PENERAPAN ALGORITMA BRANCH AND BOUND UNTUK PENENTUAN RUTE OBJEK WISATA

SKRIPSI

Diajukan untuk Menempuh Ujian Akhir Sarjana
Program Strata Satu Jurusan Teknik Informatika
Fakultas Teknik dan Ilmu komputer
Universitas Komputer Indonesia

EKA RIYANTI
10199034



JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK DAN ILMU KOMPUTER
UNIVERSITAS KOMPUTER INDONESIA
BANDUNG
2004

LEMBAR PENGESAHAN

PENERAPAN ALGORITMA BRANCH AND BOUND
UNTUK PENENTUAN RUTE OBJEK WISATA

Penyusun : Eka Riyanti
NIM : 10199034

Pembimbing I	Pembimbing II
Ir. Bambang Siswoyo, M.Si	**Andri Heryandi, S.T.**
NIP : 41277006010	**NIP : 41277006007**

Mengetahui,
Ketua Jurusan Teknik Informatika

Budhi Irawan, S.Si
NIP : 41277006003

LEMBAR PENGESAHAN

PENERAPAN ALGORITMA BRANCH AND BOUND
UNTUK PENENTUAN RUTE OBJEK WISATA

Penyusun : Eka Riyanti
NIM : 10199034

Penguji I

Ir. Bambang Siswoyo, M.Si
NIP : 41277006010

Penguji II

Andri Heryandi, S.T.
NIP : 41277006007

Penguji III

Khusnul Novianingsih, S.Si
NIP : 41277006013

KATA PENGANTAR



Dengan memanjatkan segala puji dan syukur kehadirat Alloh SWT yang telah memberikan rahmat dan hidayah-Nya kepada penulis sehingga berbagai bentuk halangan dan rintangan yang selama penulisan skripsi ini mampu penulis lalui dan akhirnya dapat menyelesaikan penyusunan skripsi dengan baik.

Laporan Skripsi ini berjudul "Penerapan Algoritma Branch And Bound Untuk Penentuan Rute Objek Wisata", dimana didalamnya berisi tentang analisis pencarian rute yang menggunakan algoritma Branch And Bound, rancangan yang akan dibuat dan analisis dari hasil program aplikasi yang telah dibuat. Adapun maksud dari penyusunan skripsi ini adalah untuk menempuh Ujian Akhir Sarjana program Strata Satu Jurusan Teknik Informatika, Fakultas Teknik dan Ilmu Komputer di Universitas Komputer Indonesia (UNIKOM), Bandung.

Dalam penyusunan skripsi ini, tidak lepas dari bantuan dan dukungan dari berbagai pihak, oleh karena itu penulis mengucapkan terima kasih yang sebesar-besarnya kepada :

1. Mamah, Ayah, dan Nuri adikku tercinta yang selalu memberikan motivasi dan bantuan kepada penulis baik dalam bentuk moril, materil maupun spiritual mulai dari awal perkuliahan hingga penulisan skripsi ini.
2. Bapak Ir. Eddy Soeryanto Soegoto, M.Sc. selaku rektor Universitas Komputer Indonesia.

3. Bapak Prof. Dr. Ir. Ukun Sastraprawira, M.Sc. Dekan Fakultas Teknik Universitas Komputer Indonesia.
4. Bapak Budhi Irawan, S.Si, Ketua Jurusan Teknik Informatika Universitas Komputer Indonesia.
5. Bapak Ir. Bambang Siswoyo, M.Si, selaku pembimbing I yang telah banyak membantu dalam metodologi penyusunan skripsi ini.
6. Bapak Andri Heryandi, S.T, selaku pembimbing II yang telah banyak membantu dan memberikan bimbingan yang berarti kepada penulis.
7. Ibu Khusnul Novianingsih, S.Si, selaku penguji, penulis sangat berterimakasih atas penjelasan algoritma *Branch And Bound*-nya.
8. Dosen-dosen Teknik Informatika yang tidak bisa disebutkan satu persatu.
9. Masedy tersayang, atas do'a, kasih sayang, pengertian, motivasi dan bantuan yang selama ini telah diberikan kepada penulis.
10. Teman-teman IF-1, IF-2 angkatan '99 yang tidak dapat disebutkan namanya satu-persatu yang telah membantu selama ini.

Semoga bantuan dan dukungan yang telah diberikan baik moril maupun materil, mendapat imbalan dari Alloh SWT, Amin. Penulis berharap skripsi ini dapat bermanfaat bagi penulis sendiri dan juga bagi pembaca pada umumnya. Terimakasih

Bandung, Agustus 2004

Penulis

DAFTAR ISI

Halaman

Kata Pengantar i

Daftar Isi..... iii

Daftar Gambar..... vi

Daftar Tabel..... viii

Daftar Simbol..... ix

ABSTRAK xi

ABSTRACT..... xii

BAB I. PENDAHULUAN

1.1 Latar Belakang 1

1.2 Rumusan Masalah..... 2

1.3 Batasan Masalah 3

1.4 Maksud dan Tujuan 3

1.5 Metodologi Penelitian..... 4

1.6 Sistematika Penulisan 4

BAB II. LANDASAN TEORI

2.1 Graph 6

2.1.1 Definisi Graph 6

2.1.2 Terminologi Graph 7

2.1.3 Representasi Graph..... 11

2.2 Travelling Salesman Problem 13

2.3 Algoritma Branch And Bound 14
2.3.1 Terminologi Branch And Bound 16
2.3.1.1 Fungsi Pembatas (Bounding) 21
2.3.1.2 Strategi Seleksi 22
2.3.1.3 Aturan Pencabangan (Branching) 24
2.4 Metodologi Pengembangan Perangkat Lunak 25
2.4.1 Proses, Metode dan Alat Bantu..... 25
2.4.2 Pandangan Umum Rekayasa Perangkat Lunak 26
2.4.3 Proses dan Model Perangkat Lunak 27
2.4.3.1 Model Prototipe..... 27
BAB III. ANALISIS DAN PERANCANGAN SISTEM
3.1 Deskripsi Masalah..... 29
3.2 Data Pengujian..... 30
3.3 Pemecahan Dengan Algoritma Branch And Bound..... 30
3.4 Perhitungan Algoritma Branch And Bound Pada Matrik .. 33
3.5 Perancangan Sistem 38
3.5.1 Diagram Konteks (DFD Level 0) 38
3.5.2 DFD Level 1 39
3.5.3 DFD Level 2 Proses 2 40
3.5.4 DFD Level 2 Proses 3 41
3.5.5 DFD Level 3 Proses 3.5 42
BAB IV. IMPLEMENTASI
4.1 Spesifikasi Kebutuhan 44

4.2 Tahap Implentasi Algoritma Branch And Bound 45

4.3 Hasil Pengujian Program 47

4.4 Rekapitulasi Hasil Pengujian 51

BAB V. KESIMPULAN DAN SARAN

5.1 Kesimpulan 53

5.2 Saran 54

DAFTAR PUSTAKA

LAMPIRAN

DAFTAR GAMBAR

Halaman

Gambar 2.1 Contoh graph dengan 6 buah titik 7

Gambar 2.2 Contoh graph lengkap 9

Gambar 2.3 Contoh graph berarah..... 10

Gambar 2.4 Graph berbobot..... 11

Gambar 2.5 Matriks adjacent untuk graph berbobot..... 12

Gambar 2.6 Pencarian menggunakan algoritma branch and bound..... 15

Gambar 2.7 Hubungan antara fungsi bounding *g* dengan fungsi objektif *f* pada satuan S dan P dari solusi yang potensial dan mungkin pada suatu masalah..... 17

Gambar 2.8 Pemetaan kota dan jarak antara masing-masing kota..... 18

Gambar 2.9 Strategi pencarian pada pohon branch and bound..... 24

Gambar 2.10 Prototipe paradigma 28

Gambar 3.1 Diagram alir penyelesaian algoritma branch and bound 32

Gambar 3.2 Pohon algoritma branch and bound..... 37

Gambar 3.3 Skema global sistem pencarian 38

Gambar 3.4 Diagram konteks (DFD Level 0) 38

Gambar 3.5 DFD Level 1 39

Gambar 3.6 DFD Level 2 Proses 2 40

Gambar 3.7 DFD Level 2 Proses 3 41

Gambar 3.8 DFD Level 3 Proses 3.5 42

Gambar 4.1 Form tampilan awal..... 46

Gambar 4.2 Form tampilan 5 objek wisata 47

Gambar 4.3 Form tampilan 10 objek wisata 48

Gambar 4.4 Form tampilan 15 objek wisata 49

Gambar 4.5 Form tampilan 20 objek wisata 50

DAFTAR TABEL

Halaman

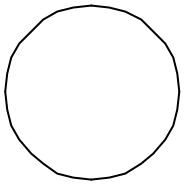
Tabel 3.1 Matrik Jarak 20 objek wisata yang akan dicari..... 30

Tabel 3.2 Contoh Kasus TSP 33

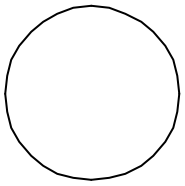
Tabel 4.1 Hasil Pengujian Program..... 51

DAFTAR SIMBOL

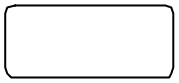
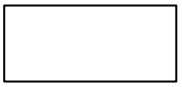
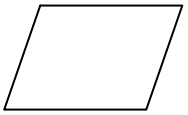
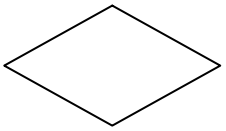
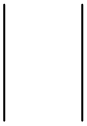
A. Simbol Dalam Diagram Konteks

Simbol	Keterangan
	Data flow (arus data)
	Process (proses)
	External entity (kesatuan luar)

B. Simbol Dalam Data Flow Diagram (DFD)

Simbol	Keterangan
	Data flow (arus data)
	Process (proses)
	External entity (kesatuan luar)

C. Simbol Dalam Diagram Alir

Simbol	Keterangan
	Menunjukkan awal atau akhir dari program
	Menunjukkan kegiatan proses operasi komputer
	Menunjukkan Input dan Output
	Menunjukkan kegiatan pengambilan keputusan dari pilihan yang diberikan
	Data flow (arus data)

ABSTRAK

Sejumlah besar masalah perencanaan dalam dunia nyata yang disebut masalah optimasi kombinatorial yang merupakan masalah optimasi yang mudah untuk dinyatakan dan mempunyai suatu keterbatasan tetapi biasanya memiliki sejumlah besar kemungkinan solusi. Salah satu masalah optimasi yang sering dijumpai adalah *Travelling Salesman Problem* (TSP) yang merupakan masalah pencarian rute perjalanan terpendek dari sejumlah kota atau tempat dimana seseorang menggunakan waktunya untuk mengunjungi n kota (*nodes*) secara siklus perputaran. Di dalam satu kali perjalanan, ia harus menentukan urutan dari sejumlah kota yang harus dilaluinya, setiap kota hanya boleh dilalui sekali dan hanya sekali dalam perjalanan, dan perjalanan berakhir pada kota awal dimana ia memulai perjalanan. Dalam penyelesaian masalah TSP banyak algoritma yang dapat digunakan. Algoritma yang akan penulis pakai untuk masalah TSP diatas adalah algoritma *Branch and Bound*.

ABSTRACT

A large number of real world planning problems called combinatorial optimization problems they are optimization problems, are easy to state, and have a finite but usually very large number of feasible solutions. One of the problem of optimization which is often met by is Travelling Salesman of Problem (TSP) representing the problem of seeking of short journey route from a number of place or town where someone use its time to visit town n (nodes) cyclely rotation. In once journey, he have to determine sequence from a number of town which must pass by of each every town shall only pass by once and for once on the way, and journey end at town early where he start journey. In solving of the problem of TSP many algorithm able to be used. Writer algorithm to wear to the problem of TSP above is algorithm of Branch and Bound.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Objek wisata merupakan tempat yang sering dikunjungi oleh wisatawan yang ingin rekreasi mencari hiburan atau hanya sekedar melepas kepenatan dari aktifitas sehari-hari. Objek wisata sangat beragam macamnya tergantung dari letak, kondisi alam maupun jenis hiburannya. Di daerah Jawa Barat sendiri banyak terdapat objek wisata yang terkenal dan sering dikunjungi wisatawan, seperti Pantai Pangandaran, Gunung Tangkuban Perahu, Taman Bunga Nusantara, Kebun Raya Bogor, Ciater dan lain-lain.

Wisatawan baik domestik maupun mancanegara banyak berkunjung ke tempat wisata, baik menggunakan kendaraan pribadi atau kendaraan umum. Tidak sedikit wisatawan yang menggunakan jasa dari biro perjalanan yang banyak terdapat di kota-kota besar. Umumnya wisatawan tersebut ingin mengunjungi salah satu atau beberapa tempat wisata sekaligus dalam waktu singkat. Banyak biro perjalanan menyediakan jasa perjalanan objek wisata ke beberapa tempat atau dalam bentuk paket wisata.

Dalam pencarian rute terpendek merupakan masalah yang rumit dipandang dari segi komputasinya. Salah satu masalah pencarian rute perjalanan terpendek adalah mencari rute terpendek dari sejumlah objek

wisata dan jarak antar objek wisata yang harus dilalui oleh wisatawan bila wisatawan itu berangkat dari sebuah biro perjalanan dan menyinggahi setiap tempat objek wisata tepat satu kali dan kembali lagi ke tempat biro perjalanan. Secara teoritis, untuk n objek wisata maka terdapat n! rute yang harus dicari. Apabila terdapat n = 5 maka yang harus dicari sebanyak 120 rute, tetapi apabila jumlah n = 30 maka rute yang harus dicari lebih dari 4×10^{30}. Maka pencarian pun sangat lama.

Berdasarkan permasalahan di atas, yang biasanya disebut Travelling Salesman Problem (TSP), penulis mencari solusinya dengan menggunakan algoritma *Branch and Bound* untuk mendapatkan rute perjalanan objek wisata yang terbaik, baik dari sisi efisiensi waktu maupun biaya.

Berdasarkan latar belakang permasalahan di atas, maka penulis mengambil judul **" PENERAPAN ALGORITMA BRANCH AND BOUND UNTUK PENENTUAN RUTE OBJEK WISATA "**.

1.2 Rumusan Masalah

Berdasarkan pertimbangan di atas, maka pada Tugas Akhir ini akan dibahas suatu pendekatan yang berbeda yaitu algoritma *Branch and Bound* untuk menyelesaikan sirkuit terpendek atau disebut juga dengan Travelling Salesman Problem dengan kriteria minimasi jalur dan waktu pencarian.

1.3 Maksud dan Tujuan

Maksud penulisan Tugas Akhir ini adalah untuk memenuhi salah satu syarat menyelesaikan studi pada program Strata I Fakultas Teknik dn Ilmu Komputer, Jurusan Telnik Informatika, Universitas Komputer Indonesia, Bandung.

Tujuan dari penulisan Tugas Akhir ini yaitu :

a. Membantu mengatasi kesulitan pencarian rute objek wisata yang dapat dikunjungi dalam waktu singkat dan efisiensi biaya.
b. Untuk mengetahui performansi dan cara kerja dari algoritma *Branch and Bound* dalam menentukan rute tercepat sehingga dapat digunakan sebagai alternatif pemecahan persoalan Travelling Salesman Problem.
c. Membuat program aplikasi dari algoritma yang digunakan untuk pengujian dan pengembangan sehingga dapat dimanfaatkan untuk menyelesaikan masalah Travelling Salesman Problem.

1.4 Batasan Masalah

Batasan masalah pada Tugas Akhir ini adalah :

a. TSP diterapkan hanya untuk pencarian rute terpendek dari beberapa objek wisata dari sebuah kota asal dan menyinggahi setiap objek wisata tepat satu kali dan kembali lagi ke kota asal keberangkatan.
b. Jarak objek wisata ke objek wisata lain dianggap lurus (ditarik garis dari objek wisata satu ke objek wisata yang lainnya) dan seluruh jalan yang digunakan dianggap jalur dua arah.

c. Graph yang digunakan adalah graph lengkap tak berarah dan bentuk TSP yang digunakan adalah TSP simetris.
d. Tidak ada prioritas objek wisata yang akan dilalui terlebih dahulu.
e. Objek wisata yang dapat dihubungkan maksimal 20 objek wisata dalam satu lingkup daerah, yaitu Jawa Barat.
f. Peta yang digunakan adalah objek-objek wisata yang berada di wilayah Jawa Barat.
g. Objek-objek wisata yang masuk dalam kategori diklasifikasikan terkenal dan banyak dikunjungi wisatawan.

