TAAKVERSLAG

DEMONSTRATIE PROGRAMMA

voor

NETWERK BEWERKINGSALGORITHMEN

Auteur : R.V. van de Ree

Betreft : Een raamwerk-definitie en een demonstratie-programma voor het uitvoeren van diverse bewerkingsalgorithmen op netwerken.

Technische Universiteit Delft
Faculteit der Electrotechniek

Inhoudsopgave

1 Inleiding

Naar aanleiding van de 60-middagen-taak, uit te voeren ten behoeve van de opleiding tot ingenieur in de electrotechniek, is bij de vakgroep Automatische Verkeerssystemen de volgende taakopdracht uitgevoerd :

> Definiëer een raamwerk waarbinnen algoritmen voor netwerk-operaties geïmplementeerd kunnen worden. Vervolgens dient dit raamwerk gebruikt te worden om een demonstratie-programma te ontwerpen.

Dit programma kan gebruikt worden naast het vak 'Datacommunicatie Netwerken' (vakcode 1102B).
Het bij dit college gebruikte boek van Tanenbaum [Lit.1] bevat tal van procedures voor netwerk-operaties. De meeste van deze procedures zullen in het demonstratie-programma opgenomen worden. Het demonstratie-programma beoogt het vergroten van het inzicht in de stof, zoals die op het college behandeld wordt.

2 Taakopdracht

Zoals in de inleiding is vermeld, luidt de taakopdracht :

> Definiëer een raamwerk waarbinnen algoritmen voor netwerk-operaties geïmplementeerd kunnen worden. Vervolgens dient dit raamwerk gebruikt te worden om een demonstratie-programma te ontwerpen.

De bedoeling van deze opdracht is het specificeren en standaardiseren van de gegevens die een rol spelen binnen een netwerk (zoals onderlinge afstanden, verkeersaanbod).
Daarnaast dient het raamwerk van een dusdanige opzet te zijn dat nieuwe algoritmen eenvoudig en met zo min mogelijk handelingen te integreren zijn in een bestaand programma dat op het raamwerk gebaseerd is.
Tenslotte dient er een mogelijkheid gemaakt te worden om snel en overzichtelijk de betreffende netwerkgegevens in te voeren.
Al deze eisen dienen gerealiseerd te worden in het demonstratie-programma.

3 Raamwerk definitie

3.1 Gegevensstructuur.

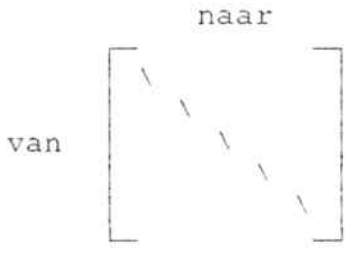
Allereerst is een analyse gemaakt van de benodigde gegevens voor de verschillende netwerk-operaties. Bij de analyse is gebruik gemaakt van het boek van Tanenbaum [Lit.1]. Deze gegevens zijn :

- afstand tussen knooppunten
- capaciteit van verbindingen
- verkeer(sintensiteit) tussen knooppunten
- kosten van knooppuntstations en verbindingen
- routering van packets over het netwerk
- vertraging van packets op verbindingen
- packetlengte (1/μ).

Daarnaast zijn nog een aantal criteria van belang :

- gewenste arc-connectivity
- gewenste node-connectivity
- maximaal toegestane vertraging
- maximaal toegestane kosten.

Voor die gegevens die per verbinding hetzelfde karakter hebben is een matrix-organisatie ingevoerd. Deze matrices zijn als volgt ingedeeld :



matrix[van,naar]

De index 'van' staat voor het bron-knooppunt en de index 'naar' staat voor het 'bestemmings'-knooppunt. Zo zijn dus verschillende waarden mogelijk voor de heen- en terugweg tussen twee knooppunten. Zijn deze waarden gelijk, dan is de matrix symmetrisch.

Voor het demonstratie-programma zijn de volgende matrices gedefiniëerd :

- afstandsmatrix (symmetrisch)
- capaciteitsmatrix (symmetrisch)
- verkeersmatrix (niet-symmetrisch)
- kostenmatrix (zie hieronder)
- routeringsmatrix (niet-symmetrisch)
- vertragingsmatrix (niet-symmetrisch).

De kostenmatrix wijkt iets af van de overige matrices en is als volgt ingedeeld :

- hoofddiagonaal : kosten van de knooppuntstations
- benedendriehoeksmatrix : vaste lijnkosten
- bovendriehoeksmatrix : variabele lijnkosten.

Hierbij is afgeweken van het "heenweg-terugweg" principe voor de matrix-organisatie. Aanleiding hiertoe is de aanname van het bestaan van slechts één (fysieke) verbinding tussen twee knooppunten. Dit sluit de mogelijkheid van verschillende capaciteiten voor de beide richtingen uiteraard niet uit.

3.2 De editor.

Een onderdeel van de taak is het verzorgen van de invoer van netwerken en netwerk-gegevens. Hiertoe is een grafische editor ontworpen waarmee netwerken snel zijn in te voeren. De gegevens omtrent verbindingen worden één voor één ingevoerd en bij de betreffende verbinding geschreven.

4 Demonstratie-programma

4.1 De Programmastructuur.

Teneinde de verschillende programmadelen gemakkelijk aan elkaar te koppelen, is gekozen voor een modulaire opzet. De modules worden bestuurd vanuit één centrale menu-module. De keuze van de systeem-functies geschiedt aan de hand van de bekende keuze-boom structuur.

Het toevoegen van een nieuwe module aan het systeem vergt slechts vijf stappen :

1 schrijf en test de nieuwe module
2 zet de naam van de nieuwe module bij het rijtje van module-namen in de menu-module
3 zet de nieuwe mogelijkheden bij de andere in die menu-groep waaraan de nieuwe functies moeten worden toegevoegd
4 zet de beschrijving van de nieuwe mogelijkheden in de betreffende menu-presentatie procedure
5 verhoog de 'aantal_mogelijkheden'-teller met het aantal toegevoegde mogelijkheden.

Al deze toevoegingen en veranderingen dienen in de menu-module te worden uitgevoerd. Als een bestaande module wordt uitgebreid, dan komt stap 2 te vervallen.

4.2 Indeling van het demonstratie-programma.

Het demonstratie-programma bestaat uit vier delen :

- menu module (uit het raamwerk)
- editor module (uit het raamwerk)
- procedure module
- ontwerp module.

De menu-module zorgt voor het aanroepen van de verschillende submodulen en procedures daarin.
De editor-module zorgt voor grafische invoermogelijkheden. De knooppunten zowel als de verbindingen tussen knooppunten kunnen snel ingevoerd worden. Voorts kunnen afstands-, kosten-, verkeers- en capaciteits-matrices ingevoerd worden. Dit geschiedt aan de hand van het netwerkplaatje. Ook opslagmogelijkheden ontbreken niet. De gegevens worden (uiteraard) conform de raamwerk-definitie opgeleverd.
De procedure-module bevat de meeste bewerkingsalgorithmen die in het boek van Tanenbaum [Lit.1] genoemd worden (zoals Dijkstra's kortste route algorithme, Even's node-connectivity-bepalend algorithme). Deze procedures maken gebruik van de gegevensstructuur zoals die in het raamwerk is gedefiniëerd (zie §3.1).
De ontwerp-module tenslotte, is een realisatie van de suggestie genoemd op pagina 69 van het boek van Tanenbaum [Lit.1]. Deze

suggestie is uitgebreid met de mogelijkheid op verschillende plaatsen in het ontwerpproces uit alternatieve bewerkingen te kiezen. Zo kan voor het bepalen van de capaciteiten van de knooppuntverbindingen gekozen worden uit een uniforme, proportionele of optimale verdeling.

4.3 Handleiding bij het demonstratie-programma.

Na het opstarten van het programma verschijnt het volgende keuze-menu:

```
Main Menu :

 1 : Editor
 2 : Procedure Menu
 3 : Design Topology
 4 : End of program

make your choice ( 1 - 4 ) : _
```

Keuze-mogelijkheid 1 start de grafische editor op. Keuze 2 brengt de gebruiker in een nieuw keuze-menu voor de keuze van de geïmplementeerde netwerk-bewerkingsprocedures. Mogelijkheid 3 roept de netwerk ontwerp procedure aan. De laatste keuze-mogelijkheid spreekt voor zichzelf. In figuur 1 is weergegeven hoe de verschillende programma onderdelen met elkaar in verbinding staan.

4.3.1 De editor.

Als de editor aangeroepen wordt, wordt de scherm-uitvoer in de grafische mode geschakeld. Het type beeldscherm wordt automatisch gedetecteerd.
Als er nog geen netwerk-naam is opgegeven, dan wordt daar op dit moment naar gevraagd.
Daarna wordt een kader in het beeld getekend met daarbinnen de tekening van het netwerk, vermits de opgegeven netwerk-naam reeds bekend is. Is er nog geen netwerk-tekening bekend onder de opgegeven netwerk-naam, dan blijft het kader leeg.
Tenslotte verschijnt onder in beeld een balk met de beschrijving van de functie-toetsen die in de editor gebruikt worden.
De editor is nu klaar voor gebruik. De editor-functies worden met behulp van functie-toetsen geactiveerd.

De functie-toetsen roepen de volgende mogelijkheden aan:

F 1 : invoer van de kosten
F 2 : invoer van de afstanden
F 3 : invoer van de capaciteiten
F 4 : invoer van het verkeer
F 5 : invoer van netwerk parameters (Ca, Cn, μ)
F 6 : aanroepen van het teken-gedeelte van de editor
F 7 : opslaan van de huidige netwerk-gegevens op schijf
F 8 : inlezen van netwerk-gegevens bij de opgegeven netwerk-naam van schijf
F 9 : invoer van een nieuwe netwerk-naam en de keuze uit "invoeren" of "laten-zien" van de gegevens
F10 : verlaten van de editor.

De invoer van de gegevens betreffende knooppunten en verbindingen daartussen (F1..F4), geschiedt aan de hand van de tekening van het netwerk. Het item waarbij een gegeven moet worden ingevoerd, wordt aangeduid door een merkteken. Dit merkteken bestaat uit drie sterretjes ("***"). Onder in beeld verschijnt de bijbehorende vraag aan de gebruiker. Deze wordt geacht op dit punt een getal in te voeren, voorstellende de maat van de in te voeren grootheid.
Bij het invoeren van gegevens betreffende het gehele netwerk (F5 en F9), blijft het bij de vragen onder in beeld. Bij F5 dienen getallen ingevoerd te worden. Bij F9 wordt eerst naar een nieuwe naam gevraagd (men dient hier een string in te voeren, voorstellende de nieuw te gebruiken netwerk-naam). Vult men niets in, dan blijft de oude naam aangehouden. Vervolgens wordt de keuze geboden om te schakelen tussen de "invoeren"-mode en de "laten-zien"-mode van de gegevens. De editor start in de "invoeren"-mode. In de "laten-zien"-mode wordt de invoer-cyclus overgeslagen en kan de gebruiker snel zien wat er reeds bekend is.
Het opslaan en inlezen van netwerk-gegevens vergt slechts één druk op de knop (F7 en F8). De gegevens staan op schijf met als bestands-naam de netwerk-naam plus het achtervoegsel ".NET".
Bij het verlaten van de editor (F10) wordt de scherm-uitvoer teruggeschakeld naar de tekst mode. Tevens wordt de gebruiker gevraagd of hij/zij de netwerk-gegevens wil opslaan of niet. Na beantwoording van deze vraag, keert het programma terug naar het hoofdmenu.
Bij het invoeren van strings en getallen kan de correctie-toets gebruikt worden om fouten te herstellen. Voert de gebruiker geen of een foutief getal in (met niet-cijfer tekens), dan blijft de oude waarde voor de betreffende parameter gehandhaafd.

4.3.2 Het teken-gedeelte van de editor.

Als vanuit de editor de keuze F6 gegeven wordt, dan komt de gebruiker terecht in het teken-gedeelte van de editor. In dit gedeelte kan de gebruiker het netwerk invoeren.
Op het beeldscherm wordt nu een kader ingeruimd voor het tekenwerk. Als er reeds een netwerk is ingevoerd, dan blijft dit in het kader staan.
Links boven in beeld verschijnt een kruisje. Dit kruisje is de tekencursor en wordt bestuurd door de cursor-toetsen.

Het invoeren en/of wijzigen van het netwerk gaat met behulp van de commando's p(ut), e(rase), m(ark), u(nmark), c(onnect), d(isconnect) en r(eady) :

p : plaats op het aangegeven punt een knooppunt
e : verwijder het knooppunt dat zich op het aangegeven punt bevindt
m : markeer het knooppunt op het aangegeven punt
u : demarkeer het knooppunt op het aangegeven punt
c : verbindt het gemarkeerde knooppunt met het knooppunt op het aangegeven punt
d : verwijder de verbinding tussen het gemarkeerde knooppunt en het knooppunt op de aangegeven plaats
+ : verhoog de cursor-stapgrootte met 10 dots/stap
- : verlaag de cursor-stapgrootte met 10 dots/stap
r : verlaat het teken-gedeelte van de editor.

