

Contrôle continu du 7 octobre 2019

Durée : 0h45 Pages : 5

Nom :

CORRIGÉ

- * Le barème est indicatif. Les documents sont interdits.
- * Les réponses doivent **obligatoirement** être fournies dans les cadres prévus. Elles doivent être **justifiées, précises et concises**.
- * Les algorithmes du cours, s'ils ne sont pas modifiés, peuvent être utilisés en les appelant juste par leur nom, et en récupérant leur sortie dans la structure de données qui vous convient. S'ils subissent la moindre modification, **ils doivent être réécrits**.
- * Les algorithmes doivent être écrits en **pseudo-code**.

Exercice 1. Soit $n \geq 2$ un entier. On considère un graphe orienté G_n défini comme suit :

- Ses sommets sont les entiers $2, 3, \dots, n$.
- Il existe un arc du sommet a vers le sommet b si et seulement si
 - a est un diviseur de b , et
 - $a < b$, et
 - il n'existe pas de sommet c tel que l'on ait (1) $a < c < b$, et (2) a soit un diviseur de c et (3) c soit un diviseur de b . *Exemple.* 7 est un diviseur de 42, mais il n'y aura pas d'arc de 7 à 42 puisque 7 divise 21 qui divise 42.

Nous supposons que G_n , pour un n fixé, est représenté sous la forme d'une matrice d'adjacence M . On souhaite écrire un algorithme *DiviseursPremiers* qui :

- prenne en entrée M et un sommet s
- et qui retourne les diviseurs premiers de s (Rappel : il s'agit des entiers p dont les seuls diviseurs sont p lui-même et 1, et qui divisent s).

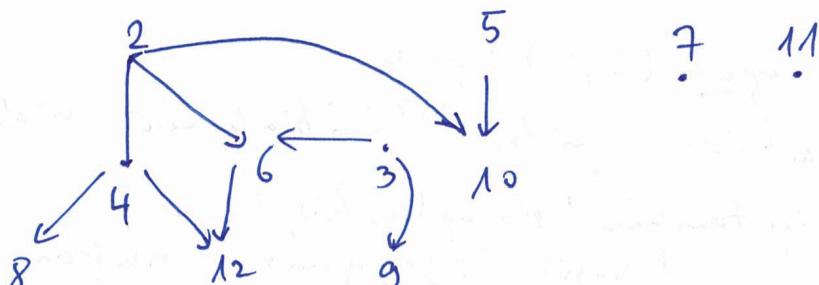
Cet algorithme doit être aussi efficace que possible. Pour cela il doit avoir une complexité proportionnelle au nombre de ses diviseurs : moins il a de diviseurs, moins la complexité est grande. Nous noterons donc d le nombre de diviseurs de s . Une fois cet algorithme écrit, on modifie le graphe et on calcule les composantes fortement connexes du graphe ainsi obtenu.

La Partie 2 peut donc être réalisée même si vous n'avez pas écrit tous les algorithmes de la partie 1.

Partie 1.

Questions :

1. Dessiner le graphe G_{12} (obtenu pour $n = 12$).



2. La première variante de l'algorithme, que nous appellerons *DiviseursPremiersA*, utilise deux étapes :
- L'étape 1 calcule le graphe transposé G_n^t de G_n , qui est le graphe obtenu de G_n en inversant tous les arcs. A la différence de G_n , le graphe G_n^t n'est pas représenté sous forme de matrice, mais de listes de successeurs. Nous appelons L la représentation par liste de successeurs de G_n^t .
 - L'étape 2 utilise le graphe transposé G_n^t (uniquement) pour trouver les diviseurs premiers de s . Ecrire deux procédures *Etape1(M)* (qui retourne L) et *Etape2(L, s)* (qui retourne les diviseurs premiers de s). Puis écrire *DiviseursPremiersA(M, s)*.

Etape 1 (M) : liste d'adjacence ;
 // comme en cours, les cellules de la liste
 // sont d'un type cellule avec champs val et suiv.
 var L : tableau [1..n] de pointeurs sur cellule;
 // supposé initialisé à NULL pour tout L[i]
 pour i de 1 à n faire
 pour j de 1 à n faire
 si M[i,j] = 1 alors
 P ← L[j].suiv;
 Q ← cellule-vide;
 L[j] ← Q;
 Q.val ← i;
 Q.suiv ← P
 fini
 fini
 fini

Etape 2 les diviseurs premiers de s sont
 les sommets, trouvés lors d'un parcours
 à partir de s , qui n'ont pas de
 successeurs dans G_n^t .

Etape2 (L, S) : pile;
 var Div : pile // initialement vide
 pour i de 1 à n faire
 visite[i] ← faux fini
 retourner (RemplirDiv(L, S));

RemplirDiv(L, s')

$P \leftarrow L(s')$; visite[s] \leftarrow non

Si $P = \text{NULL}$ alors

Empiler(s' , Div);

sinon

tant que $P \neq \text{NULL}$ faire

si non visite[P.val] alors

RemplirDiv(L, P.val)

fin si

fin si; $P \leftarrow P.\text{succ}$

fin si; fin tant que

3. Calculer la complexité de $\text{DiviseursPremiersA}$, en ordre de grandeur. La complexité peut être une fonction du nombre de sommets n , du nombre d'arêtes m et du nombre de diviseurs d de s .

$\Theta(n^2)$ pour Etape 1 (avec initiation en début de liste)

$\Theta(d^2)$ pour Etape 2 (noter que tout n'a rien qu'un

diviseur premier mais parvenir un
long chemin: $2^{10} \rightarrow 2^9 \rightarrow 2^8 \rightarrow \dots \rightarrow 2$)

4. La deuxième variante de l'algorithme, que nous appellerons $\text{DiviseursPremiersB}(M, s)$ calcule directement à partir de la matrice M les diviseurs premiers, sans calculer le transposé de G_n . Ecrire cet algorithme et calculer sa complexité.

