Mini-projet : décongestion d'un réseau.

L'objectif de ce projet est de créer et d'implémenter des algorithmes permettant d'améliorer une infrastructure routière, afin de faire face aux problèmes de trafic sans cesse croissants. Des jeux de tests seront également fournis sous la forme de doctests pour vous aider à vérifier vos fonctions ; votre travail sera évalué sur ces tests *et* sur d'autres jeux de tests aléatoires.

L'échéance est fixée au **dimanche 15 mai 2022 à 23h55**.

Avant de commencer, lisez intégralement le sujet et consultez les tests donnés en exemple afin de ne pas partir dans une mauvaise direction ! !

Notions de base

Les réseaux routiers sont modélisés par des graphes orientés, simples, et pondérés. Chaque sommet du graphe représente une intersection entre plusieurs routes, qui sont elles-mêmes représentées par des arcs dont les capacités correspondent au débit que la route peut fournir. Plus précisément, elles correspondent au *taux moyen journalier de trafic* sur chaque route, c'est-à-dire le nombre moyen de véhicules y circulant chaque jour.

Afin de réduire les embouteillages, il convient de répartir les utilisateurs sur le réseau de manière à maximiser le taux moyen journalier de trafic global. Comme on l'a vu au cours, ceci peut s'obtenir en calculant un flot maximum pour le graphe représentant le réseau, qui pourra éventuellement contenir plusieurs sources et plusieurs puits.

Notre but étant d'améliorer la fluidité du réseau routier, nous souhaitons modifier ce réseau pour qu'il soit en mesure de fournir un meilleur débit, ce qui signifie que nous chercherons à augmenter la valeur du flot maximum. Nous nous y prendrons de deux façons différentes :

1. soit en augmentant la capacité d'une ou plusieurs routes (ce qui en pratique pourrait se traduire par exemple par l'ajout d'une nouvelle bande) ;
2. soit en créant de nouvelles routes entre deux points de raccordement existants.

Bien entendu, chacune de ces opérations aura un coût, et il nous faudra veiller à rentabiliser nos actions. Dans la suite, nous manipulerons un réseau de flot G muni d'un flot maximum f déjà calculé. Les notations suivront celles du cours.

Augmentation des capacités

On dira qu'une augmentation de capacité d'un arc (u, v) de $G = (V, A, c)$ est *utile* s'il existe une valeur $k \in \mathbb{N}$ telle qu'une augmentation de k de $c(u, v)$ transforme G en un réseau G' pour lequel il existe un flot maximum f' avec $|f'| > |f|$. La figure 1 montre un exemple d'augmentation de capacité menant à une augmentation de la valeur du flot maximum. Remarquons qu'une augmentation de k en capacité n'implique pas forcément une augmentation de k de la valeur du flot maximum ; et qu'une augmentation de capacité sur d'autres arcs (par exemple (d, t)) n'aurait eu aucun impact sur la valeur du flot.

Le coût d'une augmentation de capacité sera directement proportionnel à la valeur de cette augmentation (une augmentation de capacité de k unités coûte donc k), et l'on cherchera à en maximiser la rentabilité. Dans l'exemple de la figure 1, il aurait suffi d'augmenter la capacité de l'arc (c, t) de 4 pour obtenir une augmentation de flot de 4, ce qui aurait donc constitué une meilleure solution.

Ajout d'arcs

La figure 2 montre un exemple d'ajout d'arc menant à une augmentation de la valeur du flot maximum. Ici encore, un ajout d'un arc de capacité k n'implique pas forcément une augmentation de k de la valeur du flot maximum ; et de même, l'ajout de certains arcs (par exemple (e, a)) n'aurait eu aucun impact

