Examen Final - Juin 2021 – RO03 et AI25

Documents autorisés : Polycopié, notes de cours et de TD
Tout appareil électronique est interdit.

Rédigez vos réponses sur deux copies différentes : une pour chaque partie.

PARTIE I.

Exercice 1 (8 points):

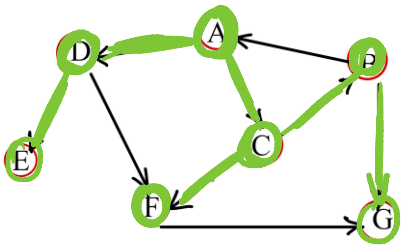
Nous considérons des graphes orientés G=(V, E) ou |V|=n et |E| = m. Une file F de type FIFO est une suite finie d'objets gérée selon le principe « premier arrivé, premier examiné ». CréerFile(F) consiste à créer une file F vide, donc avec 0 objet. Dans Enfiler(F,v) l'on rajoute l'objet v à la fin de la file F. Defiler(F) consiste à retirer de F l'objet du début de la file. Toutes ces opérations s'exécutent en O(1).

On étudiera un algorithme de « parcours en largeur » utilisant une structure File de type FIFO. Les fonctions **CreerFile(F), Enfiler(F,v)** et **Défiler(F)** permettent respectivement d'initialiser la File F, d'ajouter un élément v à la fin de la file F et d'enlever l'élément du début de la file F. La fonction **Père(j)**=i permet d'affecter le sommet i comme père du sommet j (i étant le sommet permettant d'atteindre j), auquel correspondra une arborescence de racine s (par convention, s est son propre père). Marquer un sommet j consiste à lui attribuer une marque. Initialement les sommets ne sont pas marqués.

Algorithme **ParcoursLargeur**(Graphe G, Sommet s):

```
Début
Lire un  Graphe G=(V,E) //codage alpha/beta
F=CreerFile();
Enfiler(F,s);
//initialement les sommets ne sont pas marqués//
Marquer(s);
Père[s]=s;   Dist[s] = 0 ;
    tant que la file F est non vide
        i = Defiler(F);
        pour tout successeur j (suivant ordre alphabétique) de i dans G
            si j non marqué
                Enfiler(F,j);
                marquer(j);
                Père[j] =i;
                ↳ Dist[j] = Dist[i]+1 ;
        Fin pour;
    Fin tant que;
Fin.
```

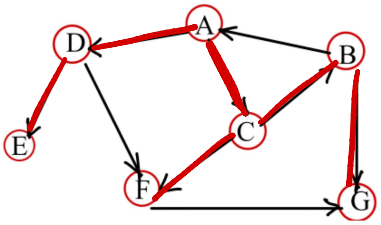
1) Faites tourner l'algorithme ParcoursLargeur (G1, A). Donner la suite des files construites par l'algorithme et les arborescences obtenues.



Graphe G1

2) Donner la complexité de l'algorithme. Justifier.
3) Modifier l'algorithme de façon à ce qu'il calcule le plus court chemin **en nombre d'arcs** de s à tout autre sommet en ajoutant le tableau **Dist[v]**, (Dist[v] donne la distance la plus courte en nombre d'arcs de s vers v). Montrer que cet algorithme calcule les plus courts chemins (en nombre d'arcs) issus de s. Rapporter le tableau des distances pour les sommets du graphe G1.
4) Ce dernier algorithme peut être vu comme un cas spécial de l'algorithme de *Dijkstra*. A quoi correspondrait dans *ParcoursLargeur* l'ensemble S utilisé dans la formulation de l'algorithme de *Dijkstra* présenté dans le polycopié ? Justifier.

1)



Graphe G1

2) $O(m)$

3) initialiser Dist[s] = 0 ;
ensuite : Dist[j] = Dist[i] + 1

4) l'ensemble des sommets examinés = S.

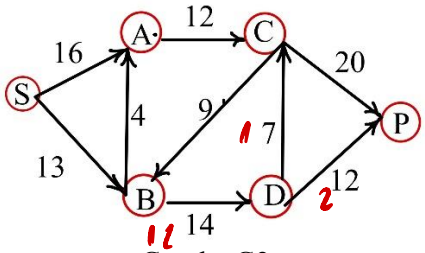
PARTIE II – COPIE SEPAREE

Exercice 2. (12 points)

Algorithme de flot maximal (Edmonds et Karp) :

(I) Initialiser f sur tous les arcs : f (u)= 0, pour tout u.
(II) Construire le graphe d'écart $G_e(f)$ et chercher le chemin le plus court en nombre d'arcs de s à p dans $G_e(f)$. Aller en (III) si un chemin a été trouvé sinon FIN. (Ce chemin est appelé chemin améliorant)
(III) Améliorer le flot f le long du chemin obtenu et retourner en (II).