1.5 Metodologi Penelitian

Teknik pengumpulan data yang dilakukan dalam penyusunan laporan skripsi ini adalah berupa studi kepustakaan, yaitu mengumpulkan literatur bacaan yang bisa berupa buku, internet, majalah, jurnal, *textbook*, artikel-artikel yang berhubungan dengan permasalahan yang diambil dan melakukan simulasi dengan menggunakan algoritma *Branch And Bound* untuk penyelesaian masalah *Travelling Salesman Problem*.

1.6 Sistematika Penulisan

Sistematika penulisan dapat memberikan informasi secara umum tentang pembatasan yang terdapat dalam setiap bab sehingga pembahasannya akan lebih mudah dan terarah, sistematika penulisannya adalah sebagai berikut :

BAB I : PENDAHULUAN

Bab ini berisi tentang latar belakang dari masalah, rumusan masalah, maksud dan tujuan penulisan, batasan masalah, metodologi penelitian, sistematika penulisan.

BAB II : LANDASAN TEORI

Bab ini membahas mengenai teori-teori yang akan digunakan menyelesaikan permasalahan yang meliputi teori tentang graph, Travelling Salesman Problem, algoritma *Branch and Bound*, metodologi pengembangan perangkat lunak dan teori-teori penunjang lainnya.

BAB III : ANALISIS MASALAH

Bab ini berisi tentang deskripsi masalah, data hasil pengujian, pemecahan Travelling Salesman Problem menggunakan *algoritma Branch and Bound* dan perhitungan algoritma *Branch and Bound* pada matrik dan perancangan sistem.

BAB IV : IMPLEMENTASI

Bab ini menjelaskan mengenai cara mengimplementasikan setiap prosedur yang telah dirancang pada bab sebelumnya ke dalam bentuk bahasa pemrograman untuk membuat aplikasi. Kemudian akan dilakukan pengujian dan pembahasan tentang kemampuan dari aplikasi tersebut.

BAB V : KESIMPULAN DAN SARAN

Bab terakhir ini akan memberikan suatu kesimpulan dan saran terhadap analisa yang dilakukan pada penyelesaian Travelling Salesman Problem menggunakan algoritma *Branch and Bound*.

BAB II

LANDASAN TEORI

2.1 Graph

Graph bisa dibayangkan sebagai kumpulan obyek atau aktifitas. Sebagai contoh, rute bis kota dari satu terminal ke terminal lain, rute perjalanan seorang tukang pos pada saat mengantar surat dari satu rumah ke rumah lain, dan contoh-contoh lain yang bisa disajikan sebagai suatu graph. Contoh diatas merupakan contoh klasik dari teori graph yang lebih dikenal dengan persoalan *travelling salesman problem* atau *shortest path problem*, yang pada prinsipnya mencari jalur terpendek dari semua tempat yang harus dikunjungi, sehingga dapat menghemat waktu, tenaga, maupun biaya. Selain contoh persoalan di atas, masih banyak persoalan lain yang bisa disajikan sebagai persoalan graph.

2.1.1 Definisi graph

Graph secara umum dapat didefinisikan sebagai kumpulan titik (*nodes* atau *vertices*) yang dihubungkan dengan garis (*arcs* atau *edges*). Secara singkat graph dapat ditulis sebagai G = (V, E), yang dalam hal ini :

V : Sekumpulan dari titik (*nodes* atau *vertices*).

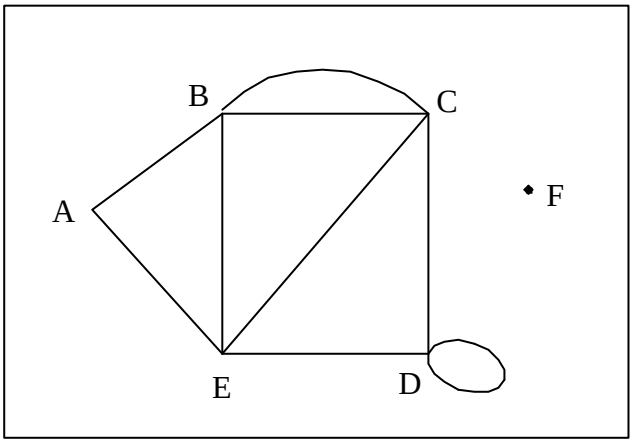
: $\{v_1, v_2, v_3, \ldots v_n)$

E : Sekumpulan garis yang menghubungkan titik ke titik yang lain.

: $\{e_1, e_2, e_3, \ldots e_n)$

2.1.2 Terminologi graph

Suatu garis selalu diawali pada suatu titik dan diakhiri pada titik yang lain, maka garis dapat dituliskan sebagai pasangan antara dua titik. Dalam notasi graph, garis dituliskan sebagai $e = [u,v]$, yang berarti bahwa garis e berawal pada titik u dan berakhir pada titik v. Dua buah titik yang menghubungkan sebuah garis dikatakan sebagai titik yang bertetangga. Meskipun demikian, tidak harus semua titik yang ada pada sebuah graph dipasangkan dengan titik lain untuk membentuk garis. Titik-titik yang tidak dihubungkan dengan titik lain disebut dengan titik terisolir (*isolated node*). Gambar 2.1 menunjukkan contoh sebuah graph yang mempunyai 6 buah titik.



Gambar 2.1 Contoh graph dengan 6 buah titik.

Pada Gambar 2.1 dapat dilihat bahwa graph tersebut mempunyai 6 buah titik {A, B, C, D, E, F}, dan 9 buah garis { [A, B], [A, E], [B, C], [C, B], [C, D], [D, E], [C, E], [D, D], [E, B] }, dengan sebuah titik yang terisolir yaitu titik F. Pada gambar di atas juga dapat dilihat bahwa dari titik B ke titik C (atau sebaliknya) terdapat dua buah garis. Kondisi ini

disebut sebagai garis ganda (*multiple edges*). Selain itu juga terdapat sebuah garis yang berujung dan berpangkal pada titik yang sama, yaitu pada titik D, garis ini disebut dengan kalang (*loop*).

Sebuah jalur (*path*) dengan panjang *n* dari titik *u* ke titik *v* didefinisikan sebagai deretan $n + 1$ buah titik, dan ditulis dengan notasi :

$P = (v_0, v_1, v_2, ..., v_n)$

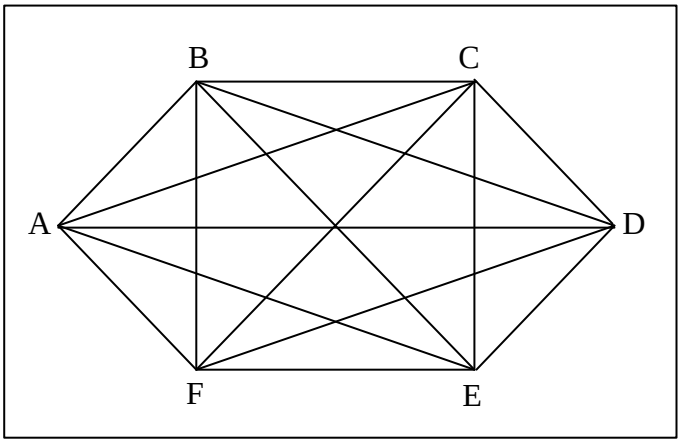
sedemikian rupa sehingga titik yang merupakan pangkal dari suatu garis menjadi ujung dari garis berikutnya kecuali titik pertama dan terakhir (titik v_0 dan v_n). Dalam notasi di atas, garis pertama akan terbentuk dari titik v_0 dan v_1, garis kedua terbentuk dari titik v_1 dan v_2, dan seterusnya. Jalur yang terbentuk dari titik yang berbeda disebut jalur sederhana (*simple path*). Dalam Gambar 2.1 (C, D, D, E, A, B) adalah sebuah jalur dengan panjang 5, dan (C, D, E, A, B) disebut sebagai jalur sederhana dengan panjang 4.

Jalur yang kedua titik ujungnya sama ($v_0 = v_n$) disebut dengan jalur tertutup (*closed path*). Jalur tertutup yang terbentuk dari jalur sederhana disebut lingkar (*cycle*). Pada Gambar 2.1 (C, D, D, E, B, C) disebut sebagai jalur tertutup, dan (C, D, E, B, C) disebut sebagai lingkar.

Graph disebut terhubung (*connected graph*) apabila dari sembarang titik yang ada dapat dibuat jalur ke titik yang lain. Dengan demikian, graph pada Gambar 2.1 bukan merupakan graph terhubung, karena terdapat sebuah titik yang terisolir (titik F). Jika titik terisolir tersebut dihapus, maka graph akan menjadi graph terhubung. Misalnya

apabila titik F dihubungkan dengan titik C, maka graph menjadi graph terhubung.

Suatu graph disebut graph lengkap (*complete graph*) apabila setiap titik yang ada dihubungkan ke titik-titik yang lain. Graph pada Gambar 2.2 adalah graph lengkap, dan graph pada Gambar 2.1 bukan graph lengkap. Sembarang graph lengkap dengan n buah titik akan mempunyai garis sebanyak $n(n-1)/2$ buah.



Gambar 2.2 Contoh graph lengkap.

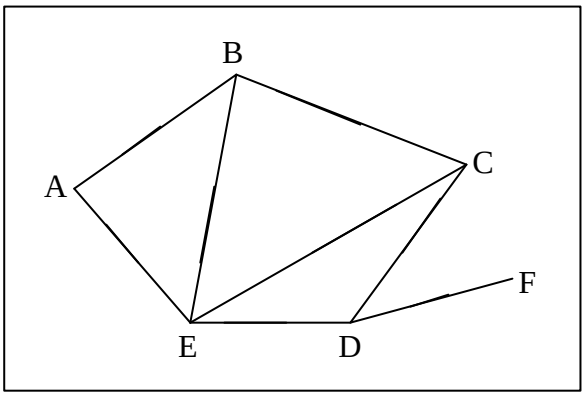
Graph terhubung yang tidak mempunyai lingkar disebut sebagai graph pohon (*tree graph*) atau pohon bebas (*free tree*) atau pohon. Dengan demikian, suatu pohon adalah salah satu bentuk khusus dari graph.

Graph berarah (*directed graph* atau *digraph*) adalah merupakan bentuk yang lebih khusus dari graph seperti dijelaskan di atas, karena di dalam graph berarah terkandung suatu aliran (*flow*), misalnya aliran beban, dari satu titik ke titik yang lain, dalam gambar biasanya disajikan menggunakan anak panah. Dengan demikian, pasangan titik yang menyatakan suatu garis harus disusun secara berurutan (*ordered pair*). Hal

ini bisa dipahami dengan membayangkan suatu jalan yang lalu lintasnya hanya bisa dalam satu arah (*one-way traffic*). Dalam pasangan berurutan ini titik pertama menunjukkan titik asal aliran (*source*), dan titik kedua menunjukkan titik tujuan (*sink*). Dengan kenyataan di atas, maka graph dalam Gambar 2.3 harus dituliskan sebagai :

N = {A, B, C, D, E, F}

E = {[A, B], [A, E], [B, C], [E, B], [C, E], [C, D], [E, D], [D, F]}



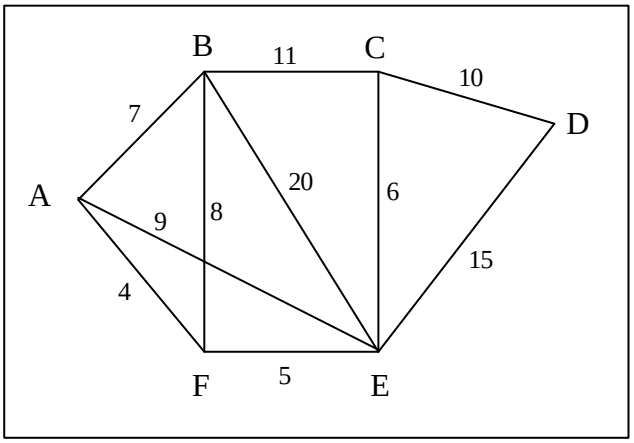
Gambar 2.3 Contoh Graph Berarah

selain sebutan titik asal dan titik tujuan, titik pertama juga sering disebut sebagai predesesor dari titik kedua, dan titik kedua adalah suksesor dari titik pertama. Dengan demikian, titik A adalah predesesor dari titik B dan E, dan titik B adalah suksesor dari titik A. Dengan memperhatikan banyaknya anak panah yang keluar dan masuk pada suatu titik, dikenal dua istilah yang lain, yaitu *indegree* dan *outdegree*. *Indegree* adalah banyaknya anak panah yang masuk ke suatu titik, dan *outdegree* adalah banyaknya anak panah yang meninggalkan suatu titik. Dengan mengambil contoh graph berarah pada Gambar 2.3 maka *indegree* dari titik A adalah 0, dan *outdegree* dari titik A adalah 2. Dengan cara yang sama dapat

diketahui banyaknya *indegree* maupun *outdegree* dari setiap titik yang ada pada suatu graph berarah.

Sehubungan dengan graph berarah, graph tak berarah sering dipakai untuk menggambarkan graph berarah yang setiap garis pada graph tak berarah mewakili sepasang garis pada graph berarah yang mempunyai arah aliran yang saling berlawanan.

Graph berbobot (*weighted graph*) adalah graph yang setiap sisinya diberi sebuah nilai atau bobot. Bobot tiap sisi dapat menyatakan jarak antar dua buah kota atau panjang suatu jalan pada sebuah kota, biaya perjalanan antara dua buah kota, ongkos produksi dan sebagainya. Gambar 2.4 adalah contoh dari graph berbobot.

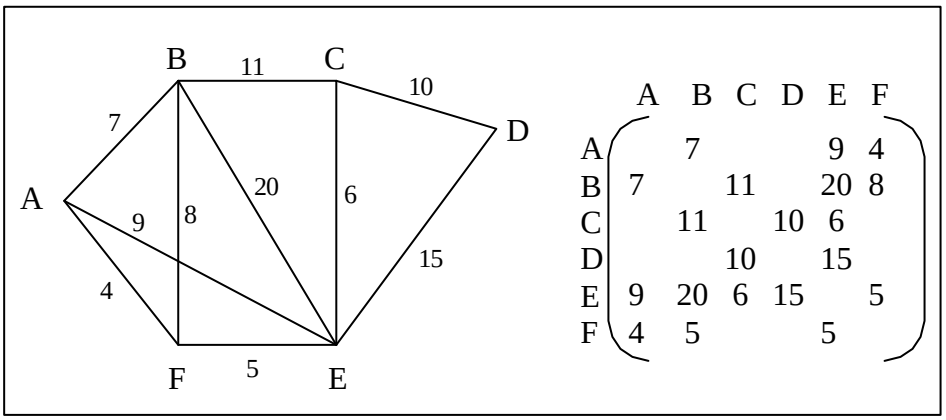


Gambar 2.4 Graph berbobot

2.1.3 Representasi graph

Pemrosesan graph dengan program komputer memerlukan representasi graph dalam memori. Ada beberapa representasi graph yang mungkin untuk dilakukan, salah satunya adalah dengan matriks *adjacent* (*adjacency matrix*).

Matriks *adjacent* adalah representasi graph yang paling umum digunakan. Misalkan G = (V, E) adalah graph dengan n simpul, n i. Matriks *adjacent* G adalah matriks yang berukuran n x n, bila matriks tersebut dinamakan matriks A = [a, j] maka bernilai 1 (satu) jika simpul dan j berbatasan dan bernilai 0 (nol) jika simpul i dan simpul j tidak berbatasan. Matriks *adjacent* untuk graph sederhana dan tidak berarah selalu simetris dan diagonal utamanya selalu bernilai 0 (nol) karena tidak ada sisi perulangan. Jumlah elemen matriks *adjacent* untuk graph dengan n simpul adalah n^2. Jika tiap elemen membutuhkan ruang memori sebesar p maka ruang memori yang dibutuhkan seluruhnya adalah pn^2. Pada matriks *adjacent* untuk graph tak berarah sederhana simetris, cukup dengan menyimpan elemen segitiga atas saja, karena matriksnya simetris, sehingga ruang lingkup memori akan dapat dihemat sebesar pn^2/2. Pada graph berbobot, a_{ij} menyatakan tiap sisi yang menghubungkan simpul dengan simpul j. Gambar 2.5 merupakan contoh graph berbobot beserta matriks *adjacent*.



Gambar 2.5 Matrik *adjacent* untuk graph berbobot

2.2 Travelling Salesman Problem

Travelling Salesman Problem termasuk ke dalam persoalan yang sangat terkenal dalam teori graph. Nama persoalan ini diilhami oleh masalah seorang pedagang yang akan mengunjungi sejumlah kota. Uraian persoalannya adalah sebagai berikut:

Suatu pedagang menggunakan waktunya untuk mengunjungi n kota (*nodes*) secara siklus perputaran. Didalam satu kali perjalanan keliling, ia harus menentukan urutan dari sejumlah kota yang harus dilaluinya, setiap kota hanya boleh dilalui sekali dan hanya sekali dalam perjalanan, dan perjalanan berakhir pada kota awal dimana ia memulai perjalanan.

