EXO1
La distance entre deux points peut se mesurer avec la distance euclidienne :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}.$$

Pour savoir si le point est rouge ou bleu avec la technique du « 1-nearest neighbor classifier », on recherche le point le plus proche de la croix. Si le point le plus proche est rouge, on considère que la croix est rouge (et inversement).

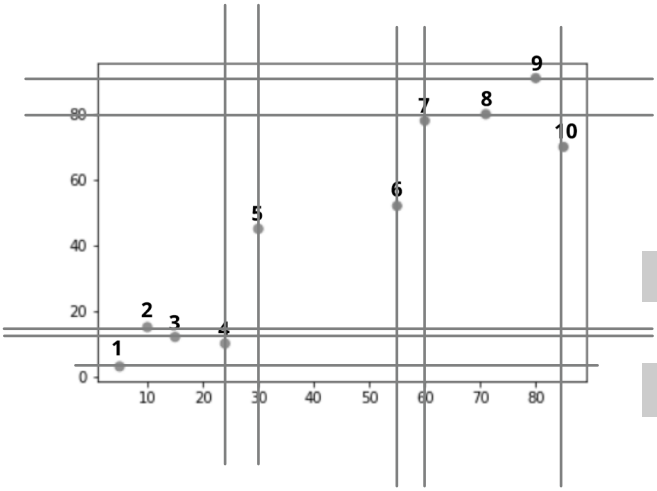
Exo2 :
C'est un R-tree.

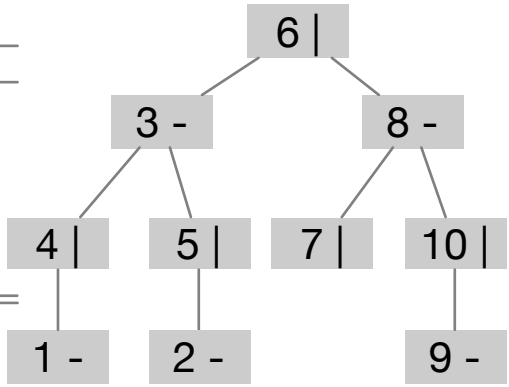
La recherche fonctionne comme suit :
On commence à la racine de l'arbre.

- « Query Range » intersecte-t-il les enfants ?
    - Pour T la réponse est non.
    - Pour U la réponse est oui.
- On fait de même pour chaque réponse positive.
    - « Query Range » intersecte-t-il les enfants ?
        - Pour R la réponse est oui.
        - Pour S la réponse est oui.
- On fait de même pour chaque réponse positive
    - Pour H, la réponse est non
    - Pour I la réponse est non
    - Pour J la réponse est oui
    - Pour K la réponse est oui
    - Pour L la réponse est non
    - Pour M la réponse est non
- Les nœuds ne possédant pas d'enfant, l'algorithme se termine.

Cette approche réduit grandement le nombre d'intersections à tester (une approche « brute-force » testerait chaque objet (13). Ici nous avons effectué 10 intersections. Si le nombre d'objets étaient plus important, le gain aurait été plus marqué.

Exo 3





L'avantage de subdivisez en k-d-tree réside dans le fait que nous réduisons le nombre de point à tester.

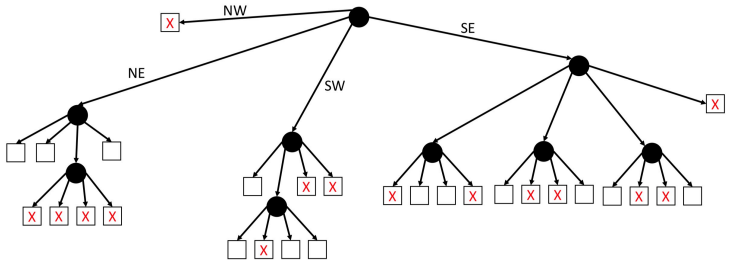
Ex4

```
import math
def find_min_dist(arr, p):
    min_dist = math.sqrt((abs(p[0]-arr[0][0]))**2+(abs(p[1]-arr[0][1]))**2)
    point = 0
    for i in range(len(arr)):
        dist = math.sqrt((abs(p[0]-arr[i][0]))**2+(abs(p[1]-arr[i][1]))**2)
        if dist < min_dist:
            min_dist = dist
            point = i
    return arr[point]

d2_points = [[8, 9], [10, -3], [2, 4], [-12, -13], [9, 9], [3,3]]
print(find_min_dist(d2_points, [1,1]))
```

Ex5

1.



2.