CRC

COMPÉTITION DE PROGRAMMATION

PROBLÈME PRÉLIMINAIRE 4

QUELQUES NOTES

- Les règles complètes sont dans la section 4 du livret des règlements.
- Vous avez jusqu'au **dimanche 26 janvier, 23h59** pour remettre votre code.
- N'hésitez pas à utiliser le forum de programmation sur le discord de la CRC pour poser vos questions et discuter des problèmes. Il est là pour ça!
- **On vous donne des fichiers modèles faciles à utiliser pour votre code et pour faire vos tests. Vous devez les utiliser pour résoudre le problème!**

UTILISATION DU FICHIER MODÈLE

- Le fichier de test appelle la fonction associée avec en paramètre les informations du test et compare sa sortie avec ce qui est attendu pour vous permettre de voir si les tests réussissent. **Tout votre code (sauf fonctions additionnelles que vous créez) devrait être écrit dans la fonction prévue à cet effet.**
- Les points mis dans le document indiquent la difficulté et le pointage attribué pour la réussite pour chaque défi. Ce problème préliminaire aura une valeur globale de 4% du défi principal.

STRUCTURE

Une petite mise en situation comme celle-ci explique les fondements de chaque défi et offre les bases nécessaires pour résoudre celui-ci.

Spécification d'entrée et de sortie:

Contient les caractéristiques des entrées fournies ainsi que les critères attendus pour les sorties du programme.

Exemple d'entrée et de sortie:

Contient un exemple d'entrée, parfois constitué lui-même de plusieurs sous-exemples, pour que vous puissiez tester votre programme. Chaque exemple de sortie donne la réponse attendue pour l'entrée correspondante.

Explication de la première sortie:

Décortique davantage le défi en expliquant comment la première entrée est traitée et en montrant le chemin menant à cette réponse.

Il faut réduire les risques de feu!

Quand des villes sont créées, il faut s'assurer qu'il y ait des services essentiels pour tout couvrir. Naturellement, il faut que les services d'urgence soient en assez grande quantité et aussi bien placés. Par contre, on ne peut pas juste déplacer une ville! Alors on doit choisir comment on assigne nos services aux villes voisines!

Dans ce problème, vous ferez l'assignation de villes à des casernes de pompiers pour vous assurer de minimiser le score de risque!

Partie 1: Calculer la distance (5 points)

Pour vous assurer de minimiser les risques, il faut que la caserne de pompiers soit proche de la ville à desservir. Vous devrez calculer la distance entre la ville avec la station de pompier ainsi que la ville choisie. Comme un camion de pompier doit emprunter des routes et non nécessairement passer par le chemin le plus court vous calculerez la distance entre deux points en utilisant la distance de manhattan. (https://fr.wikipedia.org/wiki/Distance_de_Manhattan)

Spécification d'entrée et de sortie:

En entrée vous recevrez:

- **city_a:** un *tuple* contenant 2 *int* qui sont les coordonnées en x et en y de la ville a
- **city_b:** un *tuple* contenant 2 *int* qui sont les coordonnées en x et en y de la ville b

En sortie vous devrez fournir:

- une variable *int* de la distance de manhattan entre les deux villes

Exemple d'entrée:

```
[0, 0], [1, 5]
[20, 18], [14, 15]
[2, 3], [4, 5]
```

Exemple de sortie:

```
6
9
4
```

N.B. Il n'y aura pas d'explications comme vous avez 3 semaines pour ce problème! Il y a des tests pytest pour les 3 premières parties.

Partie 2: Où sont les casernes? (5 points)

Vous recevrez le problème complet sous forme d'une carte en 2D et donc ce sera à vous d'aller chercher la position des stations. Vous devrez donc trouver les coordonnées des casernes dans la carte qui vous est donnée. Les casernes ont la **valeur 1** dans la carte. Vous devrez les renvoyer triées en ordre croissant de coordonnées selon les **x** et en cas d'égalité, triées en ordre croissant aussi selon les **y**.

Pour mieux comprendre visuellement la carte et les stations vous pouvez aller dans le fichier **benchmark.py** et changer la valeur à la **ligne 15** pour la grosseur de la carte (10-25 sont des bonnes valeurs pour visualiser) Les points rouges sont les casernes et les points bleus sont les villes à répartir. Quand vous fermez la visualisation des villes, votre algorithme est appelé pour résoudre la situation.

Spécification d'entrée et de sortie:

En entrée vous recevrez:

- map: un *array* de *array* de *int* qui représente la carte en 2D
- n: un *int* de la grandeur d'un côté de la carte

En sortie vous devrez donner un *array* de *tuple* de 2 *int* qui sont les coordonnées en **x** et en **y** des casernes de pompier.

Exemple d'entrée:

```
[[0, 0, 2, 0, 0],
[0, 0, 0, 0, 0],
[0, 1, 0, 0, 0],
[0, 0, 2, 2, 0],
[0, 2, 0, 2, 0]]

5
```

Exemple de sortie:

```
[(2, 1)]
```

N.B. Il n'y aura pas d'explications comme vous avez 3 semaines pour ce problème! Il y a des tests pytest pour les 3 premières parties.

