RO03

TD 9

Recherche en profondeur d'abord dans un graphe

L'algorithme ci-dessous permet de déterminer avec une faible complexité les descendants d'un sommet i_0. Initialement n(i) est le nombre de successeurs de i. On initialise également p(i)=0 pour i différent de i_0 et $p(i_0)=i_0$.

Au cours de l'exploration, on associe à chaque sommet i, un nombre p(i) $\in$ [0,n] et on dit que **le sommet i est atteint** quand p(i) est strictement positif (p(i) est alors le sommet ayant permis d'atteindre le sommet i) et que le sommet i est **non atteint** si p(i) est nul.

Un sommet atteint se trouvera, par convention, dans l'un des deux états suivants : **ouvert**, quand n(i) est strictement positif, **fermé** quand n(i) est nul.

Algorithme :

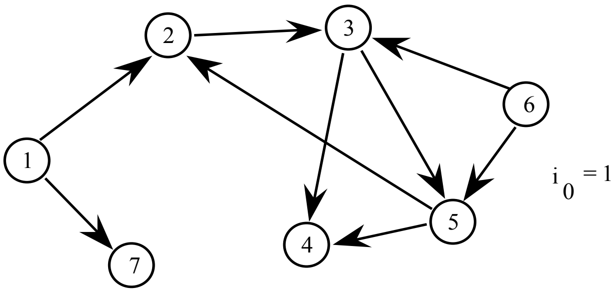
```
Début
    Pour i =1 à n faire {les sommets sont numérotés de 1 à n.}
        Début
            p(i) := 0;
            n(i) := d+(i);
        Fin
    p(i0) := i0;
    i := i0;
    Tant que (n(i0) + ABS(i-i0)) > 0 faire /*ABS(x) est la valeur absolue de x */
        Début
            Si n(i) différent de 0 alors {le sommet  i est ouvert}
                Début
                    Sélectionner parmi les successeurs de i le sommet j venant en position n(i)
                    n(i) := n(i)-1;
                    Si p(j)=0 alors {le sommet j est non atteint}
                        Début
                            p(j) := i; {le sommet j devient atteint}
                            i := j;
                        Fin
                Fin
            sinon {le sommet i est fermé car on a examiné tous ses successeurs}
                i := p(i)
        Fin
Fin
```

Première partie: étude de l'algorithme

Question 1 : On admettra provisoirement que la restriction de la fonction p à X-i_0-{i/p(i)=0} définit une arborescence partielle de racine i_0 à tout instant de l'algorithme. Si p(i) est différent de 0, p(i) est le père de i. Rappelons qu'une telle fonction père définit une arborescence si le graphe associé à sa restriction est connexe.

On demande de faire tourner l'algorithme sur le graphe ci-dessous en traçant autant d'arborescences obtenues en cours d'algorithme que d'exécutions de la boucle tant que avec les conventions suivantes :

- initialement, l'arborescence est réduite à sa racine i_0 = 1;
- quand on sélectionne un sommet j avec p(j)=0, on ajoute l'arc "plein" (i,j) à l'arborescence;
- quand on ferme le sommet i, on met en pointillé ou en rouge l'arc (p(i),i).



Question 2 : Expliquer intuitivement ce que l'on fait dans la boucle Tant Que. Interpréter le test : $(n(i_0) + ABS(i-i_0)) > 0$.

Question 3

A quoi correspondent dans la boucle Tant que les cas n(i) différent de 0 et n(i) égal 0 ?

Question 4

Combien de fois exécute-t-on la boucle Tant que pour l'exemple ci-dessus ? Combien de fois au pire dans le cas général ? Pour le démontrer on distinguera les cas n(i) différent de 0 et n(i) = 0.

Deuxième partie : preuve de l'algorithme.

Question 1: Montrer qu'en cours d'algorithme on construit une arborescence de racine i_0.

Question 2: Montrer que la restriction de cette arborescence aux arcs pleins est une arborescence, réduite à un chemin, dont i est le sommet pendant (cette arborescence est dite arborescence courante) et i_0 le sommet initial.

