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 graphique *sans croisement d'arêtes* du graphe (non-orienté) donné par les listes d'adjacences suivantes :

```
G = [ [0,2],
      [2],
      [0,1,3,4],
      [2],
      [2,5,6],
      [4],
      [4],
]
```

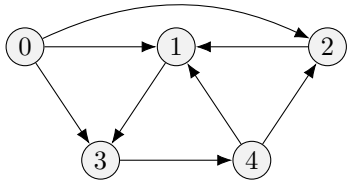
[2] *Question 2.* Écrivez une fonction en (pseudo) Python qui renvoie le nombre total d'arêtes (non-orientées) dans un graphe donné sous forme de listes d'adjacences, comme dans la question précédente.

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.* Comment peut-on modifier ce parcours pour obtenir un *parcours en largeur* ? Donnez précisément ces modification en (pseudo) Python.

[2] *Question 3.* Les parcours de questions précédentes calculent un tableau `Pred` contenant l'arbre de parcours depuis le sommet `s0`. Écrivez en (pseudo) Python une fonction qui prend `s0`, `Pred` et un autre sommet `s1` et renvoie la taille du chemin de `Pred` qui va de `s0` vers `s1`. S'il n'y a pas de tel chemin, votre fonction devra renvoyer `-1`.

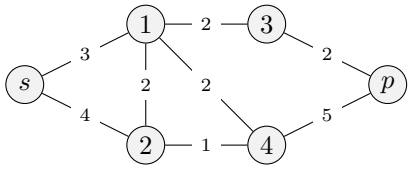
Cette fonction peut-elle renvoyer `-1` lorsqu'elle est appelé sur des sommets `s0` et `s1` du graphe de la question 1 (et avec le tableau `Pred correspondant`) ?

[2] *Question 4.* Qu'est-ce qu'une *composante connexe* dans un graphe non-orienté ?

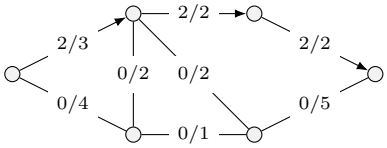
Comment pourriez vous calculer la liste des sommets de la composante connexe contenant un sommet s dans un graphe G ?

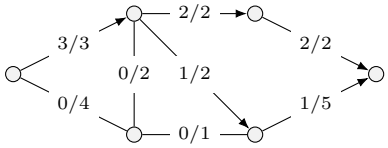
Partie 3 : flot maximal

[2] *Question 1.* On considère le réseau suivant :



Voici les 2 premières étapes de l'algorithme de Ford-Fulkerson pour trouver un flot maximal.[1]



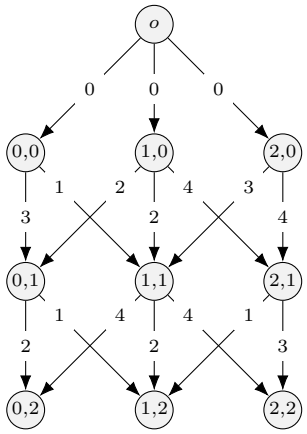


- Quels étaient les deux chemins augmentants utilisés pour ces deux premières étapes ?
- Détaillez la suite de l'exécution de l'algorithme de Ford-Fulkerson sur ce réseaux.

[1] *Attention,* les arêtes avec un flot strictement négatif ne sont pas notées sur le graphe des flots.

Partie 4 : graphes pondérés et plus court chemin

[2] *Question 1.* Donnez, pour chacun des sommets du graphe suivant, la longueur du plus court chemin depuis le sommet o.[2]



Votre réponse devra être un tableau 3×3 de la forme suivante :

```
D = [ [0,  0,  0],
      [2,  ?,  ?],
      [?,  ?,  ?] ]
```

Les 3 premiers "0" correspondent aux plus courtes distance de o vers les sommets $(0,0)$, $(1,0)$ et $(2,0)$, et le "2" correspond à la plus courte distance de o vers $(0,1)$.

[3] *Question 2.* Le graphe de la question précédente est découpé en lignes et les voisins d'un sommet de la ligne k sont sur la ligne $k+1$. C'est exactement la situation rencontrée lors du TP2 sur le recadrage d'images.

Dans ce cas, nous pouvons faire un calcul plus direct que l'algorithme de Dijkstra : la première ligne de `D` est initialisée à `0`, et chaque ligne `D[`$k+1$`]` est construite à partir de la ligne `D[`k`]`.

Écrivez le code correspondant pour le cas général où :

- le graphe est composé de `h` lignes et `colonnes` (sans compter le sommet o),
- le sommet (x, y) a 3 voisins : $(x-1, y+1)$, $(x, y+1)$ et $(x+1, y+1)$, sauf sur les bords,
- le poids de l'arête de (x, y) vers $(x', y+1)$ s'obtient avec `G.poids(`x`,`y`,`x'`)`

[1] *Question 3.* Comment se comporte cet algorithme si le graphe contient des arêtes de poids négatif ?

[*] *Question 4.* Bonus : discutez de la complexité de votre programme et comparez là à celle de l'algorithme de Dijkstra.

[2] La "longueur" d'une arête est égale à son poids.