BAB V

IMPLEMENTASI PERANGKAT LUNAK

5.1 Implementasi secara Umum

Persoalan *minimum spanning tree* dengan algoritma genetik diimplementasikan dengan bahasa pemrograman Borland Delphi 6.0. Implementasi merupakan salah satu tahap di mana sistem dioperasikan pada keadaan yang sebenarnya, sehingga akan diketahui apakah sistem yang dibuat benar-benar sesuai dengan yang diharapkan.

Sebelum program diimplementasikan, maka program harus bebas dari kesalahan. Kesalahan program yang mungkin terjadi antara lain karena kesalahan penulisan (*coding*), kesalahan proses, atau kesalahan logika.

5.2 Alasan Pemilihan Bahasa Pemrograman

Borland Delphi 6.0 adalah bahasa pemrograman yang dirancang untuk bekerja di dalam *platform* Microsoft Windows. Selain itu Borland Delphi menggunakan bahasa *Object* Pascal yang berarti merupakan bahasa pemrograman terstruktur, sehingga memudahkan dalam *coding* maupun *debugging*. Disamping itu pula Borland Delphi juga mempunyai kemampuan operasi numerik serta menyediakan fasilitas grafik dan Windows API yang sangat menunjang dalam penyelesaian persoalan *minimum spanning tree* dengan algoritma genetik.

Sistem optimasi ini dapat berjalan dengan baik, apabila memenuhi standar minimal dari perangkat keras (*hardware*). Perangkat keras yang digunakan minimal memiliki spesifikasi sebagai berikut :

1. Satu unit komputer dengan spesifikasi minimum Prosesor Pentium II 300 MHz, RAM 128 MB, Hard Disk 2 GB.
2. Satu unit printer Dot Matrix, LaserJet atau Deskjet (optional) dan kertas
3. Monitor VGA atau SVGA
4. Mouse
5. Keyboard

Perangkat lunak yang dibutuhkan untuk pengembangan dan implementasi persoalan *minimum spanning tree* dengan algoritma genetik ini menggunakan :

1. Sistem Operasi Windows 9.x atau 2000 atau yang lebih tinggi
2. Bahasa Pemrograman Borland Delphi 6.0 untuk pembuatan perangkat lunak
3. Microsoft Word 97 atau yang lebih tinggi.
4. Software untuk membuka gambar (Paint, ACDSee, Imaging).

5.3 Tahap Pembuatan Perangkat Lunak

Dalam pembuatan perangkat lunak ini terdapat beberapa tahapan yang dapat dibedakan menjadi 3 tahap yaitu :

1. Tahap Pemrograman Visual

   Pada tahap ini yang dilakukan adalah merancang *form* yang akan digunakan program serta kontrol kontrol yang diperlukan. Perancangan tersebut ditangani oleh paket-paket yang disediakan oleh Borland Delphi 6.0.

2. Tahap Penulisan Kode (*Coding*)

   Pada tahap ini yang dilakukan adalah menuliskan kode – kode yang diimplemetasikan dengan *procedure* untuk selanjutnya ditempatkan atau dipanggil pada kontrol-kontrol (*object*) yang dipakai. Penulisan kode ini dilakukan dengan menggunakan *Text Editor* milik Borland Delphi 6.0.

3. Tahap *Debugging*

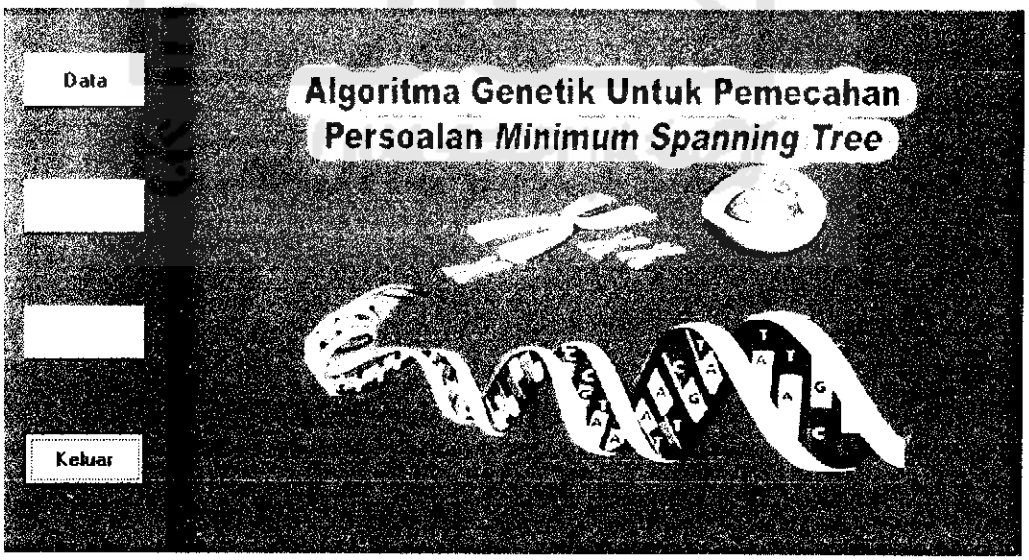
   Pada tahap ini dilakukan pengujian terhadap perangkat lunak yang dibuat. Metode pengujian yang digunakan ialah metode *White Box*, dimana tiap langkah yang menghasilkan *output* ditulis ulang ke dalam *FileLog* dan diteliti keabsahannya melalui *software* Notepad yang disediakan oleh Sistem Operasi Windows.

5.4 Implementasi Antarmuka / Form

5.4.1 Form Menu Utama

Form utama ini adalah form awal yang ditampilkan begitu perangkat lunak dijalankan. Form ini merupakan gabungan dari beberapa antarmuka yaitu : form data, form proses dan form keluaran. Pada bagian form ini terdapat pilihan menu berbasis tombol. Tombol-tombol yang tersedia didalam form ini adalah :

Gambar 5.1 Tampilan Menu Utama



Gambar 5.2 Tampilan Awal Keseluruhan Program

1. Tombol *Data*

   Tombol ini digunakan untuk menampilkan form *Data*.

2. Tombol *Genetik*

   Tombol ini digunakan untuk menampilkan form *Genetik*.

