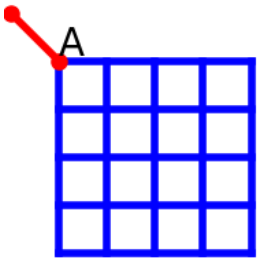
ENSAE TD noté, mercredi 6 novembre 2024

L'accès à internet est autorisé mais l'utilisation d'intelligences artificielle génératives de type ChatGPT est interdite.

Toutes les questions valent 2 points.



25 maisons sont positionnées aux 25 intersections du quadrillage ci-dessus. Le courant (rouge) arrive à un angle du carré (point A). Il faut relier chaque maison au courant pour un coût minimal. Pour cela il faut tirer un câble entre le point A et chaque intersection. Les cibles ne peuvent passer que par les routes (les lignes du quadrillage), pas de diagonales donc.
Les assertions du sujets (exprimées avec le mot clé `assert`*) doivent être satisfaites par vos solutions.*

Question 1 : Implémenter une fonction qui calcule la distance L1.
La distance L1 est définie par $d(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$.

```python
def distance(x1, y1, x2, y2):
    # ...
    return ...

assert distance(0, 0, 3, 4) == 7
```

Question 2 : Calculer la longueur de câble pour relier les 25 maisons.

```python
def longueur_cable(n=5):
    # ...

assert longueur_cable(5) == 100
```

On suppose dans la suite de l'énoncé que le quadrillage reste carré.

Question 3 : Adapter la fonction pour un rectangle 8x9.

```python
assert longueur_cable(8, 9) == 540
```

Question 4 : Avec deux câbles...
On dispose de deux câbles :

1. un câble ne pouvant relier qu'une maison avec un coût c_1 par mètre (un quadrillage $n \times m$ est de dimension n mètres par m mètres)
2. un câble ne pouvant relier qu'une ou deux maisons avec un coût c_2 par mètre

Par conséquent, on peut relier une maison *M1* avec un câble c_2 puis relier *M1* à une autre maison *M2* avec un câble c_1. On veut savoir quand utiliser tel ou tel câble pour minimiser les coûts. Ecrire une fonction qui retourne le coût du câblage décrit ci-dessus.

```
def cout_cablage(x1,y1, x2,y2, c1, c2):
    # ...

assert cout_cablage(1,2, 2,4, 1, 1.5) == 7.5
```

Question 5 : Que fait le code suivant et que montre-t-il ?

```
def position_m1(n, c1, c2):
    x = []
    y = []
    for i in range(2*n):
        x.append(i)
        y.append(cout_cablage(0,i, 0,n, c1, c2))
    return x, y

x, y = position_m1(5, 1, 1.5)
print(x)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(y)  # [5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 10.0, 12.5, 15.0, 17.5]
```

Question 6 : Idée d'algorithme

1. On prend une maison non câblée M1 la plus proche de A.
2. On prend ensuite une maison non câblée M2 la plus proche du coin opposé
3. On continue jusqu'à la fin.

On crée une matrice M, $n \times n$ telle que $M[i][j]$ représente le numéro du câble (1 pour c_1 et 2 pour c_2) qui relie la maison à l'intersection i, j (sans information sur l'origine du cable). La maison 0, 0 est initialement reliée avec le câble 0, les autres valeurs sont initialisées à -1.
Écrire une fonction qui initialise la matrice.

```
def init(n=5):
    # ...
```

Question 7 : Ecrire une fonction qui retourne la maison la plus proche d'une position (i, j) et non câblée.

```
def maison_proche(M, i, j):
    # ...

M = init()
assert maison_proche(M, 0,0) == (0, 1)
M[0][1] = 1
M[1][0] = 2
M[1][1] = 3
assert maison_proche(M,0,1) == (0, 2)
```

Question 8 : On veut tirer un nouveau câble comme suit :

1. on prend la maison m_1 la plus proche de A non cablée, on la cable avec le câble c_2
2. on met à jour la matrice M
3. on prend une maison proche du coin opposé (non câblée), qu'on cable avec c_1 depuis m_1
4. on met à jour la matrice M
5. on renvoie le coût de ces deux câbles ainsi que les deux maisons choisies

Écrire une fonction qui tire un nouveau câble de cette manière :

```
def nouveau_cable(M, c1, c2):
    # ...

M = init()
g = nouveau_cable(M, 1, 1.5)
assert g == ((0, 1), (4, 4), 8.5)
```

Question 9 : Terminer l'algorithme.
Il suffit d'une boucle pour câbler toutes les maisons. La trouverez-vous ?

```
def algorithme_cablage(n, c1, c2):
    M = init(n)
    n_cables = n * n // 2
    cables = []
    # for ...
    # ...
    return cables

print(algorithme_cablage(5, 1, 1.5))
```

Question 10 : L'algorithme a un défaut, trouverez-vous lequel ?