Algorithmique
Correction Partiel n° 1 (P1)

Info-spé (s3) – Epita

22 déc. 2014 - 10 :00

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

1. Le principe est le suivant :

   Puisqu'il faut un graphe connexe dont tous les sommets soient pairs, nous allons effectuer un parcours profondeur du graphe. La détection de la connexité se fera dans l'algorithme principal*ExisteCycleEulerien*. Si lors du parcours, la fonction *Parcycleurer* doit être rappelée (nombre de sommets rencontrés différents du nombre total de sommets), c'est que le graphe est non connexe. Dans ce cas, une sortie *fausse* s'imposera. La détection des degrés pairs se fera, quant à elle dans l'algorithme de parcours *Parcycleuler*, lors de la détermination du degré du sommet actuel. Si celui-ci (le d$^\circ$) est impair, une sortie *fausse* s'imposera là aussi.

   Dans tous les autres cas, l'algorithme renverra *Vrai* signifiant par là l'existence d'un *cycle eulérien* dans le graphe traité.

2. Les algorithmes :
   L'algorithme principal *ExisteCycleEulerien* :

```
algorithme fonction ExisteCycleEulerien : booleen
parametres locaux
    Graphe G
variables
    t_vect_booleens  M
    entier    nb             /* nb compteur de sommets */
debut
    pour nb ← 1 jusqu'a N faire
       M[nb] ← faux
    fin pour
    nb ← 0                                          /* appel arbitraire sur sommet 1 */
    si non(Parcycleurer(1,G,M,nb)) ou nb<>N alors   /*  sommet impair rencontré ou */
       retourne faux                                 /* sommets pas tous parcourus */
    fin si
    retourne vrai
fin algorithme fonction ExisteCycleEulerien
```

   L'algorithme de parcours *ParcCycleEuler* :

```
algorithme fonction ParcCycleEuler : booleen
parametres locaux
    entier  s
    Graphe G
parametres globaux
    t_vect_booleens  M
    entier  nb
variables
    entier  i,t,deg
debut
    M[s] ← vrai
    nb ← nb + 1
    deg ← d°(s,G)    /* d°: degré-de s dans G */
    si deg mod 2=0 alors
       pour i ← 1 jusqu'a deg faire
          t ← i ème-succ-de s dans G
          si non(M[t]) alors
             si non(ParcCycleEuler(t,G,M,nb)) alors /* Un sommet impair rencontré */
                retourne faux
             fin si
          fin si
       fin pour
       retourne vrai
    sinon
       retourne faux
    fin si
fin algorithme fonction ParcCycleEuler
```

Solution 3 **(Chemin – *9 points*)**

1. Le parcours largeur donnera un des chemins les plus courts.

2. **Spécifications :**
   La fonction `path (t_listsom` *ps*`, entier` *dst*`, t_vect_entiers` *p*`)` effectue un parcours à partir du sommet pointé par *ps*. Elle retourne un booléen indiquant si *dst* atteint. Le vecteur *p* contient le père de chaque sommet pour la partie du graphe visitée.

```
algorithme fonction path : booleen
    parametres locaux
        t_listsom     ps
        entier         dst
    parametres globaux
        t_vect_entiers     p

    variables
        t_file      f
        t_listadj  pa
debut
    p[ps↑.som] ← -1
    f ← enfiler (ps, file_vide ())
    faire
        ps ← defiler (f)
        pa ← ps↑.succ
        tant que pa <> NUL faire
            si p[pa↑.vsom↑.som] = 0 alors
                p[pa↑.vsom↑.som] ← ps↑.som
                si pa↑.vsom↑.som = dst alors
                    vide_file (f)
                    retourne vrai
                fin si
                f ← enfiler (pa↑.vsom, f)
            fin si
            pa ← pa↑.suiv
        fin tant que
    tant que non est_vide (f)
    retourne faux
fin algorithme fonction path
```

Spécifications :

La fonction `find_path (t_graph_dyn` G`, entier` src, dst`)` recherche un chemin à partir du sommet src jusqu'au sommet dst dans le graphe orienté G. Si un chemin a été trouvé elle l'affiche et retourne *vrai*, *faux* sinon.

```
algorithme fonction find_path : booleen
    parametres locaux
        t_graph_dyn   G
        entier        src, dst

    variables
        t_vect_entiers     parent
        entier          i
        t_pile           p
debut
    pour i ← 1 jusqu'a G.ordre faire
        parent[i] ← 0
    fin pour

    si path (recherche (src, G), dst, parent) alors
        p ← pile_vide ()
        tant que dst <> -1 faire
            p ← empiler (dst, p)
            dst ← parent[dst]
        fin tant que
        ecrire ("Path: \n")
        tant que non est_vide (p) faire
            ecrire (", ", sommet (p))
            p ← depiler (p)
        fin tant que
        retourne vrai
    sinon
        retourne faux
    fin si
fin algorithme fonction find_path
```