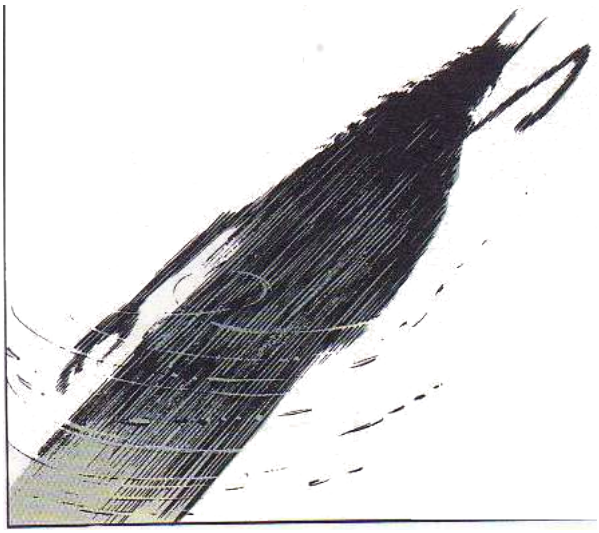
Algorithmique
Partiel n° 4 (P4)

Info-spé S4
Epita

15 mai 2018 - 10 : 00

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (classes, fonctions, méthodes) est indiqué en **annexe** !
  - — Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
- ☐ Durée : 2h00



Exercice 1 (Gisement épuisant... – *5 points*)

Des mineurs veulent sécuriser la circulation entre les différents points d'extraction (représentés par des sommets) reliés par des galeries. Tous ces points d'extraction sont accessibles séparément depuis l'extérieur et le réseau est suffisamment complexe pour qu'en général plusieurs itinéraires permettent de passer d'un point d'extraction à un autre. Il n'est donc pas nécessaire que chaque galerie soit sécurisée pour qu'il existe toujours entre deux points d'extraction au moins un itinéraire qui le soit.

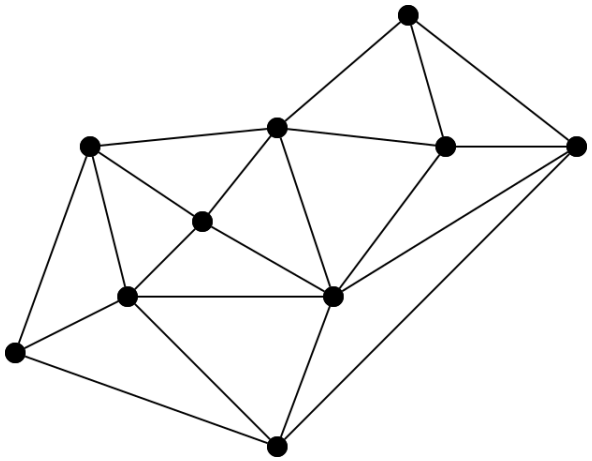


FIGURE 1 – Graphe associé à une mine et son réseau.

1. On veut déterminer le plus petit nombre de galeries à sécuriser :
   (a) À quoi correspond la solution en termes de graphes ?
   (b) Dans le cas de la figure 1, combien faut-il sécuriser de galeries ?
   (c) Proposer une solution graphique.
   (d) En généralisant à un réseau de N points d'extraction, combien faudrait-il sécuriser de galeries ?
2. On affine l'analyse du problème :
   Pour chaque galerie, nous avons évalué le coût des travaux de sécurisation (voir figure 2).

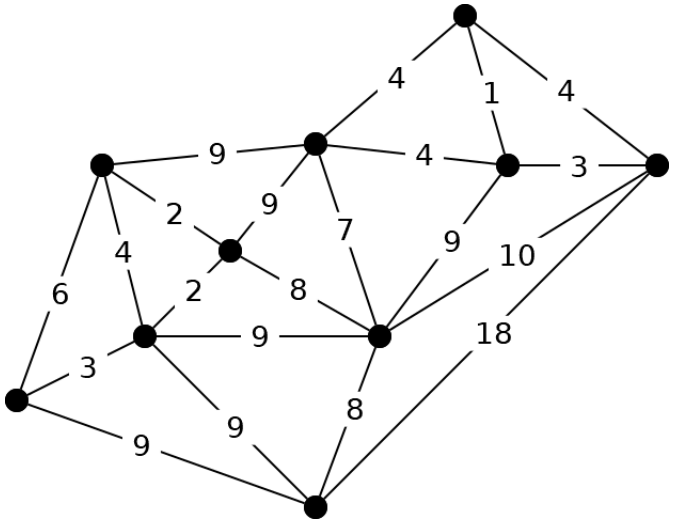


FIGURE 2 – Graphe valué : sécurisation des galeries

   (a) Proposer une solution graphique qui minimise le coût des travaux de sécurisation.
   (b) Cette solution est-elle unique ?
   Pourquoi ?