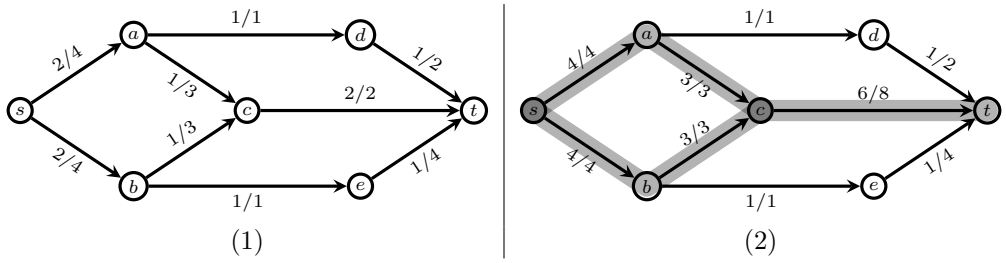

FIGURE 1 – (1) Un réseau muni d'un flot maximum de valeur 4 ; (2) le même réseau muni d'un nouveau flot maximum après augmentation de 6 de la capacité de l'arc (c, t), qui permet d'augmenter le flot de 4 unités. Les arcs surlignés sont ceux sur lesquels le changement a permis d'augmenter le flot y circulant.

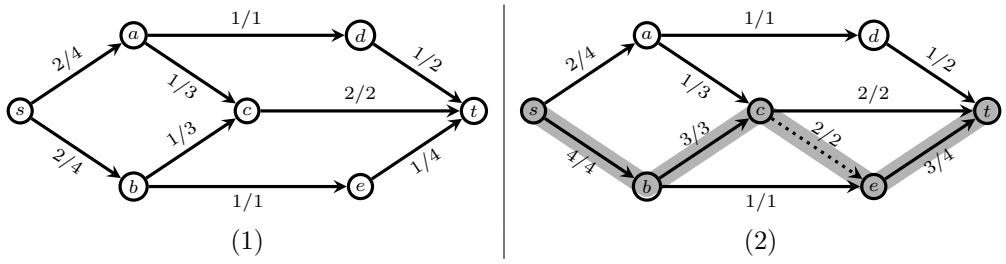

FIGURE 2 – (1) Un réseau muni d'un flot maximum de valeur 4 ; (2) le même réseau muni d'un nouveau flot maximum après l'ajout de l'arc (c, e) de capacité 2, qui permet d'augmenter le flot de 2 unités. Les arcs surlignés sont ceux sur lesquels le changement a permis d'augmenter le flot y circulant.

sur le flot, quelle que soit la capacité choisie. On qualifiera d'*utile* un ajout permettant d'augmenter la valeur du flot maximum.

Le coût de l'ajout d'un arc sera égal à sa longueur. Pour mesurer cette longueur, il nous faudra donc utiliser les coordonnées des points de raccordement que l'on désire relier, qui devront donc être précisées lors de l'ajout des sommets au graphe. Consultez les exemples d'exécutions pour y voir plus clair.

Avant de vous lancer dans l'implémentation, répondez aux questions théoriques ci-dessous, qui vous aideront à répondre efficacement aux questions suivantes.

Questions théoriques

1. Donnez un exemple de graphe muni d'un flot maximum n'admettant aucune augmentation utile sur un seul arc.
2. Généralisez votre exemple à des graphes nécessitant l'augmentation de plus de k arcs ; autrement dit, donnez un exemple de graphe muni d'un flot maximum pour lequel aucune augmentation de capacité de k arcs ne permet d'augmenter le flot donné.
3. Prouvez que si le flot donné est maximum, augmenter la capacité d'un arc non saturé ne peut pas augmenter le flot.

Implémentation

1. Pour chaque fonction et méthode implémentée, écrivez un test supplémentaire au format doctest. Vous rajouterez tous ces tests dans un seul fichier nommé `mes_tests.txt`, que vous rendrez également.
2. Développez une classe nommée `Graphe` qui vous permettra de stocker les propriétés du réseau qu'il vous faudra construire. Il est fortement conseillé de partir de la classe `DictionnaireAdjacence` qui vous a été fournie. Parmi les adaptations à réaliser, il faudra notamment que cette classe

`Graphe` soit capable de gérer les noms et les coordonnées des sommets (quitte à leur donner des valeurs par défaut si ces données ne sont pas fournies).

3. Implémentez les fonctions suivantes (consultez les exemples des doctests pour vous aider à comprendre ce qui est attendu). Les fonctions détecteront elles-mêmes les sources et les puits de votre réseau, en faisant appel aux méthodes `Graphe.sources()` et `Graphe.puits()` que vous implémenterez.