Kebanyakan Travelling Salesman Problem merupakan suatu simetris yang berarti untuk dua kota A dan B, jarak dari kota A ke kota B adalah sama dengan jarak dari kota B ke kota A. Dalam hal ini, kita akan mendapatkan panjang perjalanan keliling yang sama persis jika kita membalikkan rute perjalanan tersebut. Jadi tidak ada perbedaan antara suatu perjalanan keliling dan kebalikannya.

Fungsi objektif dari persoalan Travelling Salesman Problem adalah sebagai berikut :

$$f(x) = \sum_{i \neq j}^{n} W_{ij} \cdot X_{ij}$$

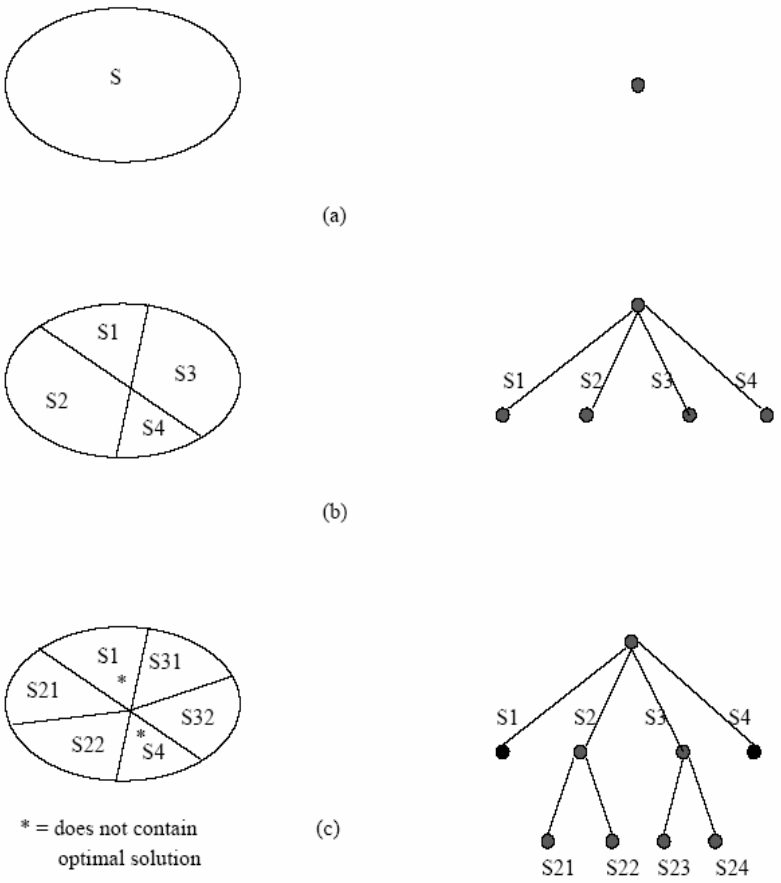
Xij : 0 jika i dan j tidak terpilih
1 jika i dan j terpilih

2.3 Algoritma *Branch and Bound*

Pemecahan masalah optimasi Travelling Salesman Problem merupakan pekerjaan yang membutuhkan algoritma yang efisien dan algoritma *Branch and Bound* merupakan salah satu algoritma untuk memecahkan masalah tersebut. Algoritma *Branch and Bound* mencari sejumlah solusi yang lengkap untuk masalah yang ada dengan hasil yang terbaik. Walaupun begitu, penggunaan satu per satu secara eksplisit tidak mungkin dilakukan dalam kaitan penambahan sejumlah solusi yang potensial. Penggunaan batas (*bound*) untuk fungsi yang akan dioptimalkan dikombinasikan dengan nilai solusi terbaik yang ada memungkinkan algoritma untuk mencari bagian-bagian dari sejumlah solusi secara implisit.

Pada titik sembarang sepanjang proses solusi, status solusi yang berkenaan dengan pencarian sejumlah solusi dijelaskan oleh sekelompok ahli yang mempelajari dan belum seluruhnya dieksplorasi tetapi sejauh ini merupakan solusi terbaik yang ada saat ini. Pada awalnya hanya ada subset yaitu ruang solusi lengkap (*complete solution space*) dan solusi terbaik sejauh ini yang ditemukan baru 1 (satu). *Subspace* yang belum diperiksa direpresentasikan sebagai titik-titik dalam sebuah pohon pencarian yang dihasilkan secara dinamis, dimana awalnya hanya berisi *root*, dan setiap iterasi dari algoritma *Branch and Bound* klasik memproses satu titik. Iterasi memiliki tiga komponen utama yaitu pemilihan titik untuk diproses,

kalkulasi batasan (*bound*), dan pencabangan. Pada Gambar 2.6, dijelaskan situasi awal dan langkah berikutnya dari proses.



Gambar 2.6 Pencarian menggunakan algoritma *Branch and Bound*

Urutan dari pencarian ini dapat dipertukarkan sembarang sesuai dengan strategi yang dipilih untuk memilih node berikutnya yang akan diproses. Jika pemilihan subproblem berikutnya didasarkan pada nilai batas (*bound*) dari subproblem, maka operasi pertama dari iterasi setelah pemilihan node adalah pencabangan (*branching*), yaitu pembagian ruang solusi dari node menjadi dua atau lebih *subspace* untuk diperiksa dalam sebuah iterasi sub rangkaian. Untuk setiap rangkaian, akan diperiksa apakah *subspace* terdiri dari satu solusi, yang kemudian dibandingkan

dengan solusi terbaik yang ada selama pencarian. Jika tidak, pembatasan fungsi untuk *subspace* dihitung dan dibandingkan dengan solusi terbaik yang diperoleh. Jika pencarian tidak dapat dilanjutkan dimana *subspace* tidak berisi solusi yang optimal, keseluruhan *subspace* akan dibuang, selain itu *subspace* akan disimpan dalam kelompok node bersama-sama dengan batasannya (*bound*). Alternatif lainnya adalah dengan memulai menghitung batas (*bound*) dari node yang terpilih dan kemudian mencabangkannya jika diperlukan. Node-node yang dibuat kemudian disimpan bersamaan dengan batas dari node yang diproses. Pencarian berakhir saat tidak ada lagi bagian dari ruang solusi yang diperiksa, dan solusi optimal kemudian dicatat sebagai solusi terbaik.

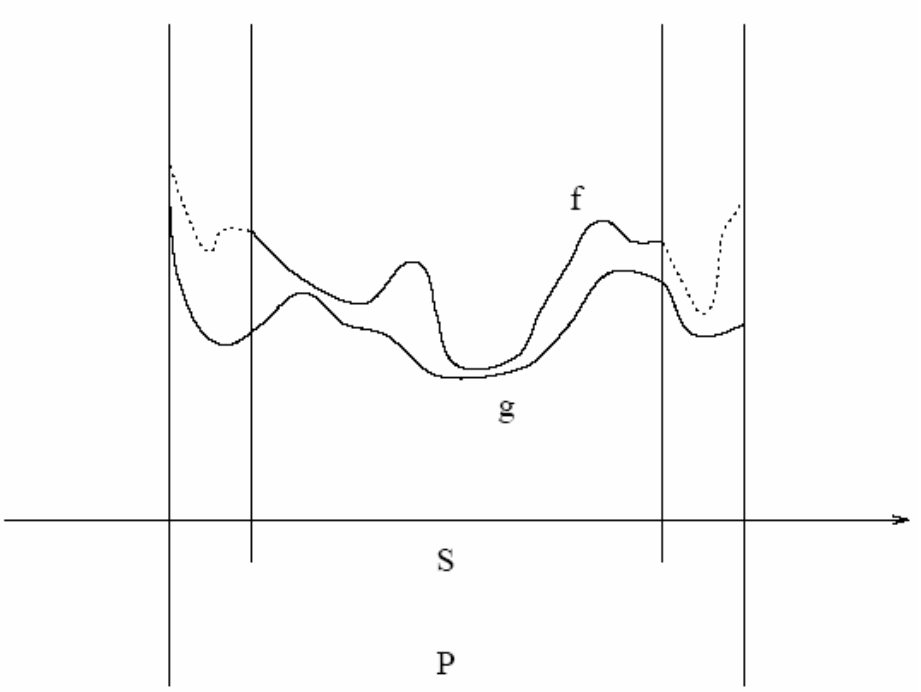
2.3.1 Terminologi *branch and bound*

Dalam *Branch and Bound* terdapat metode untuk meminimalkan masalah dimana kasus maksimalisasi masalah dapat dikerjakan dengan cara yang sama. Masalahnya adalah meminimalkan fungsi $f(x)$ dari variabel ($x_1 \ldots x_n$) sepanjang ruang solusi yang mungkin, S :

$$\min_{x \in S} \; f(x)$$

Fungsi f disebut fungsi objektif (*objective function*) dan dapat berupa berbagai tipe. Suatu kemungkinan solusi umumnya ditentukan oleh kondisi umum pada variabel, meskipun begitu variabel ini harus bukan bilangan bulat atau biner negatif dan batasan khusus menentukan struktur dari set yang mungkin. Pada satu kesempatan, satu set potensial, P, berisi

S dimana *f* masih dibutuhkan. Fungsi *g(x)* diperlukan pada *S* (atau *P*) dengan $g(x) \quad f(x)$ untuk semua *x* dalam S (atau P). Baik *P* dan *g* sangat diperlukan pada konteks *Branch and Bound*. Gambar 2.7 menjelaskan situasi dimana S dan P merupakan interval dari *real.*

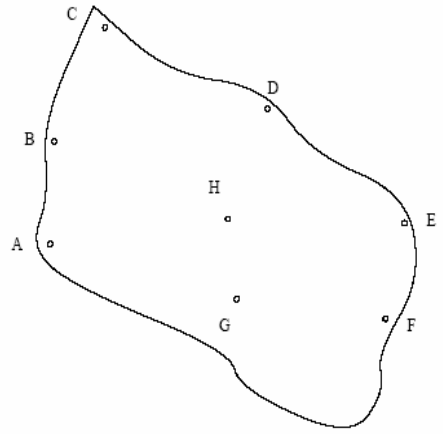


Gambar 2.7 Hubungan antara fungsi *bounding g* dan fungsi objektif *f* pada satuan S dan P dari solusi yang potensial dan mungkin pada suatu masalah.

Istilah subproblem digunakan untuk menandakan masalah yang berasal dari masalah sebenarnya melalui penambahan batasan baru. Subproblem sesuai dengan *subspace* dari ruang solusi asli, dan dua terminologi yang digunakan dapat dipertukarkan dan dalam konteks dari pohon pencarian dapat dipertukarkan dengan terminologi titik.

Untuk lebih jelas mengenai algoritma *Branch and Bound*, akan diaplikasikan pada masalah optimalisasi kombinatorial yang terkenal yaitu

Travelling Salesman Problem (TSP). Masalah yang muncul dari TSP berhubungan dengan rute perjalanan untuk mengantarkan atau menjual barang pada beberapa kota dengan seminimal mungkin waktu dan jarak perjalanan. Berikut ini merupakan contoh dari kasus Travelling Salesman Problem Simetris.



	A	B	C	D	E	F	G	H
A	0	11	24	25	30	29	15	15
B	11	0	13	20	32	37	17	17
C	24	13	0	16	30	39	29	22
D	25	20	16	0	15	23	18	12
E	30	32	30	15	0	9	23	15
F	29	37	39	23	9	0	14	21
G	15	17	29	18	23	14	0	7
H	15	17	22	12	15	21	7	0

Gambar 2.8 Pemetaan kota dan jarak antara masing-masing kota

Dalam Gambar 2.8 digambarkan peta berikut tabel jarak yang menunjukkan jarak diantara kota-kota yang ada. Masalah dari seorang wisatawan yang ingin mengunjungi semua kota yang ada, dengan perjalanan yang minimum dan mengunjungi setiap kota hanya sekali serta mengakhirinya dikota tempat ia memulai. Perjalanan seperti ini disebut Hamilton *cycle*. Masalah ini disebut TSP simetris saat tabel dari jarak membentuk simetris.

Umumnya TSP simetris dihasilkan oleh simetrik $n \times n$ matrik D dengan jarak tidak negatif dan tujuannya adalah mencari panjang minimum dari perjalanan Hamilton (*Hamilton tour*). Dalam kaitannya dengan grafik, digunakan *undirected graph* lengkap dengan n simpul Kn,

dan panjang tidak negatif diberikan pada sisi-sisinya, dan tujuannya adalah menentukan panjang minimum dari perjalanan Hamilton (*Hamilton tour*). Masalah dapat juga dinyatakan secara matematika menggunakan variabel keputusan untuk menguraikan sisi-sisi yang akan disertakan didalam perjalanan. Digunakan nilai 0,1 dengan variabel x_{ij} ; 1 $i < j$ n, dan mengintepretasikan nilai 0 menjadi "Tidak dalam perjalanan" dan sebaliknya untuk nilai 1. Masalahnya kemudian menjadi :

$$min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_{ij}$$

sehingga menjadi

$$\sum_{k=1}^{i-1} x_{ki} + \sum_{k=i+1}^{n} x_{ik} = 2, \qquad i \in \{1, ..., n\}$$

$$\sum_{i,j \in Z} x_{ij} < |Z| \qquad \emptyset \subset Z \subset V$$

$$x_{ij} \in \{0, 1\}, \qquad i, j \in \{1, ..., n\}$$

Satuan pertama dari pembatas memastikan bahwa untuk masing-masing i tepatnya dua variabel yang dihubungkan sisi dengan i yang terpilih. Saat masing-masing sisi mempunyai dua titik akhir yang mengimplikasikan bahwa variabel n dibolehkan untuk mengambil nilai 1. Satuan kedua dari pembatas terdiri dari batasan sub perjalanan yang di eliminasi. Masing-masing dari batasan ini ditentukan untuk menspesifikasikan subset S dari V dimana nilai dari sisi-sisinya dihubungkan dengan simbol dalam S yang harus kurang dari | S | dengan

demikian mengesampingkan batasan dari bentuk sub perjalanan. Tetapi banyak yang bersifat eksponen pada pembatasan ini.

Batasan yang diberikan menentukan satuan solusi yang mungkin (S). Satu cara yang jelas memudahkan dalam mengatur sekumpulan solusi yang potensial adalah dengan meringankan batasan eliminasi dari sub perjalanan. Satuan solusi potensial P merupakan anggota dari semua satuan sub perjalanan, seperti masing-masing *i* mengacu pada tepatnya salah satu dari sub perjalanan pada setiap satuan yang ditetapkan.

Suatu subproblem dari TSP simetris yang diberikan dibangun dengan menentukan subset A dari sisi G yang harus disertakan dalam perjalanan yang akan dibangun. Pengeluaran dari suatu sisi (*i, j*) biasanya dimodelkan dengan mengatur c_{ij} ke , dimana pemasukan dari suatu sisi dapat ditangani dalam berbagai cara. Nilai dari solusi yang mungkin untuk masalah diatas adalah (n 1)!=2, dengan n=50 adalah $3x10^{62}$.

Algoritma *Branch and Bound* digunakan untuk meminimalkan masalah. Oleh karena itu algoritma ini terdiri dari tiga komponen, yaitu :

1. Fungsi pembatas (*bounding*) : fungsi yang disediakan *subspace* dari ruang solusi dengan batas rendah untuk nilai solusi terbaik yang diperoleh dalam *subspace*.
2. Strategi seleksi : suatu strategi untuk memilih solusi *subspace* aktif untuk diperiksa dalam iterasi, dan
3. Aturan pencabangan (*branching*) : Suatu aturan yang diaplikasikan jika *subspace* setelah diperiksa tidak dapat dibatalkan, karena itu

pembagian *subspace* kedalam dua atau lebih *subspace* untuk diperiksa dalam sub rangkaian iterasi.

2.3.1.1 Fungsi pembatas (*Bounding*)

Fungsi pembatas merupakan kunci komponen dari berbagai algoritma *Branch and Bound* dalam arti bahwa fungsi pembatas yang berkualitas rendah tidak dapat dikompensasikan melalui pilihan yang baik dari pemilihan strategi dan pencabangan. Idealnya nilai dari fungsi pembatas yang diberikan pada subproblem seharusnya sama dengan nilai solusi terbaik dari masalah, tetapi saat mendapatkan nilai ini biasanya dimasukan sendiri kedalam *NP-hard.* Tujuannya adalah untuk mendapatkan kemungkinan sejumlah perhitungan komputasi terbatas.

Fungsi pembatas disebut kuat (*strong*), jika fungsi menghasilkan nilai mendekati nilai optimal untuk batasan subproblem, dan lemah (*weak*), jika nilai yang dihasilkan jauh dari optimal. Satu hal yang sering dialami dari pertukaran antara kualitas dan waktu saat berhubungan dengan fungsi pembatas. Karena itu semakin banyak waktu dihabiskan untuk menghitung batas, biasanya semakin baik nilai dari batas (*bound*).