Met "het aangegeven punt" wordt dat punt bedoeld, dat door het cursor-kruisje wordt aangegeven.
Een knooppunt wordt aangegeven door het kruisje binnen zijn cirkel te plaatsen. De preciese plaats binnen de cirkel is niet belangrijk. Een knooppunt-cirkel heeft een straal van 20 dots. Valt het kruisje buiten een cirkel, dan wordt geen actie ondernomen op het geven van een commando.
Dit met uitzondering van het "p" commando; nu wordt juist wel actie ondernomen. Als er zich geen knooppunt bevindt binnen een straal van 50 dots, dan wordt een knooppunt geplaatst. De minimum afstand tussen twee knooppunten is groter gekozen dan de dubbele cirkelstraal, opdat ook de kortste verbindingen nog duidelijk zichtbaar zijn.
Als er nog geen knooppunt gemarkeerd is op het moment dat het commando "c" of "d" gegeven wordt, dan wordt het aangegeven knooppunt alleen gemarkeerd.
De cursor-stapgrootte heeft een minimum van 10 dots/stap en een maximum van 100 dots/stap.
Als het netwerk is ingevoerd, dan verlaat de gebruiker het teken-gedeelte door middel van het commando "r".
In verband met de beperkte breedte van het beeldscherm (80 lettertekens) is er een beperking gelegd op het maximale aantal knooppunten. Het maximale aantal bedraagt 10 knooppunten. Nu past een matrix van integer getallen precies in één beeld.

4.3.3 Het netwerk-procedure menu.

Als deze optie uit het hoofd-menu gekozen wordt, dan verschijnt het volgende menu op het beeldscherm :

```
Procedure Menu :    ( operations on network :  NAME  )

 1 : Dijkstra
 2 : Malhotra
 3 : Kleitman
 4 : Even
 5 : Arcdisjoint
 6 : Nodedisjoint
 7 : Monte Carlo
 8 : Mean Delay
 9 : Steiglitz
10 : Assign Flow
11 : Assign Capacities
12 : Calculate Cost
13 : Branch Exchange
14 : Saturated Cut
15 : Connectivity Restore
16 : Last Menu

make your choice ( 1 - 16 ) : _
```

Dit zijn vrijwel alle bewerkings-algoritmen die in het boek van Tanenbaum [Lit.1] genoemd worden. Door een keuze te maken wordt de betreffende procedure opgestart. De **NAME** staat voor de naam van het netwerk.
Als er nog aanvullende gegevens nodig zijn om het algorithme te kunnen gebruiken, dan wordt hier eerst naar gevraagd. Als dit gegevens zijn die normaal gesproken via de editor ingevoerd worden, dan wordt de editor aangeroepen. Onder in het editor-kader verschijnt dan een commentaar-regel die vermeldt welke gegevens nog moeten worden ingevoerd. Nadat de ontbrekende gegevens zijn ingevoerd, keert de editor rechtstreeks terug naar de netwerk-procedure. Zijn alle gegevens benodigd voor het algorithme ingevoerd, dan gaat het algorithme aan de slag.
Na verloop van tijd verschijnt het resultaat op het beeldscherm. Als de gebruiker kennis heeft genomen van de resultaten, dan is het aanslaan van een willekeurige toets voldoende om terug te keren naar het procedure-menu.
Keuze 16 brengt de gebruiker terug naar het hoofd-menu.

4.3.4 Het ontwerpen van een topologie.

De derde optie uit het hoofd-menu biedt de mogelijkheid om een compleet netwerk te laten ontwerpen.
Zoals reeds eerder vermeld (zie §4.1), is deze procedure

gebaseerd op het programma op pagina 69 uit het boek van Tanenbaum [Lit.1].
Op sommige plaatsen in dit programma, is het mogelijk alternatieven te gebruiken voor de gebruikte bewerkings-algoritmen. De gebruiker wordt dan ook gevraagd naar zijn keuze voor die alternatieven. De volgende alternatieven zijn beschikbaar :

- Starttopologie-generatiemethode :
  "S" : Steiglitz link-deficit algorithme
  "O" :

- Routeringsmethode :
  "D" : Dijkstra's algorithme voor kortste route bepaling

- Capaciteiten-toekenningsmethode :
  "O" : optimale verdeling
  "P" : proportionele verdeling
  "U" : uniforme verdeling

- Netwerkvariatie-methode :
  "B" : branch exchange
  "S" : saturated cut heuristic.

Er kan slechts één alternatief per keer gekozen worden.
Na een variatie-operatie is het mogelijk om het connectivity-restoring algorithme toe te passen. De gebruiker wordt hiernaar gevraagd en kan antwoorden met "y"(es) of "n"(o).
Tevens kan de gebruiker aangeven of hij/zij er prijs op stelt het resultaat van een tussenstap uit het programma op het scherm te zien. De resultaten die getoond kunnen worden zijn :

"R" : de toegekende routering binnen het netwerk (bv: de kortste route berekend via het algorithme van Dijkstra)
"C" : de toegekende capaciteiten aan de verbindingen
"V" : het netwerk na een netwerkvariatie-bewerking
"F" : de toegekende verkeersstromen door de verbindingen.

Op deze manier kan de gebruiker na gaan wat het verschil in resultaat is tussen de alternatieven. Er kunnen meerdere tussenresultaten tegelijk opgegeven worden. Dit kan door het (als één woord) invoeren van de gewenste letters. De letters mogen in willekeurige volgorde staan.
Nadat alle keuzes zijn gemaakt, start de ontwerp-procedure. Deze procedure bestaat uit het itereren naar het goedkoopste netwerk dat aan de specificaties voldoet.
Deze specificaties bestaan uit de in de editor opgegeven gewenste arc-connectivity en de gewenste node-connectivity.

5 Conclusie

Ten behoeve van het college 'Datacommunicatie netwerken' (vakcode 1102B) is een demonstratie-programma ontworpen ter bevordering van het inzicht in netwerk-operaties en -manipulaties.
Dit programma is ontworpen volgens een van te voren gedefiniëerd raamwerk voor de gegevensrepresentatie en programma-organisatie.

Literatuur

[Lit.1] "Computer Networks"
A.S. Tanenbaum
Vrije Universiteit Amsterdam
The Netherlands.
Prentice-Hall, ISBN 0-13-165183-8.

Bijlage Commando's

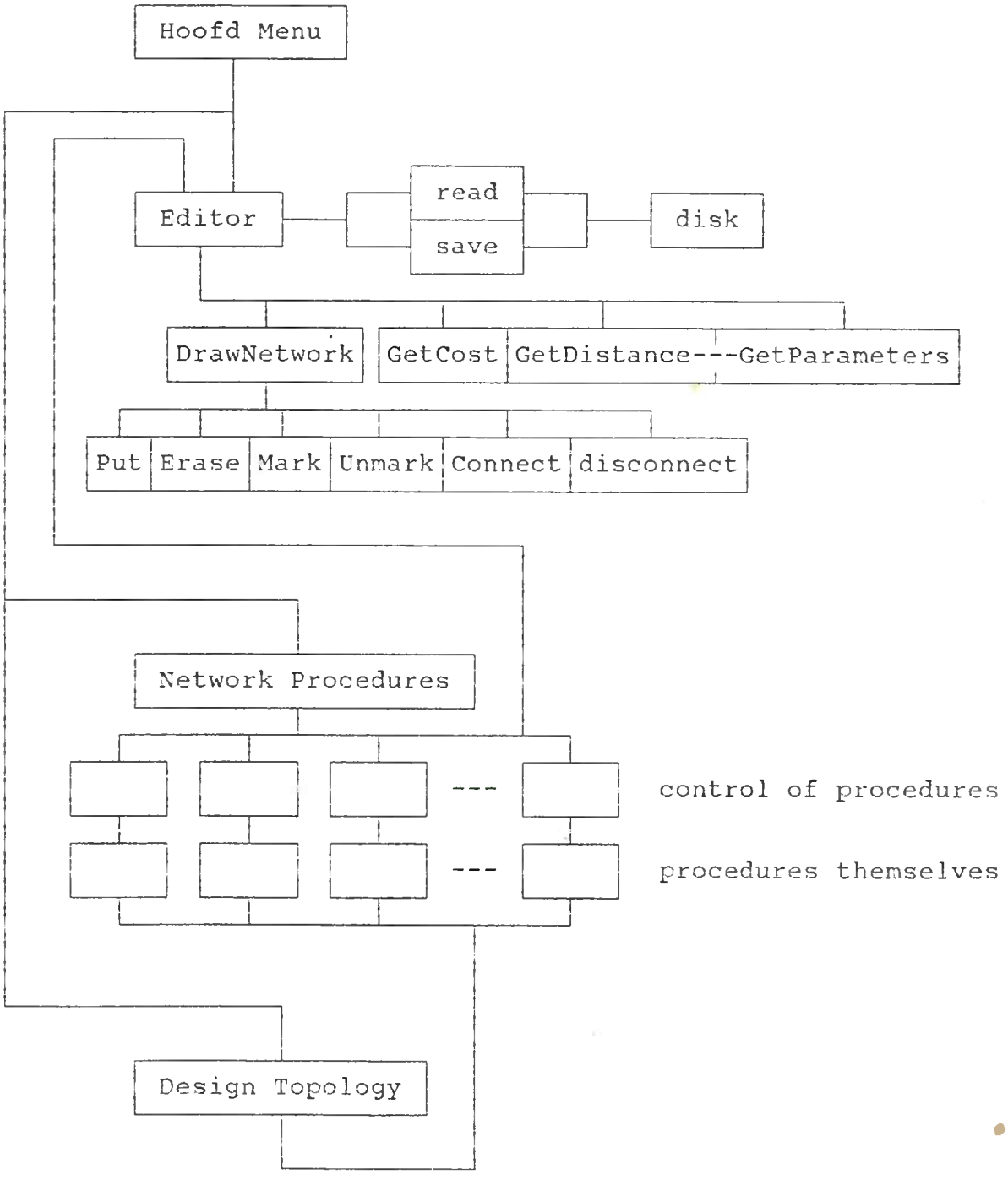
Kies de verschillende menu-opties door het invoeren van het bijbehorende getal. De correctie-toets kan gebruikt worden om foutieve invoer te corrigeren. De invoer wordt gecontroleerd op correctheid en zinnigheid.

In de editor roepen de functie-toetsen de volgende mogelijkheden aan:

F 1 : invoer van de kosten
F 2 : invoer van de afstanden
F 3 : invoer van de capaciteiten
F 4 : invoer van het verkeer
F 5 : invoer van netwerk parameters (Ca, Cn, μ)
F 6 : aanroepen van het teken-gedeelte van de editor
F 7 : opslaan van de huidige netwerk-gegevens op schijf
F 8 : inlezen van netwerk-gegevens bij de opgegeven netwerk-naam van schijf
F 9 : invoer van een nieuwe netwerk-naam en de keuze uit "invoeren" of "laten-zien" van de gegevens
F10 : verlaten van de editor.

In het teken-gedeelte van de editor zijn de volgende commando's mogelijk :

p(ut)	: plaats op het aangegeven punt een knooppunt
e(rase)	: verwijder het knooppunt dat zich op het aangegeven punt bevindt
m(ark)	: markeer het knooppunt op het aangegeven punt
u(nmark)	: demarkeer het knooppunt op het aangegeven punt
c(onnect)	: verbindt het gemarkeerde knooppunt met het knooppunt op het aangegeven punt
d(isconnect)	: verwijder de verbinding tussen het gemarkeerde knooppunt en het knooppunt op de aangegeven plaats
+	: verhoog de cursor-stapgrootte met 10 dots/stap
-	: verlaag de cursor-stapgrootte met 10 dots/stap
r(eady)	: verlaat het teken-gedeelte van de editor.



figuur 1. Verbindings-diagram.

Appendix De programmatuur.

Alle source-code is geschreven in pascal. Hiervoor is gebruik gemaakt van het pakket "Turbo Pascal" van de firma Borland. Dit pakket kent de mogelijkheid van "units". Een "unit" is een onafhankelijk stuk code en vergelijkbaar met een "library". Een "unit" wordt op zich gecompileerd. De "unit" kan opgenomen worden in een ander stuk code (een "unit" of een "program"). De in de "unit" gedefiniëerde procedures en functies kunnen dan gebruikt worden, zonder dat deze als source-code aanwezig zijn.

Bij de ontwikkeling van het demonstratie-programma zijn de volgende "units" gebruikt :

- data: hierin zijn alle gebruikte pascal-typen en constantes gedefiniëerd, tesamen met de globale variabelen; tevens worden de globale variabelen hier geïnitialiseerd.
- funcs: hierin staan alle hulp en in- en uitvoer procedures en functies.
- draw: hierin staan de procedures en functies die al het tekenwerk verrichten en de daarbijbehorende sturing (zie §4.3.2).
- editor: dit is de editor-module.
- netprocs: dit is de procedure-module.
- des_topo: dit is de ontwerp-module.
- netdemo: dit is de menu-module.

Aangezien het programma in zijn huidige vorm dient als basis om op voort te bouwen, zijn de bij de ontwikkeling gebruikte "units" niet samengevoegd tot de vier modules die in hoofdstuk 4 genoemd worden. Dit vergemakkelijkt het voortbouwen aan het programma.

Hier volgen de listings van de 7 pascal-files.