Partie 3: Où sont les villes? (5 points)

Vous recevrez le problème complet sous forme d'une carte en 2D et donc ce sera à vous d'aller chercher la position des villes. Vous devrez donc trouver les coordonnées des villes dans la carte qui vous est donnée. Les villes ont la **valeur 2** dans la carte. Vous devrez les renvoyer triées en ordre croissant de coordonnées selon les **x** et en cas d'égalité, triées en ordre croissant aussi selon les **y**.

Pour mieux comprendre visuellement la carte et les stations vous pouvez aller dans le fichier **benchmark.py** et changer la valeur à la **ligne 15** pour la grosseur de la carte (10-25 sont des bonnes valeurs pour visualiser) Les points rouges sont les casernes et les points bleus sont les villes à répartir. Quand vous fermez la visualisation des villes, votre algorithme est appelé pour résoudre la situation.

Spécification d'entrée et de sortie:

En entrée vous recevrez:

- map: un *array* de *array* de *int* qui représente la carte en 2D
- n: un *int* de la grandeur d'un côté de la carte

En sortie vous devrez donner un *array* de *tuple* de 2 *int* qui sont les coordonnées en x et en y des villes.

Exemple d'entrée:

```
[[0, 0, 2, 0, 0],
[0, 0, 0, 0, 0],
[0, 1, 0, 0, 0],
[0, 0, 2, 2, 0],
[0, 2, 0, 2, 0]]

5
```

Exemple de sortie:

```
[(0, 2), (3, 2), (3, 3), (4, 1), (4, 3)]
```

N.B. Il n'y aura pas d'explications comme vous avez 3 semaines pour ce problème! Il y a des tests pytest pour les 3 premières parties.

Partie 4: Répartition des villes (20 points)

Maintenant que vous savez où sont chacune des villes et des casernes, vous pouvez maintenant procéder à l'assignation des villes aux casernes! Voici une série de règles importantes à suivre pour minimiser votre score de risque.

La première règle est essentielle et doit être respectée pour avoir une solution valide. **Toutes les villes doivent être associées à exactement une caserne**. Vous ne pouvez pas avoir de villes qui ne sont pas desservies, mais vous pouvez avoir des casernes qui ne desservent aucune ville, n'oubliez pas de mettre un *array* vide pour une caserne qui ne sert personne.

La seconde règle est de garder le nombre de villes par casernes le plus près possible de 5. Dans un exemple avec un n qui vaut 20, il y aura 20 villes et 4 casernes. Dans tous les exemples pour valider vos solutions il y aura exactement 5 villes à desservir en moyenne par caserne. Vous pouvez assigner un nombre différent de 5 villes pour une caserne, mais le risque augmente exponentiellement quand vous déviez de ce nombre. Une caserne avec 3 villes de trop à s'occuper aura un score de risque associé à son nombre de villes de $3^2 = 9$. Le score de risque monte doucement au début quand le nombre de villes est inexact, mais peut grimper rapidement si une caserne doit s'occuper de trop de villes!

La troisième règle du calcul de score de risque est de limiter la distance entre la caserne et ses villes qu'elle couvre. Une caserne doit être près de la ville en cas de feu pour réagir rapidement. Vous aurez donc un score de risque plus élevé plus vos villes sont loin de leur caserne assignée.

Vous devrez donc minimiser le score de risque de la répartition des casernes pour différentes grosseurs de carte. Les différentes grosseurs sont: [10, 25, 50, 100, 250, 500, 1000, 2500, 5000, 10000]. Vous obtiendrez **1 point** pour chaque grandeur de carte pour laquelle vous fournissez une solution valide en **moins d'une minute de temps** de calcul. Vous serez par la suite comparés aux autres équipes et l'équipe avec le meilleur score aura **1 point supplémentaire** par grosseur de problème. Pour toutes les autres équipes, elles auront une portion de ce point dépendant à quel point elles sont proches de la meilleure solution en descendant proportionnellement jusqu'à 0 points pour l'équipe avec le score de risque le plus élevé.

Vous aurez donc directement 10 des 20 points si vous donnez une solution valide pour chacune des grosseurs de n.

Spécification d'entrée et de sortie:

En entrée vous recevrez:

- map: un *array* de *array* de *int* qui représente la carte en 2D
- n: un *int* de la grandeur d'un côté de la carte

En sortie vous devrez donner un *array* de *array* de *tuple* de 2 *int* qui sont les coordonnées en x et en y des villes pour chacune des casernes de pompier.

Exemple d'entrée:

```
[[0, 0, 0, 0, 2, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 2, 0, 0, 0, 0, 0, 2, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 2, 0, 2, 0, 0, 0],
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0],
[0, 0, 2, 0, 0, 0, 0, 2, 0, 0],
[0, 0, 1, 2, 0, 0, 0, 0, 0, 0]]

10
```

Exemple de sortie:

```
[[(0, 4), (1, 9), (4, 2)],
[(4, 8), (6, 4), (6, 6), (7, 5), (8, 2), (8, 7), (9, 3)]]
```

Explication de la première sortie:

On a pris les 3 premières villes et on les a assignées à la caserne 0 et les 7 autres ont été assignées à la station 1. Vous pouvez trouver de meilleurs algorithmes, bonne chance!!

score de risque de répartition: 8
score de risque de distance: 5.5

score de risque total: 13.5