Numérique et sciences informatiques
Baccalauréat 2024
Épreuves écrites
Asie
Journée 1
Correction

Exercice 1

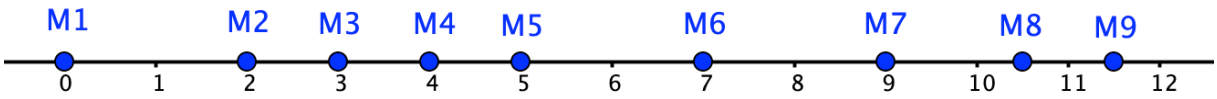
1. On écrit les instructions suivantes :

```
m1 = Maison(1)
m2 = Maison(3.5)
```

2. On écrit l'instruction suivante :

```
a = Antenne(2.5, 1)
m1 = Maison(1)
```

3. On obtient le schéma suivant :



4. On complète le code de la façon suivante :

```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(Maison(p))
    return maisons
```

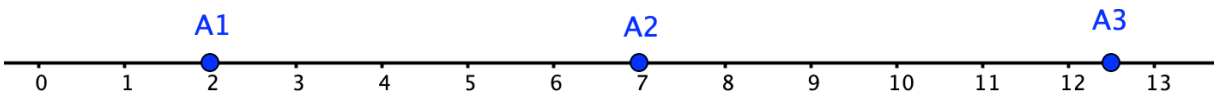
5. On peut complèter le code de la façon suivante :

```
def couvre(self, maison):
    pos_m = maison.get_pos_maison()
    pos_a = self.get_pos_antenne()
    ray = self.get_rayon()
    return abs(pos_m - pos_a) <= ray
```

6. Ces instructions engendrent l'affichage suivant :

```
[0, 3, 7, 10.5]
```

7. On obtient le schéma suivant :



8. En s'inspirant du code de `stratégie_1`, on peut écrire le code suivant :

```
def strategie_2(maisons, rayon):
    pos_antenne = maisons[0].get_pos_maison()+rayon
    antennes = [Antenne(pos_antenne, rayon)]
    for m in maisons[1:]:
        if not antennes[-1].couvre(m):
```

```
                pos_antenne = m.get_pos_maison()+rayon
                antennes.append(Antenne(pos_antenne, rayon))
    return antennes
```

9. Dans les deux cas, le coût est proportionnel au nombre n de maisons. Il s'agit donc d'une complexité linéaire (en $O(n)$).

Exercice 2

1. Il s'agit d'un graphe orienté.

2. On peut réaliser la tâche (f) puis la tâche (g) mais on ne peut pas réaliser la tâche (g) puis la tâche (f).
On peut réaliser la tâche (i) puis la tâche (j) mais on peut aussi réaliser la tâche (j) puis la tâche (i).

3. Il faut réaliser les tâches suivantes avant de réaliser la tâche (k) :

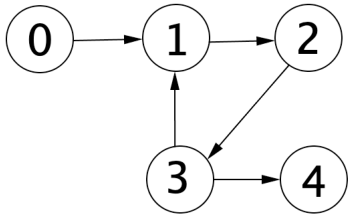
a, h, i, c, j

4. Le graphe ne contient pas de cycle.

5. Un ordre possible est :

0, 2, 1, 3, 5, 4

6. On obtient le graphe suivant :



7. Il est impossible de trouver un ordre permettant de réaliser les tâches car pour réaliser la tâche 3 il faut auparavant avoir réalisé la tâche 1 mais pour réaliser la tâche 1 il faut avoir réalisé la tâche 3. L'impossibilité provient donc du fait que le graphe comporte un cycle.

8. On a le tableau suivant des appels successifs suivant :

Appel `mystere`	Variable `ouverts`	Variable `fermes`
Avant l'appel `mystere`	`[F,F,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,1,5,[F,F,F,F,F], [F,F,F,F,F],None)`	`[F,T,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,2,5,[F,T,F,F,F], [F,F,F,F,F],None)`	`[F,T,T,F,F]`	`[F,F,F,F,F]`
`mystere(M,3,5,[F,T,T,F,F], [F,F,F,F,F],None)`	`[F,T,T,T,F]`	`[F,F,F,F,F]`
`mystere(M,1,5,[F,T,T,T,F], [F,F,F,F,F],None)`	`[F,T,T,T,F]`	`[F,F,F,F,F]`

Le dernier appel conduira au cas de base car on aura alors `ouverts[1]` à `True` et cet appel va donc renvoyer `False`. Lors de la remontée des appels récursifs, la variable `var` recevra à chaque fois la valeur `False`. La variable `ok` recevra donc finalement la valeur `False`.

9. La fonction `mystere` renvoie `False` lorsque le graphe comporte un cycle.

10. On empile successivement 3, 2 et 10 puis on dépile deux fois. Après exécution des instructions, la variable `elt` contient le nombre dépilé la deuxième fois donc 2.

11. En ligne 24 de la fonction `mystere` il faut écrire l'instruction suivante :

```
    resultat.empiler(s)
```