Fungsi pembatas biasanya muncul dalam hubungan dengan satuan solusi potensial P dan fungsi g. Berkaitan dengan fakta bahwa S Ñ P, dan $g(x)$ $f(x)$ pada P, berikut ini merupakan merupakan fungsi lengkapnya.

$$min_{x \in P}\, g(x) \leq \left\{ \begin{array}{l} min_{x \in P}\, f(x) \\ min_{x \in S}\, g(x) \end{array} \right\} \leq min_{x \in S}\, f(x)$$

Jika *P* dan *g* aktif, maka sekarang akan ada pilihan untuk memilih diantara tiga masalah optimasi, dimana setiap solusi optimal akan menyediakan batas terendah untuk fungsi objektif yang diberikan. Kemampuan disini adalah untuk memilih antara *P* atau *g* sehingga salah satunya mudah untuk memecahkan dan menyediakan batasan yang mendekati.

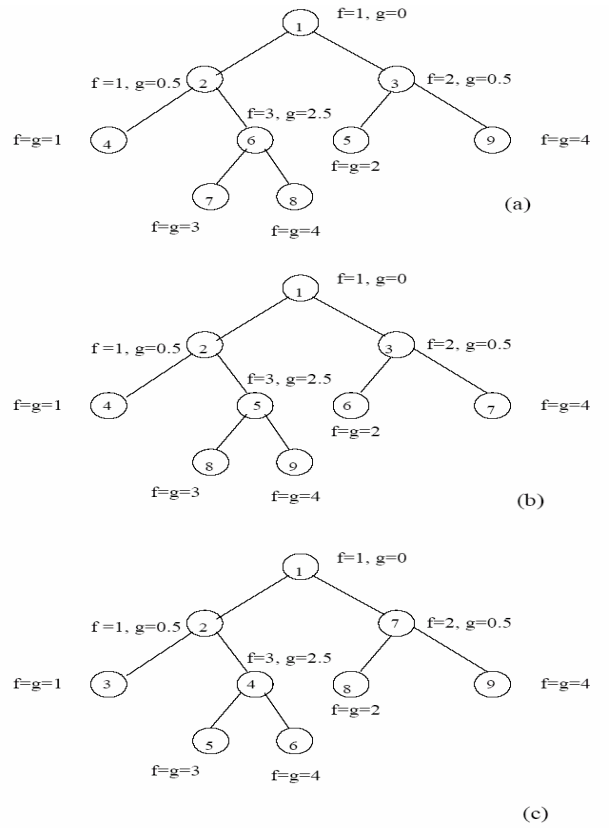
2.3.1.2 Strategi seleksi

Strategi untuk menyeleksi subproblem aktif berikutnya untuk diperiksa biasanya menggambarkan pertukaran diantara menjaga nilai dari titik yang diperiksa dalam pohon pencarian tetap rendah, dan tinggal di dalam kapasitas memori dari komputer yang digunakan. Jika yang satu selalu memilih diantara salah satu subproblem aktif, salah satu dari subproblem dengan batas terendah disebut *best first search strategy*, BeFS, tidak ada kalkukasi batas (*bound*) yang berlebihan terjadi setelah solusi optimal ditemukan. Gambar 2.9 (a) menunjukan sebuah pohon pencarian kecil dengan nilai pada setiap node berkaitan dengan rangkaian, dimana node-node diproses saat BeFS digunakan.

Meskipun pilihan dari subproblem dengan batas terendah (*lower bound*) saat ini berhasil mengenai kemungkinan menghasilkan suatu kemungkinan solusi yang baik, masalah memori muncul jika nilai dari subproblem kritis dari masalah yang diberikan menjadi terlalu besar. Situasinya kurang lebih berkaitan dengan strategi *breath first search*,

dimana seluruh node pada satu level dari pohon pencarian diproses sebelum node berada pada level tertinggi. Gambar 2.9 (b) menunjukkan pohon pencarian dengan nilai-nilai dalam setiap node berkaitan dengan rangkaian proses BFS. Nilai dari setiap node pada masing-masing level dari pohon pencarian tumbuh secara eksponen dengan level yang membuat tidak mungkin untuk melakukan *breath first search*, untuk masalah yang lebih besar.

Alternatif yang digunakan adalah *depth first search*, DFS. Disini node yang aktif dengan level tertinggi dalam pohon pencarian dipilih untuk eksplorasi/pelacakan. Gambar 2.9 (c) menunjukan DFS memproses rangkaian nilai dari node. Kebutuhan memori dalam kaitan dengan jumlah subproblem untuk menyimpan pada saat yang bersamaan sekarang dibatasi diatas oleh banyaknya level dalam pohon pencarian dikalikan dengan jumlah maksimum *child* dari node, dimana biasanya sejumlah besar yang dapat dikendalikan.



Gambar 2.9 : Strategi pencarian dalam *Branch and Bound* : (a) *Best First Search*, (b) *Breadth First Search*, dan (c) *Depth First Search*.

2.3.1.3 Aturan Pencabangan

Seluruh aturan pencabangan dalam konteks *Branch and Bound* dapat terlihat sebagai *subdivision* dari bagian ruang pencarian melalui penambahan batasan, sering dalam bentuk penandaan nilai pada variabel. Pemusatan dari *Branch and Bound* dipastikan jika ukuran dari setiap subproblem yang dihasilkan lebih kecil dari masalah awal, dan sejumlah kemungkinan solusi untuk masalah awal terbatas. Secara normal, subproblem yang dihasilkan terpisah sehingga perlu dihindari adanya masalah dari kemungkinan solusi yang sama yang muncul dalam *subspace* yang berbeda dari pohon pencarian.

2.4 Metodologi Pengembangan Perangkat Lunak

Proses perangkat lunak telah menjadi perhatian yang serius selama dekade terakhir karena proses ini menentukan pendekatan yang digunakan ketika perangkat lunak dikembangkan. Pengembangan perangkat lunak juga meliputi teknologi yang mempopulasikan proses, metode teknis serta alat-alat otomatis.

2.4.1 Proses, metode dan alat bantu

Rekayasa perangkat lunak merupakan sebuah teknologi yang dibentangkan. Banyak pendekatan keteknikan yang harus berada pada sebuah komitmen dasar menuju kualitas. Manajemen kualitas total serta filosofinya mengangkat budaya pengembangan proses yang terus menerus, dan budaya itu sendiri membawa kepada perkembangan pendekatan yang semakin matang terhadap rekayasa perangkat lunak.

Proses di dalam rekayasa perangkat lunak membatasi kerangka kerja untuk serangkaian area proses yang harus dibangun demi keefektifan penyampaian teknologi pengembangan perangkat lunak. Metode dalam rekayasa perangkat lunak memberikan teknik untuk membangun perangkat lunak. Metode-metode itu menyangkut serangkaian tugas mengenai analisis kebutuhan, konstruksi program, desain, pengujian, dan pemeliharaan. Rekayasa perangkat lunak mengandalkan pada serangkaian prinsip dasar yang mengatur setiap area teknologi dan menyangkut aktivitas pemodelan serta teknik-teknik deskriptif yang lain.

2.4.2 Pandangan umum tentang rekayasa perangkat lunak

Rekayasa merupakan analisis, desain, konstruksi, verifikasi, dan manajemen kesatuan teknik. Usaha yang berhubungan dengan rekayasa perangkat lunak dapat dikategorikan ke dalam tiga fase umum dengan tanpa memperdulikan area aplikasi, ukuran proyek, atau kompleksitasnya.

Fase definisi (*definition phase*) merupakan fase dimana pengembang perangkat lunak harus mengidentifikasi informasi yang akan di proses, fungsi dan unjuk kerja yang dibutuhkan, tingkah laku sistem yang diharapkan, interface yang akan dibangun, batasan desain dan kriteria validasi yang dibutuhkan untuk mendefinisikan sistem yang sukses.

Fase pengembangan (*development phase*) merupakan fase dimana selama masa pengembangan perangkat lunak, teknisi harus mendefinisikan bagaimana data dikonstruksikan, fungsi-fungsi diimplementasikan sebagai sebuah arsitektur perangkat lunak, detail prosedur yang akan diimplementasikan, rancangan yang akan diterjemahkan ke dalam bahasa pemrogramanan , serta cara pengujian yang akan dilakukan.

Fase pemeliharaan (*maintenance phase*) merupakan fase yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan dari pengguna.

2.4.3 Proses dan model perangkat lunak

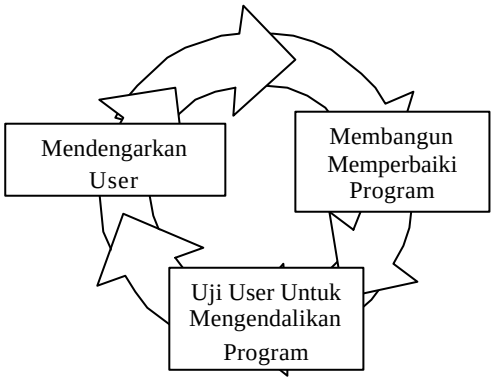
Sebuah kerangka kerja proses umum dibangun dengan mendefinisikan sejumlah aktivitas kerangka kerja yang bisa diaplikasikan ke semua proyek perangkat lunak tanpa melihat ukuran dan kompleksitasnya. Rekayasa perangkat lunak atau tim perekayasa harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode, dan alat-alat bantu. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang akan dipakai, dan kontrol serta penyampaian yang dibutuhkan.

Di dalam perangkat lunak terdapat bermacam-macam model proses. Masing-masing model sudah ditandai dengan cara tertentu sehingga bisa membantu di dalam kontrol dan koordinasi dari proyek perangkat lunak yang nyata. Beberapa model proses yang ada diantaranya adalah model sekuensial linier, model prototipe, model RAD, dan lain-lain. Di dalam tugas akhir ini, penulis menggunakan model prototipe sebagai model proses pengembangan perangkat lunak.

2.4.3.1 Model prototipe

Sering seorang *user* mendefinisikan serangkaian sasaran umum bagi perangkat lunak, tetapi tidak melakukan identifikasi kebutuhan input, pemrosesan, ataupun output secara detail. Pada kasus yang lain, *programmer* mungkin tidak memiliki kepastian terhadap efisiensi algoritma, kemampuan penyesuaian dari sebuah sistem operasi, atau

bentuk-bentuk yang harus dilakukan oleh interaksi manusia dengan mesin. Dalam hal ini, *prototyping paradigma* mungkin menawarkan pendekatan yang terbaik. *Prototyping paradigma* (Gambar 2.10) dimulai dengan pengumpulan kebutuhan. *Programmer* dan *user* bertemu dan mendefinisikan obyektif keseluruhan dari perangkat lunak, mengidentifikasi segala kebutuhan yang diketahui, baru kemudian dilakukan perancangan. Perancangan membawa pada konstruksi sebuah prototipe. Prototipe tersebut dievaluasi oleh *user* dan dipakai untuk menyaring kebutuhan pengembangan perangkat lunak. Iterasi terjadi pada saat prototipe disetel untuk memenuhi kebutuhan *user*, dan pada saat yang sama memungkinkan *programmer* untuk secara lebih baik memahami apa yang harus dilakukannnya.



Gambar 2.10 Prototipe paradigma.

Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak. Bila prototipe yang sedang bekerja dibangun, *programmer* harus menggunakan bagian-bagian program yang ada sehingga memungkinkan program yang bekerja untuk dimunculkan secara cepat.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Deskripsi Masalah

Permasalahan yang akan dibahas dalam tugas akhir ini adalah Travelling Salesman Problem (TSP). Masalah yang muncul dari TSP berhubungan dengan rute perjalanan untuk mengantarkan atau menjual barang pada beberapa kota dengan waktu dan jarak perjalanan seminimal mungkin. Uraian persoalannya adalah diberikan sejumlah kota dan jarak antar kota. Tentukan rute terpendek yang harus dilalui oleh pedagang bila pedagang itu berangkat dari sebuah kota asal dan menyinggahi setiap kota tepat satu kali, dan kembali ke kota asal keberangkatan dengan pencarian yang relatif singkat.

Travelling Salesman Problem (TSP) merupakan masalah optimasi kombinatorial yang terkenal. Untuk menyelesaikan masalah TSP terdapat banyak algoritma yang dapat digunakan, diantaranya adalah *Simulated Annealing*, *Generate and Test*, *Simple Hill Climbing*, *Genetic Algorithms*, *Branch and Bound*, dan lain-lain. Dalam tugas akhir ini penulis menggunakan algoritma *Branch and Bound* untuk menyelesaikan masalah TSP. *Branch and Bound* merupakan algoritma yang harus disesuaikan dengan setiap tipe masalah yang spesifik dan mempunyai banyak pilihan untuk setiap komponen yang ada.

3.2 Data Pengujian

Pada TSP ini menggunakan graph lengkap berbobot yaitu jika setiap titik (*node*) mempunyai sisi atau jarak ke titik (*node*) yang lain. Graph ini terdiri dari 20 simpul yang mewakili banyaknya objek wisata yang akan dicari. Dibawah ini merupakan matrik jarak antar objek wisata yang akan dicari. Matrik lengkapnya dapat di lihat pada lampiran.

Tabel 3.1 Matrik jarak 20 objek wisata yang akan dicari.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**			**T**
A	M	26	21	44	31	43	47	50			253
B	26	M	36	51	52	56	57	61			246
C	21	36	M	25	15	20	26	29			236
D	44	51	25	M	27	12	7	14			212
E	31	52	15	27	M	17	24	23			237
F	43	56	20	12	17	M	8	7			220
G	47	57	26	7	24	8	M	7			213
H	50	61	29	14	23	7	7	M			215
									M		
										M	
T	253	246	236	212	237	220	213	215			M

3.3 Pemecahan Dengan Algoritma *Branch And Bound*

Algoritma *Branch and Bound* yang digunakan untuk menyelesaikan masalah Travelling Salesman Problem adalah sebagai berikut :

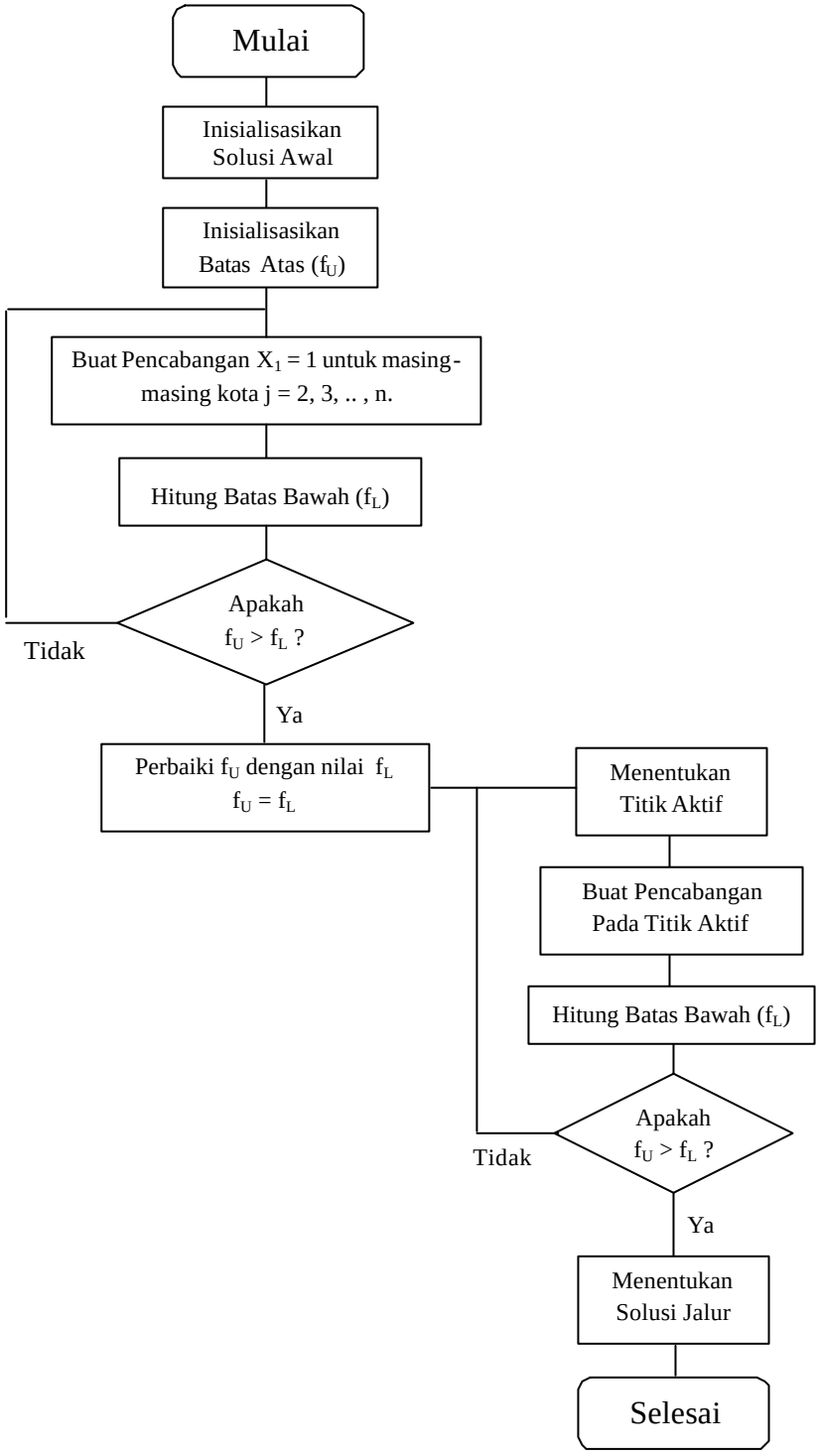
Langkah 1 : Buat penyelesaian masalah awal sebagai penetapan masalah. Solusi yang ditetapkan merupakan suatu perjalanan lengkap, buat batas tertinggi (*Upper bound*) pada nilai minimum fungsi objektif dengan mencari berbagai kemungkinan perjalanan. Batas ini ditunjukkan dengan f_U, dan lanjutkan ke langkah 2.

