Nom	
Prénom	
Groupe	

Note	

Algorithmique - Info-SPE

partiel n^o 2

D.S. 313424.3 BW (4 mai 2010)

Feuilles de réponses

Réponses 1 **(Dénombrement et Graphes non orientés – *6 points*)**

1. Le nombre d'arêtes d'un graphe simple non orienté complet de n sommets est
   (a) pour un graphe n'admettant pas les arêtes réflexives de :
   Justifiez

   (b) pour un graphe admettant les arêtes réflexives de :
   Justifiez

2. Peut-on construire un graphe non orienté simple de (entourer la bonne réponse) :
   (a) 4 sommets et 7 arêtes ? OUI – NON
   (b) 5 sommets et 9 arêtes ? OUI – NON
   (c) 10 sommets et 46 arêtes ? OUI – NON

3. Règle entre n sommets et p arêtes permettant la construction d'un graphe simple :

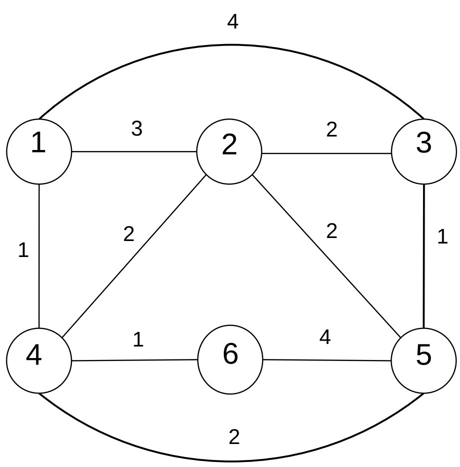
Réponses 2 **(ARM et autres... – *4 points*)**

1. Un arbre de recouvrement d'un graphe G non orienté est :

2. Nombre d'arbres de recouvrement :

   Justifiez

3. Proposer une solution graphique (Surligner les galeries que vous vous proposez de sécuriser).



4. Pourquoi tout graphe non orienté valué connexe admet au moins un arbre couvrant minimum ?

5. Un exemple d'application au problème des arpm pourrait être :

Réponses 3 **(Plus court chemin et parcours profondeur – *17 points*)**

1. Condition(s) pour que l'arc $s \rightarrow sa$ soit un arc retour (pas de phrase, la formule) :

2. **Spécification :** la procédure `pprof_rec(ps,cpt,op,os,p)` effectue le parcours profondeur depuis le sommet `ps`, numérote les sommets (en ordre préfixe de rencontre) dans `op` et (en ordre suffixe de rencontre) dans `os` à l'aide du compteur `cpt` et empile les sommets en suffixe dans la pile de pointeur (de type `t_listsom`) `p`.

algorithme procedure `pprof_rec`
  parametres locaux
    `t_listsom` `ps`
  parametres globaux
    `entier` `cpt`
    `t_vect_entiers` `op, os`
    `t_pile` `p`
  variables

debut

fin algorithme procedure `pprof_rec`

3. **Spécification :** la procédure `bellman_prof(g, src, op, os, p, pere, dist)` calcule l'ensemble des *plus courts* chemins (en ignorant les arcs responsables des circuits) depuis le sommet `src` dans le graphe `g` dont les sommets sont triés dans la pile `p`. Les vecteurs `op` et `os` donnent respectivement l'ordre préfixe et suffixe de rencontre (du parcours profondeur) des sommets. L'algorithme remplit le vecteur de pères (`pere`) et le vecteur de distances (`dist`) avec les chemins calculés.

algorithme procedure `bellman_prof`
parametres locaux
```
t_graphe_d       g
entier           src
t_vect_entiers   op, os
```
parametres globaux
```
t_pile           p
t_vect_entiers   pere
t_vect_reels     dist
```
variables

debut

fin algorithme procedure `bellman_prof`

4. Ordres préfixes (`op`) et ordres suffixes (`os`) dans le graphe G_1 :

	1	2	3	4	5	6	7	8
`op`								
`os`								

Pile remplie en suffixe pendant le parcours :

`p`								

5. Arcs ignorés (sous la forme $s \to sa$) :

6. Vecteurs de père et de distance obtenus par la fonction `bellman_prof` pour le graphe G_1 :

	1	2	3	4	5	6	7	8
`pere`								
`dist`								

7. Chemin obtenu par l'algorithme entre les sommets 1 et 2 :

8. Existe-t-il un meilleur chemin du sommet 1 vers le sommet 2 : `OUI` `NON`

Si oui, lequel :

Réponses 4 **(Graphe réduit – *3 points*)**

Spécifications :

La procédure `graphe_reduit (t_graphe_s` G, `t_vect_entiers` cfc, `entier` nb_cfc, `t_graphe_s` Gr`)` construit Gr graphe réduit de G à partir du vecteur des composantes fortement connexes de G, cfc, ainsi que leur nombre nb_cfc.

algorithme procedure `graphe_reduit`
parametres locaux
`t_graphe_s      G`
`t_vect_entiers  cfc`   */* le vecteur des composantes fortement connexes */*
`entier          nb_cfc`   */* le nombre de composantes */*
parametres globaux
`t_graphe_s      Gr`

variables

debut

fin algorithme procedure `graphe_reduit`