DATA.PAS

```
unit data;

interface

const
   COS         =  1;   { index to array "specified" }
   DIS         =  2;
   TRA         =  3;
   CAP         =  4;
   PAR         =  5;

   TRVAR       =  1;   { index to array "intermediate_result" }
   TRFLOW      =  2;
   TRCAP       =  3;
   TRROUTE     =  4;

   MAXNODES    = 10;   { maximum number of nodes in the network }
   TWOMAXNODES = 22;  { = 2 * ( MAXNODES + 1 ) }

   BLANK       = ' ';
   BACKSPACE   = #8;
   TAB         = #9;
   NEWLINE     = #13;

   F1          = #59;  { these are the key-codes of the function-keys }
   F2          = #60;
   F3          = #61;
   F4          = #62;
   F5          = #63;
   F6          = #64;
   F7          = #65;
   F8          = #66;
   F9          = #67;
   F10         = #68;

type
   booleanfive        = array [1..5] of boolean;
   booleanarray       = array [1..MAXNODES] of boolean;
   booleanmatrix      = array [1..MAXNODES,1..MAXNODES] of boolean;
   integermatrix      = array [1..MAXNODES,1..MAXNODES] of integer;
   realmatrix         = array [1..MAXNODES,1..MAXNODES] of real;
   shortstring        = string [20];    { intended for names (of files, networks) }
   largebooleanmatrix = array [1..TWOMAXNODES,1..TWOMAXNODES] of boolean; { for use }
   largeintegermatrix = array [1..TWOMAXNODES,1..TWOMAXNODES] of integer; { see netprocs.pas }
   coordinate         = record
                           x          : integer;
                           y          : integer;
                        end;
   coordinates        = array [1..MAXNODES] of coordinate;
   coordinatematrix   = array [1..MAXNODES,1..MAXNODES] of coordinate;
```

```
   parameters        = record
                         number_of_nodes          : 0..MAXNODES;
                         nodecoordinates          : coordinates;
                         line_positioncoordinates : coordinatematrix;
                         connectionsmatrix        : booleanmatrix;
                         Cn                       : integer;
                         Ca                       : integer;
                         mu                       : integer;
                         T                        : real;
                         total_cost               : real;
                       end;
   networkdata       = record
                         specified   : booleanfive;
                         distance    : integermatrix;
                         traffic     : integermatrix;
                         cost        : integermatrix;
                         capacity    : integermatrix;
                         routering   : integermatrix;
                         networkpars : parameters;
                       end;

var
   name         : shortstring;
   filename     : shortstring;
   netfile      : file of networkdata;
   result       : networkdata;
   resultfile   : file of networkdata;
   driver, mode : integer;
   Crosshair    : pointer;  { for use in draw.pas to record a graphic-image }
   max_flow     : realmatrix;
   traffic_flow : realmatrix;   { matrices to record results in the design process }
   delay        : realmatrix;
   routing      : integermatrix;

implementation

{ Initialise variables }

var
   i, j : integer;

begin
   name := '';
   with result do
   begin
      for i := 1 to MAXNODES do
         for j := 1 to MAXNODES do
         begin
            cost[i,j] := 0;
            traffic[i,j] := 0;
            capacity[i,j] := 0;
            distance[i,j] := MAXINT;
         end;
      for i := 1 to 5 do
         specified[i] := false;
```

```
      with networkpars do
      begin
         number_of_nodes := 0;
         Cn := 0;
         Ca := 0;
         mu := 0;
         T  := 0;
         total_cost := 0;
         for i := 1 to MAXNODES do
         begin
            nodecoordinates[i].x := 0;
            nodecoordinates[i].y := 0;
            for j := 1 to MAXNODES do
            begin
               line_positioncoordinates[i,j].x := 0;
               line_positioncoordinates[i,j].y := 0;
               connectionsmatrix[i,j] := false;
            end;
         end;
      end;
   end;
   driver := 0; { DETECT graphics board }
end.  { data unit }
```

FUNCS.PAS

```
unit funcs;

interface

uses Crt, graph, data;

function  GetChar : char;
function  GetKey : char;
function  GetFunctionKey ( var functionkey : boolean ) : char; { was char function-key ? }
function  GetName : shortstring;
function  GetNameGraph : shortstring;  { for use in graphics-mode }
function  GetInt : integer;
function  GetIntGraph : integer;       { for use in graphics-mode }
procedure Wait_for_key;
procedure ClrLines ( from, onto : integer );
function  Sgn ( number : integer ) : integer;
function  FileExists ( name : shortstring ) : boolean;
function  AskChoice ( max : integer ) : integer;
procedure PrintIntegermatrix ( number_of_nodes : integer; matrix : integermatrix );
procedure PrintLargeIntegermatrix ( number_of_nodes : integer; matrix : largeintegermatrix );
procedure PrintRealmatrix ( number_of_nodes : integer; matrix : realmatrix );
procedure Erase_Char ( charnumber : integer );
procedure Erase_CharXY ( cpx, cpy, charnumber : integer );
procedure MoveTo_Graphline ( linenumber : integer );
procedure GraphComment ( linenumber : integer; comment : string );
function  Long_to_Int ( number : longint ) : integer;
procedure DrawBorder;
procedure DrawCrosshair ( xpos, ypos : integer );
procedure MoveCrosshairTo ( x, y : integer );
function  NodeNearby ( var x, y : integer; node : coordinates; number_of_nodes : integer ) : boolean;
function  NodeNearby50 ( var x, y : integer; node : coordinates; number_of_nodes : integer ) : boolean;


implementation

function Getc ( var line : shortstring ) : char;

begin
   Getc := line[1];
   Delete ( line, 1, 1 );
end;  { Getc }


procedure Strip_spaces ( var comment : shortstring );

begin
   while ( Length ( comment ) > 0 ) and ( comment[1] in [BLANK, TAB] ) do
      Delete ( comment, 1, 1 );
   while ( Length ( comment ) > 0 ) and ( comment[Length ( comment )] in [BLANK, TAB] ) do
      Delete ( comment, Length ( comment ), 1 );
end;  { Strip_spaces }
```

```
function Make_standard_shortstring ( comment : shortstring ) : shortstring;

var
   temp : shortstring;

begin
   temp := '';
   Strip_spaces ( comment );
   while Length ( comment ) > 0 do
   begin
      temp := temp + Getc ( comment );
      if comment[1] = TAB then
         comment[1] := BLANK;
      if comment[1] = BLANK then
      begin
         while ( Length ( comment ) > 1 ) and ( comment[2] in [BLANK, TAB] ) do
            Delete ( comment, 2, 1 );
         if comment[2] = ',' then
            Delete ( comment, 1, 1 );
      end;
      if comment[1] = ',' then
         Insert ( BLANK, comment, 2 );
   end;
   Make_standard_shortstring := temp;
end;  { Make_standard_shortstring }


function GetChar : char;

var
   c : char;

begin
   c := ReadKey;
   if c = #0 then
      c := ReadKey;
   GetChar := c;
end;  { GetChar }


function GetKey : char;

var
   c : char;

begin
   c := UpCase ( ReadKey );
   if c = #0 then
   begin
      c := ReadKey;
      c := Chr ( Ord(c) + 128 );
   end;
   GetKey := c;
end;  { GetKey }
```

```
procedure Erase_Char ( charnumber : integer );

var
   cpx, cpy             : integer;
   CharHight, CharWidth : integer;
   x, y                 : integer;

begin
   cpx := GetX;
   cpy := GetY;
   CharWidth := GetMaxX DIV 80;
   CharHight := 8;
   SetFillStyle ( EmptyFill, Black );
   Bar ( cpx, cpy, cpx + ( charnumber * CharWidth ), cpy + ( CharHight - 1 ) );
end;  { Erase_Char }


function GetFunctionKey ( var functionkey : boolean ) : char;

var
   c : char;

begin
   functionkey := false;
   c := ReadKey;
   if c = #0 then
   begin
      functionkey := true;
      c := ReadKey;
   end;
   GetFunctionKey := c;
end;  { GetFunctionKey }


function GetName : shortstring;

var
   i    : integer;
   c    : char;
   temp : string;

begin
   readln ( temp );
   temp := Make_standard_shortstring ( temp );
   while Length ( temp ) > 20 do
      Delete ( temp, Length ( temp ), 1 );
   GetName := temp;
end;  { GetName }
```

```
function GetNameGraph : shortstring;

var
   i    : integer;
   c    : char;
   line : string;

begin
   line := '';
   i := 1;
   c := GetChar;
   while ( not ( c = NEWLINE ) ) and ( i <= 20 ) do
   begin
      if c <> BACKSPACE then
         OutText ( c );
      if ( c = BACKSPACE ) and ( Length ( line ) > 0 ) then
      begin
         Delete ( line, Length ( line ), 1 );
         MoveTo ( GetX-8, GetY );
         Erase_Char ( 1 );
         i := i - 1;
      end
      else
         if ( c <> BACKSPACE ) then
         begin
            line := line + c;
            i := i + 1;
         end;
      c := GetChar;
   end;
   GetNameGraph := line;
end;  { GetNameGraph }
```

```
function GetInt : integer;

var
   sum   : integer;
   c     : char;
   error : boolean;
   code  : integer;
   line  : shortstring;

begin
   sum := 0;
   error := false;
   line := '';
   readln ( line );
   Strip_spaces ( line );
   if Length ( line ) = 0 then
      line := 'a';
   while Length ( line ) > 0 do
   begin
      c := Getc ( line );
      if not ( c in ['0'..'9'] ) then
         error := true
      else
         sum := ( 10 * sum ) + ord(c) - ord('0');
   end;
   if error then
      GetInt := -1
   else
      GetInt := sum;
end;  { GetInt }
```

```
function GetIntGraph : integer;

var
   sum   : integer;
   c     : char;
   error : boolean;
   line  : shortstring;

begin
   sum := 0;
   line := '';
   error := false;
   c := GetChar;
   while not ( c = NEWLINE ) do
   begin
      if c <> BACKSPACE then
         OutText ( c );
      if ( c = BACKSPACE ) and ( Length ( line ) > 0 ) then
      begin
         Delete ( line, Length ( line ), 1 );
         MoveTo ( GetX-8, GetY );
         Erase_Char ( 1 );
      end
      else
         if ( c <> BACKSPACE ) then
            line := line + c;
      c := GetChar;
   end;
   if line = '' then
      error := true
   else
      while ( Length ( line ) > 0 ) do
      begin
         c := Getc ( line );
         if not ( c in ['0'..'9'] ) then
            error := true
         else
            sum := ( 10 * sum ) + ord(c) - ord('0');
      end;
   if error then
      GetIntGraph := -1
   else
      GetIntGraph := sum;
end;  { GetIntGraph }


procedure Wait_for_key;

var
   dummy : char;

begin
   dummy := GetChar;
end;  { Wait_for_key }
```

```
procedure ClrLines ( from, onto : integer );

var
   i : integer;

begin
   for i := from to onto do
   begin
      GotoXY ( 1, i );
      ClrEol;
   end;
   GotoXY ( 1, from );
end;  { ClrLines }


function Sgn ( number : integer ) : integer;

begin
   if number > 0 then
      Sgn := 1
   else
      if number < 0 then
         Sgn := -1
      else
         Sgn := 0;
end;  { Sgn }


function FileExists ( name : shortstring ) : boolean;

var
   fileid : file;
   exists : boolean;

begin
   assign ( fileid, name );
   {$I-} reset ( fileid ) {$I+};
   exists := ( IOResult = 0 );
   if exists then
      close ( fileid );
   FileExists := exists;
end;  { FileExists }
```

```
function AskChoice ( max : integer ) : integer;

var
   option : integer;

begin
   write ( 'make your choice ( 1 - ', max:2, ' ) : ' );
   option := GetInt;
   while ( option < 1 ) or ( option > max ) do
   begin
      write ( 'No option, try again : ' );
      option := GetInt;
   end;
   AskChoice := option;
end;  { AskChoice }


procedure PrintIntegermatrix ( number_of_nodes : integer; matrix : integermatrix );

var
  i, j : integer;

begin
   for i := 1 to number_of_nodes do
   begin
      for j := 1 to number_of_nodes do
         write ( matrix[i,j]:6 );
      writeln;
   end;
end;  { PrintIntegermatrix }


procedure PrintLargeIntegermatrix ( number_of_nodes : integer; matrix : largeintegermatrix );

var
  i, j : integer;

begin
   for i := 1 to number_of_nodes do
   begin
      for j := 1 to number_of_nodes do
         write ( matrix[i,j]:6 );
      writeln;
   end;
end;  { PrintLargeIntegermatrix }
```

```
procedure Printrealmatrix ( number_of_nodes : integer; matrix : realmatrix );

var
  i, j : integer;

begin
   for i := 1 to number_of_nodes do
   begin
      for j := 1 to number_of_nodes do
         write ( matrix[i,j]:6 );
      writeln;
   end;
end;  { Printrealmatrix }


procedure Erase_CharXY ( cpx, cpy, charnumber : integer );

var
   CharHight, CharWidth : integer;
   x, y                 : integer;

begin
   CharWidth := GetMaxX DIV 80;
   CharHight := 8;
   SetFillStyle ( EmptyFill, Black );
   Bar ( cpx, cpy, cpx + ( charnumber * CharWidth ), cpy + ( CharHight - 1 ) );
end;  { Erase_CharXY }


procedure MoveTo_Graphline ( linenumber : integer );

var
   CharHight : integer;
   Graphline : integer;
   i, j      : integer;

begin
   CharHight := Round ( GetMaxY / 25 );
   Graphline := CharHight * ( linenumber - 1 );
   MoveTo ( 5, Graphline + 2 );
   SetFillStyle ( EmptyFill, Black );
   Bar ( 1, Graphline + 1, GetMaxX - 1, Graphline + CharHight - 1 );
end;  { MoveTo_Graphline }


procedure GraphComment ( linenumber : integer; comment : string );

var
   cpx, cpy : integer;

begin
   cpx := GetX;
   cpy := GetY;
   MoveTo_Graphline ( linenumber );
   OutText ( comment );
   MoveTo ( cpx, cpy );
end;  { GraphComment }
```

```
function Long_to_Int ( number : longint ) : integer;

type
   WordRec = record
               Low, High : word;
             end;

begin
   Long_to_Int := WordRec ( number ).Low;
end;  { Long_to_Int }


procedure DrawBorder;

var
   ViewPort : ViewPortType;

begin
   SetLineStyle ( SolidLn, 0, NormWidth );
   GetViewSettings ( ViewPort );
   with ViewPort do
      Rectangle ( 0, 0, x2-x1, y2-y1 );
end;  { DrawBorder }


procedure DrawCrosshair ( xpos, ypos : integer );

var
   ulx, uly : integer;
   lrx, lry : integer;
   Size     : word;

begin
   MoveTo ( xpos, ypos );
   Line ( xpos-5, ypos, xpos+5, ypos );
   Line ( xpos, ypos-5, xpos, ypos+5 );
   ulx := xpos-5;
   uly := ypos-5;
   lrx := xpos+5;
   lry := ypos+5;
   Size := ImageSize ( ulx, uly, lrx, lry );
   GetMem ( Crosshair, Size );
   GetImage ( ulx, uly, lrx, lry, Crosshair^ );
end;  { DrawCrosshair }


procedure MoveCrosshairTo ( x, y : integer );

var
   xold, yold : integer;

begin
   xold := GetX;
   yold := GetY;
   PutImage ( xold-5, yold-5, Crosshair^, XORput );
   MoveTo ( x, y );
   PutImage ( x-5, y-5, Crosshair^, XORput );
end;  { MoveCrosshairTo }
```

```
function NodeNearby ( var x, y : integer; node : coordinates; number_of_nodes : integer ) : boolean;
var
   found : boolean;
   i     : integer;

begin
   i := 1;
   found := false;
   if number_of_nodes > 0 then
      repeat
         if ( x <= ( node[i].x + 20 ) ) and
            ( x >= ( node[i].x - 20 ) ) and
            ( y <= ( node[i].y + 20 ) ) and
            ( y >= ( node[i].y - 20 ) ) then
         begin
            x := node[i].x;
            y := node[i].y;
            MoveCrosshairTo ( x, y );
            found := true;
         end
         else
            i := i + 1;
      until found or ( i > number_of_nodes );
   NodeNearby := found;
end;  { NodeNearby }


function NodeNearby50 ( var x, y : integer; node : coordinates; number_of_nodes : integer ) : boolean;

var
   found : boolean;
   i     : integer;

begin
   i := 1;
   found := false;
   if number_of_nodes > 0 then
      repeat
         if ( x <= ( node[i].x + 50 ) ) and
            ( x >= ( node[i].x - 50 ) ) and
            ( y <= ( node[i].y + 50 ) ) and
            ( y >= ( node[i].y - 50 ) ) then
         begin
            x := node[i].x;
            y := node[i].y;
            MoveCrosshairTo ( x, y );
            found := true;
         end
         else
            i := i + 1;
      until found or ( i > number_of_nodes );
   NodeNearby50 := found;
end;  { NodeNearby50 }

end.  { functies unit }
```

