberikan oleh sistem.

4. Kesimpulan

1. Aplikasi ini dapat merekomendasikan spesifikasi komputer yang optimal dengan tidak melebihi anggaran yang dimiliki dan tetap menjaga kompatibilitas antara komponen dengan baik.

2. Penyusunan spesifikasi komputer dilakukan dengan menelusuri ruang solusi kombinasi perangkat keras yang dibuat berdasarkan prioritas komponen dan menggunakan fungsi pembatas (*Bounding Function*) yaitu harga untuk membunuh simpul yang tidak mengarah ke solusi, sehingga kalkulasi yang tidak perlu dapat dihindari dan dapat mengurangi lamanya komputasi
3. Semua fungsionalitas sistem dapat berfungsi dengan baik setelah melalui pengujian *white-box* dan *black-box*.
4. Algoritma Backtracking ini mempunyai *cyclomatic complexity* yang terstruktur dan baik dengan *V(G) = 9*. Dengan demikian algoritma ini sudah memperbaiki teknik DFS yang dapat mengurangi lamanya waktu komputasi.

Daftar Pustaka

[1] E. Haryanty, "Aplikasi Algoritma Genetika Dalam Menentukan Spesifikasi PC Berdasarkan Kemampuan Finansial Konsumen," *Teknika*, vol. 1, no. 1, pp. 21–25, 2012, doi: 10.34148/teknika.v1i1.3.
[2] M. T. Utomo and H. Mustafidah, "Penentuan Spesifikasi Komputer Berdasarkan Kebutuhan Pemakai Dan Harga Menggunakan Basis Data Fuzzy," *Juita*, vol. IV, no. 1, pp. 28–36, 2016.
[3] A. Herwanto and B. E. Purnama, "Penerapan Metode Depth First Search Pada Pencarian Rute Bus Kota Berbasis Web Mobile Di Solo," *Ilm. Go Infotech*, 2013.
[4] R. B. Sirait, "Perancangan Aplikasi Game Labirin Dengan Menggunakan Algoritma Backtracking," *Pelita Inform. Budi Darma*, vol. 5, no. 2, pp. 100–103, 2013.
[5] F. F. Coastera and A. Nomansa, "Backtracing Pada Aplikasi Permaianan Tradisional Dam-Daman," *Rekursif Inform.*, vol. 1, no. 3, 2019.
[6] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach Third Edition*. New Jersey: Prentice Hall, 2010.
[7] B. Bruegge and A. H. Dutoit, *Object-oriented software engineering : using UML, patterns and Java*. New Jersey: Prentice Hall, 2010.
[8] T. C. Lethbridge and R. Laganiere, *Object-Oriented Software Engineering*. Glasgow: McGraw-Hill Education, 2005.
[9] S. Nidhra and J. Dondeti, "Black Box and White Box Testing Techniques," *Int. J. Embed. Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012.
[10] M. E. Khan, "Different approaches to white box testing technique for finding errors," *Int. J. Softw. Eng. its Appl.*, vol. 2, no. 4, pp. 31–40, 2011, doi: 10.5121/ijsea.2011.2404.
[11] A. Z. Pitoyo, G. Djuwadi, and P. Yudho, "Nilai Cyclomatic Complexity Konflik Kerja terhadap Pengaruh Pimpinan dan Beban Kerja Karyawan dengan Menggunakan Model Reflektif PLS SEM," *J. Pendidik.*, vol. 3, no. 5, pp. 648–655, 2018.
[12] H. L. P. Y. T. A. Candra, "Cyclomatic Complexity Test Design Flowgraph Registration of Emergency Installation Patients in Wava Husada Hospital Using SEM," *Int. J. Sci. Res.*, vol. 6, no. 8, pp. 1983–1987, 2017, doi: 10.21275/ART20176323.
[13] M. E. Khan, "Different Approaches to Black Box," vol. 2, no. 4, pp. 31–40, 2011.