Question 3: Montrer que tout sommet atteint est descendant de i_0.

Question 4: A quelle condition un sommet est-il retiré de l'arborescence courante ?

Question 5: Que devient l'arborescence courante à la fin de l'algorithme ?

Question 6: Montrer en utilisant le fait que les n(i) décroissent en restant positifs ou nuls, et que l'algorithme ne se bloque jamais en distinguant les cas $i=i_0$ et $i\neq i_0$. Déduire que l'algorithme se termine.

Question 7: Montrer que tout sommet descendant de i_0 est atteint à la fin de l'algorithme.

Question 8 : Montrer que l'algorithme détermine les descendants de i_0.

Troisième partie : complexité de l'algorithme

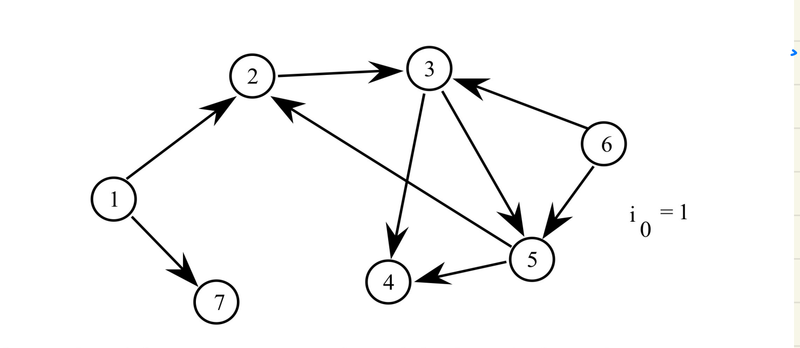
Question 1: Quel codage du graphe proposez-vous?

Question 2: Evaluer la complexité globale de l'algorithme en justifiant votre réponse.

Quatrième partie: extensions.

On a donc démontré que cet algorithme permet de déterminer les descendants d'un sommet i_0. Quelle méthode proposez-vous pour déterminer les composantes connexes d'un graphe? Ecrire un tel algorithme utilisant comme sous-programme l'algorithme précédent. Evaluer sa complexité.

Q1 On fera tourner l'algorithme sur le graphe suivant:

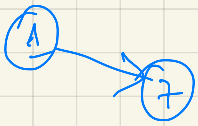


Initialement: $p_i = 0 \ \forall i \neq 1$, $p_1 = 1$; $l_0 = 1$

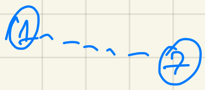
$n_1 = 2$, $n_2 = 1$, $n_3 = 2$, $n_4 = 0$, $n_5 = 2$, $n_6 = 2$, $n_7 = 0$, $i = 1$

fin de première itération: lors de la première itération on choisit $j = 7$ car c'est le $n(1)^{ème}$ successeur, donc le $2^{ème}$ successeur.

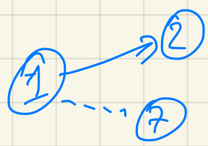
$i = 7$, $n(1) = 1$. $p(7) = 1$



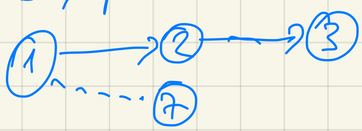
fin de deuxième itération: $i = 1$,



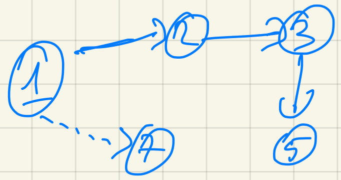
fin de troisième itération: $i = 2$, $p(2) = 1$, $n(1) = 0$



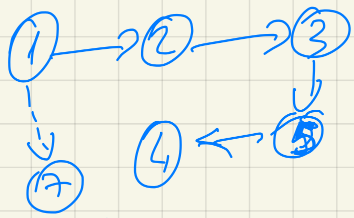
fin de $4^{ème}$ itération: $i = 3$, $p_3 = 2$, $n(2) = 0$



