Les GRAPHES

1- Définition

Un graphe est un ensemble d'objets appelé sommet et de relation entre ses sommets.

Exemple

Carte de liaison aériennes où les villes sont les sommets du graphe et l'existence d'une liaison aérienne entre 2 villes et une relation de graphe.

- Un graphe orienté est le couple $G = \langle S, A \rangle$ où S : ensemble de sommet et A : ensemble des arcs, arc : est une paire de sommets ordonnée
- Un graphe non orienté est le couple $G = \langle S, A \rangle$

  S : sommet

  A : ensemble d'arêtes

  arête : couple de sommets
- Un graphe valué orienté (respectivement non orienté est le triplet $G = \langle S, A, C \rangle$

  S : sommet

  A : ensemble des arc (resp arêtes)

  C : fonction de $A \rightarrow R$ appelé fonction de côut

. La valuation des graphes permet (mettre des valeurs ds le graphe) par exemple la recherche du plus court chemin entre les sommet des graphes dans un réseau

⚠ Remarque :

Certains graphes admettent des boucles c.à.d des arcs ou arêtes qui connectent un sommet à lui mm

2- Terminologie

* on note $x \to y$ l'arc (x, y)

x : extrémité initiale

y : " terminale

y est le successeur de x

x est le predecesseur de y

* on note $x \longrightarrow y$ l'arête (x, y)

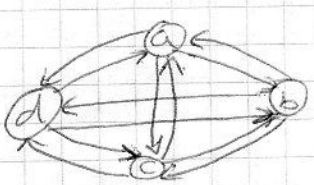
x, y : extremité de l'arête

* Soit $G = \langle S, A \rangle$ le sous graphe de G engendré par $S' \subset S$ est le graphe G' dont les sommets sont des elts de S' et dont les arcs ou arêtes sont de G ayant leur extremité dans S'

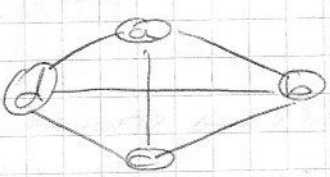
* 2 arcs (respectivement arêtes) d'un graphe orienté (resp non orienté) sont dit adjacents si ils ont au moins une extremité commune

* 2 sommets d'un graphe non orienté sont dits adjacent s'il existe une arête qui les joignent

* Un graphe orienté (resp non orienté) est dit complet si pour tout couples de sommets (x, y) il existe un arc $x \to y$ (resp une arête $x - y$)



graphe orienté complet



g. non orienté complet

* Dans un graphe orienté si x est l'extrémité initiale d'un arc $u = x \to y$, on dit que l'arc u est incident à x vers l'extérieur

- le nombre d'arc ayant cette extrémité initiale x est appelé :
demi-degré extérieur de x est noté $d^{\circ +}(x)$.

* De la même manière on définie un arc incident à x vers l'intérieur si x est l'extrémité terminal de l'arc $y \to x$.
le degré d'un sommet noté : $d^{\circ}(x) = d^{\circ +}(x) + d^{\circ -}(x)$

Exo

$d^{\circ}(a) = d^{\circ +}(a) + d^{\circ -}(a)$

$d^{\circ}(a) = 2 + 1 = 3$

* Dans un graphe orienté G (resp non orienté) on appele chemin (resp chaîne) de longueur λ une suite de $\lambda + 1$ sommet $\{s_0, s_1, \ldots, s_\lambda\}$ tel que : $\forall 0 \leqslant i \leqslant \lambda - 1$

$s_i \to s_{i+1}$ est un arc (resp arête) de G

par convention la longueur d'un chemin d'un sommet vers lui même est nul

* Un chemin (resp chaine) est dite élémentaire s'elle ne contient pas plusieurs fois le même sommet

3 - Type abstrait graphe :

Pour distinguer les sommets d'un graphe il faut les étiquetter en leur attribuant des numéros par exemple.

Sate sommet
utilise entier
opérations
Som : entier → sommet
N° : sommet → entier

3-1 Spécification des graphes orienté :

On prend les conventions suivantes :

- Quand on ajoute un sommet il est isolé (aucun arc incident)
- Quand on ajoute un arc si les sommets adjacent à cet arc n'appartiennent pas aux graphes, on doit les ajouter
- Quand on retire un arc les sommets adjacents ne sont pas retirés
- " " " un sommet : tous les arc incident sont retiré.

