Algorithmique
Partiel n° 1 (P1)

Info-spé (s3)
Epita

D.S. 309973.74 BW (22 déc. 2014 - 10 :00)

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo (pas de C, Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !
- Durée : 2h00

Exercice 1 (Graphes Eulériens... – *7 points*)

Soit un graphe G=<S,A> orienté ou non. Un *parcours eulérien* du graphe G emprunte une fois, et une seule, chaque arc ou arête de A. En revanche, il peut passer plusieurs fois par un même sommet.

Nous nous intéresserons au *cycle eulérien* dans le cas d'un graphe non orienté.

Définition : Un graphe non orienté fini G sans sommet isolé possède un cycle eulérien si, et seulement si, G est connexe et si tous les sommets de G sont pairs[1]

1. En vous basant sur un parcours profondeur, donner un principe algorithmique permettant de déterminer si un graphe donné présente un cycle eulérien.

2. Ecrire l'algorithme abstrait correspondant. Vous remplirez, selon que vous l'estimez nécessaire ou non, les différents espaces laissés dans les deux algorithmes : La fonction principale *(ExisteCycleEulerien)* et la fonction récursive *(ParcCycleEuler)*.

   On considérera le nombre de sommets du graphe représenté par N.
   Votre algorithme doit permettre de débrancher avec une valeur de retour fausse dès qu'une impossibilité d'obtenir un cycle eulérien est détectée.

Types abstraits : Les graphes non orientés

SORTE Graphe

UTILISE Sommet, Entier, Booléen

OPERATIONS

graphe_vide	:	$\rightarrow$ Graphe
ajouter-le-sommet _ à _	:	Sommet $\times$ Graphe $\rightarrow$ Graphe
ajouter-l'arête <_,_> à _	:	Sommet $\times$ Sommet $\times$ Graphe $\rightarrow$ Graphe
_ est-un-sommet-de _	:	Sommet $\times$ Graphe $\rightarrow$ Booléen
<_,_> est-une-arête-de _	:	Sommet $\times$ Sommet $\times$ Graphe $\rightarrow$ Booléen
d°	:	Sommet $\times$ Graphe $\rightarrow$ Entier
_ ème-succ-de _ dans _	:	Entier $\times$ Sommet $\times$ Graphe $\rightarrow$ Sommet
retirer-le-sommet _ de _	:	Sommet $\times$ Graphe $\rightarrow$ Graphe
retirer-l'arête <_,_> de _	:	Sommet $\times$ Sommet $\rightarrow$ Graphe
nb-sommets	:	Graphe $\rightarrow$ Entier
nb-arêtes	:	Graphe $\rightarrow$ Entier

1. Les sommets dont le degré est impair sont dits "impairs" et ceux de degré pair sont dits "pairs".

Exercice 2 (Dans les profondeurs de la forêt couvrante – *4 points*)

On se propose d'effectuer un parcours profondeur sur un graphe orienté. L'objectif est de construire la forêt couvrante du parcours, d'ajouter les différents types d'arcs rencontrés et de calculer l'ordre suffixe de rencontre de chaque sommet.

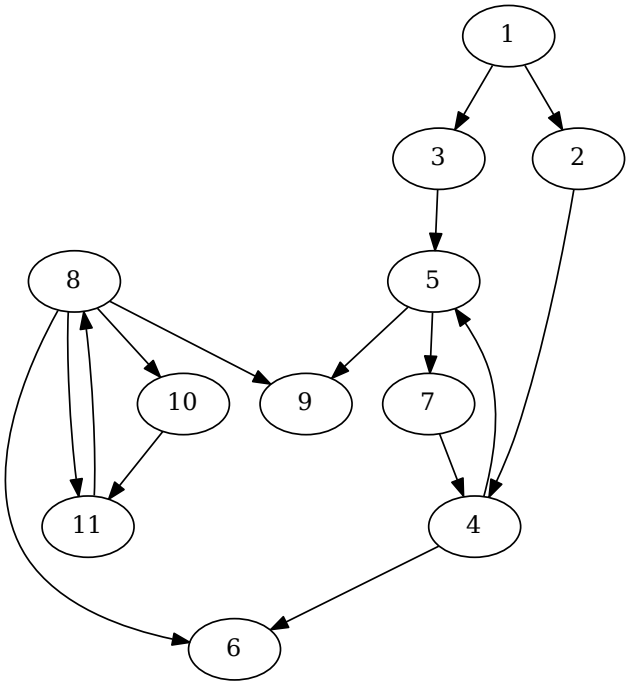

FIGURE 1 – Un graphe orienté

1. Construire la forêt couvrante du graphe de la figure 1 correspondant à un parcours profondeur depuis le sommet numéro 1. On rencontrera les sommets dans l'ordre croissant. Ajouter à la forêt obtenue les différents types d'arcs rencontrés lors du parcours.

2. Remplir les vecteurs de pères et d'ordres suffixes pour chacun des sommets, correspondants au parcours de la question précédente. Les racines de la forêt seront indiquées par la valeur -1 dans le vecteur de pères.

Exercice 3 (Chemin – *9 points*)

1. Quel parcours utiliser pour trouver un chemin (ou une chaîne) le plus court (en nombre de liaisons) entre deux sommets donnés dans un graphe ?

2. On cherche un chemin (le plus court possible) entre deux sommets, avec l'implémentation dynamique. Si un chemin a été trouvé, il devra être affiché.
Compléter les deux algorithmes donnés : le premier effectue le parcours choisi et le deuxième est l'algorithme d'appel qui lance le parcours et affiche l'éventuel chemin trouvé.

Annexes

Implémentation dynamique des graphes

Les graphes utilisés ici sont non valués, les coûts ont donc été enlevés de la représentation.

```
types
    t_listsom = ↑ s_som      /* liste des sommets */

    t_listadj = ↑ s_ladj     /* liste d'adjacence */

    s_som     = enregistrement      /* un sommet */
       entier      som
       t_listadj   succ
       t_listadj   pred
       t_listsom   suiv
    fin enregistrement s_som

    s_ladj      = enregistrement     /* un successeur (ou prédécesseur) */
       t_listsom   vsom
       entier      nb
       t_listadj   suiv
    fin enregistrement s_ladj

    t_graph_dyn  = enregistrement     /* le graphe */
       entier     ordre
       booleen    orient
       t_listsom  lsom
    fin enregistrement t_graph_dyn
```

Les vecteurs :

```
types                                  /* Max > ordre (G) */
   t_vect_entiers  = Max entier
   t_vect_booleens  = Max booleen
```

Routines autorisées

Toutes les opérations sur les files et les piles peuvent être utilisées à condition de préciser le type des éléments.

Files

- `file_vide():t_file`
- `est_vide(t_file` f`):booleen`
- `enfiler(t_eltFile` e`, t_file` f`):t_file`
- `defiler(t_file` f`):t_eltFile`
- `vide_file(t_file` f`)`

Piles

- `pile_vide():t_pile`
- `est_vide(t_pile` p`):booleen`
- `empiler(t_eltPile` elt`, t_pile` p`):t_pile`
- `depiler(t_pile` p`):t_pile`
- `sommet(t_pile` p`):t_eltPile`

Autres

- `recherche (entier` s`, t_graph_dyn` G`) : t_listsom`
  retourne le pointeur sur le sommet s dans G.