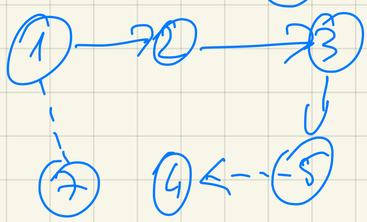
fin de $5^{ème}$ itération: $i = 5$, $p(5) = 3$, $n(3) = 1$



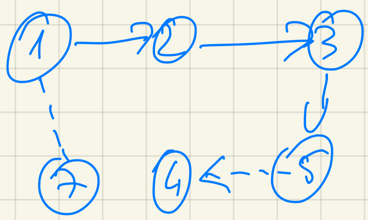
fin de $6^{ème}$ itération: $i = 4$, $p(4) = 5$, $n(5) = 1$



fin de $7^{ème}$ itération: $i = 5$:

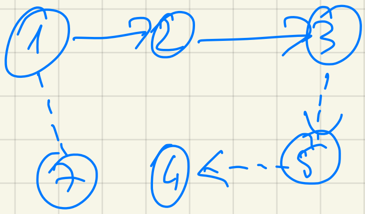


fin de $8^{ème}$ itération: $i = 5$, $n(5) = 0$

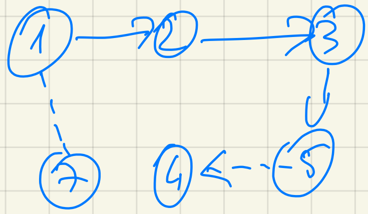


lors de cet itération on examine $j = 2$ mais le sommet 2 est déjà atteint.

fin de la $9^{ème}$ itération : $i=3$

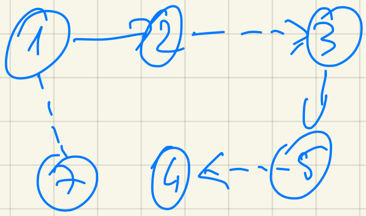


fin de la $10^{ème}$ itération : $i=3$, $n(3)=0$

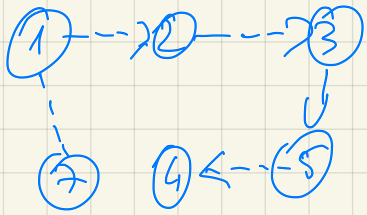


$j=4$: on examine le (successeur) sommet 4 mais ce sommet est déjà atteint.

fin de la $11^{ème}$ itération : $i=2$



fin de la $12^{ème}$ itération : $i=1$



Fin de la boucle Tant Que.

Question 2. Le test de l'arrêt de la boucle Tant Que correspond au fait que l'algorithme se termine quand on se trouve à la racine et l'ensemble de ses successeurs ont été atteint.

Question 3. $n(i)=0$ correspond au cas où le sommet i est fermé et il faudra revenir au sommet père. $n(i)>0$ correspond au cas où il reste encore des successeurs de i à explorer/atteindre.

Question 4. On exécute la boucle T.Q. $d^+(i)$ fois pour chaque sommet atteint ; ce qui correspond à l'exploration de ses successeurs. + 1 fois pour chaque sommet atteint, sauf i_0.

Donc : $d^+(1)+d^+(2)+d^+(3)+d^+(4)+d^+(5)+d^+(7)+5 = 7+5 = 12$.

Partie 2

Deuxième partie : preuve de l'algorithme.

Question 1: Montrer qu'en cours d'algorithme on construit une arborescence de racine i_0.

Question 2: Montrer que la restriction de cette arborescence aux arcs pleins est une arborescence, réduite à un chemin, dont i est le sommet pendant (cette arborescence est dite arborescence courante) et i_0 le sommet initial.

Question 3: Montrer que tout sommet atteint est descendant de i_0.

Question 4: A quelle condition un sommet est-il retiré de l'arborescence courante ?

Question 5: Que devient l'arborescence courante à la fin de l'algorithme ?

Question 6: Montrer en utilisant le fait que les n(i) décroissent en restant positifs ou nuls, que l'algorithme ne se bloque jamais en distinguant les cas $i=i_0$ et $i \neq i_0$. Déduire que l'algorithme se termine.