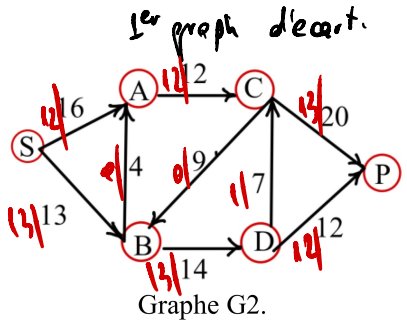
1. Application : faites tourner l'algorithme sur le graphe ci-dessous. Les arcs sont valués par leur capacité. Rapporter le graphe d'écart $G_e(f)$ et le chemin améliorant calculé à chaque itération. Rapportez la coupe minimale.



Graphe G2.

2. L'algorithme ci-dessus permet d'obtenir le flot de valeur maximale. Démontrer rigoureusement que pour toute chaine améliorante dans le réseau de transport il y a un chemin dans le graphe d'écart correspondant au flot courant et vice-versa. Déduire la validité de l'algorithme d'Edmonds et Karp.
3. Nous allons étudier dans la suite la complexité (polynomiale) d'un tel algorithme. La preuve sera construite pas à pas.
   a. Soit v un sommet quelconque du graphe différent de s et p. Nous noterons **Dist[fk,v]** la distance minimale en terme de nombre d'arcs entre s et v dans le graphe d'écart associé au flot fk (itération k). **On va accepter sans preuve le résultat suivant : la distance Dist[f_k,v] ne diminue pas avec l'augmentation du flot aux itérations suivantes.** En se basant sur le résultat ci-dessus, démontrez qu'un arc ne pourra faire partie comme arc saturé d'un chemin améliorant qu'au plus |V|/2 fois. (*Remarque : lorsqu'un chemin améliorant est utilisé, le flot devient maximal sur au moins l'un de ses arcs, p.ex. (i,j) et celui-ci disparaitra du prochain graphe d'écart mais l'arc inverse (j,i) y sera présent. L'arc (i,j) réapparaitra dans un prochain graphe d'écart seulement si l'arc (j,i) fera partie d'un chemin améliorant. Raisonnez sur la distance de s à i et j pour prouver la propriété.*)
   b. Quel est alors le nombre d'itérations (en fonction de n et m) de l'algorithme d'Edmonds et Karp ?
   c. Déduire la complexité de l'algorithme. Justifier.
4. Comment allez-vous modifier cet algorithme pour calculer le flot maximum de coût minimum ? Que devient la complexité ?

1)



Graphe G2.

flot(1) : S → A → C → P (12)

flot(2) : S → B → D → P (12)

flot(3) : S → B → D → C → P (1)

Coupe min = {S, A}.

2) Tt chemin dans le graphe d'ecart est composé soit d'arcs avec capacité residuelle > 0, soit avec des arcs dans le sens inverse avec flot non nul. Or, ces mêmes arcs sont présents dans le graph initial et peuvent être combinés pour construire une chaine améliorante.

3) a) Soit (i,j) un arc saturé dans un chemin améliorant lors du flot f_k

$d(f_{k+1}, j) = d(f_k, i) + 1$. lors d'une itération suivante, l'arc (j,i) fera parti d'un nouveau chemin améliorant tel que

$d(f_{k'}, i) = d(f_{k'}, j) + 1 \geq d(f_k, j) + 1 = d(f_k, i) + 2 \Rightarrow$

à chaque nouvelle apparition de (i,j) en tant que arc saturé la distance a été augmentée de +2, et puisque la longueur du chemin max est $\leq n-1$ → il y aura au plus $O(|V|/2)$ Chemins où (i,j) apparaîtra comme arc saturé. →

il y aura au plus $m * |V|/2$ chemins ameliorants.

b) $O(n * m)$

c) $O(n * m^2)$

4) Au lieu de calculer les chemins les plus courts en nombre d'arcs, on calculera les chemins de cout minimum (dans le graphe d'ecart). La complexité deviendra

$O(n * m)$ → complexité de l'algorithme de cheminement utilisé.

p.ex. $O(n^2 * m^2)$ si on utilisera Fourd.