3. Tombol *Keluaran*

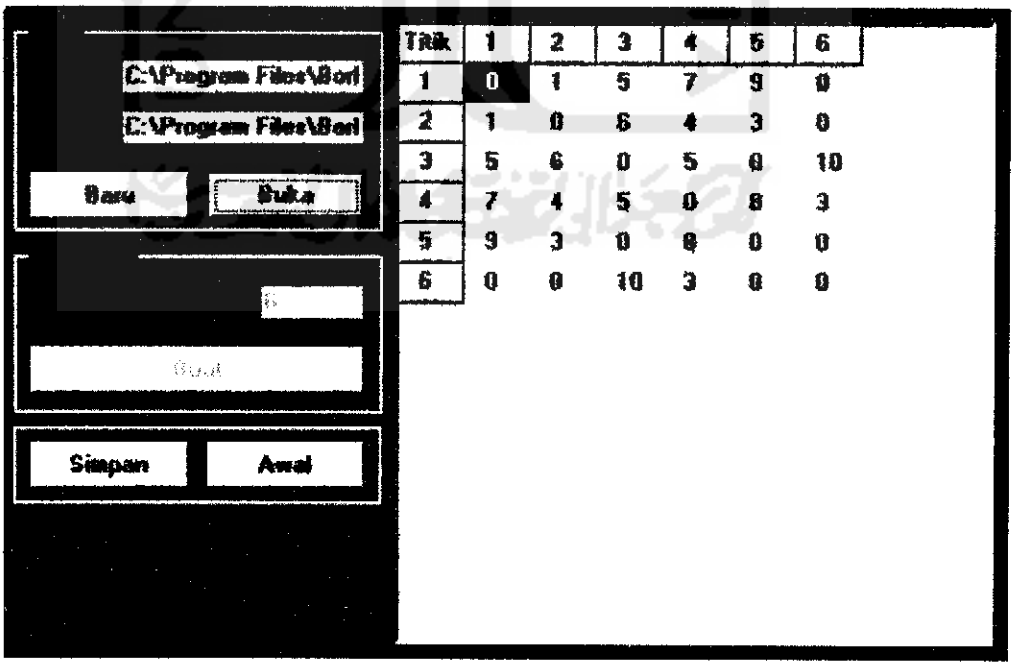
   Tombol ini digunakan untuk menampilkan form *Keluaran*.

4. Tombol *Keluar*

   Tombol ini digunakan untuk keluar dari *Program*.

5.4.2 Form Data

Form *Data* ini digunakan untuk menginisialisasi data yang akan diproses. Pada form ini dibagi menjadi 5 bagian yaitu bagian inisialisasi file, inisialisasi jaringan, data koordinat, peta awal dan input data.



Trk	1	2	3	4	5	6
1	0	1	5	7	9	0
2	1	0	6	4	3	0
3	5	6	0	5	0	10
4	7	4	5	0	8	3
5	9	3	0	8	0	0
6	0	0	10	3	0	0

Gambar 5.3 Tampilan Keseluruhan Form *Data*

Pada bagian inisialisasi *File*, terdapat 2 tombol yaitu :

1. Tombol *Baru*

   Tombol ini digunakan untuk membuat file data baru yang akan disimpan ke dalam ekstensi *.txt*. Jika tombol ini ditekan maka akan ditampilkan *Save Dialog*, kemudian diisikan nama file yang baru.

2. Tombol *Buka*

   Tombol ini digunakan untuk membuka *file* yang telah disimpan atau dibuat sebelumnya. Jika tombol ini ditekan maka akan ditampilkan *Open Dialog*, kemudian dipilih nama *file* yang akan dibuka.

Nama file yang dibuka akan ditampilkan ke dalam *texbox Nama*, sedangkan nama file log akan ditampilkan ke *texbox log* dan disimpan ke dalam suatu file yang mempunyai nama yang sama dengan nama file yang dibuka atau file buat, hanya ekstensinya diubah ke *.log*.

Gambar 5.4 Tampilan Bagian Inisialisasi *File* Form Data

Pada bagian inisialisasi jaringan, terdapat 2 tombol :

1. Tombol *Simpan*

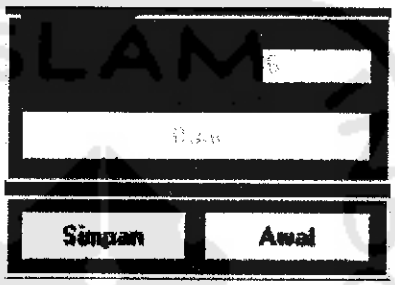
   Tombol ini digunakan untuk menyimpan *file* atau data yang telah diinputkan ke dalam sistem.

2. Tombol *Buat*

   Tombol ini digunakan untuk membuat ukuran *Grid* sesuai dengan jumlah titik.

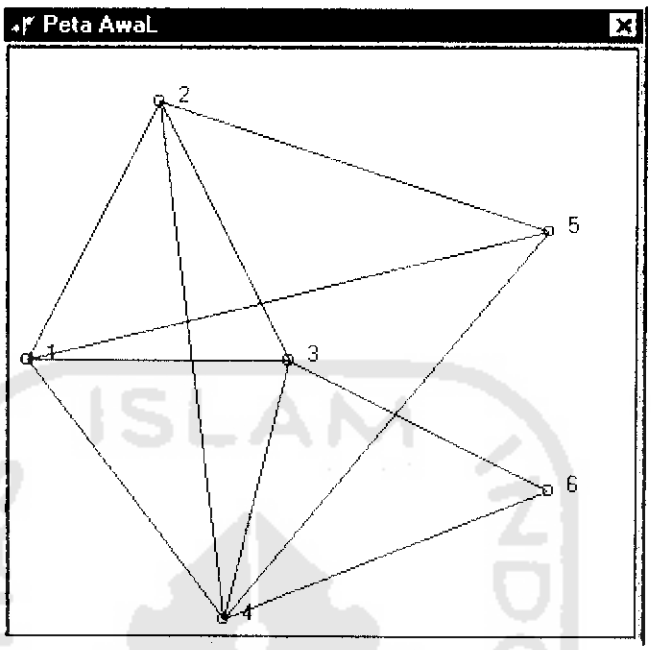
3. Tombol *Awal*

   Tombol ini digunakan untuk kembali ke menu awal.



Gambar 5.5 Tampilan Bagian Inisialisasi *Jaringan* Form Data.