Question 7: Montrer que tout sommet descendant de i_0 est atteint à la fin de l'algorithme.

Question 8 : Montrer que l'algorithme détermine les descendants de i_0.

Question 1 : On démontre cette propriété (c.à.d. on construit une arborescence de racine i_0 défini par la fonction père (p) au cours de l'algorithme) par recurrence sur le nombre d'iterations de la boucle Tant Que.
Clairement à $k=0$ la propriété est vraie car l'arborescence est reduite à la racine i_0.
Supposons la propriété vraie à la fin de la k-ème itération et prouvons que cela reste vraie à la fin de la $k+1$ ème itération. Lors de l'iteration $k+1$ Soit la fonction p() reste inchangé, et donc la propriété est toujours vraie, soit p() est modifié et on mettre $p(j)=i$ où j est un successeur de i non atteint. Dans ce cas on rajoutera l'arc (i,j) à l'arborescence courante ce qui donne une nouvelle arborescence.
En effet la fonction père definit une arborescence de racine i_0 si le graphe défini par $\{(p(i),i) \mid i \neq i_0\}$ est connexe.

Question 2 : On raisonne de la même manière que pour Question 1. La seule différence est que on rajoute ou retire un arc à l'extremité d'un chemin ce qui donnerait toujours un chemin.

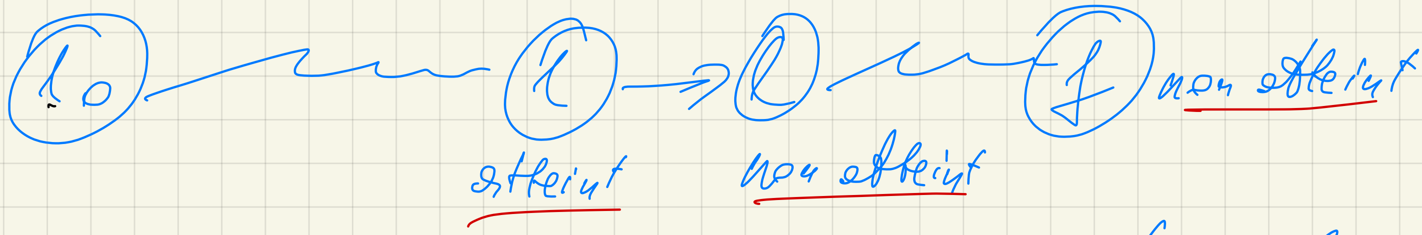
Question 3 Tout sommet atteint est dans l'arborescence, par conséquent il $\exists$ un chemin dans l'arborescence depuis la racine, donc tout sommet atteint est descendant de i_0.

Question 4. Un sommet est retiré de l'arborescence courante si tous ses successeurs ont été atteint et fermé.

Question 5 A la fin de l'algorithme l'arborescence courante est réduite à la racine i_0.

Question 6. Initialement $n(i) = d^+(i)$. A chaque itération de la boucle Tant que soit on diminue par 1 le $n(i)$ du sommet courant i quand $n(i) > 0$, soit on remonte dans l'arborescence au sommet $p(i)$ quand $n(i) = 0$. On fait ceci pour tous les sommets ~~sauf~~ i_0 quand on sort de la boucle Tant que. On en déduit que l'algorithme se termine après un nombre fini d'itérations

Question 7 On demontrera la propriété par l'absurd : On suppose qu'à la fin de l'algorithme il existe le sommet j, descendant de i_0, qui n'a pas été atteint. Puisque j est un descendent de i_0 il existe un chemin μ de i_0 à j. Parcourons les sommets de μ en partant de i_0 et notons avec i le dernier sommet atteint de μ et l le successeur de i dans μ. (voir figure).



Or à la fin de l'algorithme tous les sommets atteints seraient fermés, y compris le sommet i. Par conséquent tous les successeurs de i auraient dû être atteints, et ainsi k aurait dû être atteint aussi, ce qui mène à une contradiction.