DRAW.PAS

```
unit draw;

interface

uses Crt, data, funcs, graph;

procedure ShowNetwork ( networkparameters : parameters );
procedure DrawNetwork ( var name : shortstring; var network : networkdata; show : integer );

implementation

var
   number_of_nodes      : integer;
   node                 : coordinates;
   line_position        : coordinatematrix;
   connection           : booleanmatrix;
   x, y                 : integer;
   xstep, ystep         : integer;
   xasp, yasp           : word;
   marked_node          : integer;
   first_node           : boolean;
   text                 : string;
   CharHight, CharWidth : integer;
   showvalue            : integer;
   currentnetwork       : networkdata;

procedure Initialise_Matrices;

var
   i,j : integer;

begin
   for i := 1 to MAXNODES do
   begin
      for j := 1 to MAXNODES do
      begin
         line_position[i,j].x := 0;
         line_position[i,j].y := 0;
         connection[i,j] := false;
      end;
      node[i].x := 0;
      node[i].y := 0;
   end;
end;  { Initialise_Matrices }
```

```
function GetNodeNumber ( x, y : integer ) : integer;

var
   i : integer;

begin
   GetNodeNumber := 0;
   for i := 1 to number_of_nodes do
      if ( node[i].x = x ) and ( node[i].y = y ) then
         GetNodeNumber := i;
end;  { GetNodeNumber }


procedure Draw_Node ( x, y, number : integer );

begin
   MoveCrosshairTo ( 10, 10 );
   Str ( number:2, text );
   OutTextXY ( x-9, y, text );
   Circle ( x, y, 20 );
   MoveCrosshairTo ( x, y );
end;  { Draw_Node }


procedure Draw_Line ( x1, y1, x2, y2 : real );

var
   hypothenusa, dx, dy : real;
   px1, px2, py1, py2  : integer;

begin
   if ( x1 <> x2 ) or ( y1 <> y2 ) then
   begin
      hypothenusa := Sqrt ( Sqr ( y2-y1 ) + Sqr ( x2-x1 ) );
      dx := ( 20 * ( x2-x1 ) ) / hypothenusa;
      dy := ( 20 * ( y2-y1 ) ) / hypothenusa * ( xasp / yasp );
      px1 := Round ( x1 + dx );
      px2 := Round ( x2 - dx );
      py1 := Round ( y1 + dy );
      py2 := Round ( y2 - dy );
      Line ( px1, py1, px2, py2 );
   end;
end;  { Draw_Line }
```

```
procedure Remove_Line ( source, destination : integer );

var
   hypothenusa, x1, y1, x2, y2, dx, dy : real;
   px1, px2, py1, py2                  : integer;

begin
   if source <> destination then
   begin
      x1 := node[source].x;
      y1 := node[source].y;
      x2 := node[destination].x;
      y2 := node[destination].y;
      hypothenusa := Sqrt ( Sqr ( y2-y1 ) + Sqr ( x2-x1 ) );
      dx := ( 20 * ( x2-x1 ) ) / hypothenusa;
      dy := ( 20 * ( y2-y1 ) ) / hypothenusa * ( xasp / yasp );
      px1 := Round ( x1 + dx );
      px2 := Round ( x2 - dx );
      py1 := Round ( y1 + dy );
      py2 := Round ( y2 - dy );
      SetColor ( Black );
      Line ( px1, py1, px2, py2 );
      if driver in [1..4] then     { if color-graphics card then }
         SetColor ( Yellow )
      else
         SetColor ( 1 );
   end;
end;  { Remove_Line }


function Length_Line ( source, destination : integer ) : integer;

var
   x1, y1, x2, y2 : real;

begin
   if source <> destination then
   begin
      x1 := node[source].x;
      y1 := node[source].y;
      x2 := node[destination].x;
      y2 := node[destination].y;
      Length_Line := Round ( Sqrt ( Sqr ( y2-y1 ) + Sqr ( x2-x1 ) ) );
   end
   else
      Length_Line := 0;
end;  { Length_Line }
```

```
procedure Put_node ( x, y : integer );

begin
   if GetNodeNumber ( x, y ) = 0 then
   begin
      if number_of_nodes < MAXNODES then
      begin
         number_of_nodes := number_of_nodes + 1;
         node[number_of_nodes].x := x;
         node[number_of_nodes].y := y;
         Draw_Node ( x, y, number_of_nodes );
      end
      else
         GraphComment ( 22, 'past limit on number of nodes!' );
   end;
end;  { Put_Node }


procedure Mark_Node ( x, y : integer );

var
   i : integer;

begin
   if not first_node then
   begin
      MoveCrosshairTo ( 10, 10 );
      MoveTo ( node[marked_node].x-9, node[marked_node].y );
      Str ( marked_node:2, text );
      Erase_Char ( 2 );
      OutText ( text );
      MoveTo ( 10, 10 );
      MoveCrosshairTo ( x, y );
      first_node := true;
   end;
   marked_node := GetNodeNumber ( x, y );
   if marked_node > 0 then
   begin
      MoveCrosshairTo ( 10, 10 );
      first_node := false;
      MoveTo ( x-9, y );
      Erase_Char ( 2 );
      OutText ( ' *' );
      MoveTo ( 10, 10 );
      MoveCrosshairTo ( x, y );
   end;
end;  { Mark_Node }
```

```
procedure Unmark_Node;

var
   cpx, cpy : integer;

begin
   if not first_node then
   begin
      cpx := GetX;
      cpy := GetY;
      MoveCrosshairTo ( 10, 10 );
      MoveTo ( node[marked_node].x-9, node[marked_node].y );
      Erase_Char ( 2 );
      Str ( marked_node:2, text );
      OutText ( text );
      first_node := true;
      MoveTo ( 10, 10 );
      MoveCrosshairTo ( cpx, cpy );
   end;
end;  { Unmark_Node }


procedure Write_by_line_position ( value : integer );

var
   source, destination : integer;

begin
   if value > 0 then
      with currentnetwork do
         for source := 1 to number_of_nodes do
            for destination := 1 to source - 1 do
               if connection[source,destination] then
               begin
                  Erase_CharXY ( line_position[source,destination].x - 8, line_position[source,destination].y, 3 );
                  case value of
                     COS : Str ( cost[source,destination]:3, text );
                     DIS : Str ( distance[source,destination]:3, text );
                     CAP : Str ( capacity[source,destination]:3, text );
                     TRA : Str ( traffic[source,destination]:3, text );
                  end;
                  OutTextXY ( line_position[source,destination].x - 8, line_position[source,destination].y, text );
               end;
end;  { Write_by_line_position }
```

```
procedure Connect_Node ( x, y : integer );

var
   node_to_connect, length : integer;

begin
   node_to_connect := GetNodeNumber ( x, y );
   if first_node then
      Mark_Node ( x, y )
   else
      if ( marked_node <> node_to_connect ) then
      begin
         Draw_Line ( node[marked_node].x, node[marked_node].y, x, y );
         line_position[marked_node,node_to_connect].x := ( x + node[marked_node].x ) DIV 2 - 8;
         line_position[node_to_connect,marked_node].x := line_position[marked_node,node_to_connect].x;
         line_position[marked_node,node_to_connect].y := ( y + node[marked_node].y ) DIV 2;
         line_position[node_to_connect,marked_node].y := line_position[marked_node,node_to_connect].y;
         connection[marked_node,node_to_connect] := true;
         connection[node_to_connect,marked_node] := true;
         currentnetwork.distance[marked_node,node_to_connect] := Length_Line ( marked_node, node_to_connect );
         currentnetwork.distance[node_to_connect,marked_node] :=
                                                    currentnetwork.distance[marked_node, node_to_connect];
         currentnetwork.specified[DIS] := true;
         Write_by_line_position ( showvalue );
      end;
end;  { Connect_Node }
```

```
procedure Disconnect_Node ( x, y : integer );

var
   node_to_disconnect : integer;

begin
   node_to_disconnect := GetNodeNumber ( x, y );
   if first_node then
      Mark_Node ( x, y )
   else
      if ( marked_node <> node_to_disconnect ) then
      begin
         Remove_Line ( marked_node, node_to_disconnect );
         if showvalue > 0 then
            Erase_CharXY ( line_position[marked_node,node_to_disconnect].x - 8,
                           line_position[marked_node,node_to_disconnect].y, 3 );
         line_position[marked_node,node_to_disconnect].x := 0;
         line_position[node_to_disconnect,marked_node].x := 0;
         line_position[marked_node,node_to_disconnect].y := 0;
         line_position[node_to_disconnect,marked_node].y := 0;
         connection[marked_node,node_to_disconnect] := false;
         connection[node_to_disconnect,marked_node] := false;
         with currentnetwork do
         begin
            cost[marked_node,node_to_disconnect] := 0;
            cost[node_to_disconnect,marked_node] := 0;
            capacity[marked_node,node_to_disconnect] := 0;
            capacity[node_to_disconnect,marked_node] := 0;
            traffic[marked_node,node_to_disconnect] := 0;
            traffic[node_to_disconnect,marked_node] := 0;
            distance[marked_node,node_to_disconnect] := MAXINT;
            distance[node_to_disconnect,marked_node] := MAXINT;
         end;
         Write_by_line_position ( showvalue );
      end;
end;  { Disconnect_Node }
```

```
procedure ShowNetwork ( networkparameters : parameters );

var
   source, destination : integer;
   number_of_nodes     : integer;
   node                : coordinates;
   connection          : booleanmatrix;

begin
   ClearViewPort;
   GetAspectRatio ( xasp, yasp );
   number_of_nodes := networkparameters.number_of_nodes;
   node := networkparameters.nodecoordinates;
   connection := networkparameters.connectionsmatrix;
   MoveTo ( node[number_of_nodes].x, node[number_of_nodes].y );
   if number_of_nodes <> 0 then
   begin
      DrawCrosshair ( GetX, GetY );
      for source := 1 to number_of_nodes do
      begin
         Draw_Node ( node[source].x, node[source].y, source );
         for destination := 1 to source - 1 do
            if connection[source,destination] then
               Draw_Line ( node[source].x,node[source].y,node[destination].x,node[destination].y );
      end;
      if not first_node then
         Mark_node ( node[marked_node].x, node[marked_node].y );
      PutImage ( GetX-5, GetY-5, Crosshair^, XORput );
   end;
end;  { ShowNetwork }
```

```
procedure Erase_Node ( x, y : integer );

var
   node_to_erase, i, j : integer;
   newparameters       : parameters;

begin
   node_to_erase := GetNodeNumber ( x, y );
   for i := node_to_erase to number_of_nodes - 1 do
      for j := 1 to number_of_nodes do
      begin
         with currentnetwork do
         begin
            cost[i,j] := cost[i+1,j];
            capacity[i,j] := capacity[i+1,j];
            traffic[i,j] := traffic[i+1,j];
            distance[i,j] := distance[i+1,j];
         end;
         line_position[i,j].x := line_position[i+1,j].x;
         line_position[i,j].y := line_position[i+1,j].y;
         connection[i,j] := connection[i+1,j];
      end;
   for j := node_to_erase to number_of_nodes - 1 do
      for i := 1 to number_of_nodes do
      begin
         with currentnetwork do
         begin
            cost[i,j] := cost[i,j+1];
            capacity[i,j] := capacity[i,j+1];
            traffic[i,j] := traffic[i,j+1];
            distance[i,j] := distance[i,j+1];
         end;
         line_position[i,j].x := line_position[i,j+1].x;
         line_position[i,j].y := line_position[i,j+1].y;
         connection[i,j] := connection[i,j+1];
      end;
   for i := node_to_erase to number_of_nodes - 1 do
   begin
      node[i].x := node[i+1].x;
      node[i].y := node[i+1].y;
   end;
```

```
   for i := 1 to number_of_nodes do
   begin
      with currentnetwork do
      begin
         cost[i,number_of_nodes] := 0;
         cost[number_of_nodes,i] := 0;
         capacity[i,number_of_nodes] := 0;
         capacity[number_of_nodes,i] := 0;
         traffic[i,number_of_nodes] := 0;
         traffic[number_of_nodes,i] := 0;
         distance[i,number_of_nodes] := MAXINT;
         distance[number_of_nodes,i] := MAXINT;
      end;
      line_position[i,number_of_nodes].x := 0;
      line_position[number_of_nodes,i].x := 0;
      line_position[i,number_of_nodes].y := 0;
      line_position[number_of_nodes,i].y := 0;
      connection[i,number_of_nodes] := false;
      connection[number_of_nodes,i] := false;
   end;
   number_of_nodes := number_of_nodes - 1;
   newparameters.number_of_nodes := number_of_nodes;
   newparameters.nodecoordinates := node;
   newparameters.line_positioncoordinates := line_position;
   newparameters.connectionsmatrix := connection;
   ShowNetwork ( newparameters );
   Write_by_line_position ( showvalue );
   MoveTo ( x, y );
   PutImage ( GetX-5, GetY-5, Crosshair^, XORput );
end;  { Erase_Node }


procedure DrawNetwork ( var name : shortstring; var network : networkdata; show : integer );

var
   action, save     : char;
   ready, cursorkey : boolean;
   filename         : shortstring;
   present          : boolean;

begin
   first_node := true;
   marked_node := 0;
   ready := false;
   CharHight := Round ( GetMaxY / 25 );
   CharWidth := GetMaxX DIV 80;
   SetViewPort ( 0, 0, GetMaxX, GetMaxY - ( CharHight * 3 ), ClipOn );
   DrawBorder;
   SetViewPort ( 1, 1, GetMaxX-1, GetMaxY - ( CharHight * 3 ) - 1, ClipOn );
   currentnetwork := network;
   number_of_nodes := currentnetwork.networkpars.number_of_nodes;
   node := currentnetwork.networkpars.nodecoordinates;
   line_position := currentnetwork.networkpars.line_positioncoordinates;
   connection := currentnetwork.networkpars.connectionsmatrix;
   showvalue := show;
   GetAspectRatio ( xasp, yasp );
   if number_of_nodes > MAXNODES then
      number_of_nodes := 0;
```

```
if number_of_nodes = 0 then
   Initialise_Matrices
else
begin
   ShowNetwork ( currentnetwork.networkpars );
   Write_by_line_position ( showvalue );
end;
x := 10;
y := 10;
xstep := 10;
ystep := 10;
DrawCrosshair ( x, y );
repeat
   action := GetFunctionKey ( cursorkey );
   case action of
      #80 : if cursorkey and ( y < ( 320 - ystep ) ) then y := y + ystep;
      #72 : if cursorkey and ( y > ystep )           then y := y - ystep;
      #77 : if cursorkey and ( x < ( 720 - xstep ) ) then x := x + xstep;
      #75 : if cursorkey and ( x > xstep )           then x := x - xstep;
      '+' : if xstep < 100 then
            begin
               xstep := xstep + 10;
               ystep := ystep + 10;
            end;
      '-' : if xstep > 10 then
            begin
               xstep := xstep - 10;
               ystep := ystep - 10;
            end;
      'p' : if not NodeNearby50 ( x, y, node, number_of_nodes ) then
               Put_Node ( x, y )
            else
               GraphComment ( 22, 'Other node(s) too close' );
      'e' : if NodeNearby ( x, y, node, number_of_nodes ) then
               Erase_Node ( x, y )
            else
               GraphComment ( 22, 'No node present to erase' );
      'm' : if NodeNearby ( x, y, node, number_of_nodes ) then
               Mark_Node ( x, y )
            else
               GraphComment ( 22, 'No node present to mark' );
      'u' : if not first_node then
               Unmark_Node;
      'c' : if NodeNearby ( x, y, node, number_of_nodes ) then
               Connect_Node ( x, y )
            else
               GraphComment ( 22, 'No node present to connect to' );
      'd' : if NodeNearby ( x, y, node, number_of_nodes ) then
               Disconnect_Node ( x, y )
            else
               GraphComment ( 22, 'No node present to disconnect from' );
      #13, 'r' : ready := true;
   end;
   MoveCrosshairTo ( x, y );
until ready;
```

```
    Unmark_Node;
    PutImage ( GetX-5, GetY-5, Crosshair^, XORput );
    GraphComment ( 22, '' );
    currentnetwork.networkpars.number_of_nodes := number_of_nodes;
    currentnetwork.netwcrkpars.nodecoordinates := node;
    currentnetwork.networkpars.line_positioncoordinates := line_position;
    currentnetwork.networkpars.connectionsmatrix := connection;
    network := currentnetwork;
    SetViewPort ( 1, 1, GetMaxX-1, GetMaxY - ( Round ( GetMaxY / 25 ) * 2 ), ClipOn );
end;  { DrawNetwork }

end.  { draw }
```

