```
$ ----------------------------------------------------
$ - La prog   G g                                  -
$ ----------------------------------------------------

$ ----------------------------------------------------
$ Les graphes de tests
$ ----------------------------------------------------

VC5 := {1..5};
EC5 := {[x,y] : x,y in VC5 : x<y};
PC5 := [VC5,EC5];

$ ----------------------------------------------------

VGG := {1..7};
EGG := {[x,y] : x,y in VGG : x < y - 1};
PGG := [VGG,EGG];

$ ----------------------------------------------------

VGG2 := {-3..7};
EGG2 := EGG + {[0,y] : y in {2..7}}+ {[a,b] : a in {-3..-1}, b in  {a+1..7}};
PGG2 := [VGG2,EGG2];

$ ----------------------------------------------------

Vex := {0..7} + {"x"};
Eex := {[0,2],[1,2],[1,3],[2,"x"],[3,"x"],["x",4],["x",5],[4,6],[4,7],[5,7]};
Pex := [Vex,Eex];

$ ----------------------------------------------------
$ Pour entrer directement le graphe orient  de
$ couverture,
$ Calcul de la fermeture transitive
$ ----------------------------------------------------

fermTrans := func(P);
        local V,E,newE,Etmp;
        [V,E] := P;
        newE := E;

        Etmp := {[a,b] : a in V, b in V : [a,b] notin newE and
                                exists z in V : [a,z] in newE and [z,b] in newE};


        while (#Etmp /= 0) do
        newE := newE + Etmp;
        Etmp := {[a,b] : a in V, b in V : [a,b] notin newE and
                exists z in V : [a,z] in newE and [z,b] in newE};

        end;
        return [V,newE];
```

```
end;
$ ----------------------------------------------------

$ ----------------------------------------------------
$ Les fonctions auxilliaires
$ ----------------------------------------------------

$ predecesseurs d'un point x dans l'ordre P ----------
predecesseur := func(P,x);
        local V,E;
        [V,E] := P;
        return {y : y in V : [y,x] in E};
end;
$ ----------------------------------------------------


$ successeurs d'un point x dans l'ordre P ------------
successeur := func(P,x);
        local V,E;
        [V,E] := P;
        return {y : y in V : [x,y] in E};
end;
$ ----------------------------------------------------


$ predecesseurs d'un ensemble de points Ens dans
$ l'ordre P ------------------------------------------
predecesseurE := func(P,Ens);
        local Preds;
        Preds := {y : x in Ens, y in predecesseur(P,x)};
        return Preds - Ens;
end;
$ ----------------------------------------------------


$ successeurs d'un ensemble de points Ens dans
$ l'ordre P ------------------------------------------
successeurE := func(P,Ens);
        local Succs;
        Succs := {y : x in Ens, y in successeur(P,x)};
        return Succs - Ens;
end;
$ ----------------------------------------------------


$ minimum d'un ordre P -------------------------------
minimumO := func(P);
        local V,E;
        [V,E] := P;
        return {x : x in V : #predecesseur(P,x) = 0};
end;
$ ----------------------------------------------------
```

```
$ maximum d'un ordre P ------------------------------
maximumO := func(P);
        local V,E;
        [V,E] := P;
        return {x : x in V : #successeur(P,x) = 0};
end;


$ sous ordre de l'ordre P induit par l'ensemble de
$ points SE ----------------------------------------
sousOrdre := func(P,SE);
        local V,E;
        [V,E] := P;
        return [SE * V, E * {[x,y] : x,y in SE}];
end;
$ ----------------------------------------------------


$ hauteur de l'ordre P ------------------------------
$ d finition de la hauteur selon la CRAS
hauteurAux := func(P,val);
        local V,E;
        [V,E] := P;
        if #E = 0  then return val;
        else
        V := V - minimumO(P);
        P := sousOrdre(P,V);
        return hauteurAux(P, val + 1);
        end;
end;

hauteur := func(P);
        return hauteurAux(P,1);
end;
$ ----------------------------------------------------


$ estUneChaine --------------------------------------
estUneChaine := func(P);
        local V,E;
        [V,E] := P;
        return forall x,y in V : (([x,y] in E or [y,x] in E) or (x = y));
end;
$ ----------------------------------------------------


$ ----------------------------------------------------
$ Calcul de l'ensemble des anticha nes maximales
$ pour l'inclusion d'un ordre Px
$ ----------------------------------------------------
```

```
EnsAntMax := func(Px);
        local Vx,V,E,x,P,EnsAnt,EnsAntPx,A,Ax;

        [Vx,E] := Px;
        if #E = 0 then return {Vx};
        else
        x:=arb (maximumO(Px));
        V := Vx - {x};
        P := sousOrdre(Px, V);
        EnsAnt := EnsAntMax(P);
        EnsAntPx := {};

        for A in EnsAnt do

        Ax := A * predecesseur(Px,x);
        if #Ax = 0
        then
        EnsAntPx := EnsAntPx + {A + {x}};
        else
        EnsAntPx := EnsAntPx + {A};
        if successeurE(P,Ax) subset successeurE(P,(A-Ax))
        then
        EnsAntPx := EnsAntPx + {(A + {x})-Ax};
        end;
        end;
        end for;
        return EnsAntPx;
        end;
end;
$ ----------------------------------------------------


$ ----------------------------------------------------
$ Reconstruction de l'ordre avec les anticha nes
$ ----------------------------------------------------

KR := func(P);
        local vp,ep,vp1,ep1;
        [vp,ep] := P;

        vp1:=EnsAntMax(P);
        ep1 := { [a,b] : a in vp1, b in vp1 : forall x in a | (exists y in b | [x,y] in ep)};
        return [vp1,ep1];
end;

$ ----------------------------------------------------

$ ----------------------------------------------------
$ Pour un meilleur affichage...
$ ----------------------------------------------------
```

```
$ Calcul de la r duction transitive ------------------

ReducTrans := func(P);
        local V,E,newE;
        [V,E] := P;
        newE := {[a,b] : a in V, b in V : [a,b] in E and
                                        (not exists z in V : [a,z] in E and
                                                                [z,b] in E)};
        return [V,newE];
end;
$ ---------------------------------------------------


$ ---------------------------------------------------
$ Calcul de la borne des it rations
$ ---------------------------------------------------

Borne := func(P);
        local b;

        b := 0;

        while (not(estUneChaine(P))) do
        b := b + 1;
        P := KR(P);
        end;
        print ReducTrans(P);
        return b;
end;
$ ---------------------------------------------------
```