   (a) `flot_maximum(graphe)` : renvoie un couple `(valeur, flot)`, où `valeur` est la valeur du flot maximum calculé sur le graphe et `flot` est flot maximum pour le graphe, représenté par un dictionnaire possédant comme clés les arcs du graphe et comme valeurs le flot circulant sur l'arc correspondant.

   (b) `reseau_residuel(reseau, flot)` : renvoie le réseau résiduel correspondant au réseau et au flot donnés en paramètres. Le flot est un dictionnaire au même format que dans la question (a).

   (c) `coupe_minimum(residuel, sources)` : renvoie une coupe minimum pour le réseau de départ sur base du résiduel et des sources données en paramètres. On supposera que ce résiduel est obtenu à partir d'un graphe pour lequel on a calculé un flot maximum.

   (d) `augmentations_uniques_utiles(graphe, flot_max)` : renvoie l'ensemble des arcs du graphe dont l'augmentation de capacité est utile par rapport au flot maximum précalculé. `flot_max` est un flot maximum au format de la question (a). Votre solution doit être en $O(|V| + |A|)$.

   (e) `augmentations_uniques_utiles_calibrees(graphe, flot_max)` : renvoie un dictionnaire possédant comme clés les arcs du graphe dont l'augmentation est utile et comme valeurs la valeur maximale dont le flot maximum précalculé augmente *via* l'arc correspondant (le paramètre `flot_max` est identique à celui de la question précédente).

   (f) `ensemble_minimum_augmentations_utiles_calibrees(graphe, flot_max, cible)` : cherche un ensemble d'arcs permettant d'augmenter la valeur du flot maximum `flot_max` d'au moins `cible` unités à moindre coût, et renvoie un dictionnaire ayant pour clés les arcs de cet ensemble et pour valeurs les augmentations à réaliser. Attention, le nombre d'arcs de cet ensemble n'est pas fixé au préalable : ce n'est pas ce nombre que l'on veut minimiser, mais bien la somme des augmentations d'arcs nécessaires à l'augmentation voulue du flot.

   (g) `rajouts_uniques_utiles(graphe, flot_max)` : renvoie l'ensemble des arcs dont l'ajout au graphe permet d'augmenter le flot maximum donné en paramètre. Chaque élément de cet ensemble sera représenté par un tuple `(u, v, longueur, augmentation)`, contenant donc les extrémités de l'arc (u, v) suivies de la longueur de la nouvelle route (donnée par la distance euclidienne entre u et v) et l'augmentation maximale du flot maximum que permet le rajout de cette route. On interdit la création d'arcs parallèles.

   (h) `ensemble_optimal_rajouts(graphe, flot_max, budget)` : renvoie un dictionnaire ayant pour clés les nouveaux arcs à ajouter et pour valeurs leurs capacités. La somme des longueurs n'excède pas le budget donné, et doit permettre d'augmenter au maximum la valeur du flot maximum donné en paramètre.

Vous pouvez bien entendu implémenter toutes les fonctions supplémentaires que vous voulez, mais votre rendu devra au moins contenir les fonctions ci-dessus avec la signature demandée et respectant les formats d'entrée et sortie imposés.

Exemples d'exécution

Quelques exemples d'exécution des fonctions demandées sont donnés ci-dessous.

`augmentations_uniques_utiles`

Voici le résultat à obtenir sur l'exemple de la figure 1 :

```
>>> G = Graphe()
>>> for u, v, c in [('s', 'a', 4), ('s', 'b', 4), ('a', 'd', 1), ('a', 'c', 3), ('b',
↪  'c', 3), ('b', 'e', 1), ('c', 't', 2), ('d', 't', 2), ('e', 't', 4)]:
...     G.ajouter_arc(u, v, capacite=c)
>>> flot = {('s', 'a'): 2, ('s', 'b'): 2, ('a', 'd'): 1, ('a', 'c'): 1, ('b', 'c'): 1,
↪  ('b', 'e'): 1, ('c', 't'): 2, ('d', 't'): 1, ('e', 't'): 1}
>>> sorted(augmentations_uniques_utiles(G, flot))
[('a', 'd'), ('b', 'e'), ('c', 't')]
```

