2024 asie jour 1

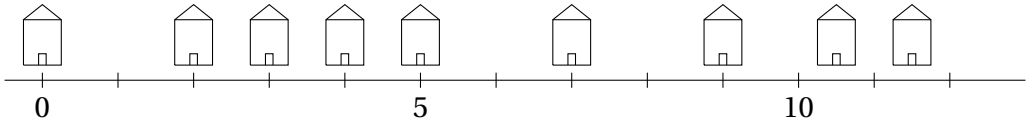
Exercice 1

1.

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

2. `a = Antenne(2.5, 1)`

3.



4.

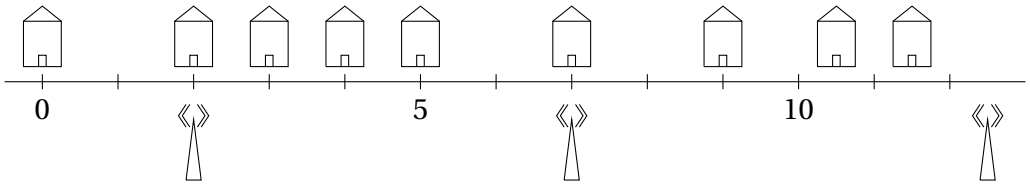
```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(m)
    return maisons
```

5.

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

6. `[0, 3, 7, 10.5]`

7.



8.

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

9. Le coût asymptotique est le même pour les deux stratégies. Le nombre d'opérations est linéaire avec n.

Exercice 2

1. C'est un graphe orienté.

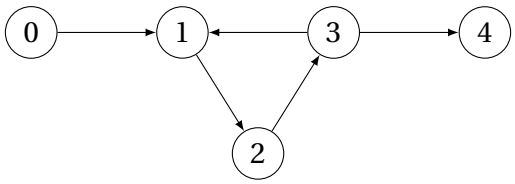
2.
- Oui
- Non
- Oui
- Oui

3. a, c, h, i et j.

4. Il ne contient pas de cycle.

5. Par exemple : 0, 2, 1, 3, 5, 4

6.



7. Non, ça n'est pas possible car il y a un cycle.

8.

Appel `mystere`	var `ouverts`	var `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]`

Comme `ouverts[1] == False` alors on renvoie `False`.

9. Elle renvoie False s'il y a un cycle.

10. 2

11.

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

Exercice 3

Partie A

1. `personneA = Personne(112, "Lesieur", "Isabelle", 1982, 2005);`

2. `personneA.num_badge`

3.

```
def annee_anciennete(self):
    return 2024 - self.annee_entree
```

4.

```
def ajouter(self, p):
    self.liste.append(p)
```

5.

```
def effectif(self):
    return len(self.liste)
```

6.

```
def donne_nom(self, num):
    for elt in self.liste:
        if elt.num_badge == num:
            return elt.nom
    return None
```

7.

```
def nb_personne_honneur(self, annee):
    nb = 0
    for elt in self.liste:
        if annee - elt.annee_entree == 10:
            nb += 1
    return nb
```

8.
[Cette méthode fonctionnera jusqu'en l'an 10000]

```
def plus_anciens(self):
    annee_entree = 10000
    anciens = []
    for elt in self.liste:
        if elt.annee_entree == annee_entree:
            anciens.append(elt.num_badge)
        elif elt.annee_entree < annee_entree:
            annee_entree = elt.annee_entree
            anciens = [elt.num_badge]
    return anciens
```

Partie B

9. La requête renvoie les nom et prénom des personnels travaillant au centre 2.

10.

```
UPDATE Personnel
SET num_centre = 3
WHERE num_badge = 135;
```

11. Cela évite une redondance des informations. En mettant les informations sur le centre dans la table `Personnel` il faudrait écrire plusieurs fois les informations sur le centre 1. Cela pourrait créer des erreurs et cela alourdi les mises à jour.

12. L'attribut `num_centre` de la table `Personnel` est une clé étrangère et fait donc référence à la clé primaire `num` de la table `centre`.

13.

```
SELECT nom
FROM Personnel
JOIN Centre
ON num_centre = num
WHERE Centre.ville = 'Lille'
AND annee_debut >= 2015 AND annee_debut <= 2022;
```

14. S'il y a encore des personnes affectées au centre de Normandie, il y a la clé étrangère `num_centre` qui pointe vers la clé primaire `num` avec la valeur 1. Si la ligne du centre de Normandie et supprimée cette clé étrangère pointera dans le vide et cela produira une erreur.
De plus la syntaxe n'est pas correcte car on ne doit pas mettre d'étoile dans une requête `DELETE`.