Sorte graphe (orienté)

Utilise sommet, entier, booléen

Opérations

graphe-vide : → graphe

ajouter sommet ∪ à ∪ : sommet × graphe → graphe

ajouter arc <-,-> à ∪ : sommet × sommet × graphe → graphe

∪ est sommet de ∪ : sommet × graphe → booléen

<-,-> est-arc de ∪ : sommet × sommet × graphe → booléen

d^{0+} du sommet ∪ de ∪ : sommet × graphe → entier

d^{0-} " " ∪ de ∪ : sommet × graphe → entier

i ème successeur du sommet ∪ de ∪ : entier × sommet × graphe → sommet

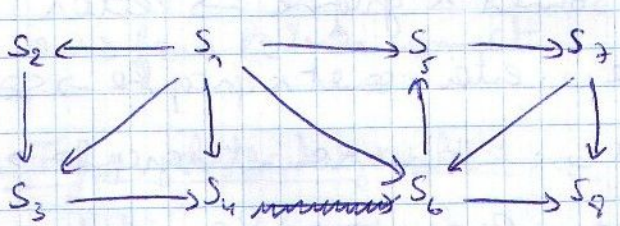
" predecesseur " ∪ de ∪ : entier × sommet × graphe → sommet

retire sommet ∪ de ∪ : sommet × graphe → graphe

" arc <-,-> de ∪ : sommet × sommet × graphe → graphe.

5-1-1 DFS pour graphe représenté par matrice d'adjacence:

```
Procedure profondeur (g: graphe);
Var marquage : array [1..n] of boolean;
    i: integer;
  Procedure prof (s: integer);
  Var i: integer;
  Begin
    marquage [s]:= True;
  For i:=1 to n do
   if g[s,i] and marquage [i] then prof (i);
     end;
  Begin
  for i:=1 to n do
  marquage [i]:= False;
   for i:=1 to n do
  if not (marquage [i]) then prof (i);
     end;
```



5-2 Parcours en longueur:

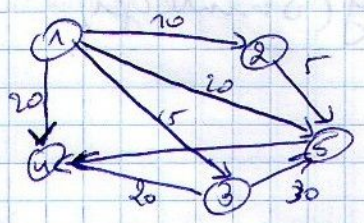
A partir d'un sommet S on commence à visiter tout les successeurs de S, appelant distance de S à y la longueur du plus cours chemin à partir de S vers y, le parcours en largeur consiste à visiter d'abord tout les sommets qui sont à la distance 1 vers S puis ceux qui sont à la distance 2 de S puis ceux qui sont à la distance 3 de S etcainsi de suite

```
Procedure largeur (g: graphe);
 var i: integer;
marquage : array [1..n] of boolean.
 Procedure larg (S: integer);
  var F: File;
  v, w, i: integer;
  Begin
  F:= File-vide;
  marquage[S]:= True;
  ajouter (F, S)
  while not (File vide(F)) do
Begin
V:= Premier (F);
retirer (F);
For i:=1 to d°+(v) do
  Begin
  w:= i ème succ(v) dans g;
  if not( marquage[w]) then
  Begin
  marquage[w]:= True;
  ajouter (F, w);
  end;
end;
end;
end;
end;
  Begin
  for i:=1 to n do
```

```
marquage [i]:= False;
For i:=1 to n do
if not
```

Exo 1:

Ecrire un ss pgme qui calcule la somme des poids des arcs d'un graphe orienté valué




```
const n=5
type Ptr = ^elt;
  elt = record
  som : 1..n;
  Poids : integer;
  suiv : ptr;
  end;
Graphe = array[1..n] of Ptr;
  Function Some (G:graphe): integer;
  Var i,S : integer; P:Ptr;
  Begin
  S:=0
  for i:=1 to n do
  P:= G[i]
  while P <> nil do
  Begin
  S:= S+1 P^.Poids
  P:= P^.suiv;
  end;
  end;
Some := S
```

Exo 2

Ecrire un S/ pgme qui calcule le nombre de predecesseurs d'un sommet dans un graphe orienté

```
Function Nb_Pred ( G : graphe ; som integer ) : integer ;
Var Nb, i : integer;
Function cherche ( P : Ptr ) : boolean;
  Begin
  while (P<>nil) and (P^.suiv <> som ) do P := P^.suiv
   cherche := P<>nil;
   end;
   Begin
   Nb := 0
  For i := 1 to som -1 do
  if cherche (G[i]) then
   Nb := Nb+1
    For i := somet + 1 to n do
  if cherche (G[i]) then
  Nb := Nb + 1;
  Nb_Pred := Nb;
   end.
```

Exo3

Soit $G = \langle S, A \rangle$ un graphe orienté ; r est dite racine de G s'il existe un chemin ~~issu~~ issu de r qui permet d'atteindre tous les sommet du graphe.