Langkah 2 : Buat pencabangan awal dengan mengatur x1 = 1 untuk masing-masing kota j = 2, 3, ..., n (kecuali cij = M, mengidentifikasikan rute tidak mungkin). Hitung batas terendah (f_L) pada nilai minimum fungsi objektif di setiap titik sebagai berikut. Dari data awal, hapus baris pertama dan kolom ke j. Selanjutnya, set c_{j1} = +M. Pecahkan hasil masalah yang ditetapkan dan tambahkan harga (f) ke c_{lj}, untuk memperoleh f_L sehingga $f_L = c_{lj} + f$. Jika $f_L > f_U$, ukur titiknya dan lanjutkan ke langkah 3.

Langkah 3 : Jika tidak ada titik aktif maka solusi terbaik saat ini adalah optimal. Jika tidak, pilih titik dengan nilai terkecil f_L dan buat pencabangan baru dengan mengatur $x_{jk} = 1$ untuk setiap kota k yang belum dikunjungi sebelumnya pada sebagian perjalanan. Lanjutkan pada langkah 4.

Langkah 4 : Buat batasan f_L pada setiap titik dengan menghapus baris j dan kolom k dari pemecahan masalah pada titik saat ini diatas titik yang ada, atur c_{kj} = n dan tambahkan nilai f ke c_{jk} dan seluruh nilai sebelumnya dalam sebagian perjalanan.

Dengan mengacu pada langkah-langkah diatas, diagram alurnya sebagai berikut:



Gambar 3.1 Diagram alir penyelesaian algoritma *branch and bound*

3.4 Perhitungan Algoritma *Branch And Bound* Pada Matrik

Misalkan ada 5 objek wisata dengan jarak tiap-tiap objek wisata seperti pada Tabel 3.2.

Tabel 3.2 Contoh kasus TSP

	1	2	3	4	5
1	M	29	18	42	29
2	29	M	37	51	52
3	18	37	M	25	16
4	42	51	25	M	28
5	29	52	16	28	M

Langkah 1 : Solusi untuk masalah yang ditetapkan adalah $x_{12} = x_{23} = x_{34} = x_{45} = x_{51} = 1$ adalah mungkin dengan nilai 29 + 37 + 25 + 28 + 35 = 148, sehingga kita atur $f_U^1 = 148$.

Langkah 2 : Buat percabangan $x_{12} = 1$ (titik 2), $x_{13} = 1$ (titik 3), $x_{14} = 1$ (titik 4), $x_{15} = 1$ (titik 5).

Pada titik 2 hapus baris pertama dan kolom ke dua dari data awal dan set $C_{21} = M$, untuk memperoleh :

	1	3	4	5
2	M	37	51	52
3	18	M	25	16
4	42	25	M	28
5	29	16	28	M

Solusinya adalah $x_{23} = x_{35} = x_{54} = x_{41} = 1$ dengan nilai 37 + 16 + 28 + 42 = 123, dengan f = 123. Jadi , $f_L = C_{12} + f = 29 + 123 = 152$. Kemudian set $f_U^2 = 152$, simpan solusi ini sebagai *incumbent* baru dan mengukur titik 2.

Pada titik 3 hapus baris pertama dan kolom ke tiga dari data awal dan set $C_{31} = M$, untuk memperoleh :

	1	2	4	5
2	29	M	51	52
3	M	37	25	16
4	42	51	M	28
5	29	52	28	M

Solusinya adalah $x_{35} = x_{54} = x_{42} = x_{21} = 1$ dengan nilai 16 + 28 + 51 + 29 = 124, dengan f = 124. Jadi , $f_L = C_{13} + f = 18 + 124 = 142$, sehingga $f_L < f_U^2$ dan titik 3 dapat diukur.

Pada titik 4 hapus baris pertama dan kolom ke empat dari data awal dan set C_{41} = M, untuk memperoleh :

	1	2	3	5
2	29	M	37	52
3	18	37	M	16
4	M	51	25	28
5	29	52	16	M

Solusinya adalah $x_{43} = x_{35} = x_{52} = x_{21} = 1$ dengan nilai 25 + 16 + 52 + 29 = 132, dengan f = 132. Jadi, $f_L = C_{41} + f = 42 + 132 = 174$, simpan solusi ini sebagai *incumbent* baru dan mengukur titik 4.

Pada titik 5 hapus baris pertama dan kolom ke lima dari data awal dan set C_{51} = M, untuk memperoleh :

	1	2	3	4
2	29	M	37	51
3	18	37	M	25
4	42	51	25	M
5	M	52	16	28

Solusinya adalah $x_{53} = x_{34} = x_{42} = x_{21} = 1$ dengan nilai 16 + 25 + 51 + 29 = 121, dengan f = 30. Jadi, $f_L = C_{41} + f = 29 +$

121 = 150, simpan solusi ini sebagai *incumbent* baru dan mengukur titik 5.

Langkah 3 : Terdapat satu titik aktif (titik 3) dengan f_L = 142. Pilih titik 3 secara berubah-ubah, buat pencabangan x_{32} = 1 (titik 6), x_{34} = 1 (titik 7), dan x_{35} =1 (titik 8).

Langkah 4 : Buat batasan pada titik 6, 7 dan 8 dengan memodifikasi data pada titik 3, karena titik-titik ini adalah turunan yang berasal dari titik 3.

Pada titik 6 hapus baris ke empat dan kolom ke dua dari data di titik 4 dan set C_{21} = M, untuk memperoleh :

	1	4	5
2	M	51	52
4	42	M	28
5	29	28	M

Dengan solusi $x_{24} = x_{45} = x_{51}$ = 1, dengan nilai 51 + 28 + 29 = 108, dengan f = 108. Jadi, $f_L = C_{13} + C_{32}$ + f = 18 + 37 + 108 = 163, simpan solusi ini sebagai *incumbent* baru dan mengukur titik 6.

Pada titik 7 hapus baris ke tiga dan kolom ke empat dari data di titik 3 dan set C_{41}= M, untuk memperoleh :

	1	2	5
2	29	M	52
4	M	51	28
5	29	52	M

Dengan solusi $x_{45} = x_{52} = x_{21}$ = 1, dengan nilai 28 + 52 + 29 = 109, dengan f = 109. Jadi, $f_L = C_{13} + C_{34}$ + f = 18 + 25 + 109

= 152, simpan solusi ini sebagai *incumbent* baru dan mengukur titik 6..

Pada titik 8 hapus baris ke tiga dan kolom ke lima dari data di titik 3 dan set C_{51}= M, untuk memperoleh :

	1	2	4
2	29	M	51
4	42	51	M
5	M	52	28

Dengan solusi $x_{54} = x_{42} = x_{21} = 1$, dengan nilai 28 + 51 + 29 = 108, dengan f = 108. Jadi, $f_L = C_{13} + C_{35} + f = 18 + 16 + 108 = 142$, sehingga $f_L < f^2_U$ dan titik 8 dapat diukur.

Perhitungan untuk batas bawah dihasilkan dalam kemungkinan perjalanan pada seluruh tiga titik dengan nilai-nilai yang dihasilkan dan untuk mencari solusi alternatif, pilih titik 8 karena merupakan titik yang memiliki batas bawah yang terendah.

Langkah 3 : Titik aktif yang dibutuhkan hanya titik 8. Buat pencabangan $x_{52} = 1$ (titik 9), x_{54}= 1 (titik 10).

Pada titik 9 hapus baris ke lima dan kolom ke dua dari data di titik 8 dan set C_{21}= M, untuk memperoleh :

	1	4
2	M	51
4	42	M

Dengan solusi $x_{24} = x_{42} = 1$, dengan nilai 51 + 42 = 93, dengan f = 93. Jadi, $f_L = C_{13} + C_{35} + C_{52} + f = 18 + 16 + 52 + 93 = 179$, hapus titik 9 karena $f_L > f^2_U$.

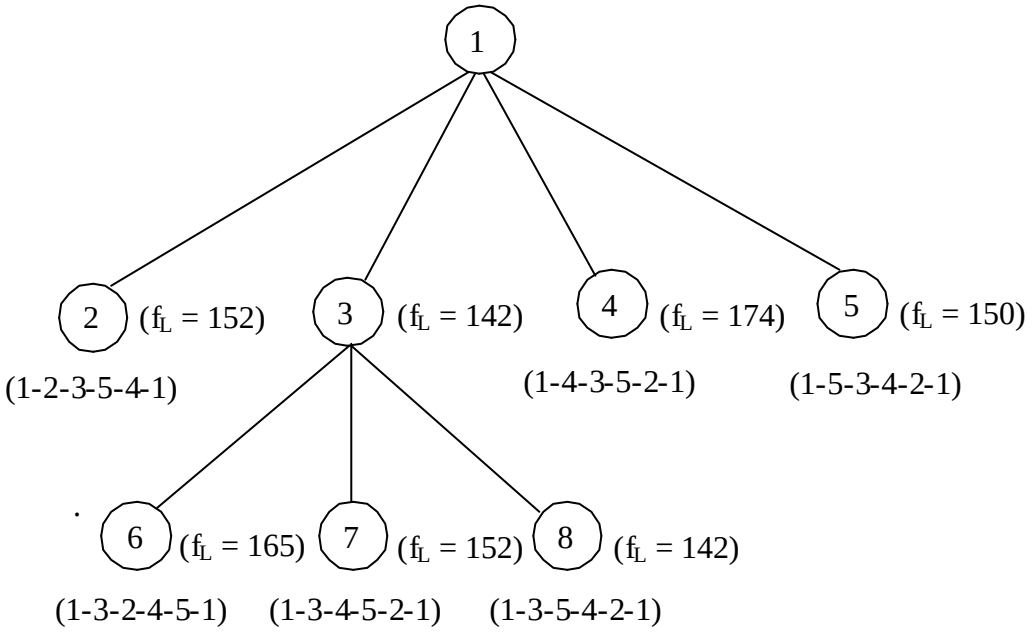
Pada titik 10 hapus baris ke lima dan kolom ke empat dari data di titik 8 dan set C_{41}= M, untuk memperoleh :

	1	2
2	29	M
4	M	51

Dengan solusi $x_{42} = x_{21} = 1$, dengan nilai 51 + 29 = 80, dengan f = 80. Jadi, $f_L = C_{13} + C_{35} + C_{54} + f = 18 + 16 + 28 + 80 = 142$.

Langkah 4 : Tidak ada titik aktif. Terdapat satu alternatif perjalan optimal : 1-3-5-4-2-1 dengan nilai 142.

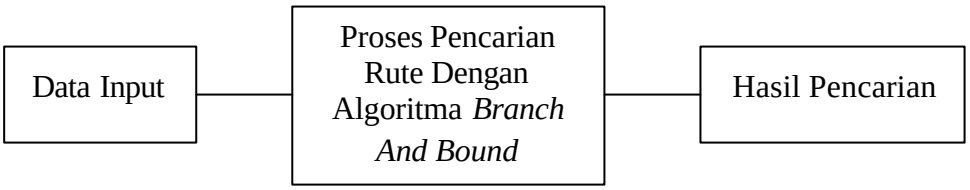
Pohon algoritma *branch and bound* pada contoh di atas ditunjukkan pada Gambar 3.8.



Gambar 3.2 Pohon algoritma *branch and bound*

3.5 Perancangan Sistem

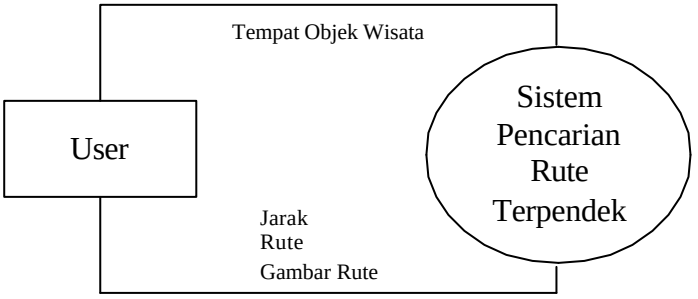
Sistem pencarian rute secara global dapat dilihat pada Gambar 3.3. Pada skema tersebut, terdapat tiga elemen utama. Pertama, data masukan yang menunjukkan awal dari proses pencarian rute, yang kedua proses pencarian yang menggunakan algoritma *Branch And Bound* dan yang ketiga adalah keluaran dari proses pencarian rute.



Gambar 3.3 Skema global sistem pencarian

Komponen selanjutnya dari skema global pencarian rute pada Gambar 3.3 adalah proses dalam sistem pencarian rute. Metode pendekatan yang dipilih untuk mendeskripsikan alur dan informasi di dalam prosesnya menggunakan diagram alir data (data flow diagram). Secara lebih rinci akan dijelaskan melalui beberapa representasi gambar dibawah ini.

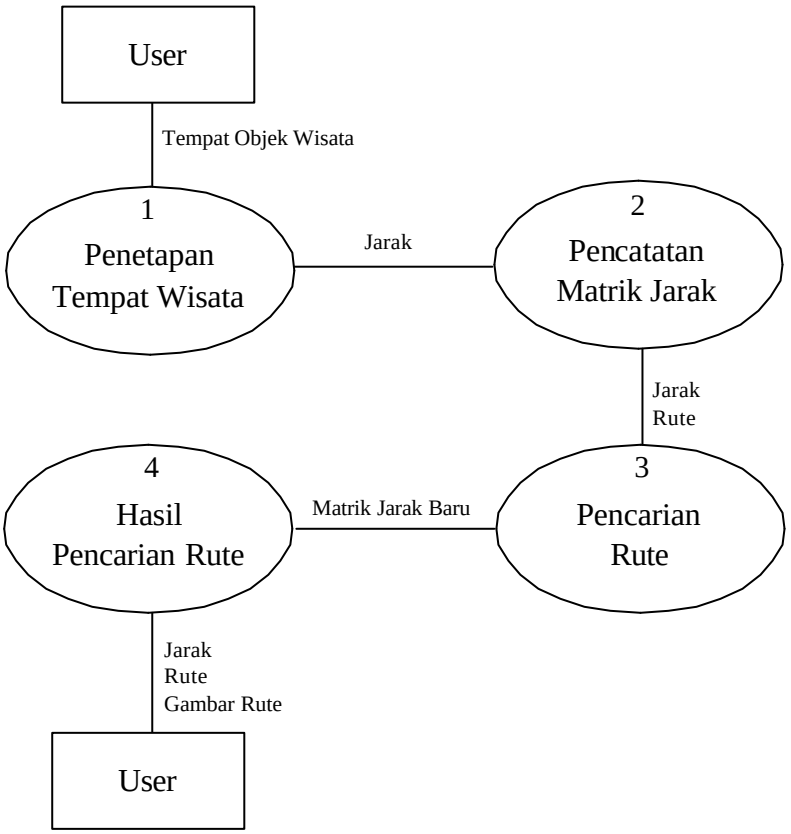
3.5.1 Diagram Konteks (DFD Level 0)



Gambar 3.4 Diagram konteks (DFD level 0)

Pada Gambar 3.4 yang menunjukkan Diagram konteks, entiti eksternalnya adalah *User*. *User* sebagai entiti eksternal berperan sebagai pemberi masukan bagi sistem (tempat objek wisata). Setelah proses pemasukan, sistem akan memproses masukan dan akan menghasilkan beberapa output diantaranya Jarak, Rute dan Gambar Rute. Selain sebagai pemberi masukan *user* berperan sebagai penerima keluaran (Jarak, Rute dan Gambar Rute) dari sistem.