Question 8 $Q_3 + Q_2$ implique que tous les sommets atteints sont des descendants et vice-versa. Donc l'algorithme détermine bien tous les descendants de i_0.

Troisième partie : complexité de l'algorithme

Question 1: Quel codage du graphe proposez-vous?

Question 2: Evaluer la complexité globale de l'algorithme en justifiant votre réponse.

Question 1 Puisque on a besoin d'accéder aux successeurs il semble plus judicieux d'utiliser la file des successeurs.

Question 2 On calcule la complexité directement sur l'algorithme:

```
Début
   Pour i =1 à n faire {les sommets sont numérotés de 1 à n.}
      Début
         p(i) := 0;
         n(i) := d+(i);
      Fin
   p(i0) := i0;
   i := i0;
   Tant que (n(i0) + ABS(i-i0)) > 0 faire /*ABS(x) est la valeur absolue de x */
      Début
         Si n(i) différent de 0 alors {le sommet  i est ouvert}
            Début
               Sélectionner parmi les successeurs de i le sommet j venant en position n(i)
               n(i) := n(i)-1;
               Si p(j)=0 alors {le sommet j est non atteint}
                  Début
                     p(j) := i; {le sommet j devient atteint}
                     i := j;
                  Fin
            Fin
         sinon {le sommet i est fermé car on a examiné tous ses successeurs}
            i := p(i)
      Fin
Fin
```

$O(n)$ (boucle Pour)

$O(1)$ ($p(i_0) := i_0$)

$O(1)$ (bloc Si)

$(O(1))$ ($i := p(i)$)

chaque sommet atteint sera examiné exactement $(d^+(i)+1)$ fois, sauf i_0; ⇒ complexité de la boucle Tant Que ~ $O\left(\sum_{\substack{i \text{ atteint} \\ i \neq i_0}} (d^+(i)+1) + d^+(i_0)\right)$

La phase d'initialisation coûte $O(n)$

La complexité de l'exécution de la boucle T.Q dependra du nombre d'itérations de la boucle.

Cela donnera $\sum_{\substack{i \text{ atteints} \\ i \neq i_0}} (d^+(i)+1) + d^+(i_0) \leq \sum_{i=1}^{n} d^+_i + n = m+n$

et chaque itération coûterait $O(1)$.

D'après ci-dessus on déduit que la complexité de la boucle T.Q est $\sim O\left(\sum_{\substack{i \text{ atteint} \\ i \neq i_0}} (d^+(i)+1) + d^+(i_0)\right) \sim O\left(\sum_i (d^+_i+1)\right) \sim O(m+n) \sim O(m)$

Complexité globale est $O(m)+O(n) \sim O(m)$

Quatrième partie: extensions.

On a donc démontré que cet algorithme permet de déterminer les descendants d'un sommet i_0. Quelle méthode proposez-vous pour déterminer les composantes connexes d'un graphe? Ecrire un tel algorithme utilisant comme sous-programme l'algorithme précédent. Evaluer sa complexité.

Pour calculer les composantes connexes il suffira de dedoubler les arcs, c.à.d. pour tout arc (i,j) existant on va ajouter l'arc (j,i). Ainsi pour toute chaîne reliant i_0 à j dans le graphe initial il existe un chemin de i_0 à j dans le nouveau graphe G' et vice-versa.

Algorithme de calcul des composantes connexes.

Ecrire Graphe G' à partir de G.
Pour $i=1$ à n faire :
  Connexe $[i]=0$;
Pour $i_0=1$ à n faire :
  Si Connexe $[i_0]=0$ faire:
  début
    appliquer l'algorithme precedent avec racine i_0 dans G';
    pour tous les descendants i de i_0 faire Connexe $[i]=i_0$;
  fin.

Complexité globale $O(m)$ car pour chaque composante connexe K on appliquera l'algorithme séparement et cela coûtera $O(m_K)$ où m_K est le nombre d'arcs de la composante. Donc, Complexité globale $= O\left(\sum_{K=1}^{P} m_K\right) \simeq O(m)$.