Même algo que dans Etape 2, mais les succèssifs dans G_n^t = les prédecesseurs dans G_n .

RemplirDiv2(M, s)

visite[s] \leftarrow non; cpt $\leftarrow 0$;

pour j de 1 à n faire

si $M[j, s] = 1$ alors

cpt \leftarrow cpt + 1;

si non visite[j] alors

RemplirDiv2(M, j)

fin si

fin si

si cpt = 0 alors Empiler(s', Div)

5. Analyser les complexités des deux algorithmes écrits et indiquer si elles dépendent ou non du nombre de diviseurs de s

L'algorithme B parcourt tous les diviseurs, et pour trouver les prédiviseurs de chacun fait $O(n)$ opérations $\rightarrow O(nd)$.
 L'algorithme A ne dépend donc pas de d , mais l'algorithme B si.

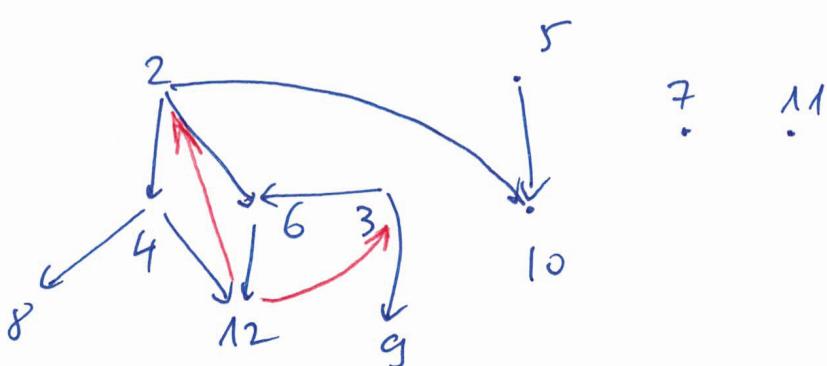
6. Modifier l'un des algorithmes *DiviseursPremiersA* et *DiviseursPremiersB* (au choix) pour que, dès qu'un diviseur premier x de s a été trouvé, l'algorithme ajoute au graphe G_n l'arc qui va de s à x . Pour cela, la matrice M sera modifiée en conséquence. Ce nouvel algorithme sera appelé *ApresModifs*(M, s) et retournera la matrice du nouveau graphe, lequel sera appelé $H_{n,s}$ (car il dépend à la fois de n et de s).

Dans RecuplierDiv2(M, s') il suffit de remplacer
 Empiler(s', div)
 par
 $M[s, s'] = 1$.
 (Pareil dans RecuplierDiv(M, s')).

Partie 2.

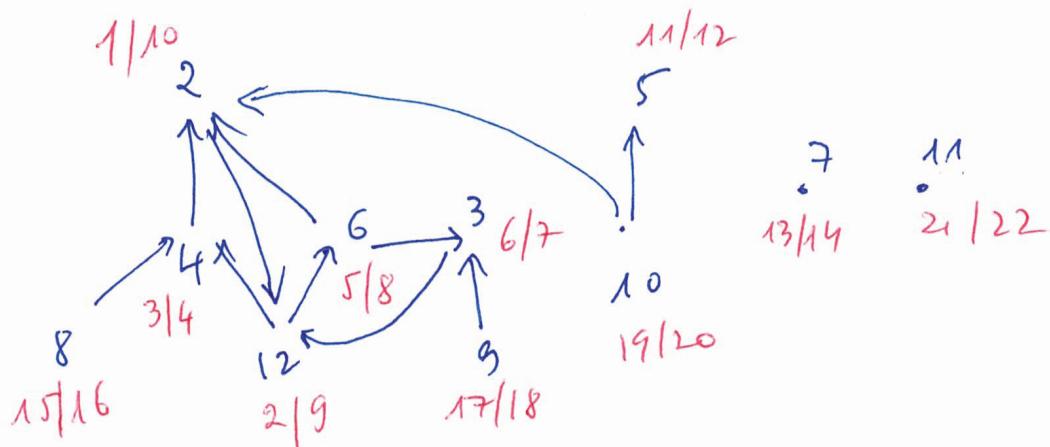
Questions :

7. Dessiner le graphe $H_{12,12}$ obtenu pour $n = 12$ à l'aide de l'algorithme *ApresModifs* utilisant $s = 12$. Si vous n'avez pas répondu aux points précédents, rappel : Le graphe $H_{12,12}$ est le graphe obtenu à partir du graphe G_{12} dessiné au point 1. en ajoutant des arcs depuis 12 vers chacun de ses diviseurs premiers.



8. Calculer, en utilisant un des algorithmes qui vous ont été présentés en distanciel, les composantes fortement connexes de $H_{12,12}$. Donner tous les détails de l'exécution.

Graphe transposé :

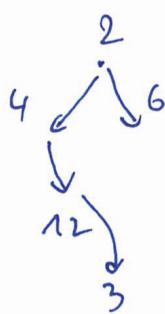


On numérote le graphe transposé utilisant ProfNum (voir cours). Le résultat est ci-dessous. Ordre : d'abord à partir de 2, puis de 5, 7, 8, 9, 10, 11.

(Remarque : l'ordre des sommets de démarage est arbitraire)

Ensuite on crée les arbres census par arête décroissant des dates de fin des sommets - racine, et en regardant dans G_u .

11 10 9 8 7 5
• • • • • •
{ { { { { {
composantes fortement connexes triunales



la seule composante non-triviale est celle des descendants de 12.