Pada bagian gambar peta awal akan ditunjukkan titik-titik yang menunjukkan sebuah jaringan. Titik-titik tersebut merupakan representasi dari data yang telah dibuka.



Gambar 5.6 Tampilan Bagian Peta Awal

Pada saat menginput data ke sebuah *Grid* yaitu data jarak, di mana data tersebut berupa angka, maka tombol keyboard yang diaktifkan hanyalah tombol angka 0 sampai 9, koma, *Enter*, *BackSpace*, dan *Arrows*.

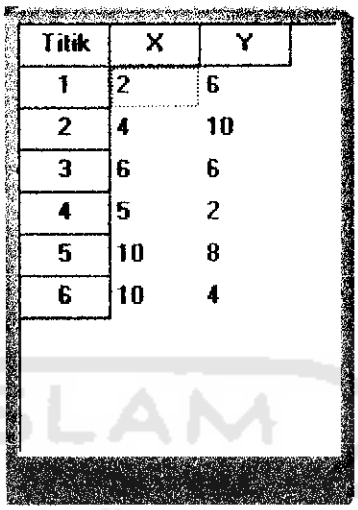
Dengan demikian kesalahan pengisian data yang bukan berupa angka dapat dihindarkan. Untuk titik-titik yang terhubung akan diberikan suatu angka tertentu. Sedangkan untuk titik yang tidak terhubung akan diberikan angka 0. Untuk menyimpan tekan tombol *Save*.

Titik	1	2	3	4	5	6
1	0	1	5	7	9	0
2	1	0	6	4	3	0
3	5	6	0	5	0	10
4	7	4	5	0	0	3
5	9	3	0	0	0	0
6	0	0	10	3	0	0

Gambar 5.7 Tampilan Bagian Input Data Jarak Form *Data*

Data koordinat digunakan untuk menentukan titik-titik dari gambar peta awal dan peta akhir. Pada saat menginput data titik ke sebuah *Grid* yaitu data koordinat (x,y), di mana data tersebut berupa angka, maka tombol keyboard yang diaktifkan hanyalah tombol angka 0 sampai 9, koma, *Enter*, *BackSpace*, dan *Arrows*.

Dengan demikian kesalahan pengisian data yang bukan berupa angka dapat dihindarkan. Untuk menyimpan dan keluar tekan *Ok*. Sedangkan untuk keluar dan tidak jadi dilakukan pengisian data tekan *Batal*.

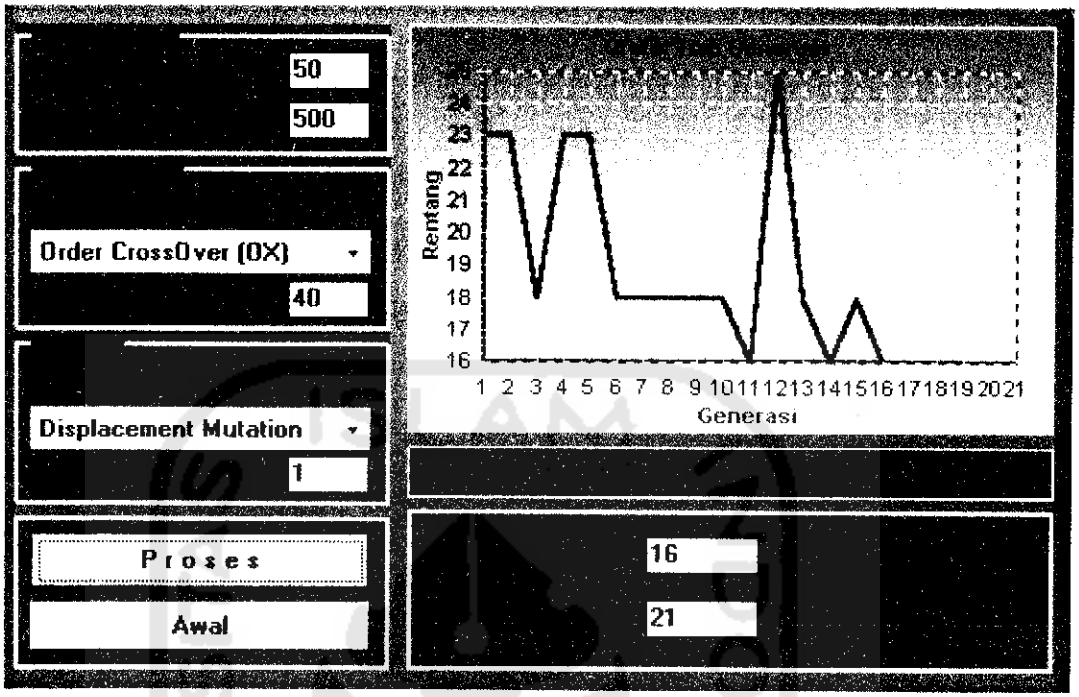


Titik	X	Y
1	2	6
2	4	10
3	6	6
4	5	2
5	10	8
6	10	4

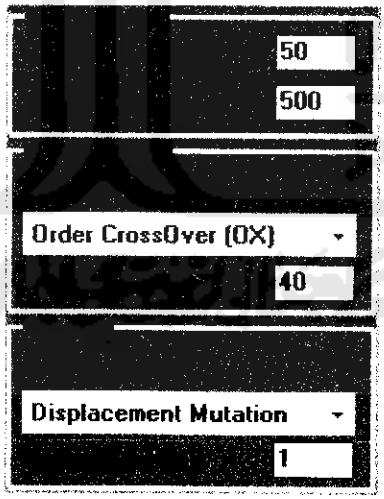
Gambar 5.8 Tampilan Bagian Input Data Titik Form *Data*

5.4.3 Form Genetik

Form *Genetik* ini digunakan untuk memproses data yang telah diinisialisasi pada Form *Data*. Pada bagian ini dibagi menjadi 5 bagian yaitu bagian Parameter Genetik, Metode *CrossOver*, Metode Mutasi, Rentang Terbaik(solusi) dan Grafik.