EDITOR.PAS

```
unit editor;

interface

uses Crt, graph, data, funcs, draw;

procedure Edit ( var name : shortstring; comment : string; var network : networkdata );

implementation

procedure Edit ( var name : shortstring; comment : string; var network : networkdata );

var
   save              : char;
   filename          : shortstring;
   i                 : 1..5;
   number_of_nodes   : integer;
   node              : coordinates;
   line_position     : coordinatematrix;
   connection        : booleanmatrix;
   choice            : char;
   exit, functionkey : boolean;
   temp              : integer;
   text              : string;
   tempstr1, tempstr2 : string;
   showvalue         : integer;
   enterdata         : boolean;


procedure GetCost ( var cost : integermatrix );

var
   x, y : 1..MAXNODES;
   save : char;

begin
   GraphComment ( 23, ' Enter Costs' );
   save := 'n';
   if number_of_nodes = 0 then
      GraphComment ( 24, 'no network known yet, please specify network' )
   else
   begin
      for x := 1 to number_of_nodes do
      begin
         Str ( cost[x,x]:3, text );
         Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
         OutTextXY ( node[x].x - 8, node[x].y, text );
         for y := x + 1 to number_of_nodes do
            if connection[x,y] then
            begin
               Str ( cost[x,y]:3, text );
               Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
               OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
            end;
      end;
```

```
      if enterdata then
      begin
         for x := 1 to number_of_nodes do
         begin
            Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
            OutTextXY ( node[x].x - 8, node[x].y, '***' );
            MoveTo_Graphline ( 23 );
            OutText ( ' Enter cost of this node : ( now ' );
            Str ( cost[x,x]:3, text );
            OutText ( text );
            OutText ( ' ) ' );
            temp := GetIntGraph;
            if temp <> -1 then
               cost[x,x] := temp;
            Str ( cost[x,x]:3, text );
            Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
            OutTextXY ( node[x].x - 8, node[x].y, text );
            for y := x + 1 to number_of_nodes do
               if connection[x,y] then
               begin
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, '***' );
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter variable cost of connection ' );
                  Str ( x:2, tempstr1 );
                  Str ( y:2, tempstr2 );
                  text := Concat ( '(from ', tempstr1, ' to ', tempstr2, ') ( now ' );
                  OutText ( text );
                  Str ( cost[x,y]:3, text );
                  OutText ( text );
                  OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     cost[x,y] := temp;
                  Str ( cost[x,y]:3, text );
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter basic cost of this connection : ( now ' );
                  Str ( cost[y,x]:3, text );
                  OutText ( text ); OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     cost[y,x] := temp;
               end;
         end;
         network.specified[COS] := true;
      end;
      showvalue := COS;
   end;
end; { GetCost }
```

```
procedure GetDistance ( var distance : integermatrix );

var
   x, y : 1..MAXNODES;
   save : char;

begin
   GraphComment ( 23, ' Enter Distances' );
   save := 'n';
   if number_of_nodes = 0 then
      GraphComment ( 24, 'no network known yet, please specify network' )
   else
   begin
      for x := 1 to number_of_nodes do
      begin
         Str ( x:2, text );
         Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
         OutTextXY ( node[x].x - 8, node[x].y, text );
         for y := x + 1 to number_of_nodes do
            if connection[x,y] then
            begin
               Str ( distance[x,y]:3, text );
               Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
               OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
            end;
      end;
      if enterdata then
      begin
         for x := 1 to number_of_nodes do
            for y := x + 1 to number_of_nodes do
               if connection[x,y] then
               begin
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, '***' );
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter distance of connection ' );
                  Str ( x:2, tempstr1 );
                  Str ( y:2, tempstr2 );
                  text := Concat ( '(from ', tempstr1, ' to ', tempstr2, ') ( now ' );
                  OutText ( text );
                  Str ( distance[x,y]:3, text );
                  OutText ( text );
                  OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     distance[x,y] := temp;
                  Str ( distance[x,y]:3, text );
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
                  distance[y,x] := distance[x,y];
               end;
         network.specified[DIS] := true;
      end;
      showvalue := DIS;
   end;
end; { GetDistance }
```

```
procedure GetCapacities ( var capacity : integermatrix );

var
   x, y : 1..MAXNODES;
   save : char;

begin
   GraphComment ( 23, ' Enter Capacities' );
   save := 'n';
   if number_of_nodes = 0 then
      GraphComment ( 24, 'no network known yet, please specify network' )
   else
   begin
      for x := 1 to number_of_nodes do
      begin
         Str ( x:2, text );
         Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
         OutTextXY ( node[x].x - 8, node[x].y, text );
         for y := x + 1 to number_of_nodes do
            if connection[x,y] then
            begin
               Str ( capacity[x,y]:3, text );
               Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
               OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
            end;
      end;
      if enterdata then
      begin
         for x := 1 to number_of_nodes do
            for y := x + 1 to number_of_nodes do
               if connection[x,y] then
               begin
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, '***' );
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter capacity of connection ' );
                  Str ( x:2, tempstr1 );
                  Str ( y:2, tempstr2 );
                  text := Concat ( '(from ', tempstr1, ' to ', tempstr2, ') ( now ' );
                  OutText ( text );
                  Str ( capacity[x,y]:3, text );
                  OutText ( text );
                  OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     capacity[x,y] := temp;
                  Str ( capacity[x,y]:3, text );
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
                  capacity[y,x] := capacity[x,y];
               end;
         network.specified[CAP] := true;
      end;
      showvalue := CAP;
   end;
end; { GetCapacities}
```

```
procedure GetTraffic ( var traffic : integermatrix );

var
   x, y : 1..MAXNODES;
   save : char;

begin
   GraphComment ( 23, ' Enter Traffic' );
   save := 'n';
   if number_of_nodes = 0 then
      GraphComment ( 24, 'no network known yet, please specify network')
   else
   begin
      for x := 1 to number_of_nodes do
      begin
         Str ( x:2, text );
         Erase_CharXY ( node[x].x - 8, node[x].y, 3 );
         OutTextXY ( node[x].x - 8, node[x].y, text );
         for y := x + 1 to number_of_nodes do
            if connection[x,y] then
            begin
               Str ( traffic[x,y]:3, text );
               Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
               OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
            end;
      end;
```

```
      if enterdata then
      begin
         for x := 1 to number_of_nodes do
            for y := x + 1 to number_of_nodes do
               if connection[x,y] then
               begin
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, '***' );
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter traffic on this connection : ' );
                  Str ( x:2, tempstr1 );
                  Str ( y:2, tempstr2 );
                  text := Concat ( '(from ', tempstr1, ' to ', tempstr2, ') ( now ' );
                  OutText ( text );
                  Str ( traffic[x,y]:3, text );
                  OutText ( text ); OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     traffic[x,y] := temp;
                  MoveTo_Graphline ( 23 );
                  OutText ( 'Enter traffic on this connection : ' );
                  text := Concat ( '(from ', tempstr2, ' to ', tempstr1, ') ( now ' );
                  OutText ( text );
                  Str ( traffic[y,x]:3, text );
                  OutText ( text ); OutText ( ' ) ' );
                  temp := GetIntGraph;
                  if temp <> -1 then
                     traffic[y,x] := temp;
                  Str ( traffic[x,y]:3, text );
                  Erase_CharXY ( line_position[x,y].x - 8, line_position[x,y].y, 3 );
                  OutTextXY ( line_position[x,y].x - 8, line_position[x,y].y, text );
               end;
         network.specified[TRA] := true;
      end;
      showvalue := TRA;
   end;
end; { GetTraffic }
```

```
procedure GetParameters ( var networkpars : parameters );

var
   save : char;

begin
   GraphComment ( 23, ' Enter parameters' );
   save := 'n';
   MoveTo_Graphline ( 23 );
   OutText ( 'Enter desired arc-connectivity of this network : ' );
   if network.specified[PAR] then
   begin
      Str ( networkpars.Ca:2, tempstr1 );
      text := Concat ( '( now ', tempstr1, ' ) ' );
      OutText ( text );
   end;
   temp := GetIntGraph;
   if temp <> -1 then
   begin
      while ( temp < 1 ) do
      begin
         MoveTo_Graphline ( 23 );
         OutText ( 'Error : number not greater than 0, try again : ' );
         temp := GetIntGraph;
      end;
      networkpars.Ca := temp;
   end;
   MoveTo_Graphline ( 23 );
   OutText ( 'Enter desired node-connectivity of this network : ' );
   if network.specified[PAR] then
   begin
      Str ( networkpars.Cn:2, tempstr1 );
      text := Concat ( '( now ', tempstr1, ' ) ' );
      OutText ( text );
   end;
   temp := GetIntGraph;
   if temp <> -1 then
   begin
      while ( temp < 1 ) do
      begin
         MoveTo_Graphline ( 23 );
         OutText ( 'Error : number not greater than 0, try again : ' );
         temp := GetIntGraph;
      end;
      networkpars.Cn := temp;
   end;
```

```
   MoveTo_Graphline ( 23 );
   OutText ( 'Enter mean packet size ( bits/packet ) : ' );
   if network.specified[PAR] then
   begin
      Str ( networkpars.mu:2, tempstr1 );
      text := Concat ( '( now ', tempstr1, ' ) ' );
      OutText ( text );
   end;
   temp := GetIntGraph;
   if temp <> -1 then
   begin
      while ( temp < 1 ) do
      begin
         MoveTo_Graphline ( 23 );
         OutText ( 'Error : number not greater than 0, try again : ' );
         temp := GetIntGraph;
      end;
      networkpars.mu := temp;
   end;
   network.specified[PAR] := true;
end;  { GetParameters }


procedure SaveData;

begin
   network.networkpars.number_of_nodes := number_of_nodes;
   network.networkpars.nodecoordinates := node;
   network.networkpars.line_positioncoordinates := line_position;
   network.networkpars.connectionsmatrix := connection;
   filename := name + '.NET';
   assign ( netfile, filename );
   rewrite ( netfile );
   write ( netfile, network );
   close ( netfile );
end;  { SaveData }


procedure GetOptions;

var
   temp : shortstring;

begin
   MoveTo_Graphline ( 23 );
   OutText ( 'Enter new name for this network : (<return> for old name ) ' );
   temp := GetNameGraph;
   if temp <> '' then
      name := temp;
   MoveTo_Graphline ( 23 );
   OutText ( 'Enter or Show networkdata ? ( e or s ) ' );
   temp := GetNameGraph;
   if temp <> '' then
      enterdata := ( temp[1] = 'e' );
end;  { GetOptions }
```

```
procedure ReadNetwork;

var
   i, j : integer;

begin
   GraphComment ( 23, ' reading network-data' );
   if FileExists ( name + '.NET' ) then
   begin
      assign ( netfile, name + '.NET' );
      reset ( netfile );
      read ( netfile, network );
      close ( netfile );
   end
   else
      with network do
      begin
         for i := 1 to MAXNODES do
         begin
            for j := 1 to MAXNODES do
            begin
               distance[i,j] := MAXINT;
               traffic[i,j] := 0;
               cost[i,j] := 0;
               capacity[i,j] := 0;
               routering[i,j] := 0;
               networkpars.line_positioncoordinates[i,j].x := 0;
               networkpars.line_positioncoordinates[i,j].y := 0;
               networkpars.connectionsmatrix[i,j] := false;
            end;
            networkpars.nodecoordinates[i].x := 0;
            networkpars.nodecoordinates[i].y := 0;
         end;
         with networkpars do
         begin
            number_of_nodes := 0;
            Cn := 0;
            Ca := 0;
            mu := 0;
            T := 0;
            total_cost := 0;
         end;
         for i := 1 to 5 do
            specified[i] := false;
      end;
   showvalue := 0;
end;  { ReadNetwork }
```

```
procedure ReadData;

var
   i       : integer;
   newname : shortstring;
   absent  : boolean;

begin
   ClearViewPort;
   ReadNetwork;
   absent := false;
   for i := 1 to 5 do
   begin
      MoveTo_Graphline ( i + 1 );
      if not network.specified[i] then
      begin
         absent := true;
         case i of
            COS : OutText ( 'Costs are unknown yet.' );
            DIS : OutText ( 'Distances are unknown yet.' );
            TRA : OutText ( 'Traffic is unknown yet.' );
            CAP : OutText ( 'Capacities are unknown yet.' );
            PAR : OutText ( 'Parameters are unknown yet.' );
         end;
      end;
   end;
   if absent then
   begin
      MoveTo_Graphline ( 15 );
      OutText ( '         ------ Press any key to continue ------ ' );
      Wait_for_key;
   end;
   if network.specified[PAR] then
   begin
      number_of_nodes := network.networkpars.number_of_nodes;
      node := network.networkpars.nodecoordinates;
      line_position := network.networkpars.line_positioncoordinates;
      connection := network.networkpars.connectionsmatrix;
      ShowNetwork ( network.networkpars );
   end
   else
   begin
      number_of_nodes := 0;
      ClearViewPort;
   end;
   showvalue := 0;
end;  { ReadData }
```

```
{ main body }

begin
   InitGraph ( driver, mode, 'c:\usr\bin' );
   if GraphResult <> grOk then
   begin
      writeln ( 'Graphics error!' );
      writeln ( 'Program aborted...' );
      Halt(1);
   end;
   ClearViewPort;
   DirectVideo := false;
   ClearDevice;
   if driver in [1..4] then       { if color-graphics card then }
      SetColor ( Yellow );
   SetViewPort ( 0, 0, GetMaxX, GetMaxY, ClipOn );
   DrawBorder;
   SetViewPort ( 1, 1, GetMaxX-1, GetMaxY-1, ClipOn );
   exit := false;
   while name = '' do
   begin
      MoveTo_Graphline ( 3 );
      OutText ( 'Enter name of network : ' );
      name := GetNameGraph;
   end;
   ClearViewPort;
   ReadNetwork;
   if not network.specified[PAR] then
   begin
      MoveTo_Graphline ( 22 );
      OutText ( 'Network does not exist yet' );
      number_of_nodes := 0;
   end
   else
   begin
      number_of_nodes := network.networkpars.number_of_nodes;
      node := network.networkpars.nodecoordinates;
      line_position := network.networkpars.line_positioncoordinates;
      connection := network.networkpars.connectionsmatrix;
      ShowNetwork ( network.networkpars );
   end;
   MoveTo_Graphline ( 25 );
   OutText ( 'F1:cost F2:dist F3:caps F4:traf F5:pars F6:draw F7:save F8:read F9:opt F10:exit' );
   if comment <> '' then
   begin
      MoveTo_Graphline ( 24 );
      OutText ( comment );
   end;
   enterdata := true;
```

```
  SetViewPort ( 1, 1, GetMaxX-1, GetMaxY - ( Round ( GetMaxY / 25 ) * 2 ), ClipOn );
  repeat                                    { leave last 2 lines unchangable }
    temp := GetMaxY - ( Round ( GetMaxY / 25 ) * 3 ) - 1;
    Line ( 0, temp, GetMaxX, temp ); { draw line between drawing and status-bar }
    MoveTo_Graphline ( 23 );
    text := Concat ( 'Network name is : ', name, '         make your choice' );
    OutText ( text );
    choice := GetFunctionkey ( functionkey );  { wait for user to press key }
    if functionkey then      { indicates function-key pressed }
      case choice of
        F1 : GetCost ( network.cost );
        F2 : GetDistance ( network.distance );
        F3 : GetCapacities ( network.capacity );
        F4 : GetTraffic ( network.traffic );
        F5 : GetParameters ( network.networkpars );
        F6 : begin
               GraphComment ( 23, ' drawing network' );
               DrawNetwork ( name, network, showvalue );
               if network.networkpars.number_of_nodes > 0 then
                 network.specified[PAR] := true;
             end;
        F7 : begin
               GraphComment ( 23, ' saving network-data' );
               SaveData;
             end;
        F8 : ReadData;
        F9 : GetOptions;
        F10 : exit := true;
      end;
    if network.specified[PAR] then
    begin
      number_of_nodes := network.networkpars.number_of_nodes;
      node := network.networkpars.nodecoordinates;
      line_position := network.networkpars.line_positioncoordinates;
      connection := network.networkpars.connectionsmatrix;
    end
    else
      number_of_nodes := 0;
  until exit;
  DirectVideo := true;
  CloseGraph;
  RestoreCrtMode;
  window ( 1, 1, 80, 25 );
  GotoXY ( 1, 3 );
  write ( 'Save data (y/n) ? ' );
  readln ( save );
  if ( save = 'y' ) or ( save = 'Y' ) then
    SaveData;
end;  { Edit }

end.  { editor unit }
```