augmentations_uniques_utiles_calibrees

Voici le résultat à obtenir sur l'exemple de la figure 1 :

```
>>> G = Graphe()
>>> for u, v, c in [('s', 'a', 4), ('s', 'b', 4), ('a', 'd', 1), ('a', 'c', 3), ('b',
↪  'c', 3), ('b', 'e', 1), ('c', 't', 2), ('d', 't', 2), ('e', 't', 4)]:
...     G.ajouter_arc(u, v, capacite=c)
>>> flot = {('s', 'a'): 2, ('s', 'b'): 2, ('a', 'd'): 1, ('a', 'c'): 1, ('b', 'c'): 1,
↪  ('b', 'e'): 1, ('c', 't'): 2, ('d', 't'): 1, ('e', 't'): 1}
>>> sorted(augmentations_uniques_utiles_calibrees(G, flot).items())
[(('a', 'd'), 1), (('b', 'e'), 3), (('c', 't'), 4)]
```

ensemble_minimum_augmentations_utiles_calibrees

Voici un exemple de résultat que l'on peut obtenir sur l'exemple de la figure 1 (vos arcs et augmentations peuvent être différents, mais le coût de la solution doit être le même) :

```
>>> G = Graphe()
>>> for u, v, c in [('s', 'a', 4), ('s', 'b', 4), ('a', 'd', 1), ('a', 'c', 3), ('b',
↪  'c', 3), ('b', 'e', 1), ('c', 't', 2), ('d', 't', 2), ('e', 't', 4)]:
...     G.ajouter_arc(u, v, capacite=c)
>>> flot = {('s', 'a'): 2, ('s', 'b'): 2, ('a', 'd'): 1, ('a', 'c'): 1, ('b', 'c'): 1,
↪  ('b', 'e'): 1, ('c', 't'): 2, ('d', 't'): 1, ('e', 't'): 1}
>>> sorted(ensemble_minimum_augmentations_utiles_calibrees(G, flot, 1).items())
[(('a', 'd'), 2)]
>>> sorted(ensemble_minimum_augmentations_utiles_calibrees(G, flot, 3).items())
[(('b', 'e'), 4)]
>>> sorted(ensemble_minimum_augmentations_utiles_calibrees(G, flot, 5).items())
[(('a', 'd'), 2), (('c', 't'), 6), (('s', 'a'), 5)]
>>> sorted(ensemble_minimum_augmentations_utiles_calibrees(G, flot, 8).items())
[(('a', 'd'), 2), (('b', 'e'), 4), (('c', 't'), 6), (('s', 'a'), 5), (('s', 'b'), 7)]
```

Le coût d'une solution est la somme des augmentations de capacité ; ainsi, le coût de la solution …

1. est $2 - 1 = 1$;
2. est $4 - 1 = 3$;
3. est $2 - 1 + 6 - 2 + 5 - 4 = 6$;
4. est $2 - 1 + 4 - 1 + 6 - 2 + 5 - 4 + 7 - 4 = 12$.

rajouts_uniques_utiles

Voici le résultat à obtenir sur l'exemple de la figure 2 :

```
>>> G = Graphe()
>>> for sommet, coordonnees in [("s", (0, 0)), ("a", (1.5, 1)), ("b", (1.5, -1)),
↪  ("c", (3, 0)), ("d", (4.5, 1)), ("e", (4.5, -1)), ("t", (6, 0))]:
...     G.ajouter_sommet(sommet, coords=coordonnees)
>>> for u, v, c in [('s', 'a', 4), ('s', 'b', 4), ('a', 'd', 1), ('a', 'c', 3), ('b',
↪  'c', 3), ('b', 'e', 1), ('c', 't', 2), ('d', 't', 2), ('e', 't', 4)]:
...     G.ajouter_arc(u, v, capacite=c)
>>> flot = {('s', 'a'): 2, ('s', 'b'): 2, ('a', 'd'): 1, ('a', 'c'): 1, ('b', 'c'): 1,
↪  ('b', 'e'): 1, ('c', 't'): 2, ('d', 't'): 1, ('e', 't'): 1}
>>> sorted(rajouts_uniques_utiles(G, flot))
[('a', 'e', 3.605551275463989, 3), ('a', 't', 4.6097722286464435, 3), ('b', 'd',
↪  3.605551275463989, 1), ('b', 't', 4.6097722286464435, 3), ('c', 'd',
↪  1.8027756377319946, 1), ('c', 'e', 1.8027756377319946, 3), ('s', 'd',
↪  4.6097722286464435, 1), ('s', 'e', 4.6097722286464435, 3), ('s', 't', 6.0,
↪  9223372036854775803)]
```