Gambar 5.9 Tampilan Keseluruhan Form *Genetik*

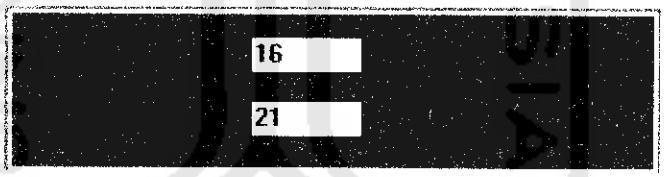


Gambar 5.10 Tampilan bagian *Parameter Values* Form *Genetik*

Pada bagian Parameter Genetik, terdapat beberapa jenis masukan yaitu Ukuran Populasi, Banyak Generasi, Probabilitas *CrossOver* beserta metodenya dan Probabilitas Mutasi beserta metodenya. Semua masukan tersebut diinputkan dengan menggunakan objek *Text*, sedangkan masukan metode diinputkan menggunakan objek *ComboList*.

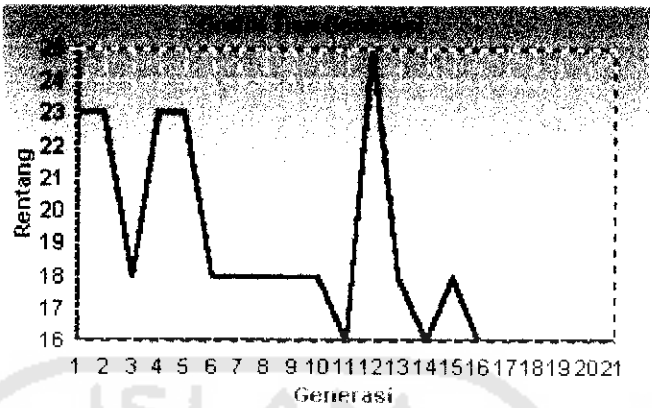
Metode *CrossOver* disediakan ialah *Partial Mapped CrossOver*, *Order CrossOver*, *Position Based CrossOver*, *Order Based CrossOver*, *Cycle CrossOver*. Metode Mutasi yang disediakan ialah *Inversion Mutation*, *Insertion Mutation*, *Displacement Mutation*, *Reciprocal Exchange Mutation*.

Untuk menjalankan proses algoritma genetik maka tombol *Proses* harus ditekan. Setelah tombol *Proses* ditekan maka *ProgressBar* yang akan mengindikasikan jalannya proses pencarian.



Gambar 5.11 Tampilan *Solusi* Form *Genetik*

Setelah proses selesai dijalankan, maka hasil pencarian akan ditampilkan pada bagian *Solusi*. Pada bagian ini ditampilkan *rentang* terbaik dari keseluruhan generasi, kemudian Generasi dimana *rentang* terbaik ditemukan. Sedangkan waktu pemrosesan Operasi Genetik dalam satuan mikrodetik (μs) akan ditampilkan dibawah *ProgressBar*.



Gambar 5.12 Tampilan *Grafik* Form *Genetik*

Pada bagian *Grafik* akan ditampilkan grafik rentang terbaik dari masing – masing generasi setelah proses dijalankan. Grafik yang ditampilkan dapat diatur pembesarannya dengan menggunakan *Mouse*.

5.4.4 Form Keluaran

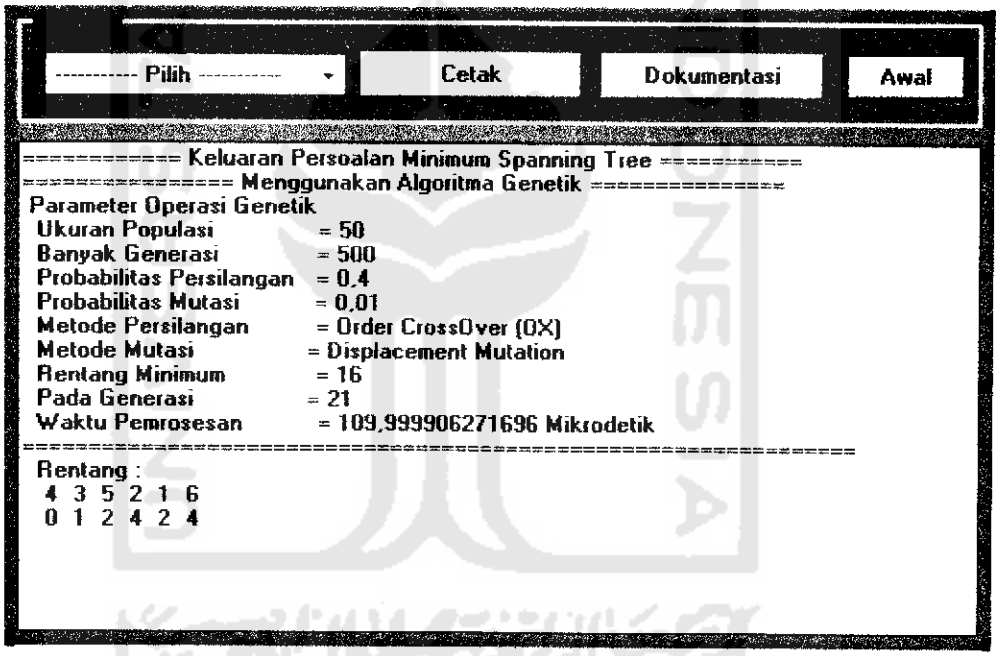
Form *Keluaran* ini digunakan untuk mencetak keluaran dari data yang telah diproses pada Form *Genetik*. Pada Form ini terdapat beberapa pilihan keluaran yang dapat dicetak yaitu keluaran grafik, keluaran file log, solusi terbaik dan keluaran peta akhir. Sedangkan untuk tombol terdapat 2 macam tombol utama yaitu

1. Tombol *Cetak*

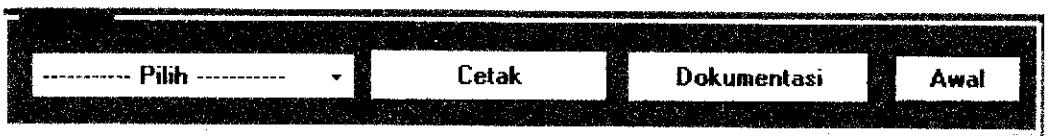
   Tombol ini digunakan untuk mencetak keluaran yang dipilih. Sebelum dicetak, akan ditampilkan *preview* dari keluaran tersebut. *Preview* yang bisa ditampilkan hanyalah keluaran grafik, solusi terbaik dan peta akhir.