NETPROCS.PAS

```
{
   Remarks to maintenance :

   As is described in Tanenbaum's book, node-connectivity and node-disjoint
   paths are calculated by splitting a node into two nodes with a directed
   link between them and by splitting a link into two directed links
   connected to the new nodes in a specific way. With respect to this program,
   this means that the size of the data-matrices is doubled. To accommodate
   this transformation process, two pascal-types are defined :
      - "largebooleanmatrix" for boolean data
      - "largeintegermatrix" for integer data.
   The function "MaxFlow" uses a "largeintegermatrix" for input of the
   link-capacities. All procedures and functions using this function "MaxFlow"
   (either direct or indirect) are to transform the normal-sized
   capacity matrix into a large-sized matrix. Examples of this transformation
   can be found in the following procedures/functions :
      - Malhotra
      - Determine_connectivity_Kleitman
      - Determine_connectivity_Even       { mind the extra node! }
      - Arcdisjoint
      - Nodedisjoint.

   All other aspects regarding the implementation are straightforward.

   Wishing following developers much success with their activities.

                                                R.V. van de Ree.
}


unit netprocs;

interface

uses Crt, data, funcs, editor;

procedure Dijkstra    ( var name : shortstring; var network : networkdata );
procedure Malhotra    ( var name : shortstring; var network : networkdata );
procedure Kleitman    ( var name : shortstring; var network : networkdata );
procedure Even        ( var name : shortstring; var network : networkdata );
procedure Steiglitz   ( var name : shortstring; var network : networkdata );
procedure Arcdisjoint ( var name : shortstring; var network : networkdata );
procedure Nodedisjoint ( var name : shortstring; var network : networkdata );
procedure MonteCarlo  ( var name : shortstring; var network : networkdata );
procedure MeanDelay   ( var name : shortstring; var network : networkdata );
procedure AssignFlow  ( var name : shortstring; var network : networkdata );
procedure AssignCapacity ( var name : shortstring; var network : networkdata );
procedure Cost        ( var name : shortstring; var network : networkdata );
procedure BranchExchange ( var name : shortstring; var network : networkdata );
procedure SaturatedCut ( var name : shortstring; var network : networkdata );
procedure ConnectivityRestoring ( var name : shortstring; var network : networkdata );
```

```
implementation

const
   INFINITY  = MAXINT;
   UNSCANNED = -MAXNODES;


procedure Dijkstra ( var name : shortstring; var network : networkdata );

type
   node       = 0..MAXNODES;
   lab        = ( perm, tent );
   NodeLabel  = record
                   predecessor : node;
                   length      : integer;
                   tag         : lab;
                end;
   GraphState = array [1..MAXNODES] of NodeLabel;

var
   number_of_nodes : integer;
   a               : integermatrix;
   s, t, i, k      : node;
   path            : array [1..MAXNODES] of node;
   state           : GraphState;
   min             : integer;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network and distance', network );
   while not network.specified[DIS] do
      Edit ( name, 'no distance known yet, please specify distance', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   a := network.distance;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Dijkstra shortest path algorithm' );
   writeln;
   writeln;
   write ( 'Enter number of source-node : ' );
   s := GetInt;
   while ( s < 1 ) or ( s > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      s := GetInt;
   end;
   write ( 'Enter number of destination-node : ' );
   t := GetInt;
   while ( t < 1 ) or ( t > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      t := GetInt;
   end;
```

```
   for i := 1 to MAXNODES do
      with state[i] do
      begin
         predecessor := 0;
         length := INFINITY;
         tag := tent;
      end;
   state[t].length := 0;
   state[t].tag := perm;
   k := t;
   repeat
{
*** calculate distances to all tentative nodes
}
      for i := 1 to number_of_nodes do
         if ( a[k,i] <> INFINITY ) and ( state[i].tag = tent ) then
            if state[k].length + a[k,i] < state[i].length then
            begin
               state[i].predecessor := k;
               state[i].length := state[k].length + a[k,i];
            end;
{
*** search for shortest intermediate path
}
      min := INFINITY;
      k := 0;
      for i := 1 to number_of_nodes do
         if ( state[i].tag = tent ) and ( state[i].length < min ) then
         begin
            min := state[i].length;
            k := i;
         end;
      state[k].tag := perm;
   until k = s;
{
*** als kortste paden destination alle sources berekend moeten worden,
*** dan ander einderiterium
}
   k := s;
   i := 0;
   ClrLines ( 1, 22 );
   writeln ( ' Shortest path : ' );
   repeat
      i := i + 1;
      path[i] := k;
      write ( k:4 );
      k := state[k].predecessor;
   until k = 0;
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Dijkstra }
```

```
function MaxFlow ( s, t : integer; c : largeintegermatrix; number_of_nodes : integer ) : integer;

type
   node     = 1..TWOMAXNODES;
   xnode    = -TWOMAXNODES..TWOMAXNODES;
   vector   = array [node] of xnode;
   WhichWay = ( push, pull );

var
   RefNode, i, j : node;
   MinPotential  : integer;
   layer         : vector;
   f             : largeintegermatrix;
   maximumflow   : integer;


function min ( x, y : integer ) : integer;

begin
   if x < y then
      min := x
   else
      min := y;
end;  { min }


procedure walk ( i : node);

var
   j  : node;
   li : xnode;

begin
   layer[i] := -layer[i];
   li := layer[i];
   for j := 1 to number_of_nodes do
      if ( j <> s ) and ( -layer[j] = li - 1 ) and ( ( f[j,i] < c[j,i] ) or ( f[i,j] > 0 ) ) then
         walk ( j );
end;  { walk }
```

```
function LayeringPossible : boolean;

var
   i, j       : node;
   k          : 0..TWOMAXNODES;
   EmptyLayer : boolean;

begin
   k := 0;
   for i := 1 to number_of_nodes do
      layer[i] := UNSCANNED;
   layer[s] := k;
   repeat
      k := k + 1;
      EmptyLayer := true;
      for i := 1 to number_of_nodes do
         if -layer[i] = k - 1 then
            for j := 1 to number_of_nodes do
               if ( layer[j] = UNSCANNED ) and ( ( f[i,j] < c[i,j] ) or ( f[j,i] > 0 ) ) then
               begin
                  layer[j] := -k;
                  EmptyLayer := false;
               end;
   until ( layer[t] <> UNSCANNED ) or EmptyLayer;
   LayeringPossible := not EmptyLayer;
   walk ( t );
end;  { LayeringPossible }


procedure FindRefNode ( i : node);

var
   j            : node;
   li,lj        : xnode;
   InCap, OutCap : integer;

begin
   li := layer[i];
   InCap := 0;
   OutCap := 0;
   for j := 1 to number_of_nodes do
   begin
      lj := layer[j];
      if ( lj = li - 1 ) and ( j <> s ) and ( ( f[j,i] < c[j,i] ) or ( f[i,j] > 0 ) ) then
         FindRefNode ( j );
      if lj = li - 1 then
         InCap := InCap + ( c[j,i] - f[j,i] ) + f[i,j];
      if lj = li + 1 then
         OutCap := OutCap + ( c[i,j] - f[i,j] ) + f[j,i];
   end;
   if ( i <> s ) and ( i <> t ) and ( min ( Incap, OutCap ) < MinPotential ) then
   begin
      MinPotential := min ( InCap, OutCap );
      RefNode := i
   end;
end;  { FindRefNode }
```

```
procedure PushPull ( i : node; FlowLeft : integer; p : WhichWay );

var
   j, k1, k2, LayerSought : 0..TWOMAXNODES;
   r                      : integer;

begin
   j := 0;
   while ( FlowLeft > 0 ) and ( j < number_of_nodes ) do
   begin
      j := j + 1;
      if p = push then
      begin
         k1 := i;
         k2 := j;
         LayerSought := layer[i] + 1;
      end
      else
      begin
         k1 := j;
         k2 := i;
         LayerSought := layer[i] - 1;
      end;
      r := min ( FlowLeft, c[k1,k2] - f[k1,k2] + f[k2,k1] );
      if ( r > 0 ) and ( layer[j] = LayerSought ) then
      begin
         FlowLeft := FlowLeft - r;
         f[k1,k2] := f[k1,k2] + r - min ( r, f[k2,k1] );
         f[k2,k1] := f[k2,k1] - min ( r, f[k2,k1] );
         if ( j <> s ) and ( j <> t ) then
            PushPull ( j, r, p )
      end;
   end;
end;  { PushPull }


begin
   for i := 1 to number_of_nodes do
      for j := 1 to number_of_nodes do
         f[i,j] := 0;
   f[s,t] := c[s,t];
   maximumflow := f[s,t];
   while LayeringPossible do
   begin
      MinPotential := INFINITY;
      FindRefNode ( t );
      PushPull ( RefNode, MinPotential, push );
      PushPull ( RefNode, MinPotential, pull );
      maximumflow := maximumflow + MinPotential;
   end;
   MaxFlow := maximumflow;
end;  { MaxFlow }
```

```
function Determine_connectivity_Kleitman ( Cn : integer; network : networkdata ) : boolean;

var
   newconnection       : largebooleanmatrix;
   unit_capacity       : largeintegermatrix;
   number_of_nodes     : integer;
   source, destination : integer;
   k, i, N             : integer;
   history             : booleanarray;
   k_connected         : boolean;
   connectivity        : integer;

function RandomNode ( number_of_nodes : integer; var history : booleanarray ) : integer;

var
   try : integer;

begin
   repeat
      try := Random ( number_of_nodes );
   until ( try > 0 ) and not history[try];
   history[try] := true;
   RandomNode := try;
end;  { RandomNode }

begin
   number_of_nodes := network.networkpars.number_of_nodes;
   for source := 1 to TWOMAXNODES do
      for destination := 1 to TWOMAXNODES do       { initialise matrices }
      begin
         newconnection[source,destination] := false;
         unit_capacity[source,destination] := 0;
      end;
   for source := 1 to number_of_nodes do          { split nodes into X and X' }
      newconnection[source,source+number_of_nodes] := true;
   for source := 1 to number_of_nodes do
      for destination := 1 to source - 1 do                { reconnect new links }
         if network.networkpars.connectionsmatrix[source,destination] then
         begin
            newconnection[source+number_of_nodes,destination] := true;
            newconnection[destination+number_of_nodes,source] := true;
         end;
   for source := 1 to 2*number_of_nodes do
      for destination := 1 to 2*number_of_nodes do   { construct matrix with }
         if newconnection[source,destination] then      { unit capacity for existing }
            unit_capacity[source,destination] := 1        { links, zero otherwise }
         else
            unit_capacity[source,destination] := 0;
   k_connected := true;
   for i := 1 to number_of_nodes do
      history[i] := false;
```

```
   for k := Cn downto 1 do
   begin
      N := RandomNode ( number_of_nodes, history );
      for i := 1 to number_of_nodes do
         if i <> N then
         begin
            connectivity := MaxFlow ( N + number_of_nodes, i, unit_capacity, 2*number_of_nodes );
            k_connected := k_connected and ( connectivity >= k );
         end;
      for i := 1 to number_of_nodes do
      begin
         unit_capacity[N,i+number_of_nodes] := 0;
         unit_capacity[i,N+number_of_nodes] := 0;
      end;
   end;
   Determine_connectivity_Kleitman := k_connected;
end;  { Determine_connectivity_Kleitman }


function Determine_connectivity_Even ( Cn : integer; network : networkdata ) : boolean;

var
   number_of_nodes      : integer;
   newconnection        : largebooleanmatrix;
   unit_capacity        : largeintegermatrix;
   source, destination  : integer;
   connectivity, X, j, i : integer;
   k_connected          : boolean;

begin
   number_of_nodes := network.networkpars.number_of_nodes;
   for source := 1 to TWOMAXNODES do
      for destination := 1 to TWOMAXNODES do            { initialise matrices }
      begin
         newconnection[source,destination] := false;
         unit_capacity[source,destination] := 0;
      end;
   for source := 1 to number_of_nodes do        { split nodes into X and X' }
      newconnection[source,source+number_of_nodes+1] := true;
   for source := 1 to number_of_nodes do
      for destination := 1 to source - 1 do              { reconnect new links }
         if network.networkpars.connectionsmatrix[source,destination] then
         begin
            newconnection[source+number_of_nodes+1,destination] := true;
            newconnection[destination+number_of_nodes+1,source] := true;
         end;
   for source := 1 to 2*( number_of_nodes + 1 ) do
      for destination := 1 to 2*( number_of_nodes + 1 ) do { construct matrix with }
         if newconnection[source,destination] then          { unit capacity for }
            unit_capacity[source,destination] := 1            { existing links }
         else
            unit_capacity[source,destination] := 0;
   k_connected := true;
```

```
   for source := 1 to Cn do
      for destination := 1 to source - 1 do   { only one of the two possible }
      begin                                   { pairs of nodes is tested }
         connectivity := MaxFlow ( source + number_of_nodes + 1, destination, unit_capacity,
                                                                    2*( number_of_nodes + 1 ) );
         k_connected := k_connected and ( connectivity >= Cn );
      end;
   if k_connected then
   begin
      X := number_of_nodes + 1;                  { introduce extra node }
      unit_capacity[X,X+number_of_nodes+1] := 1;  { and node' }
      for j := Cn + 1 to number_of_nodes do
         if k_connected then
         begin
            for i := 1 to j - 1 do
            begin
               unit_capacity[X+number_of_nodes+1,i] := 1;
               unit_capacity[i+number_of_nodes+1,X] := 1;
            end;
            connectivity := MaxFlow ( j + number_of_nodes + 1, X, unit_capacity,
                                                                    2*( number_of_nodes + 1 ) );
            k_connected := k_connected and ( connectivity >= Cn );
         end;
   end;
   Determine_connectivity_Even := k_connected;
end;  { Determine_connectivity_Even }


procedure Malhotra ( var name : shortstring; var network : networkdata );

var
   number_of_nodes              : integer;
   capacity_large               : largeintegermatrix;
   source_node, destination_node : integer;
   source, destination          : integer;
   maximum_flow                 : integer;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network and capacity', network );
   while not network.specified[CAP] do
      Edit ( name, 'no capacity known yet, please specify capacity', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Malhotra maximum flow algorithm' );
   writeln;
   writeln;
   write ( 'Enter number of source-node : ' );
   source_node := GetInt;
   while ( source_node < 1 ) or ( source_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      source_node := GetInt;
   end;
```

```
   write ( 'Enter number of destination-node : ' );
   destination_node := GetInt;
   while ( destination_node < 1 ) or ( destination_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      destination_node := GetInt;
   end;

   for source := 1 to number_of_nodes do            { transform capacity-matrix }
      for destination := 1 to number_of_nodes do    { into large-matrix }
         capacity_large[source,destination] := network.capacity[source,destination];
   maximum_flow := MaxFlow ( source_node, destination_node, capacity_large, number_of_nodes );

   writeln;
   writeln ( ' The maximum flow from node ', source_node:2, ' to node ', destination_node:2, ' is ', maximum_flow );

   writeln;
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Malhotra }


procedure Kleitman ( var name : shortstring; var network : networkdata );

var
   number_of_nodes : integer;
   Cn              : integer;
   k_connected     : boolean;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Kleitman node connectivity algorithm' );
   writeln;
   writeln;
   write ( 'Enter node-connectivity : ' );
   Cn := GetInt;
   while ( Cn < 1 ) or ( Cn > number_of_nodes ) do
   begin
      write ( 'Error : useless value, try again : ' );
      Cn := GetInt;
   end;

   k_connected := Determine_connectivity_Kleitman ( Cn, network );
```

```
   writeln;
   writeln;
   write ( ' This network ' );
   if k_connected then
      write ( 'does have a' )
   else
      write ( 'has no' );
   write ( ' node-connectivity of ', Cn:2 );
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Kleitman }


procedure Even ( var name : shortstring; var network : networkdata );

var
   number_of_nodes : integer;
   Cn              : integer;
   k_connected     : boolean;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Even node connectivity algorithm' );
   writeln;
   writeln;
   write ( 'Enter node-connectivity : ' );
   Cn := GetInt;
   while ( Cn < 1 ) or ( Cn > number_of_nodes ) do
   begin
      write ( 'Error : useless value, try again : ' );
      Cn := GetInt;
   end;

   k_connected := Determine_connectivity_Even ( Cn, network );

   writeln;
   writeln;
   write ( ' This network ' );
   if k_connected then
      write ( 'does have a' )
   else
      write ( 'has no' );
   write ( ' node-connectivity of ', Cn:2 );
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Even }
```

```
procedure Arcdisjoint ( var name : shortstring; var network : networkdata );

var
   number_of_nodes              : integer;
   source_node, destination_node : integer;
   unit_capacity                : largeintegermatrix;
   source, destination          : integer;
   connectivity                 : integer;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Number of arc-disjoint paths' );
   writeln;
   writeln;
   write ( 'Enter number of source-node : ' );
   source_node := GetInt;
   while ( source_node < 1 ) or ( source_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      source_node := GetInt;
   end;
   write ( 'Enter number of destination-node : ' );
   destination_node := GetInt;
   while ( destination_node < 1 ) or ( destination_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      destination_node := GetInt;
   end;

   for source := 1 to number_of_nodes do        { construct matrix with unit }
      for destination := 1 to number_of_nodes do    { capacity for existing links }
         if network.networkpars.connectionsmatrix[source,destination] then
            unit_capacity[source,destination] := 1
         else
            unit_capacity[source,destination] := 0;
   connectivity := MaxFlow ( source_node, destination_node, unit_capacity, number_of_nodes );

   write ( ' The number of arc-disjoint paths is : ', connectivity );
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Arcdisjoint }
```

```
procedure Nodedisjoint ( var name : shortstring; var network : networkdata );

var
   number_of_nodes              : integer;
   source_node, destination_node : integer;
   newconnection                : largebooleanmatrix;
   unit_capacity                : largeintegermatrix;
   source, destination          : integer;
   connectivity                 : integer;

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Number of node-disjoint paths' );
   writeln;
   writeln;
   write ( 'Enter number of source-node : ' );
   source_node := GetInt;
   while ( source_node < 1 ) or ( source_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      source_node := GetInt;
   end;
   write ( 'Enter number of destination-node : ' );
   destination_node := GetInt;
   while ( destination_node < 1 ) or ( destination_node > number_of_nodes ) do
   begin
      write ( 'Error : node does not exist, try again : ' );
      destination_node := GetInt;
   end;

   for source := 1 to TWOMAXNODES do
      for destination := 1 to TWOMAXNODES do
      begin
         newconnection[source,destination] := false;
         unit_capacity[source,destination] := 0;
      end;
   for source := 1 to number_of_nodes do
      newconnection[source,source+number_of_nodes] := true;
   for source := 1 to number_of_nodes do
      for destination := 1 to source - 1 do
         if network.networkpars.connectionsmatrix[source,destination] then
         begin
            newconnection[source+number_of_nodes,destination] := true;
            newconnection[destination+number_of_nodes,source] := true;
         end;
   for source := 1 to 2*number_of_nodes do
      for destination := 1 to 2*number_of_nodes do
         if newconnection[source,destination] then
            unit_capacity[source,destination] := 1
         else
            unit_capacity[source,destination] := 0;
   source_node := source_node + number_of_nodes;

   connectivity := MaxFlow ( source_node, destination_node, unit_capacity, 2*number_of_nodes );
```

```
   write ( ' The number of node-disjoint paths is : ', connectivity );
   window ( 1, 1, 80, 25 );
   GotoXY ( 20, 23 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
end;  { Nodedisjoint }


procedure MonteCarlo ( var name : shortstring; var network : networkdata );

var
   number_of_nodes                          : integer;
   max, nrounds, np, ip, discon, i, j, k, nr : integer;
   p, NumDiscon, FracDiscon, pincr, seed     : real;
   a, b                                      : booleanmatrix;
   n                                         : integer;

function random : real;

begin
   seed := 125.0 * ( seed + 1.0 );
   seed := seed - 8192.0 * trunc ( seed / 8192 );
   random := ( seed + 0.5 ) / 8192;
end;  { random }

begin
   while network.networkpars.number_of_nodes = 0 do
      Edit ( name, 'no network known yet, please specify network', network );
   number_of_nodes := network.networkpars.number_of_nodes;
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   write ( '   Monte Carlo network reliability simulation' );
   window ( 3, 5, 77, 24 );
   writeln ( ' Enter the following data : ' );
   write ( '  nrounds : ' );
   readln ( nrounds );
   write ( '  p : ' );
   readln ( p );
   write ( '  pincr : ' );
   readln ( pincr );
   write ( '  np : ' );
   readln ( np );
   write ( '  seed : ' );
   readln ( seed );
   clrscr;
   writeln (' nrounds= ',nrounds:5,' p= ',p:5,' pincr= ',pincr:5,' np= ',np:5,' seed= ',seed:5 );
   writeln;
   writeln ( 'possibility_of_defect    average_disconnect_ratio    average_disconnections' );
   window ( 3, 8, 77, 24 );
   n := number_of_nodes;
   max := n * ( n - 1 ) DIV 2;
   a := network.networkpars.connectionsmatrix;
```

```
   for ip := 1 to np do
   begin
      NumDiscon := 0;
      FracDiscon := 0;
      for nr := 1 to nrounds do
      begin
         b := a;
         for i := 1 to n do
            for j := i + 1 to n do
               if b[i,j] then
                  if random < p then
                  begin
                     b[i,j] := false;
                     b[j,i] := false;
                  end;
         for i := 1 to n do
            for j := 1 to n do
               if b[j,i] then
                  for k := 1 to n do
                     b[j,k] := b[j,k] or b[i,k];
         discon := 0;
         for i := 1 to n do
            for j := i + 1 to n do
               if not b[i,j] then
                  discon := discon + 1;
         FracDiscon := FracDiscon + discon/max;
         if discon > 0 then
            NumDiscon := NumDiscon + 1.0;
      end;
      writeln ( '        ', p:6:2,
                '                  ', FracDiscon/nrounds:6:2,
                '                  ', NumDiscon/nrounds:6:2 );
      p := p + pincr;
   end;
   writeln;
   GotoXY ( 20, 16 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { MonteCarlo }


procedure MeanDelay ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   MeanDelay' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { MeanDelay }
```

```
procedure Steiglitz ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Steiglitz' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { Steiglitz }


procedure AssignFlow ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   AssignFlow' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { AssignFlow }


procedure AssignCapacity ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   AssignCapacity' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { AssignCapacity }


procedure Cost ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   Cost' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { Cost }
```

```
procedure BranchExchange ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   BranchExchange' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { BranchExchange }


procedure SaturatedCut ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   SaturatedCut' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { SaturatedCut }


procedure ConnectivityRestoring ( var name : shortstring; var network : networkdata );

begin
   ClrLines ( 1, 25 );
   window ( 3, 3, 77, 24 );
   writeln ( '   ConnectivityRestoring' );
   GotoXY ( 27, 10 );
   write ( 'Not implemented yet!' );
   GotoXY ( 20, 21 );
   write ( '------ Press any key to continue ------' );
   Wait_for_key;
   window ( 1, 1, 80, 25 );
end;  { ConnectivityRestoring }

end.  { netprocs unit }
```

