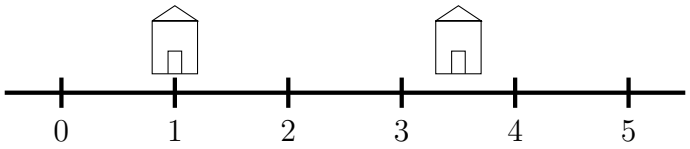
Devoir surveillé n°2 – NSI
15/10/2024

Exercice 1

Une entreprise doit placer des antennes relais le long d'une rue rectiligne. Une antenne relais de portée (ou rayon) `p` couvre toutes les maisons qui sont à une distance inférieure ou égale à `p` de l'antenne. Connaissant les positions des maisons dans la rue, l'objectif est de placer les antennes le long de la rue, pour que toutes les maisons soient couvertes, tout en en minimisant le nombre d'antennes utilisées. La rue est représentée par un axe, et les maisons sont représentées des points sur cet axe :



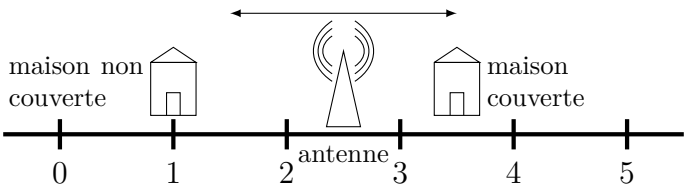
Deux maisons sur une rue, repérées par leur abscisse : 1 et 3,5

Les entités manipulées sont modélisées en utilisant la programmation orientée objet.

```
class Maison:
    def __init__(self, position):
        self._position = position
    def get_pos_maison(self):
        return self._position
class Antenne:
    def __init__(self, position, rayon):
        self._position = position
        self._rayon = rayon
    def get_pos_antenne(self):
        return self._position
    def get_rayon(self):
        return self._rayon
```

1. Donner le code qui crée et initialise deux variables `m1` et `m2` avec des instances de la classe `Maison` situées aux abscisses 1 et 3,5 (voir figure plus haut).

   On ajoute à présent une antenne ayant un rayon d'action de 1 à la position 2,5 :



L'antenne placée en 2,5 et de rayon d'action 1
couvre la maison en 3,5 mais pas celle en 1

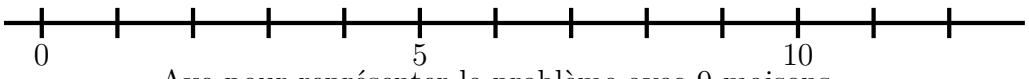
2. Donner le code qui crée la variable `a` correspondant à l'antenne à la position 2,5 avec le rayon d'action 1.

   On souhaite modéliser une rue par une liste d'objets de type `Maison`. Cette liste sera construite à partir d'une autre liste contenant des nombres correspondant aux positions des maisons. La fonction `creation_rue` réalise ce travail. Elle prend en paramètre une liste de positions et renvoie une liste d'objets de type `Maison`.

3. Recopier le schéma ci-dessous et le compléter pour donner une représentation graphique de la situation créée par :

```
creation_rue([0, 2, 3, 4, 5, 7, 9, 10.5, 11.5])
```

On pourra utiliser une simple lettre `m` au lieu de représenter une maison.



Axe pour représenter le problème avec 9 maisons

4. Compléter le code donné ci-dessous de la fonction `creation_rue`.

```
def creation_rue(pos):
    pos.sort() # rappel : la commande tab.sort() trie la liste tab
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(...)
    return ...
```

La méthode `couvre` de la classe `Antenne` prend en paramètre un objet de type `Maison` et indique par un booléen si l'antenne couvre la maison en question ou non. La méthode peut être utilisée ainsi (en supposant que les objets précédents `m1`, `m2` et `a` existent).

```
>>> a.couvre(m1)
False
>>> a.couvre(m2)
True
```

5. Compléter la fonction `couvre`, ci-dessous, en veillant à ne pas accéder directement aux attributs d'une maison depuis la classe `Antenne` (on pourra utiliser les méthodes `get_pos_maison` , `get_pos_antenne` et `get_rayon`).
Pour rappel : la fonction valeur absolue se nomme `abs()` en Python.

```
# Méthode à ajouter dans la classe Antenne
def couvre(self, maison):
    # Code à compléter (éventuellement plusieurs lignes)
     ...
```

La fonction `strategie_1` est donnée ci-dessous. L'objectif est de placer des antennes dans une rue. Elle est fournie à la société qui place les antennes. La fonction prend en paramètre une liste d'objets de type `Maison` (qu'on supposera triée par abscisse croissante) et le rayon d'action des antennes (`float`). Cette fonction renvoie une liste d'objets de type `Antenne` ayant ce rayon d'action et couvrant toutes les maisons de la rue.

```
def strategie_1(maisons, rayon):
     ''' Prend en paramètre une liste de maisons et le rayon
         d'action des antennes et renvoie une liste d'antennes
    '''
    antennes = [Antenne(maisons[0].get_pos_maison(), rayon)]
    for m in maisons[1:]:
        if not antennes[-1].couvre(m):
            antennes.append(Antenne(m.get_pos_maison(), rayon))
    return antennes
```

Pour rappel :

- `tab[1:]` correspond aux éléments de `tab` à partir de l'indice `1` jusqu'à la fin de la liste ;
- `tab[-1]` correspond au dernier élément de la liste `tab`.

6. Indiquer ce qui est affiché après l'exécution des instructions suivantes :

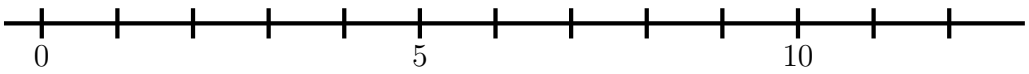
```
>>> maisons = creation_rue([0, 2, 3, 4, 5, 7, 9, 10.5, 11.5])
>>> antennes = strategie_1(maisons, 2)
>>> print([a.get_pos_antenne() for a in antennes])
```

Une amélioration est possible et la société qui pose les antennes souhaite implémenter l'algorithme suivant :

- considérer les maisons dans l'ordre des abscisses croissantes ;
- dès qu'une maison n'est pas couverte, placer une antenne à la plus grande abscisse telle qu'elle couvre cette maison. Par exemple, si la maison d'abscisse 5 est la première maison non couverte, alors, on placera l'antenne en $5 + r$ si r est le rayon d'action de l'antenne.

7. On considère la rue composée des maisons situées aux abscisses `[0, 2, 3, 4, 5, 7, 9, 10.5, 11.5]`. Recopier et compléter le schéma ci-dessous en indiquant l'emplacement des antennes selon cette nouvelle stratégie. On suppose que le rayon d'action est toujours 2.

   On pourra écrire une lettre `a` au lieu de dessiner une antenne.



Axe pour représenter le résultat de la nouvelle stratégie

8. Cet algorithme étant *a priori* plus économe en antennes, proposer une fonction `strategie_2`, sur le modèle de `strategie_1` qui implémente cette nouvelle stratégie.
9. Déterminer et comparer le coût en nombre d'opérations des deux stratégies en fonction du nombre `n` de maisons dans la rue. On admet que le coût de la fonction `append` est constant.