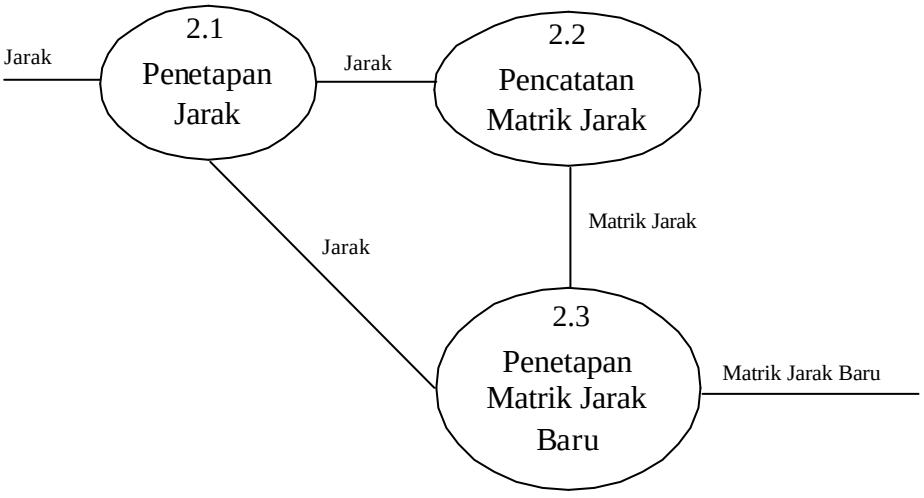
3.5.2 DFD Level 1



Gambar 3.5 DFD Level 1.

Ada empat proses utama yang terdapat pada DFD level 1 yang terdiri dari proses penetapan tempat wisata, proses pencatatan matrik jarak, proses pencarian rute, proses hasil pencarian rute. Setelah *user* memasukan data berupa tempat objek wisata maka data tersebut akan masuk ke dalam proses penetapan tempat wisata dilanjutkan dengan proses pencatatan matrik jarak dimana proses ini diambil dari variabel jarak yang menghasilkan matrik jarak baru yang siap untuk dihitung dalam proses pencarian rute. Di dalam proses pencarian rute ini menggunakan aturan algoritma *branch and bound*, proses ini akan menghasilkan rute perjalanan dengan jarak terpendek dari tempat wisata yang dipilih oleh *user*. Setelah itu masuk ke dalam proses hasil pencarian rute, proses ini akan menampilkan hasil yang berupa jarak, rute dan gambar rute yang akan diterima oleh *user.*

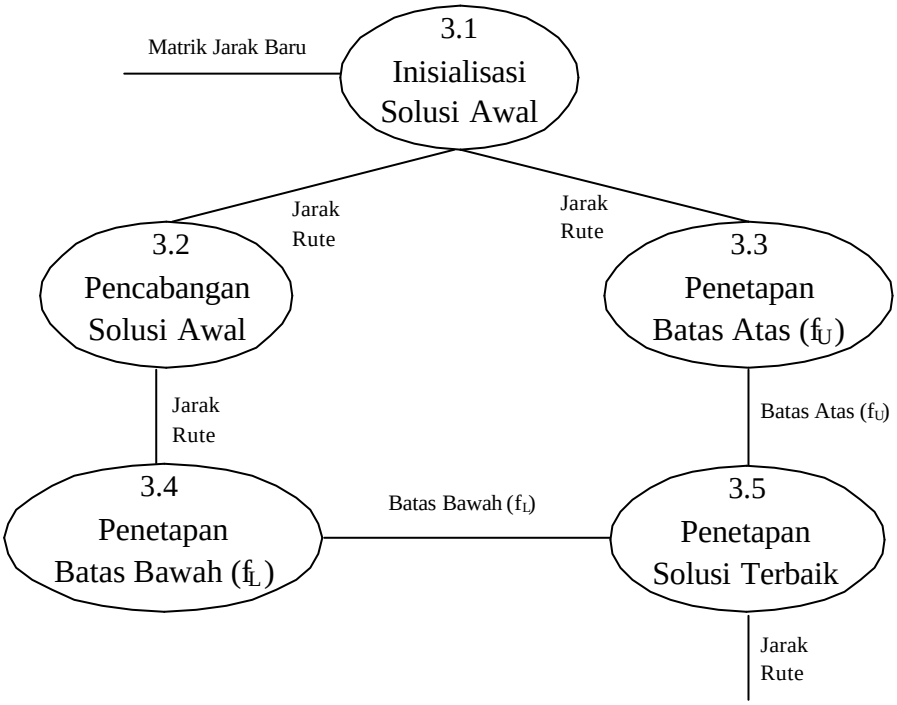
3.5.3 DFD Level 2 Proses 2



Gambar 3.6 DFD Level 2 Proses 2

Dalam DFD level 2 proses 2, masukan user dimasukkan ke proses penetapan jarak yang berupa jarak antara suatu tempat wisata ke tempat wisata lain yang telah dipilih oleh *user*. Setelah itu dilanjutkan dengan proses pencatatan matrik jarak tempat wisata yang sesuai dengan tempat wisata yang dipilih oleh *user*. Kemudian jarak dan matriks jarak ini masuk ke proses penetapan matrik jarak baru yang nantinya akan dihitung pada proses selanjutnya.

3.5.4 DFD Level 2 Proses 3

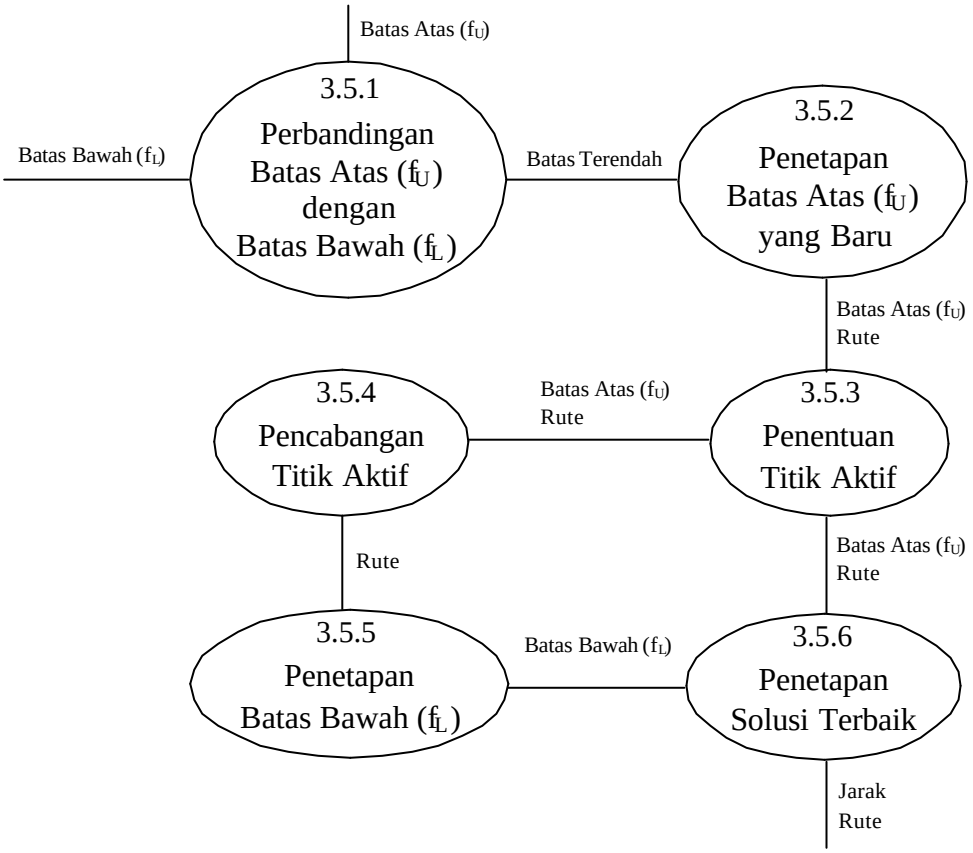


Gambar 3.7 DFD Level 2 Proses 3

Pada DFD ini matrik jarak baru dihitung dan ditetapkan pada proses inisialisasi solusi awal. Data yang mengalir berupa jarak dan rute yang masuk ke dalam proses penetapan batas atas dari solusi awal. Solusi

awal ini akan dicabangkan pada proses pencabangan solusi awal, dan kemudian hasil pencabangan ini dihitung pada proses penetapan batas bawah. Kemudian dilanjutkan ke proses penetapan solusi terbaik dengan menghitung batas atas dan batas bawah yang menghasilkan solusi jarak yang optimal.

3.5.5 DFD Level 3 Proses 3.5



Gambar 3.8 DFD Level 3 Proses 3.5

Pada DFD ini, batas atas dan batas bawah masuk ke dalam proses perbandingan batas atas dan batas bawah. Pilih batas dengan nilai jarak yang minimal, hasil ini menghasilkan suatu batas terendah yang optimal,

lalu kemudian masuk pada proses penetapan batas atas yang baru, data yang mengalir adalah batas atas dan rute. Nilai batas yang minimal ini masuk ke dalam proses penentuan titik aktif, batas baru yang minimal ini dijadikan titik aktif yang akan diperhitungkan dalam pencarian rute terpendek. Pada proses pencabangan titik aktif, titik ini dicabangkan untuk menghasilkan jarak yang minimal. Kemudian dilanjutkan proses penetapan solusi terbaik dengan menghitung batas atas dan batas bawah yang menghasilkan solusi jarak yang optimal, data yang mengalir adalah jarak dan rute. Kemudian dari keseluruhan proses ini menghasilkan gambar rute perjalanan terbaik yang akan ditampilkan kepada *user*.

BAB IV

IMPLEMENTASI

Bab ini menjelaskan mengenai cara mengimplementasikan setiap prosedur yang telah dirancang pada bab sebelumnya ke dalam bentuk bahasa pemrograman untuk membuat aplikasi. Kemudian akan dilakukan pengujian dan pembahasan tentang kemampuan dari aplikasi tersebut.

4.1 Spesifikasi Kebutuhan

Dalam proses untuk mengimplementasikan program Travelling Salesman Problem untuk pencarian rute terpendek dengan menggunakan algoritma *Branch and Bound*, dibutuhkan perangkat lunak. Perangkat lunak yang digunakan adalah sistem operasi Microsoft Windows 98 dan bahasa pemrograman Borland Delphi versi 5.0. Perangkat lunak ini membutuhkan suatu perangkat keras yang menunjang dan mempengaruhi terhadap kecepatan proses maupun daya tampung memori. Kebutuhan perangkat keras yang diperlukan adalah :

- Processor Intel Pentium 233 MMX
- HD. 2 Gb
- FDD. 1,44 Mb
- 128 Mb RAM

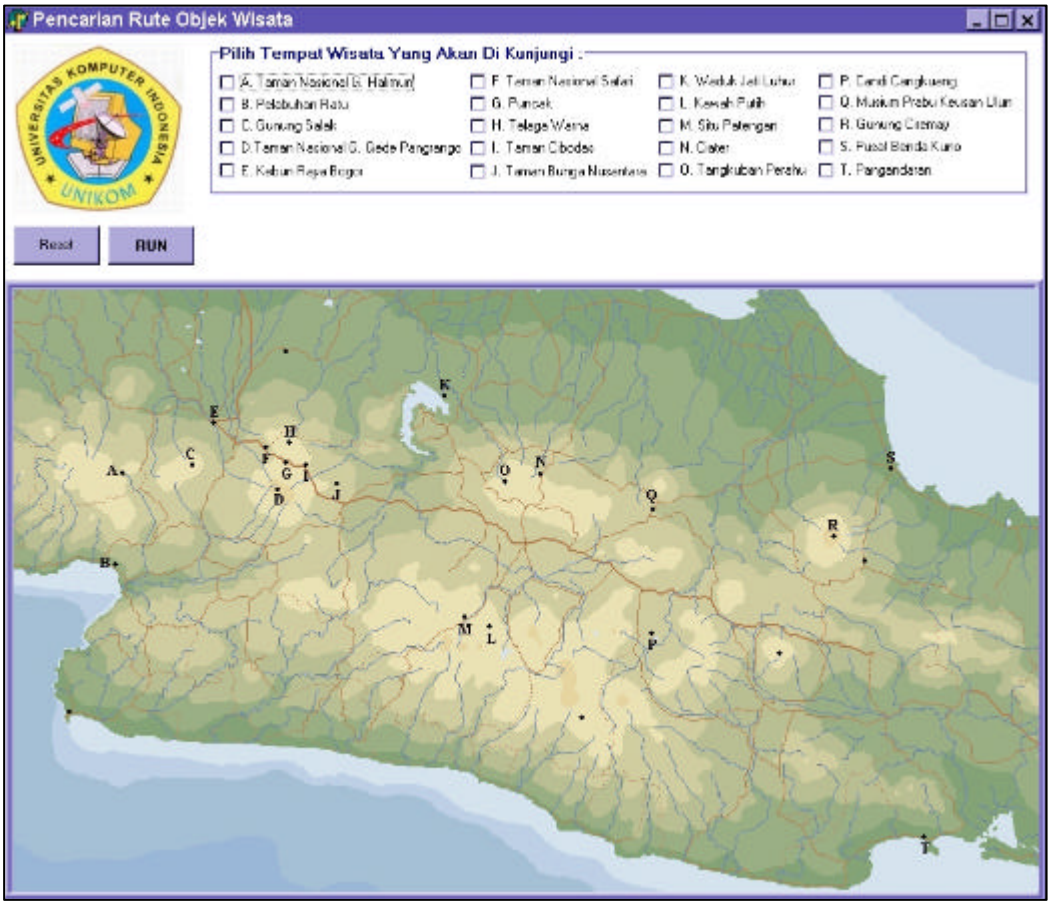
4.2 Tahap Implementasi Algoritma Branch And Bound

Pada algoritma *Branch and Bound* yang digunakan untuk menyelesaikan masalah Travelling Salesman Problem, banyak program aplikasi yang bisa digunakan, penulis mencoba mengimplementasikan

pemecahan masalah pencarian rute terpendek dengan algoritma *Branch and Bound*, tahap-tahap dan contoh algoritma *Branch and Bound* telah dibahas pada bab sebelumnya. Implementasi pada program aplikasi menggunakan perangkat lunak Borland Delphi 5.0.

Untuk membuat program, dibutuhkan beberapa variabel yang telah ditetapkan, seperti variabel matrik jarak yang tetap dan berukuran 20 x 20, matrik ini merupakan matrik jarak dari suatu objek wisata ke objek wisata lainnya. Matrik ini tidak berubah jika wisatawan memilih sembarang dari beberapa objek wisata, matrik yang telah dipilih ini menjadi matrik jarak baru yang akan dihitung pada saat pencarian rute terpendek. Variabel lainnya adalah nilai M yang telah diinisialisasikan dengan nilai 9999. Hal ini dimaksudkan bahwa jarak objek wisata A ke objek wisata A atau jarak objek wisata B ke objek wisata A jika bernilai M maka jarak objek wisata A ke objek wisata A dan jarak objek wisata B ke objek wisata A tidak ada jarak dan tidak ada perulangan, serta masih banyak lagi variabel-variabel yang dapat dilihat pada lampiran.