Ecrire un s/ pgme qui teste l'existance d'une racine

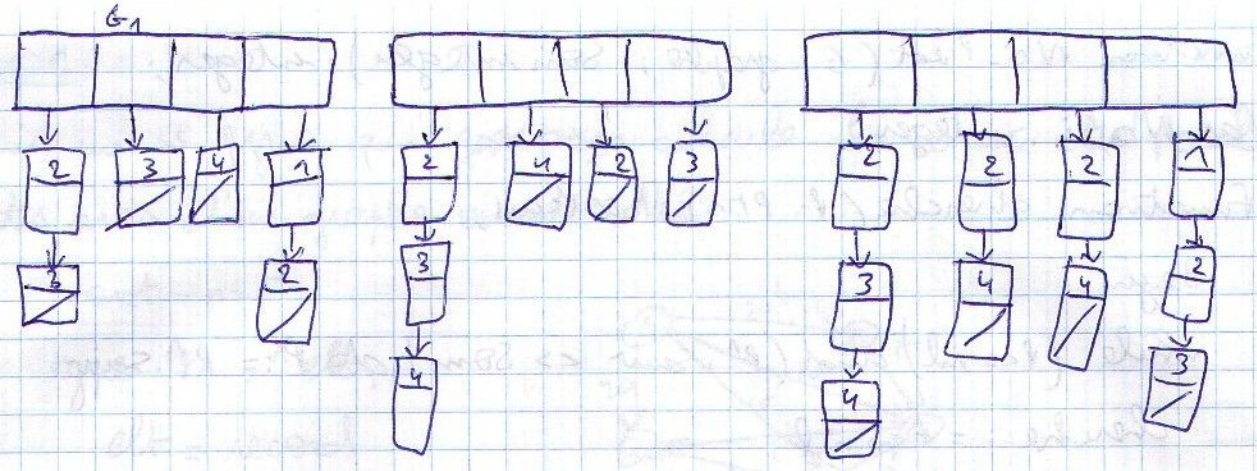
Exo4

Soient $G1 = \langle S_1, A_1 \rangle$ et $G_2 = \langle S_2, A_2 \rangle$

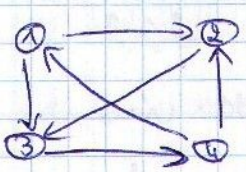
La ~~superposition~~ de G_1 et G_2 est le graphe.

$G = \langle S, A \rangle$ tq $S = S_1 = S_2$ et $A = A_1 \cup A_2$

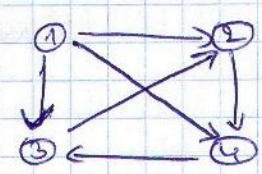
Ecrire un s/pgme qui realise la ~~superposition~~ superposition de deux graphes



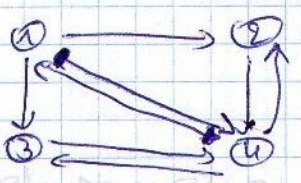
G_1 :



G_2 :



G_3 :



```
Procedure superposition (G1, G2 : graphe ; var G : graphe);
Var P, q : Ptr ; i : integer
Function cherche (P: Ptr ; som : integer) : boolean
  Begin
  While (P<>nil) and (P^.som <> som) then P := P^.suiv ;
  cherche := P<> nil;
  end;
  Begin
  for i := 0 to n do
P := G1[i] ; G[i] := nil;
while P<>nil do
  Begin
  new(q) ; q^.som := P^.som ; q^.suiv := G[i];
  G[i] := q; P := P^.suiv
  end;
  P := G2[i];
  while P<>nil do
  Begin
  if not (cherche (G[i],
  P^.som)) then
  Begin
  New(q); q^.som ; q^.som := G[i]
  G[i] := q
  end;
  P := P^.suiv;
  end;
end.
```