`ensemble_optimal_rajouts`

Voici un exemple de résultat que l'on peut obtenir sur le réseau de la figure 2 (vos arcs et augmentations peuvent être différents, mais la qualité de la solution doit être la même) :

```
>>> G = Graphe()
>>> for sommet, coordonnees in [("s", (0, 0)), ("a", (1.5, 1)), ("b", (1.5, -1)),
↪  ("c", (3, 0)), ("d", (4.5, 1)), ("e", (4.5, -1)), ("t", (6, 0))]:
...     G.ajouter_sommet(sommet, coords=coordonnees)
>>> for u, v, c in [('s', 'a', 4), ('s', 'b', 4), ('a', 'd', 1), ('a', 'c', 3), ('b',
↪  'c', 3), ('b', 'e', 1), ('c', 't', 2), ('d', 't', 2), ('e', 't', 4)]:
...     G.ajouter_arc(u, v, capacite=c)
>>> flot = {('s', 'a'): 2, ('s', 'b'): 2, ('a', 'd'): 1, ('a', 'c'): 1, ('b', 'c'): 1,
↪  ('b', 'e'): 1, ('c', 't'): 2, ('d', 't'): 1, ('e', 't'): 1}
>>> sorted(ensemble_optimal_rajouts(G, flot, 2).items())
[(('c', 'e'), 3)]
>>> sorted(ensemble_optimal_rajouts(G, flot, 4).items())
[(('c', 'd'), 1), (('c', 'e'), 3)]
```

Dans le premier cas, la distance entre les points `'c'` et `'e'` est de $1,5 < 2$ qui est le budget donné, et une capacité de 3 sur ce nouvel arc permet d'obtenir un nouveau flot maximum de valeur 7. Dans le second cas, la distance entre les points `'c'` et `'e'` est de 1, donc le budget utilisé est de $2,5 < 4$, et les capacités attribuées aux nouveaux arcs permettent d'obtenir un flot maximum de valeur 8.

Modalités

Votre projet est à réaliser en binôme. Votre rendu de projet sera constitué des fichiers suivants (noms imposés) :

- `graphe.py` - contiendra votre classe `Graphe` adaptée aux besoins du projet.
- `decongestion.py` - le programme principal qui devra contenir toutes les fonctions listées dans le sujet.
- le fichier `mes_tests.txt` contenant les tests demandés plus haut ;
- un rapport dans lequel vous donnerez vos réponses aux questions théoriques ainsi que vos algorithmes en pseudocode, en justifiant vos choix ainsi que leur complexité.

Vous avez le droit d'utiliser le module standard `itertools` (dont la documentation est disponible ici).

Compléments d'information et rappels sur Python

Paramètres avec valeurs par défaut. Python permet de préciser des paramètres prenant une certaine valeur par défaut si l'utilisateur ne précise rien. On consultera la documentation de Python pour plus d'informations à ce sujet, mais il est important de retenir au moins les deux règles suivantes :

1. ils doivent être placés après tous les autres paramètres obligatoires de la fonction ;
2. ils ne peuvent pas recevoir des valeurs par défaut variables ou non encore définies ; par exemple, `def ma_fonction(ma_liste, n=len(ma_liste))` est interdit.

Doctests. Les doctests ont été couverts dans des cours précédents, et la documentation correspondante est disponible ici. On peut placer ces doctests dans de simples fichiers texte séparés (`tests.txt`) au lieu de les mettre dans les docstrings des fonctions, et les lancer avec `python3 -m doctest tests.txt`. Si rien ne s'affiche, c'est que tous les tests ont réussi.

N'oubliez pas qu'on ne peut pas prédire l'ordre dans lequel le contenu des itérables non ordonnés sera affiché : il est donc possible que le contenu de votre résultat soit correct mais que votre doctest échoue. Une manière de contourner ce problème consiste à utiliser `sorted`, afin d'imposer un ordre d'affichage (voir les exemples de la section "Exemples d'exécution").