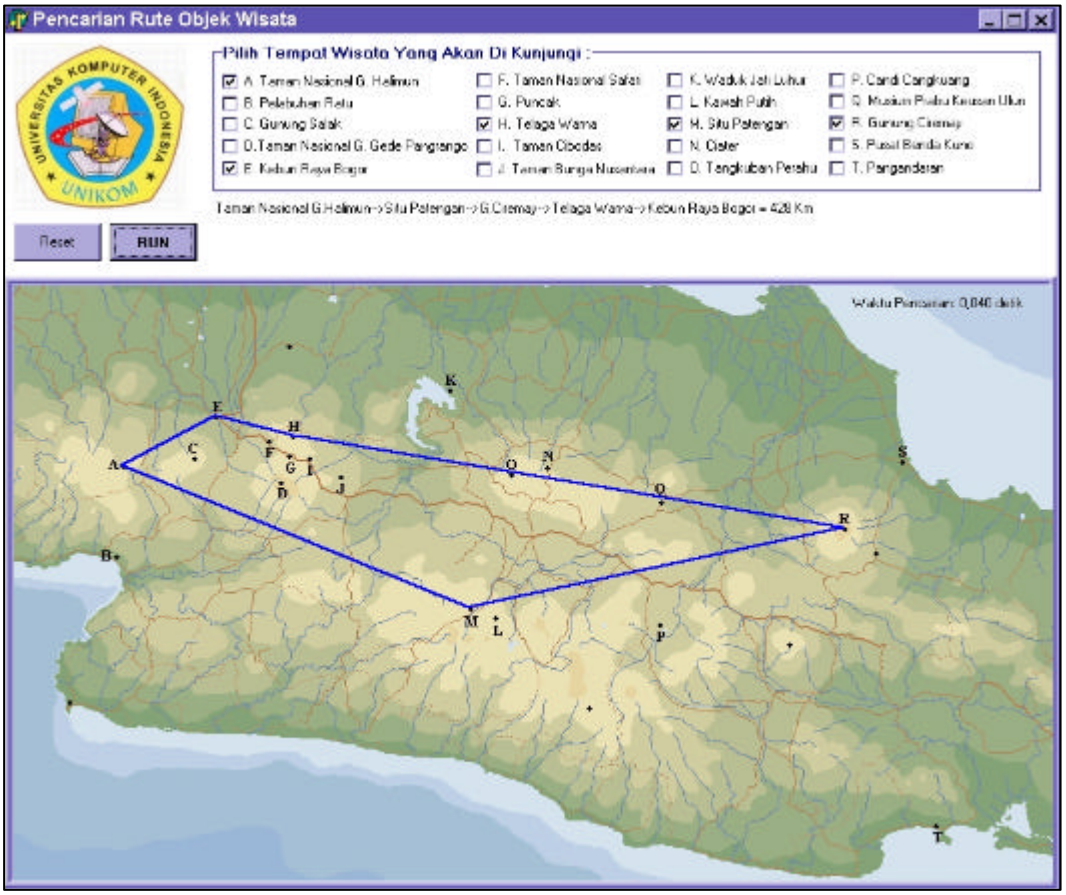
Pada program aplikasi yang penulis buat, tampilan form awal sebelum ada pemilihan tempat objek wisata adalah seperti yang ditunjukkan pada Gambar 4.1. Form awal ini berisi pilihan tempat objek wisata yang berisi 20 pilihan, label yang akan menunjukkan rute dan jarak, tombol Reset yang akan mengeset kembali pilihan tempat objek wisata, tombol Run untuk melakukan proses pencarian objek wisata yang akan dilalui bersama dengan jaraknya. Image yang akan menampung gambar peta, yang nantinya akan berisi garis-garis jalur yang terhubung.



Gambar 4.1 Form tampilan awal

4.3 Hasil Pengujian Program

Pada Gambar 4.2 merupakan tampilan form dengan 5 tempat objek wisata yang dipilih oleh wisatawan. Setelah tombol Run di klik maka program akan mencari rute terpendek dari objek wisata yang dipilih, dan kemudian menampilkannya pada peta yang berupa garis-garis jalur yang menghubungkan objek wisata satu ke objek wisata yang lain dengan jarak yang akan dilalui oleh wisatawan.



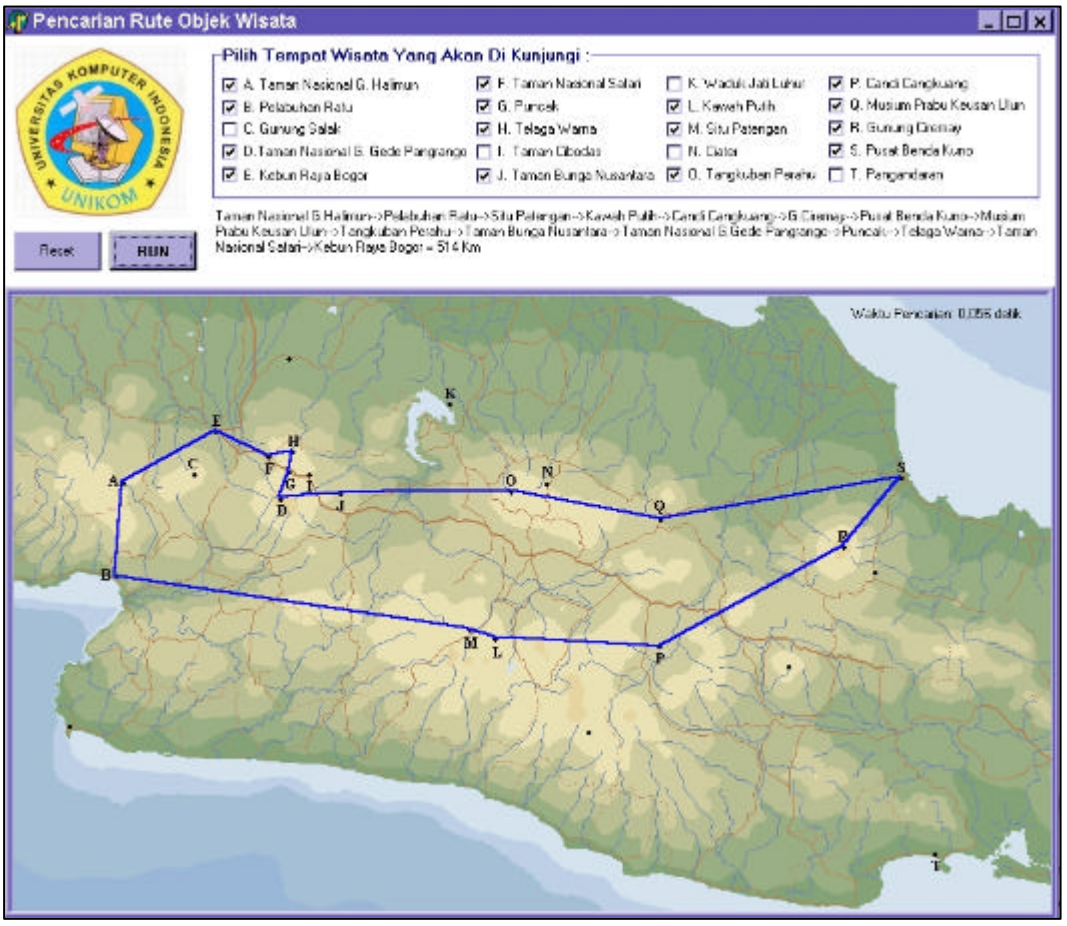
Gambar 4.2 Form tampilan 5 objek wisata.

Gambar 4.3 merupakan tampilan form dengan 10 tempat objek wisata yang dipilih oleh wisatawan. Setelah tombol Run di klik maka program akan mencari rute terpendek dari objek wisata yang dipilih, dan kemudian menampilkannya pada peta yang berupa garis-garis jalur yang menghubungkan objek wisata satu ke objek wisata yang lain dengan jarak yang akan dilalui oleh wisatawan.



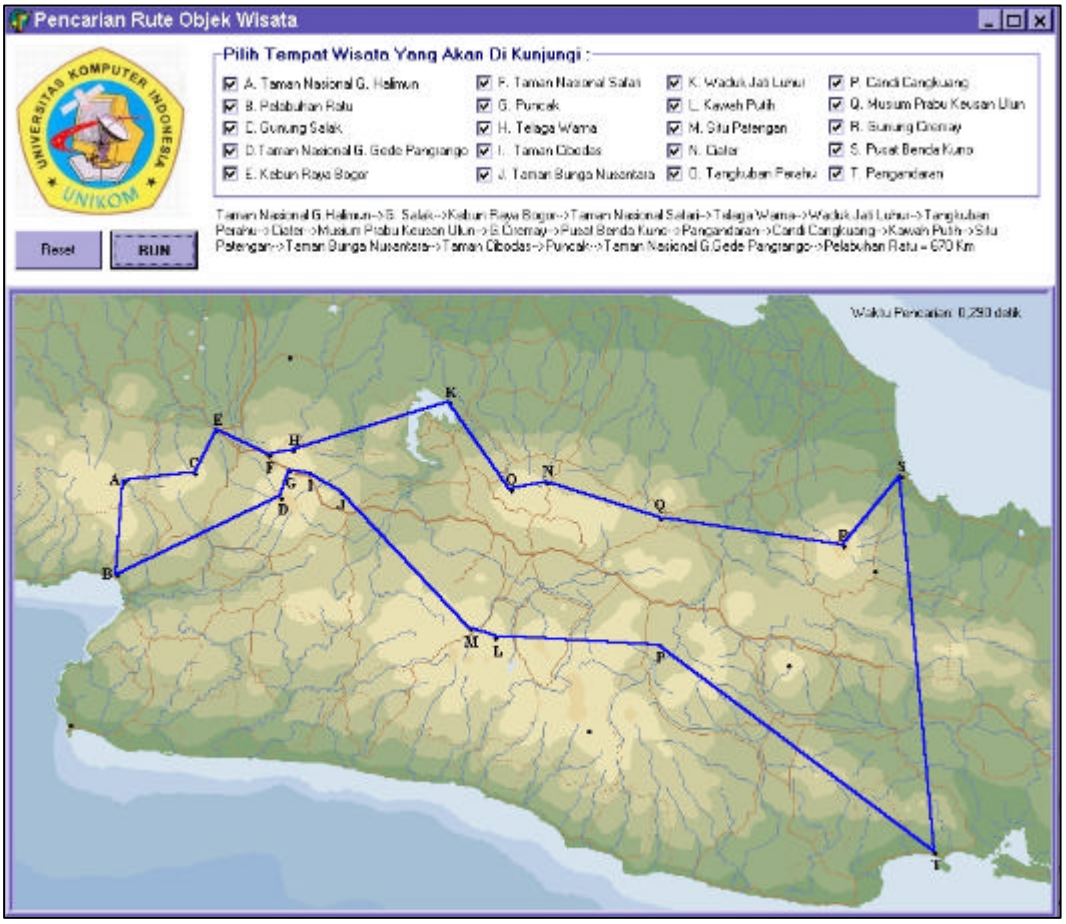
Gambar 4.3 Form tampilan 10 objek wisata.

Gambar 4.4 merupakan tampilan form dengan 15 tempat objek wisata yang dipilih oleh wisatawan. Setelah tombol Run di klik maka program akan mencari rute terpendek dari objek wisata yang dipilih, dan kemudian menampilkannya pada peta yang berupa garis-garis jalur yang menghubungkan objek wisata satu ke objek wisata yang lain dengan jarak yang akan dilalui oleh wisatawan.



Gambar 4.4 Form tampilan 15 objek wisata.

Gambar 4.5 merupakan tampilan form dengan 20 tempat objek wisata yang dipilih oleh wisatawan. Setelah tombol Run di klik maka program akan mencari rute terpendek dari objek wisata yang dipilih, dan kemudian menampilkannya pada peta yang berupa garis-garis jalur yang menghubungkan objek wisata satu ke objek wisata yang lain dengan jarak yang akan dilalui oleh wisatawan.



Gambar 4.5 Form tampilan 20 objek wisata.

4.4 Rekapitulasi Hasil Pengujian

Hasil pengujian pada program dari pencarian rute perjalanan objek wisata diperoleh jalur terbaik yang akan dilalui. Pencarian rute perjalanan ini membutuhkan waktu yang berbeda pada setiap pencarian. Dari hasil pengujian diperoleh sebagai berikut :

Tabel 4.1 Hasil Pengujian Program

Banyak Kota	**Hasil**		
	Rute Perjalanan	**Jarak (Km)**	**Waktu (detik)**
5	Taman Nasional G. Halimun Situ Patengan Gunung Ciremay Telaga Warna Kebun Raya Bogor Taman Nasional G. Halimun	428	0,040
10	Taman Nasional G. Halimun Kebun Raya Bogor Taman Cibodas Taman Bunga Nusantara Tangkuban Perahu Ciater G. Ciremay Pangandaran Kawah Putih Pelabuhan Ratu Taman Nasional G. Halimun	582	0,038
15	Taman Nasional G. Halimun Pelabuhan Ratu Situ Patengan Kawah Putih Candi Cangkuang G. Ciremay Pusat Benda Kuno Musium Prabu Keusan Ulun Tangkuban Perahu Taman Bunga Nusantara Taman Nasional G. Gede Pangrango Puncak Telaga Warna Taman Nasional Safari Kebun Raya Bogor Taman Nasional G. Halimun	514	0,056

Banyak Kota	Hasil		
	Rute Perjalanan	**Jarak (Km)**	**Waktu (detik)**
20	Taman Nasional G. Halimun G. Salak Kebun Raya Bogor Taman Nasional Safari Telaga Warna Waduk Jati Luhur Tangkuban Perahu Ciater Musium Prabu Keusan Ulun G. Ciremay Pusat Benda Kuno Pangandaran Candi Cangkuang Kawah Putih Situ Patengan Taman Bunga Nusantara Taman Cibodas Puncak Taman Nasional G. Gede Pangrango Pelabuhan Ratu Taman Nasional G. Halimun	670	0,290

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari pembahasan dan analisis yang telah dilakukan, maka dapat diambil beberapa kesimpulan, antara lain :

1. Pencarian rute perjalanan sebanyak 5, 10, 15 dan 20 tempat objek wisata dengan menggunakan algoritma *Branch And Bound* membutuhkan waktu yang singkat.
2. Di dalam program aplikasi ini, hasil output memberikan solusi rute perjalanan wisata dengan jarak yang minimal dari tempat objek wisata yang akan dikunjungi.
3. Untuk pencarian rute perjalanan tidak hanya menggunakan algoritma *Branch And Bound*, masih banyak algoritma-algoritma lain yang dapat menyelesaikan masalah Travelling Salesman Problem.
4. Dalam program aplikasi yang dibuat hanya menggunakan algoritma *Branch And Bound*, sehingga belum dapat disimpulkan bahwa algoritma *Branch And Bound* efektif dan efisien dalam pencarian rute perjalanan wisata.

5.2 Saran

Berdasarkan kesimpulan yang diperoleh, maka penulis dapat memberikan beberapa saran yang mungkin dapat digunakan untuk keperluan pengembangan selanjutnya :

1. Pada tugas akhir ini digunakan bentuk TSP simetris, untuk pengembangannya dapat digunakan bentuk TSP asimetris.
2. Untuk mengetahui kinerja algoritma Branch And Bound dalam menyelesaikan masalah Travelling Salesman Problem disarankan menggunakan lebih dari 20 tempat objek wisata.
3. Program aplikasi yang dibuat masih sederhana dan perlu pengembangan lebih lanjut pada program apilikasi.
4. Program aplikasi yang dibuat hanya membahas pemecahan masalah Travelling Salesman Problem dengan algoritma Branch And Bound saja. Untuk pengembangan lebih lanjut dapat ditambahkan algoritma lain yang dapat dijadikan bahan pertimbangan dalam menyelesaikan masalah Travelling Salesman Problem.
5. Jarak suatu tempat objek wisata ke tempat objek wisata lain dianggap garis lurus, hal ini mengurangi ketepatan jarak wisata yang akan ditempuh. Untuk perkembangan lebih lanjut, dapat dimasukan jarak satu tempat wisata ke tempat wisata lain dengan jarak yang sebenarnya.