2. Tombol *Dokumentasi*

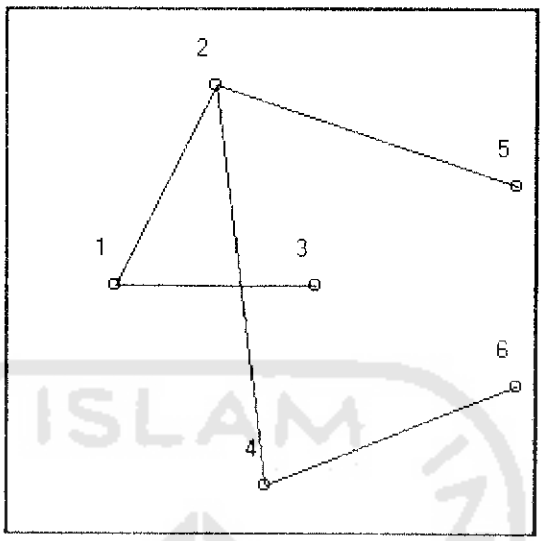
Tombol ini digunakan untuk mengekspor keluaran *File Log* dan keluaran Solusi Terbaik ke dalam format *Document* dengan ekstensi *.doc* sehingga dapat diolah atau diedit dengan software Microsoft Word dan keluaran grafik ke dalam format gambar dengan ekstensi *.bmp*. sedangkan keluaran peta akhir tidak dapat diubah ke bentuk format lain.



Gambar 5.13 Tampilan Keseluruhan Form *Keluaran*



Gambar 5.14 Tampilan bagian *Pilihan* Form *Keluaran*



Gambar 5.15 Tampilan bagian *Peta Akhir* Form *Keluaran*

Untuk memilih jenis keluaran yang akan dicetak atau diekspor digunakan objek *ComboList*, sehingga hanya satu pilihan untuk setiap kali proses.

5.5 Struktur Data

Stuktur data yang digunakan di dalam perangkat lunak ini ialah *Dynamic Arrays* (Array Dinamis), sehingga penggunaan memori dapat dialokasikan dengan tepat karena bergantung dari jumlah data masukan serta parameter genetik untuk pemrosesan solusi akhir. Hal ini dikarenakan *Dynamic arrays* tidak mempunyai ukuran atau panjang yang tetap. Tetapi besarnya memori yang digunakan untuk *Dynamic arrays* dialokasikan begitu suatu nilai di *assign* dengan prosedur *SetLength*. Contohnya suatu kromosom didefiniskan sebagai array dinamis dengan berdimensi dua dengan perintah

```
kromosom : array of array of integer;
```

selanjutnya kromosom diinisialisasi ukuran memori yang digunakan dengan perintah

```
setlength(kromosom,10,20);
```

sehingga sekarang kromosom mempunyai 10 baris yang berisi elemen-elemen dimana tiap baris mempunyai 20 elemen yang bertipe *integer*.

5.6 Prosedur dan Algoritma

5.6.1 Prosedur – Prosedur dalam Perangkat Lunak

Pemrograman di dalam persoalan *minimum spanning tree* menggunakan algoritma genetik ini terdiri atas prosedur-prosedur yang dapat memudahkan dalam pembuatan program. Berikut ini beberapa cuplikan prosedur – prosedur yang digunakan beserta pembahasannya.

5.6.1.1 Prosedur Peta Awal pada Form *Data*

Prosedur Peta Awal ini digunakan untuk merepresentasikan jarak antar titik dan koordinat titik yang telah ditampilkan pada *Grid*. Prosedur Peta Awal akan dipanggil setelah membuka data dari suatu *file*. Di dalam prosedur ini terdapat 2 bagian utama, yaitu pendeklarasian variabel dan menampilkan peta awal.

5.6.1.1.1 Bagian deklarasi variabel prosedur peta awal.

```
sumbu_x, sumbu_y, i, j :integer;
x,y, x1, y1, x2 , y2 :integer;
W,H,Skala:integer;
im:timage;
x_min, x_max, y_min, y_max, max_x, max_y:integer;
tmp:integer;
```

5.6.1.1.2 Bagian menampilkan peta awal

Pada bagian ini, data pada array akan ditampilkan ke dalam suatu peta dengan skala yang sudah ditentukan.

```
//Pembuatan peta hubungan antar node
for i:= 1 to F_Map.sgdata.RowCount-1 do
  begin
    Cari nilai x minimum ;
  end;

for i:= 1 to F_Map.sgdata.RowCount-1 do
  begin
    Cari nilai y minimum ;
  end;

for i:= 1 to F_Map.sgdata.RowCount-1 do
  begin
    Cari nilai x maximum;
  end;

for i:= 1 to F_Map.sgdata.RowCount-1 do
  begin
    Cari nilai y maximum;
  end;

Hitung Skala ;

for i:=1 to F_Map.sgdata.RowCount-1 do
  begin
    Cambar node x dan y sesuai dengan koordinat;
  end;

for i:=1 to grid.RowCount-2 do
  for j:=i+1 to grid.ColCount-1 do
   begin
     Menghubungkan garis antar node;
   end;
Tampilkan Peta ;
```

5.6.1.2 Prosedur Proses pada Form *Genetik*

Prosedur Proses ini digunakan untuk melakukan optimasi atau pencarian solusi dari persoalan *minimum spanning tree* pada Form *Data*. Di dalam prosedur Proses ini terdapat 2 bagian utama, yaitu pendeklarasian variabel dan iterasi. Dengan penjelasan sebagai berikut :

5.6.1.2.1 Bagian deklarasi variabel prosedur proses

```
i : integer;
j,counter : integer;
kon : integer;
```

5.6.1.2.2 Bagian Iterasi

Sebelum proses iterasi dijalankan terlebih dahulu dilakukan pemanggilan prosedur pembangkitan generasi awal. Setelah itu iterasi dijalankan untuk pencarian dari generasi ke dua hingga ke generasi maksimum. Di dalam iterasi tersebut terdapat 3 bagian yang utama yaitu bagian pemanggilan prosedur untuk proses *CrossOver* (persilangan), proses mutasi, generasi selanjutnya.

```
pembangkitan generasi_awal;
 try
    i := 1;
    while (i < maks) and (not found) do
    begin
       Pilih metode CrossOver dan mutasi;
       generasi_selanjutnya;
    end;
```

5.6.1.3 Prosedur Pembangkitan Generasi Awal pada Form *Genetik*

Prosedur ini digunakan untuk melakukan pembangkitan kromosom generasi awal. Di dalam prosedur ini terdapat bagian utama yaitu pendeklarasian variabel, pengacakan gen sebagai kromosom awal, perhitungan *fitness* dan probabilitas kumulatif, penentuan kandidat *parent*, penentuan *parent* yang akan di*CrossOver*kan, serta pencarian *fitness* dan kromosom terbaik.

