LINMA1691 : Théorie des graphes
Devoir 4 : Le temple maudit et l'algorithme de Ford-Harisson

Contexte

Indiana Jones se trouve actuellement en Inde à la recherche d'une des fameuses pierres de Sankara qui a été récemment volée à un village local. Mais Indiana est persuadé que la pierre a été cachée dans un temple au sud du village. Le lendemain, Indiana et ses compagnons se rendent au temple. Soudainement, alors qu'ils explorent les lieux, un éboulement se produit à la suite duquel les aventuriers se retrouvent séparés.

Heureusement, l'expédition a été bien planifiée et chacun des aventuriers possède une carte du temple (voir figure 1) indiquant où se trouvent toutes les sorties possibles ainsi qu'une radio permettant de coordonner leurs décisions. Le temple est composé de salles numérotées de 0 à $N-1$ et il est possible de se déplacer de salle en salle en prenant un couloir ou un escalier. On supposera que Indiana et ses amis ont tous été isolés les uns des autres par l'éboulement.

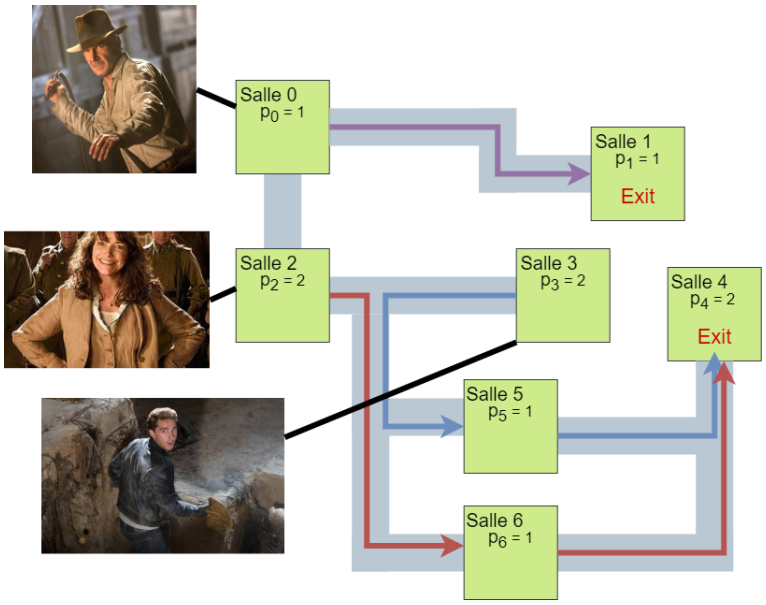

Figure 1: Exemple de configuration du temple

Cependant, se déplacer dans le temple n'est pas sans risque. En effet, il y a de nombreux pièges tels que des trappes dissimulées, des chutes de pierres, ... On fait l'hypothèse que seulement les salles sont piégées mais pas les couloirs et les escaliers. Si les aventuriers sont vigilants alors ils peuvent déjouer les pièges mais la plupart des pièges ne peuvent pas être évités indéfiniment. Pour chaque salle $i \in \{0, \dots, N-1\}$, on définit un nombre entier p_i tel que $1 \leq p_i \leq \infty$ et le piège de la salle i s'active à la $p_i + 1$ ème personne qui passe. Autrement dit, il ne doit pas y avoir plus que p_i passages par une salle pour garantir la sécurité des membres de l'expédition (salles initiales et de sorties comprises).

Dès lors, on cherche à savoir s'il est possible pour Indiana et tous ses compagnons de s'échapper du temple.

Input

On vous demande de compléter la fonction `escape_temple(N,corridor,trap,adventurer,out)` dont les arguments sont définis ci-dessous (les salles sont numérotées en commençant par zéro)

- `N` est le nombre de salles dans le temple
- `corridor` est une liste de tuple `(i,j)` où `i` et `j` sont des salles, indiquant s'il est possible de passer de `i` vers `j` et réciproquement de `j` vers `i` (les permutations `(j,i)` ne sont pas reprises dans la liste)
- `trap` est une liste de taille `N` avec `trap[i]` qui est la valeur du paramètre p_i
- `adventurer` est une liste contenant les salles dans lesquelles se trouve initialement chaque aventurier
- `out` est une liste contenant les salles qui permettent de sortir du temple

Output

Votre fonction `escape_temple(N,corridor,trap,adventurer,out)` devra renvoyer `True` si tous les aventuriers peuvent sortir du temple sans activer de piège et `False` sinon. La complexité attendue de votre solution est $\mathcal{O}(a(N + |corridor|))$ où a est le nombre d'aventuriers.

Par exemple, dans les cas de la figure 1, on a les données suivantes :

```
N          =  7
corridor   =  [(0,1),(0,2),(2,3),(2,5),(2,6),(3,6),(3,5),(6,5),(5,4),(6,4)]
trap       =  [1,1,2,2,2,1,1]
adventurer =  [0,2,3]
out        =  [1,4]
```

Pour cet input, les aventuriers peuvent s'échapper de temple sans activer de pièges.

Consignes

Vous devez soumettre votre fichier solution complété *temple.py* sur l'activité Inginious[1]
Note: Pas de texte "print" ou "import" dans le fichier.

Le langage de programmation est **Python 3**.

Deadline : vendredi 17 décembre 22h

Questions : Au tp, sur le forum moodle ou par mail :

brieuc.pinon@uclouvain.be et timothe.taminiau@student.uclouvain.be

Hint

Vous pouvez vous inspirer de l'implémentation de Ford-Fulkerson disponible sur le wikipedia anglais [2]. Il s'agit d'une version utilisant BFS pour chercher les chemins f-augmentant, cette version est connue sous le nom de Edmonds-Karp. Notez par contre que le code disponible n'a pas la complexité attendue (utilisation d'une matrice nombre de noeuds×nombre de noeuds); vous devrez donc la modifier.

[1] https://inginious.info.ucl.ac.be
[2] `https://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm`