TELECOM Bretagne

Majeure INFO - INF 435 - TP2

Graphe, Arbre, Diffusion

Cécile Bothorel

Printemps 2016

1 Objectif

Cette séance de TP s'intéresse à un problème de théorie des graphes qui a un intérêt applicatif important : la diffusion d'un message dans un réseau. Ici, le graphe est valué, la structure de données du TP1 ne peut donc pas être réutilisée telle quelle. Ne vous jetez pas sur la programmation, pensez à bien réfléchir avant.

2 Problème de diffusion dans un réseau

Vous êtes l'organisateur du WEI, et vous êtes très fier : ce week-end s'est très bien déroulé, c'est dimanche, vous savourez ce dernier jour. Votre sens de l'organisation n'a jamais été pris en défaut. Or, on vient de vous annoncer que le car doit partir une heure avant l'horaire prévu, c'est-à-dire dans quelques minutes.

Évidemment, pas de connexion correcte à internet dans votre centre de vacances. Il faut prévenir toute la promotion dans un temps record, vous n'y arriverez pas tout seul, il faut qu'on vous aide. Heureusement, vous connaissez bien toute la promotion et, comme vous êtes un peu maniaque, vous avez pris soin de noter tous les étudiants qui se connaissent. Avec des idées derrière la tête, vous avez également estimé leur relation de 100 (grosse amitié) à 1 (franche antipathie). Ces informations sont contenues dans le fichier `promotion.graph`.

Vous êtes un fainéant malin et vous vous dites que le bouche-à-oreilles est sans doute un moyen efficace de diffuser l'information : il suffirait de donner l'information à un seul étudiant, qui pourrait ensuite la transmettre à quelques autres étudiants qui, eux-mêmes, pourraient également la diffuser de la même manière, etc. Vous êtes également très organisé et vous connaissez les lascars avinés qui vous entourent. Il faut faire des choses simples et méthodiques : vous vous engagez à donner une feuille qui permet à chaque étudiant de connaître la liste des étudiants qu'il doit alerter... et de ne pas en oublier surtout !

2.1 Diffusion de l'information

Vous souhaiteriez que la communication ne concerne que des étudiants qui s'entendent bien. Vous souhaitez donc que la somme des "niveaux d'amitiés" de tous les liens utilisés pour diffuser l'information soit maximale.

Exercice 1 : Parmi tous les liens du graphe `promotion.graph`, quel est l'ensemble des liens qui vous garantit que (1) tous les étudiants recevront l'information et (2) le "coût amical total" est maximal ? Construire une telle représentation de la promotion.

Le but est de transmettre un message le moins possible et le mieux possible pour éviter qu'il ne soit (trop) déformé.

Modélisons le canal de transmission entre élèves par les arêtes d'un graphe, et les élèves par les nœuds. Le poids sur les arêtes représente la force de l'amitié.

Construire l'arbre couvrant va nous permettre d'épurer le graphe en ne gardant que le minimum d'arêtes tout en garantissant l'atteignabilité de tous les nœuds.

Si nous inversons la force des poids et cherchons l'arbre couvrant minimal, nous obtiendrons un réseau qui minimise le nombre de transmissions tout en maximisant la sympathie... et donc la fiabilité du message.

2.2 Et si un étudiant était oublié ?

Si des élèves n'ont pas été prévenus, il faut pouvoir retrouver le coupable ou les coupables qui n'ont pas transmis le message comme prévu.

Exercice 2 : A partir du résultat de votre code précédent, créez une structure de données qui permette de retrouver n'importe quelle chaîne de diffusion, de n'importe quelle source à n'importe quelle cible.

Logiquement vous n'avez rien à faire ici.

En effet, il s'agit de construire en même temps que la liste d'adjacence qui représente l'arbre couvrant (avec Prim par exemple), une liste de "prédécesseurs". Ainsi si un étudiant est oublié, il est possible de "remonter" la chaîne de diffusion et de convoquer tous les chaînons pour identifier le coupable.

Or le graphe n'est pas orienté ici. Donc la liste d'adjacence devrait DEJA encoder l'information recherchée. Vous auriez dû déjà, soit créer deux listes, soit stocker le lien bi-directionnel directement dans la liste d'adjacence comme c'est le cas dans le graphe initial.

```
def createEdge(node1, node2, weight):
  ## insert the edge linking two vertices 'node1' and 'node2'
  ## in your data structure
  ## weight is the 'friendship value'
  if node1 in setNodes and node2 in setNodes:
    listAdj[node1].append([node2, int(weight)])
    listAdj[node2].append([node1, int(weight)])
```

La structure de données est donc bien là.

Si la diffusion n'a pas atteint tout le monde, il s'agit de trouver de manière exhaustive tous les chemins à partir d'une source donnée, avec BFS par exemple et ainsi de trouver le ou les fautifs qui ont rompu la chaîne.

2.3 Optimisation de l'arbre de diffusion

Vous avez mis en place votre système et vous vous avez remarqué un problème inattendu : le téléphone arabe ! Le message se déforme au fur et à mesure qu'il est retransmis par les étudiants.

Exercice 3 : A qui devez-vous donner l'information pour réduire l'impact de ce téléphone arabe, c'est-à-dire que le nombre de retransmission du message soit minimal ?

Pour trouver le nœud source qui va minimiser l'effet du bouche à oreille, il s'agit de prendre le nœud (enfin un nœud, il y a potentiellement plusieurs candidats) à partir duquel l'arbre couvrant minimal est le moins profond.

Vous pouvez utiliser une mesure de centralité : la "closeness centrality" est celle qui trouve les nœuds qui sont les "plus proches" de chacun des autres nœuds.

Ou bien vous pouvez imaginer une solution ad'hoc avec un arbre. En partant des feuilles, on remonte petit à petit dans l'arbre jusqu'à arriver à une racine unique.

```
## return the node so that the tree depth is minimal
## 'tree' is a list of pair of nodes
def betterRoot(tree):
  # starting from the tree, we start from the leaves
  # and we discard iteratively all fathers
  # until we reach one unique root or two possible roots.

  # 'savedNodes' is a set of nodes that could be the best root
  # Initially, they are all candidates
  savedNodes = setNodes

  # we can determine the best root when there is only one
  # or two nodes in this list 'savedNodes'
  while len(savedNodes)>2:
    # compute the node degree for the tree involving only the nodes
    # in the 'savedNodes' set.
    dictNode = giveDegree(tree, savedNodes)

    # rebuild savedNodes by picking only the nodes that are not leaf.
    savedNodes = []
    for (node,degree) in dictNode.iteritems():
      if degree > 1:
        savedNodes.append(node)
  # there is one or two nodes in the 'savedNodes' list. Return one of them
  return savedNodes[0]
```

3 Instructions

Vous trouverez sur `Moodle` des fichiers à télécharger et sauvegarder dans votre espace personnel :

- un fichier `tp2.py` sur lequel vous allez travailler
- un fichier `graphviz.py` que vous n'avez pas besoin de modifier
- un fichier `promotion.graph` qui contient les données du graphe

Lorsque vous exécutez `python tp2.py`, un nouveau fichier est généré. Il s'agit d'une image `promotion.png` que vous pouvez visualiser (en utilisant par exemple `eog` sous Linux). Votre mission consiste à compléter le fichier `tp2.py`.