5.6.1.3.1 Bagian deklarasi prosedur pembangkitan generasi awal.

```
i, j, k, l : integer;
acak : integer;
prob : double;
```

5.6.1.3.2 Bagian pengacakan gen sebagai kromosom awal

```
for i := 0 to pop - 1 do
begin
 for j :  0 to titik - 1 do
   Mengacak bilangan (gen) sepanjang kromosom pada string 1;
end;
for j:=0 to (pop - 1) do
begin
 for j:= 1 to (titik-1) do
   Mengacak bilangan (gen) sepanjang kromosom pada string 2;
end;
```

5.6.1.3.3 Bagian perhitungan fitness

```
for i: 1 to pop-1 do
begin
 for j:=1 to titik-1 do
 begin
  Menghitung (fitness) tiap kromosom;
 end;
end;
```

5.6.1.3.4 Bagian perhitungan probabilitas kumulatif

```
for i :  0 to pop - 1 do
begin
 Mengkonversi fitness menjadi 1 / fitness;
end;
for i := 0 to pop - 1 do
begin
 Menghitung probabilitas kumulatif fitness;
end;
for j := 0 to pop - 1 do
begin
 Mengacak bilangan dari 0 sampai 1 sebagai prob acak;
end;
```

5.6.1.3.5 Bagian penentuan kandidat *parent*

```
for i := 0 to pop - 1 do
begin
  Menentukan kandidat parent (perbandingan prob kum dengan prob
    acak);
end;
```

5.6.1.3.6 Bagian penentuan *parent* yang akan di*CrossOver*kan

```
for j := 0 to pop - 1 do
begin
  Mengacak bilangan dari 0 sampai 1 sebagai prob acak;
end;
for i := 0 to pop - 1 do
begin
  Menentukan induk (perbandingan prob acak dengan prob
  crossover);
end;
```

5.6.1.3.7 Bagian pencarian *fitness* dan kromosom terbaik pada generasi awal

```
for i := 0 to pop - 1 do
begin
 Mencari kromosom yang mempunyai fitness terbaik;
end;
for i := 0 to titik -1 do
 Mengambil isi kromosom yang mempunyai fitness terbaik untuk
 dimasukkan ke array;
```

5.6.1.4 Prosedur *CrossOver PMX* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode *CrossOver PMX*. Di dalam prosedur ini terdapat 4 bagian utama yaitu pendeklarasian variabel, inisialisasi *parent* dan pemotongan jumlah *parent* yang ganjil, serta persilangan. Bagian bagian yang terdapat pada prosedur ini juga dimiliki oleh prosedur *CrossOver* (persilangan) yang lain, sehingga untuk penjelasan prosedur *CrossOver* yang lain hanya akan dijelaskan pada bagian deklarasi dan bagian persilangan, untuk bagian yang lain sama. Penjelasan prosedur ini sebagai berikut:

5.6.1.4.1 Bagian deklarasi variabel pada prosedur silang PMX

```
i, j, l, k : integer;
parent, temp_parent : array of array of integer;
relasi_map : array of array of integer;
substring_acak1, substring_acak2 : integer;
temp : integer;
```

5.6.1.4.2 Bagian inisialisasi *parent*

```
for i := 0 to pop - 1 do
begin
   for j := 0 to titik - 1 do
    Tampung array parent ke temporary;
end;
```

5.6.1.4.3 Bagian pemotongan *parent* yang ganjil

```
   if ((jml_induk + 1) mod 2 <> 0) then
     jml_induk :  jml_induk - 1;
```

5.6.1.4.4 Bagian persilangan antar *parent*

```
for k := 0 to jml_induk do
begin
  if k mod 2 = 0 then
  begin
   repeat
    Acak 2 bilangan hingga tidak bernilai sama;
```

```
    until substring_acak1 <> substring_acak2;
     //penyilangan parent 1 dengan parent 2 untuk memperoleh
    relasi map
    for j := substring_acak1 to substring_acak2 do
    begin
      Silangkan substring terpilih antara parent 1 dan parent 2
      untuk mencari relasi map;
    end;
      //penyilangan parent 1 dengan parent 2 berdasar relasi map
    for j := 0 to titik - 1 do
    begin
       for l := 0 to (substring_acak2-substring_acak1) do
       begin
          Silangkan parent 1 dan parent 2 berdasar relasi map;
       end;
       Kembalikan temporary ke parent;
     end;
  end;
end;
```

5.6.1.5 Prosedur *CrossOver OX* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode *CrossOver OX*. Penjelasan prosedur ini sebagai berikut :

5.6.1.5.1 Bagian deklarasi variabel pada prosedur silang *OX*

```
i, j, k, l, m : integer;
substring_acak1, substring_acak2 : integer;
parent : array of array of integer;
child1, child2 : array of integer;
temp : integer;
found : boolean;
```

5.6.1.5.2 Bagian persilangan antar *parent*

```
for k :  0 to jml_induk do
begin
   if k mod 2 = 0 then
   begin
     repeat
       Acak 2 bilangan hingga tidak bernilai sama;
     until substring_acak1 <> substring_acak2;
     for j := substring_acak1 to substring_acak2 do
     begin
        Isikan child 1 dan 2 dengan substring terpilih dari
        masing-masing parent;
     end;
     //induk yang telah mengalami penyilangan untuk posisi 1
     for j :  0 to titik -1 do
     begin
        for m :  0 to titik - 1 do
```

```
        begin
            Isikan sisa substring dari parent 2 ke child dari
            kiri ke kanan;
        end;
    end;
    //induk yang telah mengalami penyilangan untuk posisi ?
    for i :  0 to titik - 1 do
    begin
        for m :  0 to titik - 1 do
        begin
            Isikan sisa substring dari parent 1 ke child dari
            kiri ke kanan;
        end;
    end;
```

