INFO601 : graphes et algorithmes
Examen

Pierre Hyvernat, Gérald Cavallini
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 17, poste : 94 22
email : Pierre.Hyvernat@univ-smb.fr
www : http://www.lama.univ-smb.fr/~hyvernat/

Durée : 1h30.

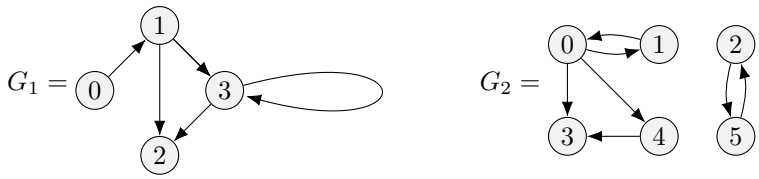
Documents et calculatrices interdits.

Un barème provisoire est donné dans la marge.

1 point négatif sera réservé à la présentation de vos réponses...

Partie 1 : Représentation des graphes

[2] *Question 1.* Donnez une représentation des graphes suivants comme listes d'adjacence *et* comme matrices d'adjacence.



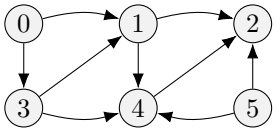
[2] *Question 2.* Écrivez une fonction en (pseudo) Python pour convertir un graphe donné sous forme de listes d'adjacence en une matrice d'adjacence.

Partie 2 : parcours simple

On rappelle l'algorithme du parcours en profondeur "basique"

```
Vu = [False] * N                # N est le nombre de sommets
Pred = [None] * N
ATraiter = [s0]
while ATraiter:                 # tant que ATraiter est non vide
    s = ATraiter.pop()
    if Vu[s]: continue          # on ignore les sommets déjà vus
    Vu[s] = True
    for v in G[s]:              # pour tous les voisins ‘v‘ de ‘s‘
        if not Vu[v]:
            ATraiter.append(v)
            Pred[v] = s
```

[2] *Question 1.* Faites tourner cet algorithme sur le graphe suivant en partant du sommet `s0=0` et en supposant que les listes de voisins sont triées dans l'ordre croissant. Donnez les valeurs de `s`, `ATraiter`, `Pred` et `Vu` à la fin de chaque passage dans la boucle principale "`while ATraiter`". Dessinez également l'arbre du parcours obtenu à la fin de l'algorithme.



[2] *Question 2.* Le parcours en *largeur* s'obtient en utilisant une *file* `ATraiter` (au lieu d'un tableau, utilisé comme une pile) et en ignorant les voisins ayant déjà un prédecesseur.

Donnez le résultat du parcours en largeur sur le graphe de la question précédente.

[2] *Question 3.* Si on ne dispose pas d'une structure de file, on peut utiliser un tableau de la manière suivante.

- L'ajout se fait en fin de tableau (`ATraiter.append(...)`).
- Le retrait se fait en début de tableau, *mais sans supprimer l'élément.* On incrémente simplement une variable `debut` pour se souvenir de la position du début du tableau.

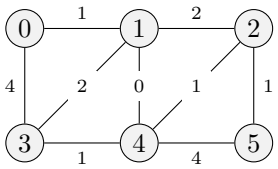
Décrivez précisemment (en pseudo Python) les modifications à apporter au parcours basique *en utilisant cette méthode.*

Cette astuce peut poser un problème de mémoire en général. Expliquez ce qui pourrait se passer, et pourquoi ce n'est pas un gros problème dans le cas du parcours en largeur.

Partie 3 : graphes pondérés et algorithme de Dijkstra

[2] *Question 1.* L'algorithme de Dijkstra calcule les distances minimales des sommets *depuis* `s0` dans un graphe pondéré.* La distance minimale de `s0` à `s` est stockée dans la case `D[s]` d'un tableau. Initialement cette distance est ∞ sauf pour `s0` lui même. L'algorithme est un parcours où le sommet `s` choisi dans `ATraiter` est le sommet avec la distance actuelle `D[s]` minimale ; et les tableau `Pred` et `D` sont mis à jours uniquement lorqu'un voisin donne une distance plus petite que la distance courante.

Faites tourner l'algorithme de Dijkstra sur le graphe suivant, en partant de `s0=0`. Précisez à chaque étape le sommet choisi et la valeur du tableau `D`. Donnez également la valeur finale du tableau `Pred` et dessinez l'arbre de parcours.

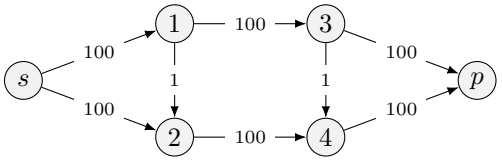


[2] *Question 2.* Dans un parcours, le tableau `Vu` permet d'éviter de boucler. Que se passera t'il si on n'utilise pas de tableau `Vu` dans l'algorithme de Dijkstra ?

[2] *Question 3.* Si on modifie l'algorithme de Dijkstra en choisissant le sommet `s` dans `ATraiter` avec `D[s] maximal`, va t'on obtenir les plus long chemins depuis `s0` ?

Partie 4 : flot maximal

[2] *Question 1.* Utilisez l'algorithme de Ford-Fulkerson pour trouver un flot maximal dans le réseau suivant. Détaillez chacune des étape en donnant le chemin augmentant trouvé et le flot correspondant. (Vous pouvez choisir les chemins augmentant comme vous voulez.)



[2] *Question 2.* Que se passe t'il si on utilise un parcours en profondeur basique pour chercher les chemins augmentants dans le graphe précédent ?

Donnez les chemins augmentants trouvés si on suppose que les voisins sont triés par ordre croissant.

---

* La "longueur" d'une arête est égale à son poids.