LAMPIRAN

Matrik Jarak Untuk 20 Tempat Objek Wisata

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
A	M	26	21	44	31	43	47	50	53	62	96	113	106	120	110	158	153	205	220	253
B	26	M	36	51	52	56	57	61	62	68	107	109	101	125	115	154	156	207	224	246
C	21	36	M	25	15	20	26	29	33	41	75	97	89	100	90	140	133	185	200	236
D	44	51	25	M	27	12	7	14	10	17	55	73	65	76	66	115	109	161	176	212
E	31	52	15	27	M	17	24	23	30	40	67	99	91	95	85	140	129	181	194	237
F	43	56	20	12	17	M	8	7	12	22	53	81	74	79	69	122	112	165	178	220
G	47	57	26	7	24	8	M	7	6	16	50	75	67	73	63	116	107	159	174	213
H	50	61	29	14	23	7	7	M	9	17	46	78	70	72	62	117	106	159	172	215
I	53	62	33	10	30	12	6	9	M	10	45	69	62	67	57	110	101	153	167	207
J	62	68	41	17	40	22	16	17	10	M	40	60	53	59	48	100	92	144	159	198
K	96	107	75	55	67	53	50	46	45	40	M	67	63	36	30	90	68	119	129	188
L	113	109	97	73	99	81	75	78	69	60	67	M	7	46	41	47	58	103	124	140
M	106	101	89	65	91	74	67	70	62	53	63	7	M	46	40	53	63	109	129	147
N	120	125	100	76	95	79	73	72	67	59	36	46	46	M	10	56	34	87	100	152
O	110	115	90	66	85	69	63	62	57	48	30	41	40	10	M	60	44	96	110	159
P	158	154	140	115	140	122	116	117	110	100	90	47	53	56	60	M	36	61	84	99
Q	153	156	133	109	129	112	107	106	101	92	68	58	63	34	44	36	M	52	69	123
R	205	207	185	161	181	165	159	159	153	144	119	103	109	87	96	61	52	M	25	91
S	220	224	200	176	194	178	174	172	167	159	129	124	129	100	110	84	69	25	M	107
T	253	246	236	212	237	220	213	215	207	198	188	140	147	152	159	99	123	91	107	M

Keterangan :

A : Taman Nasional G. Halimun

B : Pelabuhan Ratu

C : Gunung Salak

D : Taman Nasional G. Gede Pangrango

E : Kebun Raya Bogor

F : Taman Nasional Safari

G : Puncak

H : Telaga Warna

I : Taman Cibodas

J : Taman Bunga Nusantara

K : Waduk Jati Luhur

L : Kawah Putih

M : Situ Patengan

N : Ciater

O : Tangkuban Perahu

P : Candi Cangkuang

Q : Musium Prabu Keusan Ulun

R : Gunung Ciremay

S : Pusat Benda Kuno

T : Pangandaran

```
unit UnitTSP;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons, jpeg;

const
  Maks = 20;
  M = 9999;
  Waktu : Double = 1000;
Type
     Arrnn = Array[1..maks,1..maks] of integer;
     Arrn  = Array[1..maks] of integer;
const
   NamaWisata : Array[1..maks]of string = ('Taman Nasional G.Halimun','Pelabuhan Ratu','G. Salak','Taman Nasional G.Gede
                                           Pangrango','Kebun Raya Bogor','Taman Nasional Safari','Puncak','Telaga Warna',
                                          'Taman Cibodas','Taman Bunga Nusantara','Waduk Jati Luhur','Kawah Putih',
                                          'Situ Patengan','Ciater','Tangkuban Perahu','Candi Cangkuang','Musium Prabu Keusan Ulun',
                                          'G.Ciremay','Pusat Benda Kuno','Pangandaran');
  Jarak : Arrnn = (
      //  A ,  B , C ,  D , E ,  F , G , H ,   I ,   J ,  K ,  L , M , N , O ,  P , Q , R ,  S ,  T ,
      (   M,  26,  21, 44,  31,  43,  47, 50,  53,  62,  96, 113, 106, 120, 110, 158, 153, 205, 220, 253),  // TamNas G. Halimun
      (  26,   M,  36,  51,  52,  56,  57, 61,  62,  68, 107, 109, 101, 125, 115, 154, 156, 207, 224, 246),  // Pelabuhan Ratu
      (  21,  36,   M,  25,  15,  20,  26, 29,  33,  41,  75,  97,  89, 100,  90, 140, 133, 185, 200, 236),  // G. Salak
      (  44,  51,  25,   M,  27,  12,   7, 14,  10,  17,  55,  73,  65,  76,  66, 115, 109, 161, 176, 212),  // TamNas G. Gede Pangrango
      (  31,  52,  15,  27,   M,  17,  24, 23,  30,  40,  67,  99,  91,  95,  85, 140, 129, 181, 194, 237),  // Kebun Raya Bogor
      (  43,  56,  20,  12,  17,   M,   8,  7,  12,  22,  53,  81,  74,  79,  69, 122, 112, 165, 178, 220),  // TamNas Safari
      (  47,  57,  26,   7,  24,   8,   M,  7,   6,  16,  50,  75,  67,  73,  63, 116, 107, 159, 174, 213),  // Puncak
      (  50,  61,  29,  14,  23,   7,   7,  M,   9,  17,  46,  78,  70,  72,  62, 117, 106, 159, 172, 215),  // Telaga Warna
      (  53,  62,  33,  10,  30,  12,   6,  9,   M,  10,  45,  69,  62,  67,  57, 110, 101, 153, 167, 207),  // Taman Cibodas
      (  62,  68,  41,  17,  40,  22,  16,  17,  10,   M,  40,  60,  53,  59,  48, 100,  92, 144, 159, 198),  // Taman Bunga Nusantara
      (  96, 107,  75,  55,  67,  53,  50,  46,  45,  40,   M,  67,  63,  36,  30,  90,  68, 119, 129, 188),  // Waduk Jati Luhur
      ( 113, 109,  97,  73,  99,  81,  75,  78,  69,  60,  67,   M,   7,  46,  41,  47,  58, 103, 124, 140),  // Kawah Putih
      ( 106, 101,  89,  65,  91,  74,  67,  70,  62,  53,  63,   7,   M,  46,  40,  53,  63, 109, 129, 147),  // Situ Patengan
      ( 120, 125, 100,  76,  95,  79,  73,  72,  67,  59,  36,  46,  46,   M,  10,  56,  34,  87, 100, 152),  // Ciater
      ( 110, 115,  90,  66,  85,  69,  63,  62,  57,  48,  30,  41,  40,  10,   M,  60,  44,  96, 110, 159),  // Tangkuban Perahu
      ( 158, 154, 140, 115, 140, 122, 116, 117, 110, 100,  90,  47,  53,  56,  60,   M,  36,  61,  84,  99),  // Candi Cangkuang
      ( 153, 156, 133, 109, 129, 112, 107, 106, 101,  92,  68,  58,  63,  34,  44,  36,   M,  52,  69, 123),  // Musium Prabu Keusan Ulun
      ( 205, 207, 185, 161, 181, 165, 159, 159, 153, 144, 119, 103, 109,  87,  96,  61,  52,   M,  25,  91),  // G. Ciremay
      ( 220, 224, 200, 176, 194, 178, 174, 172, 167, 159, 129, 124, 129, 100, 110,  84,  69,  25,   M, 107),  // Pusat Benda Kuno
      ( 253, 246, 236, 212, 237, 220, 213, 215, 207, 198, 188, 140, 147, 152, 159,  99, 123,  91, 107,   M)); // Pangandaran

  PosX : arrn = (90,84,150,221,168,212,229,232,246,272,362,401,379,444,413,538,539,691,739,768);
  PosY : arrn = (155,234,148,168,112,133,146,129,149,164,89,285,278,156,162,293,186,209,151,466);

Type
  Trute = record
        bwisata    : integer;
        jalur : Array[1..maks]of integer;
        end;

  TTSP = class(TForm)
    GroupBox1: TGroupBox;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    CheckBox6: TCheckBox;
    CheckBox7: TCheckBox;
    CheckBox8: TCheckBox;
    CheckBox9: TCheckBox;
    CheckBox10: TCheckBox;
    CheckBox11: TCheckBox;
    CheckBox12: TCheckBox;
    CheckBox13: TCheckBox;
    CheckBox14: TCheckBox;
    CheckBox15: TCheckBox;
    CheckBox16: TCheckBox;
    CheckBox17: TCheckBox;
    CheckBox18: TCheckBox;
    CheckBox19: TCheckBox;
    CheckBox20: TCheckBox;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    TimeLb2: TLabel;
    Reset: TButton;
```

```
    Panel1: TPanel;
    Image1: TImage;
    Image2: TImage;
    Image3: TImage;
    Procedure FormCreate(Sender: TObject);
    Procedure BitBtn1Click(Sender: TObject);
    Procedure ResetClick(Sender: TObject);
    Procedure tampil(rute:arrn);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  TSP : TTSP;
  bwisata : integer;
  MJarakBaru : Arrnn;
  jalur,n : integer;
  rute : arrn;
  bjalur,count : integer;
  s,p : string;
  wisata : array[1..maks] of byte;
  StartTime, EndTime : Double;
  X : longint;

implementation
{$R *.DFM}

Procedure TTSP.FormCreate(Sender: TObject);
Begin
    Image1.Picture := Image2.Picture ;
end;

Procedure TTSP.ResetClick(Sender: TObject);
Begin
    Checkbox1.Checked:=false;
    Checkbox2.Checked:=false;
    Checkbox3.Checked:=false;
    Checkbox4.Checked:=false;
    Checkbox5.Checked:=false;
    Checkbox6.Checked:=false;
    Checkbox7.Checked:=false;
    Checkbox8.Checked:=false;
    Checkbox9.Checked:=false;
    Checkbox10.Checked:=false;
    Checkbox11.Checked:=false;
    Checkbox12.Checked:=false;
    Checkbox13.Checked:=false;
    Checkbox14.Checked:=false;
    Checkbox15.Checked:=false;
    Checkbox16.Checked:=false;
    Checkbox17.Checked:=false;
    Checkbox18.Checked:=false;
    Checkbox19.Checked:=false;
    Checkbox20.Checked:=false;
end;

Procedure TTSP.Tampil(rute:arrn);
var
    i   : integer;
    jw : arrn;
    stjalur: String;
Begin
    image1.picture:=image2.picture;
    image1.canvas.Pen.color:=clblue;
    image1.canvas.pen.width:=2;
    for i := 1 to bwisata do
    image1.canvas.brush.color := clyellow;
    image1.Canvas.MoveTo(PosX[jw[1]],PosY[jw[1]]);
    for i:=2 to bwisata do
    Begin
        image1.Canvas.LineTo(PosX[jw[i]],PosY[jw[i]]);
        stjalur:=stjalur+'-->'+Namawisata[jw[i]];
    end;
    image1.Canvas.LineTo(PosX[jw[1]],PosY[jw[1]]);
    stjalur:=stjalur+' = '+inttostr(jalur)+' Km';
    label1.caption:=stjalur;
end;
```

```
Procedure BABTSP(bwisata,M : integer;   var MJarakBaru : Arrnn;
                                        var Rute  : Arrn;
                                        var Jalur  : integer);
var
   Backptr,Fwdptr,Terbaik,Kolom,Baris : Arrn;
   i,Index :integer;

        Procedure EXPLORE(Edges,Cost : integer; var Baris,Kolom : Arrn);
        var
           Avoid,C,Colrowval,First,I,J,Last,Lowerbound,Most,R,Size :integer;
           NEWBaris,NEWKolom,BarisRED,KolomRED :Arrn;

                Function MIN(I,J:integer):integer;
                begin
                    if I <= J then
                        MIN:=I
                    else
                        MIN :=J;
                end;

                Function REDUCE(var Baris,Kolom,BarisRED,KolomRED:Arrn):integer;
                var I,J,Rvalue,Temp:integer;
                begin
                    Rvalue := 0;
                    for I := 1 to Size do
                        begin
                            for J := 1 to Size do
                                Temp := MIN(Temp,MJarakBaru[Baris[I],Kolom[J]]);
                            if Temp > 0 then
                                begin
                                    for J := 1 to Size do
                                        if MJarakBaru[Baris[I],Kolom[J]] < M then
                                            MJarakBaru[Baris[I],Kolom[J]] := MJarakBaru[Baris[I],Kolom[J]]-TEMP;
                                    Rvalue := Rvalue +Temp
                                end;
                                BarisRED[I]:=Temp
                        end;
                    for J:=1 to Size do
                        begin
                            for I := 1 to Size do
                                Temp := MIN(Temp,MJarakBaru[Baris[I],Kolom[J]]);
                            if Temp > 0 then
                                begin
                                    for I:=1 to Size do
                                        if MJarakbaru[Baris[I],Kolom[J]] < M then
                                          MJarakBaru[Baris[I],Kolom[J]]:=MJarakBaru[Baris[I],Kolom[J]]-Temp;
                                    Rvalue := Rvalue +Temp
                                end;
                                KolomRED[J]:=Temp
                        end;
                end;

                Procedure BESTEDGE(var R,C,Most:integer);
                var I,J,K,Mincolelt,Minrowelt,Zeroes : integer;
                begin
                    Most := -M;
                    for I:=1 to Size do
                        for J :=1 to Size do
                            if MJarakBaru[Baris[I],Kolom[J]] = 0 then
                                begin
                                    Minrowelt:=M;  Zeroes:=0;
                                    for K:=1 to SIZE do
                                    if MJarakBaru[Baris[I],Kolom[K]] = 0  then
                                        Zeroes := Zeroes +1
                                    else
                                        Minrowelt:=MIN(Minrowelt,MJarakBaru[Baris[I],Kolom[K]]);
                                        if Zeroes > 1 then
                                            Minrowelt:=0;
                                        Mincolelt:=M;  Zeroes:=0;
                                    for K:=1 to SIZE do
                                    if MJarakBaru[Baris[K],Kolom[J]] = 0 then
                                        Zeroes := Zeroes+1
                                    else
                                        Mincolelt:=MIN(Mincolelt,MJarakBaru[Baris[K],Kolom[J]]);
                                        if Zeroes > 1 then
                                            Mincolelt:=0;
                                end
                end;
```

```
        begin
            Size:=bwisata-Edges;
            Cost:=Cost+REDUCE(Baris,Kolom,BarisRED,KolomRED);
            if Cost < Jalur then
                if EDGES = (bwisata-2) then
                    begin
                        for I:=1 to bwisata do
                            Terbaik[I]:= Fwdptr[I];
                            if MJarakBaru[Baris[1],Kolom[1]] = M then
                                Avoid:=1
                            else
                                Avoid:=2;
                        Terbaik[Baris[1]]:=Kolom[3-Avoid];
                        Terbaik[Baris[2]]:=Kolom[Avoid];
                        Jalur:=COST
                    end
                else
                    begin
                        BESTEDGE(R,C,Most);
                        Lowerbound := Cost + Most;
                        Fwdptr[Baris[R]]:=Kolom[C];
                        Backptr[Kolom[C]]:=Baris[R];
                        Last:=Kolom[C];
                        while Fwdptr[Last] <> 0 do
                            Last:=Fwdptr[Last];
                            First:=Baris[R];
                        while Backptr[First] <> 0 do
                            First:=Backptr[First];
                        Colrowval:=MJarakBaru[Last,First];
                        MJarakBaru[Last,First]:=M;
                        for I:=1 to R-1 do
                            NEWBaris[I]:=Baris[I];
                        for I:=R to SIZE-1 do
                            NEWBaris[I]:=Baris[I+1];
                        for I:=1 to C-1 do
                            NEWKolom[I]:=Kolom[I];
                        for I:=C to SIZE-1 do
                            NEWKolom[I]:=Kolom[I+1];
                        EXPLORE(Edges+1,Cost,NEWBaris,NEWKolom);
                        MJarakBaru[Last,First]:=Colrowval;
                        Backptr[Kolom[C]]:=0;
                        Fwdptr[Baris[R]]:=0;
                        if Lowerbound < Jalur then
                            begin
                                MJarakBaru[Baris[R],Kolom[C]]:=M;
                                EXPLORE(Edges,Cost,Baris,Kolom);
                                MJarakBaru[Baris[R],Kolom[C]]:=0;
                            end
                    end;
        end;

begin
    for I:=1 to bwisata do
        begin
            Baris[I]:=I;
            Kolom[I]:=I;
            Fwdptr[I]:=0;
            Backptr[I]:=0
        end;
    Jalur:=M;
    EXPLORE(0,0,Baris,Kolom);
    Index:=1;
    for I:=1 to bwisata do
        begin
            Rute[I]:=Index;
            Index:=Terbaik[Index]
        end
end;

Procedure TTSP.BitBtn1Click(Sender: TObject);
Var
  i,j       : integer;
  rute      : arrn;
  MJarakBaru : Arrnn;

Begin
    image1.picture:=image2.picture;
    bwisata := 0;
    if Checkbox1.Checked = true then
```

```
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 1;
end;
if Checkbox2.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 2;
end;
if Checkbox3.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 3;
end;
if Checkbox4.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 4;
end;
if Checkbox5.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 5;
end;
if Checkbox6.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 6;
end;
if Checkbox7.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 7;
end;
if Checkbox8.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 8;
end;
if Checkbox9.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 9;
end;
if Checkbox10.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 10;
end;
if Checkbox11.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 11;
end;
if Checkbox12.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 12;
end;
if Checkbox13.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 13;
end;
if Checkbox14.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 14;
end;
if Checkbox15.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 15;
end;
if Checkbox16.Checked = true then
Begin
        bwisata := bwisata + 1;
        wisata[bwisata]:= 16;
end;
```

```
        if Checkbox17.Checked = true then
        Begin
                bwisata := bwisata + 1;
                wisata[bwisata]:= 17;
        end;
        if Checkbox18.Checked = true then
        Begin
                bwisata := bwisata + 1;
                wisata[bwisata]:= 18;
        end;
        if Checkbox19.Checked = true then
        Begin
                bwisata := bwisata + 1;
                wisata[bwisata]:= 19;
        end;
        if Checkbox20.Checked = true then
        Begin
                bwisata := bwisata + 1;
                wisata[bwisata]:= 20;
        end;

        for i := 1 to bwisata do
           Begin
                for j := 1 to bwisata do
                   Begin
                        MJarakBaru[i,j]:= Jarak[Wisata[i],wisata[j]];
                   end;
           end;
        begin
                x := 0;
                StartTime := GetTickCount;    { detik awal }
                while (x < 10000000) do
                     x:= x+1;
                BABTSP(bwisata, M, MJarakBaru, rute, jalur);
                EndTime := GetTickCount - StartTime;
                TimeLb2.caption := (Format('Waktu Pencarian: %0.3f detik',[EndTime/Waktu]));
                Tampil(rute);
        end;
end;

end.
```