Algorithmique
Partiel n° 4 (P4)

Info-spé S4
Epita

14 mai 2019

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 (Floyd revisité – *3 points*)

1. Expliquer simplement comment on peut modifier la procédure de Floyd (vue en cours), de sorte qu'elle s'arrête dès qu'elle détecte un circuit absorbant.
2. Dans certains problèmes de communication, on a besoin de connaître le point (ou les points) le moins éloigné de tous les autres points d'un réseau. On appelle **excentricité** d'un sommet x, dans un graphe orienté et valué par des coûts positifs, $G =< S, A, C >$, la quantité :

$$\text{exc}(x) = \max_{y \in S}\{ppdistance(x, y)\}$$

C'est la distance nécessaire pour atteindre x depuis chaque autre sommet. On appelle **centre** du graphe G, un sommet d'excentricité minimum.

Expliquer simplement comment on peut utiliser la procédure de Floyd (vue en cours) pour trouver le centre d'un graphe.

Exercice 2 (ARM ou non ? – *2 points*)

Examiner si le graphe partiel T obtenu dans la construction suivante est un arbre de recouvrement minimum de G ; si oui, expliquer pourquoi, sinon construire un contre-exemple.

Soit $G =< S, A, C >$ un graphe non orienté connexe valué. On partage G en deux sous-graphes : $G1 =< S1, A1, C >$ et $G2 =< S2, A2, C >$ avec $S = S1 \cup S2$. Soit $T1$ (respectivement $T2$) un arbre de recouvrement minimum de $G1$ (respectivement de $G2$). On choisit une arête v de coût minimum parmi les arêtes ayant une extrémité dans $S1$ et l'autre dans $S2$. T est alors le graphe partiel résultant de la réunion des arêtes de $T1$, de $T2$ et de v.

Exercice 3 (Mangez des crêpes – *11 points*)

Ci-dessous la recette de la crêpe à la banane flambée, avec pour chaque tâche sa durée en secondes.

La recette	Durée (en sec.)	Ref.
Mettre la farine dans un saladier,	3	A
ajouter deux œufs,	30	B
ajouter le lait doucement et mélanger.	600	C
Dans une poêle mettre du rhum.	3	D
Couper les bananes en fines lamelles,	300	E
les mélanger au rhum.	30	F
Faire chauffer le mélange,	120	G
faire flamber le mélange.	10	H
Faire cuire une crêpe,	10	I
verser du mélange bananes-rhum sur la crêpe.	10	J

Quelques précisions concernant l'ordre des tâches :

— la préparation de la pâte à crêpe et celle du mélange rhum-banane peuvent se faire en parallèle ;

— ce n'est que lorsque la crêpe sera cuite et que le mélange sera prêt qu'on pourra verser du mélange sur la crêpe et enfin la déguster ;

— les autres étapes se réalisent séquentiellement : on doit mettre la farine avant les œufs, on doit mettre le rhum avant les bananes (qui peuvent être coupées en avance) dans la poêle.

1. Modéliser la recette sous forme de graphe :
   — Les sommets sont les tâches.
   — Les tâches *start* et *end* représentent le début (commande de la crêpe) et la fin (dégustation) du projet.
   — La représentation du graphe doit être planaire [1].

1. Pour mémoire, cela signifie que les arcs ne doivent pas se couper !

Le graphe qui représente la recette est sans circuit. De plus, tous les sommets sont atteignables depuis un sommet donné (ici la commande de la crêpe = la tâche *start*, sommet n° 0 dans la représentation machine). Et tous les sommets peuvent atteindre la tache de fin (dernier sommet dans la représentation machine).
Dans toute la suite de l'exercice, le graphe aura ces spécifications !

2. **Le cuisinier est tout seul en cuisine :** la durée minimum est donc la somme des temps nécessaires à chaque tâche. Par contre, il faut l'aider à trouver l'ordre dans lequel il va pouvoir faire les différentes tâches : une solution de tri topologique.
   Une solution de tri topologique peut être trouvée en classant l'ensemble des sommets par ordre décroissant de rencontre en suffixe lors d'un parcours en profondeur.
   Une autre méthode utilise un vecteur contenant les demi-degrés intérieurs de chaque sommet.

   Utiliser obligatoirement cette deuxième méthode pour écrire la fonction qui retourne une solution de tri topologique (une liste de sommets) pour un graphe ayant les mêmes spécifications que ci-dessus.

3. **Le cuisinier a trouvé des aides, les tâches peuvent donc être réalisées en parallèle.**
   (a) La *date au plus tôt* de la tâche i est la date à laquelle la tâche peut commencer au plus tôt.
   Comment calculer les dates au plus tôt de chaque tâche à partir de ce type de graphe ?
   Remplir le tableau donnant les dates au plus tôt pour la recette de la crêpe.

   (b) Quel temps faudra-t'il au minimum pour pouvoir déguster notre crêpe ?

   (c) La *date au plus tard* de la tâche i est la date au plus tard où la tâche peut commencer sans entraîner de retard sur le projet.
   Comment calculer les dates au plus tard de chaque tâche à partir de ce type de graphe ?

   (d) Écrire la fonction qui détermine le temps minimum de réalisation d'un projet représenté par un graphe ayant les mêmes spécifications que ci-dessus.
   Vous devez utiliser la fonction de la question 2.

Exercice 4 (Prim, tout simplement – *5 points*)

Écrire la fonction `prim(`G`)` qui retourne un arbre de recouvrement minimum du graphe connexe G, en utilisant l'algorithme de Prim !

Annexes

Toutes les classes (`Graph`, `Heap`, `Queue`) sont supposées importées. Les graphes ne peuvent pas être vides.

Les graphes

```
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
```

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

Les files

- `Queue()` retourne une nouvelle file
- q`.enqueue(`e`)` enfile s dans q
- q`.dequeue()` retourne le premier élément de q, défilé
- q`.isempty()` teste si q est vide

`List`

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

Autres

- `range`.
- `min`, `max`, `abs` ...
- value `inf` or ∞

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.