Exercice 2 (Asterix et le devin – *13 points*)

On cherche à trouver le meilleur chemin entre une `source` et une `destination` données dans un 1−graphe orienté valué à coûts positifs en traitant un minimum de sommets inintéressants. Pour ce faire, nous requérons l'aide du devin *Heuristix*. Le devin *Heuristix* est capable de nous indiquer si un sommet est proche de la `destination` ou pas. Comme tout devin, sa réponse n'est pas exacte.

Si l'on donne à *Heuristix* un graphe, un sommet `s` et une `destination` dans le graphe, il nous rend un réel positif estimant la distance à parcourir depuis le sommet `s` vers la `destination` (plus la valeur est petite, plus le sommet est proche de la `destination`). L'estimation ne tient pas compte du chemin parcouru depuis la `source` pour atteindre le sommet `s`.

1. **L'algorithme :**

   On se propose de construire un algorithme utilisant les informations du devin, estimation donnée par la fonction `heuristix` (voir annexes). L'algorithme à écrire doit donner le meilleur chemin entre une `source` et une `destination` données. La différence avec les algorithmes de plus courts chemins classiques est que l'on utilise l'estimation du devin pour le choix des sommets à traiter : à chaque itération on choisit le sommet dont la **somme** du distance calculée depuis la `source` **avec** l'estimation du devin est la **plus petite**. Il faut de plus éviter les circuits, donc **on ne traitera pas à nouveau les sommets déjà traités** : dans le langage du devin, ces sommets sont dits "fermés", ceux non encore traités sont "ouverts".

   (a) Quel est le nom de cet algorithme ?

   (b) **Utilisation de structures de données supplémentaires :** La classe à utiliser pour le graphe est donnée en annexe (implémentation par listes d'adjacence). Vous pouvez de plus utiliser tous les types / classes (et les fonctions et méthodes associées) donnés en annexe. L'optimisation est bien entendu requise.
   Comment représenter les sommets "ouverts" ?
   Comment représenter les "fermés" ?

   (c) **Et la complexité ?** En supposant le coût de la fonction `heuristix` constant, donner un ordre de grandeur de la complexité de l'algorithme (dans le pire des cas : tous les sommets sont utilisés) avec les choix faits à la question précédente (avec n le nombre de sommets, p le nombre d'arcs).

   (d) **Implémenter l'algorithme :** c'est une fonction qui prend en paramètres le graphe (à coûts positifs), les deux sommets `source` et `destination` ; elle donne en résultat le chemin de la `source` à la `destination` obtenu sous la forme d'une liste de sommets (de la source à la destination).
   Dans un premier temps, on supposera que le chemin existe toujours.
   Bonus si la fonction déclenche une exception lorsque le chemin n'existe pas !

2. **Les devins :**

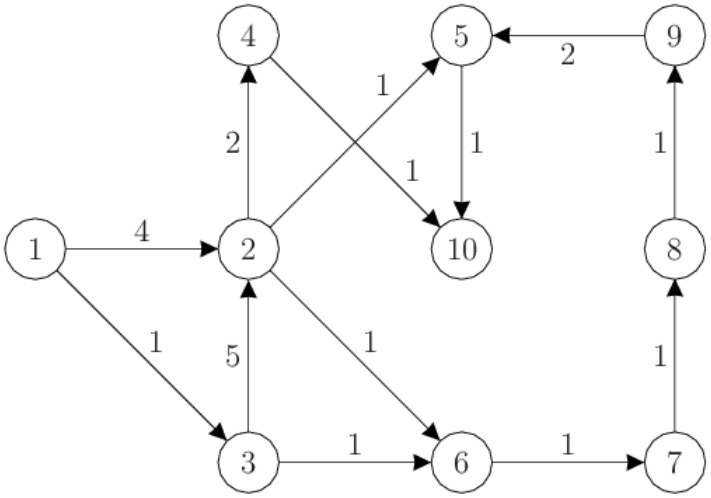


FIGURE 3 – Graphe pour les devins

Le plus court chemin de 1 à 10 ?

Lorsque l'on appelle le devin Heuristix, trois personnes se présentent :

⋆ **Heuristix le Hollandais** (appelé *HeuristixD*) : très (trop ?) optimiste, il pense que tous les sommets sont proches de la destination.

Dans le cas présent, il nous renvoie systématiquement 0.

⋆ **Heuristix du Nouveau Monde** (appelé *HeuristixM*) : germain (c'est un Angle) ayant beaucoup voyagé (d'où son surnom) il nous donne une estimation correspondant au nombre minimum d'arcs pour atteindre la destination, il aime les choses carrées.

Dans le cas présent, il nous donnera donc :

s	1	2	3	4	5	6	7	8	9	10
HeuristixM (s)	3	2	3	1	1	5	4	3	2	0

⋆ **Heuristix le Byzantin** (appelé *HeuristixB*) : comme tous les généraux byzantins, HeuristixB répond souvent n'importe quoi.