5.6.1.6 Prosedur *CrossOver PBX* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode *CrossOver PBX*. Proses persilangan hampir sama dengan metode *CrossOver OX*, perbedaanya terletak dari pengambilan gen dimana pada persilangan ini yang diambil ialah satu set posisi gen bukan *substring*. Penjelasan prosedur ini sebagai berikut :

5.6.1.6.1 Bagian deklarasi variabel pada prosedur silang *PBX*

```
i, j, k, l, m : integer;
jml_acak : integer;
parent : array of array of integer;
child1, child2, tempchild : array of integer;
acak : integer;
found : boolean;
```

5.6.1.6.2 Bagian persilangan antar *parent*

```
for k := 0 to jml_induk do
begin
    if k mod 2 = 0 then
    begin
        Acak bilangan diantara 1 hingga panjang node;
        for j :  0 to jml_acak - 1 do
        begin
            Acak satu set posisi gen dari kromosom sebanyak jumlah
            acak;
        end;
        for j := 0 to jml_acak - 1 do
        begin
            Isikan child 1 dan 2 dengan set kromosom yang sudah
            terdefinisi dari masing-masing parent;
        end;
```

```
        //induk yang telah mengalami penyilangan untuk posisi 1
        for j := 0 to titik - 1 do
        begin
            Isikan sisa gen dari parent 2 ke child 1 dari kiri ke
            kanan;
        end;
        //induk yang telah mengalami penyilangan untuk posisi 2
        for j :  0 to titik - 1 do
        begin
            Isikan sisa gen dari parent 1 ke child 2 dari kiri ke
            kanan;
        end;
      end;
    end;
```

5.6.1.7 Prosedur *CrossOver OBX* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode *CrossOver OBX*. Proses persilangan yang dilakukan hampir sama dengan metode *CrossOver PBX*. Perbedaannya terletak pada penempatan *substring* dari *parent* pertama yang dilakukan setelah penempatan satu set posisi dari *parent* kedua ke dalam *child*. Penjelasan prosedur ini sebagai berikut :

5.6.1.7.1 Bagian deklarasi variabel pada prosedur silang *OBX*

```
i, j, k, l, m : integer;
jml_acak : integer;
parent : array of array of integer;
child1, child2, tempchild1, tempchild2 : array of integer;
acak : integer;
found : boolean;
```

5.6.1.7.2 Bagian persilangan antar *parent*

```
for k := 0 to jml_induk do
begin
   if k mod 2 = 0 then
   begin
      Acak suatu bilangan diantara 1 hingga panjang node;
      for j := 0 to jml_acak - 1 do
      begin
         Acak satu set posisi gen dari kromosom parent 1
         sebanyak jumlah acak;
         Isikan child 2;
      end;
      for j := 0 to titik - jml_acak - 1 do
      begin
```

```
            Acak satu set posisi gen dari kromosom parent 2
            sebanyak jumlah acak;
            Isikan ke child 1;
        end;
        //induk yang telah mengalami penyilangan untuk posisi 1
        for j := 0 to titik - 1 do
        begin
            Isikan sisa gen dari parent 2 ke child 2 dari kiri ke
            kanan;
        end;
        //induk yang telah mengalami penyilangan untuk posisi 2
        for j := 0 to titik - 1 do
        begin
            Isikan sisa gen dari parent 1 ke child 1 dari kiri
            kanan;
    end;
  end;
end;
```

5.6.1.8 Prosedur *CrossOver CX* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode *CrossOver CX*. Penjelasan prosedur ini sebagai berikut :

5.6.1.8.1 Bagian deklarasi variabel pada prosedur silang CX

```
i, j, k, l, m : integer;
no acak : integer;
parent : array of array of integer;
child1, child2, componid : array of integer;
cycle : array of integer;
found, foundcycle : boolean;
```

5.6.1.8.2 Bagian persilangan antar *parent*

```
for x := 0 to jml induk do
begin
   if x mod 2 = 0 then
   begin
      Acak satu posisi gen;
      if cycle[0] <> cycle[1] then
      begin
         Cari hubungan gen antar parent 1 dan parent 2;
      end;
      begin
         Isikan child dengan hasil persilangan berdasarkan
         cycle;
      end;
      //induk yang telah mengalami penyilangan untuk posisi 2
      if cycle[0] <> cycle[1] then
      begin
         Cari hubungan gen antar parent 1 dengan parent 2;
      end;
```

```
        for j := 0 to titik - 1 do
        begin
           Isikan child dengan hasil persilangan berdasarkan
           cycle;
      end;
   end;
end;
```

5.6.1.9 Prosedur Mutasi *Inversion* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode Mutasi *Inversion*. Prosedur ini dibagi menjadi 3 bagian utama yaitu pendeklarasian variabel, pengacakan bilangan 0 sampai 1 yang disebut sebagai probabilitas acak untuk menentukan kromosom hasil persilangan yang akan dimutasi (proses ini berlaku untuk setiap metode mutasi yang dijalankan, sehingga untuk penjelasan metode mutasi yang lain, bagian ini tidak dijelaskan) dan mutasi kromosom. Penjelasan prosedur ini sebagai berikut :

5.6.1.9.1 Bagian deklarasi variabel pada prosedur mutasi *inversion*

```
i, j, l : integer;
substring_acak1, substring_acak2 : integer;
inversion : array of integer;
acak : integer;
prob : double;
temp : integer;
found : boolean;
```

5.6.1.9.2 Bagian pengacakan probabilitas acak

```
for j := 0 to jml_induk do
begin
 Acak bilangan antara 0 sampai 1;
end;
for i := 0 to jml_induk do
begin
  Kromosom yang termutasi;
end;
```

5.6.1.9.3 Bagian mutasi

```
for i := 0 to jml_induk do
begin
   found := false;
   begin
      Acak 2 bilangan hingga tidak bernilai sama;
```

```
        for j := substring_acak1 to substring_acak2 do
        begin
           Isikan substring terpilih ke dalam inversion;
        end;
        for j := substring_acak1 to substring_acak2 do
        begin
           Balik posisi substring di inversion, isikan ke temp
           node;
      end;
    end;
end;
```

5.6.1.10 Prosedur Mutasi *Insertion* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode Mutasi *Insertion*. Penjelasan prosedur ini sebagai berikut :