DES_TOPO.PAS

```
unit des_topo;

interface

uses Crt, data, funcs, netprocs, editor;

procedure Design_Topology ( var name : shortstring; var network : networkdata );

implementation

procedure Design_Topology ( var name : shortstring; var network : networkdata );

var
   missing            : boolean;      { used to check presence of network-data }
   filename           : shortstring;
   comment_line       : string;       { used to show user what data is missing }
   i                  : 1..5;
   route, starttopo, capa, perturb, conrestore : char; { used to record choice }
   int_res            : string[4];
   intermediate_result : array [1..4] of boolean;
   minimal_time_delay  : boolean;


procedure InitLocalData;

var
   i : 1..5;

begin
   clrscr;
   GotoXY ( 3, 3 );
   writeln ( ' Design a Topology' );
   writeln;
   for i := 1 to 4 do
      intermediate_result[i] := false;
   while name = '' do
   begin
      write ( ' Enter networkname : ' );
      name := GetName;
   end;
end;  { InitLocalData }
```

```
begin
   InitLocalData;
   repeat
      missing := false;
      for i := 1 to 5 do
         missing := missing or not network.specified[i]; { all data present ? }
      if missing then
      begin
         comment_line := 'No data on :';
         if not network.specified[PAR] then comment_line := comment_line + ' networkdrawing,';
         if not network.specified[COS] then comment_line := comment_line + ' cost,';
         if not network.specified[DIS] then comment_line := comment_line + ' distance,';
         if not network.specified[CAP] then comment_line := comment_line + ' capacities,';
         if not network.specified[TRA] then comment_line := comment_line + ' traffic,';
         comment_line := comment_line + ' please specify!';
         Edit ( name, comment_line, network );  { make user specify data }
      end;
   until not missing;
   clrscr;
   GotoXY ( 3, 3 );
   writeln ( ' Design a Topology' );
   writeln;
   writeln ( ' The networkname is : ', name );
   writeln;
   filename := name + '.RES';
   assign ( resultfile, filename );
   rewrite ( resultfile );
   write ( 'What starttopology-generation method ( S/O/- ) ? : ' );  { ask for choices }
   readln ( starttopo );
   write ( 'What routing method ( D/- ) ? : ' );
   readln ( route );
   write ( 'What capacity-assigning method ( O/E/U/- ) ? : ' );
   readln ( capa );
   write ( 'What network-variation method ( B/S/-) ? : ' );
   readln ( perturb );
   write ( 'With or without connectivity restoring ( Y(es) or N(o) ) ? : ' );
   readln ( connrestore );
   write ( 'What intermediate results do you want to see ( V,F,C,R ) ? : ' );
   readln ( int_res );       { determine what intermediate results to show }
   if pos ( 'V', int_res ) > 0 then intermediate_result[TRVAR] := true;
   if pos ( 'F', int_res ) > 0 then intermediate_result[TRFLOW] := true;
   if pos ( 'C', int_res ) > 0 then intermediate_result[TRCAP] := true;
   if pos ( 'R', int_res ) > 0 then intermediate_result[TRROUTE] := true;
   if pos ( 'v', int_res ) > 0 then intermediate_result[TRVAR] := true;
   if pos ( 'f', int_res ) > 0 then intermediate_result[TRFLOW] := true;
   if pos ( 'c', int_res ) > 0 then intermediate_result[TRCAP] := true;
   if pos ( 'r', int_res ) > 0 then intermediate_result[TRROUTE] := true;
```

```
{ Design program should be specified here }

   with network do
   begin
      minimal_time_delay := false;
      while not minimal_time_delay do
      begin
         case starttopo of
            'S','s' : { Steiglitz };
            'O','o' : ;
         else { default }
         end;
         case route of
            'D','d' : { Dijkstra };
         else { default }
         end;
         if intermediate_result[TRROUTE] then { Show_routing };
         case capa of
            'O','o' : { Optimal };
            'P','p' : { Proportional };
            'U','u' : { Uniform };
         else { default }
         end;
         if intermediate_result[TRCAP] then { Show_Capacity_Assignment };
         case perturb of
            'B','b' : { Branch_Exchange };
            'S','s' : { Saturated_Cut };
         else { default }
         end;
         if intermediate_result[TRVAR] then { Show_Network };
         if { conrestore = 'y' } or { conrestore = 'Y' } then { Restore_Connectivity };
         if intermediate_result[TRFLOW] then { Show_Flow_assignment };
         minimal_time_delay := true;
      end;

      writeln { 'The design-proces should start now, but this part' };
      writeln { 'of the program is not written yet' };

   end;
   close { resultfile };
   GotoXY { 1, 23 };
   write { '        ------ Press any key to continue ------' };
   Wait_for_key;
end;  { Design_Topology }

end.  { des_topo unit }
```