Dans le cas présent, il nous donne les sommes des coûts entrants, donc :

s	1	2	3	4	5	6	7	8	9	10
HeuristixB (s)	0	9	1	1	3	2	2	1	1	2

(a) Appliquer l'algorithme sur le graphe de la figure 3 avec les heuristiques *HeuristixD* et *HeuristixM* pour trouver le meilleur chemin entre les sommets 1 et 10 : remplir les vecteurs *dist* (les distances depuis la source, sans l'heuristique...) et *pere* et donner la liste des sommets traités (dans l'ordre). Comme d'habitude, si plusieurs choix sont possibles on choisira le sommet de numéro inférieur.

(b) À quel algorithme correspond la solution avec l'estimation d'*HeuristixD* (attention, l'orthographe compte...) ?

(c) **Bonus** Que penser de l'estimation d'*HeuristixB* ? Est-elle est meilleure que celle d'*HeuristixM* ?

Exercice 3 (What is this ? – *4 points*)

```python
def __dfs(G, s, x, y, M):
  M[s] = True
  for adj in G.adjlists[s]:
     if adj == y:
        if s != x:
           return True
     elif not M[adj]:
        if __dfs(G, adj, x, y, M):
           return True
  return False

def dfs(G, x, y):
  M = [False] * G.order
  return __dfs(G, x, x, y, M)
```

```python
def what(G):
  n = G.order
  L = []
  for x in range(n):
    for y in G.adjlists[x]:
       if x < y:
          L.append((G.costs[(x,y)], x, y))
  L.sort()  # sorts L in increasing order
  while L != []:
     (_, x, y) = L.pop()
     if dfs(G, x, y):
        G.removeedge(x, y)
```

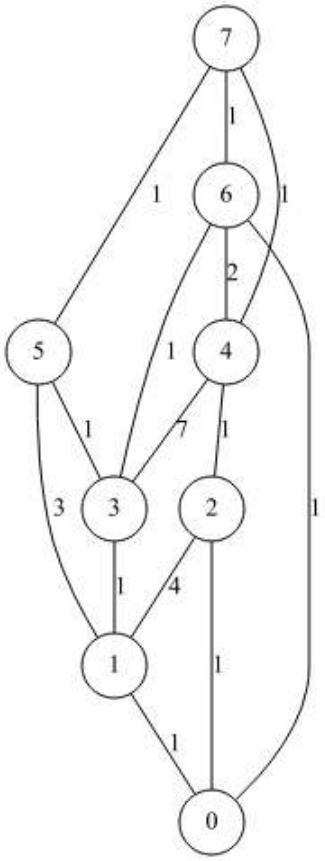


FIGURE 4 – Graph for `what`

1. Que teste la fonction `dfs(G)` ?
2. La fonction `what` :
   (a) Donner le résultat de l'application de la fonction `what` au graphe de la figure 4 (dessiner le graphe).
   (b) Quelle propriété a le graphe après application de la fonction ?
   (c) Comment optimiser cette fonction ?

Annexes

Les classes `Graph` et `Heap` sont supposées importées. Les graphes ne peuvent pas être vides.

Les graphes

```python
class Graph:
    def __init__(self, order, directed = False, costs = False):
        self.order = order
        self.directed = directed
        self.adjLists = []
        for i in range(order):
            self.adjLists.append([])
        if costs:
            self.costs = {}
        else:
            self.costs = None

    def addedge(self, src, dst):
        self.adjlists[src].append(dst)
        if not self.directed and dst != src:
            self.adjlists[dst].append(src)
        if cost:
            self.costs[(src, dst)] = cost
            if not self.directed:
                self.costs[(dst, src)] = cost

    def removeedge(self, src, dst):
        if dst in self.adjlists[src]:
            self.adjlists[src].remove(dst)
            if self.costs:
                self.costs.pop((src, dst))
            if not self.directed and dst != src:
                self.adjlists[dst].remove(src)
                if self.costs:
                    self.costs.pop((dst, src))
```

Les tas

- `Heap(`n`)`
  retourne un tas de taille n
- H`.push(`s`,` val`)`
  ajoute s de valeur val dans H
- H`.update(`s`,` $newval$`)`
  si s n'est pas présent, équivalent à `H.push(`s`,` $newVal$`)`,
  sinon met à jour le tas après minimisation de la valeur de s à $newVal$
- `(`val`,` s`) =` H`.pop()`
  supprime et retourne l'élément (s) de plus petite valeur (val)
- H`.isempty()`
  teste si H est vide

`List`

- `len(L)`
- `L.append(x)`
- `L.pop()` `L.pop(i)`
- `L.insert(i, x)`

Autres

- `range`.

Le devin

La fonction `heuristix(`G`,` s`,` dst`)` donne une estimation de la distance entre s et dst dans G.

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.