5.6.1.10.1 Bagian deklarasi variabel pada prosedur mutasi *insertion*

```
i, j, l : integer;
no_acak1, no_acak2 : integer;
insertion : array of integer;
tempinsertion : array of integer;
acak : integer;
prob : double;
temp : integer;
found : boolean;
```

5.6.1.10.2 Bagian mutasi

```
for i :  0 to jml_induk do
begin
    if found then
    begin
        Acak 2 bilangan hingga bernilai tidak sama;
        if no_acak2 < no_acak1 then
        begin
          Sisipkan gen yang terpilih ke dalam parent;
          Geser gen ke kanan posisi yang tersisip;
        end;
        if no_acak2 > no_acak1 then
        begin
          Sisipkan gen yang terpilih ke dalam parent;
          Geser gen ke kiri posisi yang tersisip;
      end;
  end;
end;
```

5.6.1.11 Prosedur Mutasi *Displacement* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode Mutasi *Displacement*. Pada prosedur ini prinsipnya sama dengan metode mutasi *insertion*, perbedaannya terletak dari jumlah gen yang disisipkan. Penjelasan prosedur ini sebagai berikut :

5.6.1.11.1 Bagian deklarasi variabel pada prosedur mutasi *Displacement*

```
i, j, l : integer;
no_acak1, no_acak2, no_acak3 : integer;
displacement : array of integer;
tempdisplacement1 : array of integer;
tempdisplacement2 : array of integer;
acak : integer;
prob : double;
found : boolean;
```

5.6.1.11.2 Bagian mutasi

```
for i := 0 to jml_induk do
begin
    if found then
    begin
        for j :  no_acak1 - 1 to no_acak2 - 1 do
        begin
            Ambil substring yang akan disisipkan ke temporary;
        end;
        for j := no_acak2 - 1 to no_acak3 - 1 do
        begin
            Pindahkan substring yang digeser ke temporary;
        end;
        for j := no_acak1 - 1 to no_acak2 - 1 do
        begin
            Sisipkan substring yang telah disimpan;
        end;
        for j :  no_acak2 to no_acak3 - 1 do
        begin
            Geser / letakkan substring yang disisipkan;
     end;
  end;
end;
```

5.6.1.12 Prosedur Mutasi *Reciprocal Exchange* pada Form *Genetik*

Prosedur ini digunakan untuk menjalankan metode Mutasi *Reciprocal Exchange*. Penjelasan prosedur ini sebagai berikut :

5.6.1.12.1 Bagian deklarasi variabel pada prosedur mutasi *Reciprocal Exchange*

```
i, j, l : integer;
no acak1, no acak2 : integer;
acak : integer;
prob : double;
temp : integer;
found : boolean;
```

5.6.1.12.2 Bagian mutasi

```
for i := 0 to jml_induk do
begin
  if found then
  begin
    //pengacakan substring
    Acak 2 bilangan 0 sampai 7 hingga tidak bernilai sama;
    Tukarkan posisi acak 1 gen dengan posisi acak 2 gen;
  end;
end;
```

5.6.1.13 Prosedur Generasi Selanjutnya pada Form *Genetik*

Prosedur ini digunakan untuk melakukan proses komputasi untuk generasi ke 2 hingga generasi maksimum. Yang menjadi perbedaan ialah kromosom untuk generasi selanjutnya ini bukan merupakan pengacakan kromosom, melainkan hasil proses (seleksi, *Cross()ver* dan mutasi) dari generasi sebelumnya.

```
for i := 0 to jml_induk do
  for j := 0 to titik - 1 do
    Transfer kromosom hasil persilangan ke temporaryl;
for i := 0 to jml_induk do
  for j := 0 to titik - 1 do
    Transfer kromosom hasil persilangan ke temporaryZ;
for i := 0 to pop - 1 do
  for j := 0 to titik - 1 do
    Transfer kromosom dari temporaryl dan temporaryZ ke
    parent dan gabung dengan kromosom yang terseleksi dari
    generasi sebelumnya;
```

5.6.1.14 Prosedur Peta Akhir pada Form *Keluaran*

Prosedur ini digunakan untuk menampilkan peta akhir yang didapat dari proses optimasi dengan algoritma genetik. Prosedur ini mempunyai 2 bagian utama yaitu pendeklarasian variabel dan menampilan peta hasil proses.

5.6.1.14.1 Bagian deklarasi variabel pada prosedur peta akhir

```
i,j : integer;
temp_output, temp_output2 : array of string;
min : single;
templog, temp, temp2 : string;
gen,line : integer;
```

5.6.1.14.2 Bagian menampilkan peta hasil proses.

Prosedur ini digunakan untuk menampilkan data hasil proses ke dalam suatu peta.

```
for i:=0 to titik-1 do
 begin
 Menampilkan node x dan y hasil proses ke dalam peta akhir;
 end;
```

5.6.1.15 Prosedur *Dokumentasi* pada Form *Keluaran*

Prosedur ini digunakan untuk mengekspor keluaran ke dalam bentuk document *.doc* untuk keluaran rentang maupun log, dan *.bmp* untuk keluaran grafik.

```
case cbprint.itemIndex of
    0 : begin
         messagedlg('Pilih lainnya !',mtinformation,[mbok],0);
        end;
    1 : begin
         ekspor grafik ke file.bmp;
        end;
    2 : begin
      ekspor file log ke file.doc;
        end;
    3 : begin
      ekspor keluaran rentang ke file.doc;
        end;
    4 : begin
         Messagedlg('Peta Akhir tidak bisa diekspor ke
         Dokumen!',mtinformation,[mook],0);
        end;
    end;
```

5.6.1.16 Prosedur *Cetak* pada Form *Keluaran*

Prosedur ini digunakan untuk mem*preview* keluaran sebelum dicetak.

```
case cbprint.ItemIndex of
  0 : begin
       messagedlg('Pilih lainnya !',mtinformation,[mbok],0);
      end;
  1 : begin
       Preview keluaran grafik;
      end;
  2 : begin
      if MessageDlg('File log tidak bisa di Preview '+#13+'
      Apakah akan diproses ?', mtConfirmation,
       [mbYes, mbNoj, 0) = mrYes then
        cetak langsung file log;
      end;
  3 : begin
      Preview keluaran kontung;
      end;
  4 : begin
      Preview keluaran peta akhir;
      end;
end;
```