NETDEMO.PAS

```
program main_menu (input,output);

uses Crt, data, funcs, netprocs, editor, des_topo;


procedure PresentMainmenu;

begin
   ClrScr;
   writeln;
   writeln ( 'Main Menu :' );
   writeln;
   writeln ( ' 1 : Editor' );
   writeln ( ' 2 : Procedure Menu' );
   writeln ( ' 3 : Design Topology' );
   writeln ( ' 4 : End of program' );
   writeln;
end;  { PresentMainmenu }


procedure PresentProceduremenu ( var name : shortstring );

begin
   ClrScr;
   writeln;
   writeln ( 'Procedure Menu : ( operations on network : ', name, ' )' );
   writeln;
   writeln ( '      1 : Dijkstra' );
   writeln ( '      2 : Malhotra' );
   writeln ( '      3 : Kleitman' );
   writeln ( '      4 : Even' );
   writeln ( '      5 : Arcdisjoint' );
   writeln ( '      6 : Nodedisjoint' );
   writeln ( '      7 : MonteCarlo' );
   writeln ( '      8 : MeanDelay' );
   writeln ( '      9 : Steiglitz' );
   writeln ( '     10 : AssignFlow' );
   writeln ( '     11 : AssignCapacity' );
   writeln ( '     12 : Cost' );
   writeln ( '     13 : BranchExchange' );
   writeln ( '     14 : SaturatedCut' );
   writeln ( '     15 : ConnectivityRestoring' );
   writeln ( '     16 : Last Menu' );
   writeln;
end;  { PresentProceduremenu }
```

```
procedure ProcedureMenu ( var name : shortstring; var network : networkdata );

const
   NUMBER_OF_PROCEDURE_CHOICES = 16;

var
   choice_made : integer;

begin
   repeat
      PresentProceduremenu ( name );
      choice_made := AskChoice ( NUMBER_OF_PROCEDURE_CHOICES );
      case choice_made of
         1 : Dijkstra ( name, network );
         2 : Malhotra ( name, network );
         3 : Kleitman ( name, network );
         4 : Even     ( name, network );
         5 : Arcdisjoint ( name, network );
         6 : Nodedisjoint ( name, network );
         7 : MonteCarlo ( name, network );
         8 : MeanDelay ( name, network );
         9 : Steiglitz ( name, network );
        10 : AssignFlow ( name, network );
        11 : AssignCapacity ( name, network );
        12 : Cost ( name, network );
        13 : BranchExchange ( name, network );
        14 : SaturatedCut ( name, network );
        15 : ConnectivityRestoring ( name, network );
        NUMBER_OF_PROCEDURE_CHOICES : ;
      end;
   until choice_made = NUMBER_OF_PROCEDURE_CHOICES;
end;  { ProcedureMenu }
```

```
const
   NUMBER_OF_CHOICES = 4;

var
   choice_made : integer;

begin
   TextColor ( yellow );
   LowVideo;
   if ParamCount > 0 then      { network-name given as commandline-parameter ? }
   begin
      name := ParamStr ( 1 );  { if so, then read corresponding file }
      if FileExists ( name + '.NET' ) then
      begin
         assign ( netfile, name + '.NET' );
         reset ( netfile );
         read ( netfile, result );
         close ( netfile );
      end;
   end;
   repeat
      PresentMainmenu;
      choice_made := AskChoice ( NUMBER_OF_CHOICES );
      case choice_made of
         1 : Edit ( name, '', result );
         2 : ProcedureMenu ( name, result );
         3 : Design_Topology ( name, result );
         NUMBER_OF_CBOICES : ;
      end;
   until choice_made = NUMBER_OF_CHOICES;
   clrscr;
end.  { netdemo, mainprogram }
```