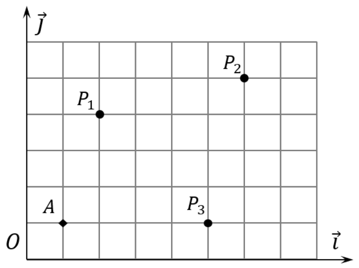
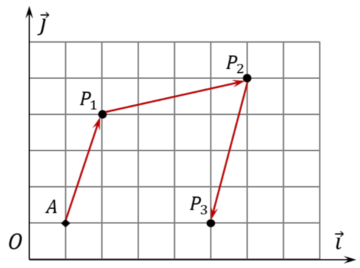
TP 9 – Tris.

Soit un plan muni d'un repère orthonormé $\left(O, \overrightarrow{i}\,\overrightarrow{j}\right)$. Soient n points appartenant à ce plan. On note (x_n, y_n) les coordonnées du point P_n.

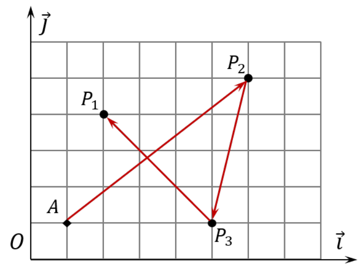
On se donne maintenant un point A de coordonnées (x_A, y_A). Le problème est le suivant : **déterminer le chemin le plus court démarrant en A et passant une seule fois par chacun des n points P_n.**



Points dans le plan



Chemin $A \to P_1 \to P_2 \to P_3$



Chemin $A \to P_2 \to P_3 \to P_1$

Réflexions autour de la résolution du problème par force brute

Question 1 *Dénombrer le nombre de chemins possibles démarrant par A et passant par 3 points.*

Question 2 *Dénombrer le nombre de chemins possibles démarrant par A et passant par n points.*

Question 3 *Conclure sur le nombre d'opérations (et donc sur le temps de calcul) nécessitant de déterminer la distance de l'ensemble des chemins passant par n points.*

Résolution du problème

Les coordonnées d'un point P sont modélisées en python par une liste de deux coordonnée. Ainisi, un point $P(x_P, y_P)$ sera déclaré par : `p = [xP,yP]`.

On note `pts` une liste de n points. On a donc `pts = [[x1,y1], [x2,y2], ... [xn,yn]]`.

Question 4 *Écrire la fonction `distance(p1:list,p2:list)-> float` permettant de calculer la distance euclidienne entre deux points. Vous importerez les fonctions que vous jugerez utile.*

On souhaite réaliser un tableau contenant :

— l'ensemble des distances entre chacun des points `pts` ;

— l'ensemble des distances entre le point A et chacun des points `pts`.

On cherche ainsi à obtenir le tableau des distances suivant.

Points		P_1	P_2	...	P_n	A
	index	0	1	...	`n-1`	n
P_1	0	`distance(p1,p1)`	`distance(p1,p2)`	...	`distance(p1,pn)`	`distance(p1,a)`
P_2	1	`distance(p2,p1)`	`distance(p2,p2)`	...	`distance(p2,pn)`	`distance(p2,a)`
...	...	...	...	...	...	...
P_n	`n-1`	`distance(pn,p1)`	`distance(pn,p2)`	..	`distance(pn,pn)`	`distance(pn,a)`
A	n	`distance(a,p1)`	`distance(a,p2)`	..	`distance(a,pn)`	`distance(a,a)`

On aura alors :

```
tab = [[distance(p1,p1), distance (p1,p2), ..., distance(p1,pn),distance(p1,pa)],
         [distance(p2,p1), distance (p2,p2), ..., distance(p2,pn),distance(p2,pa)],
         [... ,... , ..., ..., ...],
         [distance(pn,p1), distance (pn,p2), ..., distance(pn,pn),distance(pn,pa)],
         [distance(pa,p1), distance (pa,p2), ..., distance(pa,pn),distance(pa,pa)]]
```

On note que pour tout point P, `distance(p,p)=0` et que pour tous points P_1, P_2, `distance(p1,p2)=distance(p2,p1)`.

Question 5 *Écrire la fonction `distances(a:list,pts:list)-> list` permettant de calculer l'ensemble des distances entre chacun des n points P de la liste `pts` et chacune des distances entre le point `a` et chacun des points de la listes `pts` conformément à l'exemple ci-dessus.*

Question 6 *En prenant compte de la note précédente, estimer le nombre de distances à calculer par l'algorithme.*

Soit un chemin, quelconque passant par un ensemble de points contenu dans le tableau des distances. Pour modéliser ce chemin, on utilise la liste des index des points qui le constituent. Ainsi si `chemin = [n,4,0,n-1]` alors c'est qu'on a parcouru le chemin $A \to P_5 \to P_1 \to P_n$.

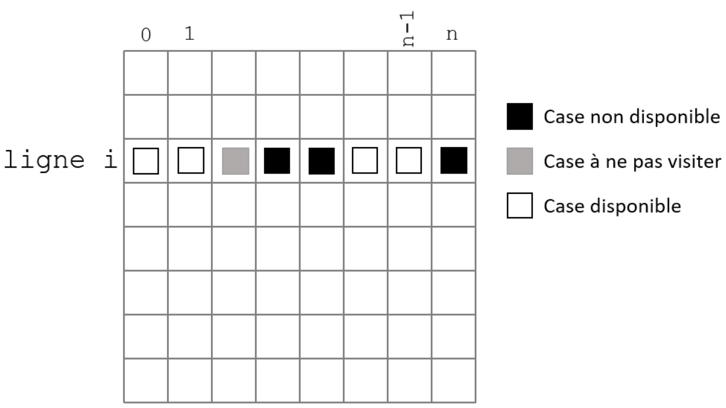
Question 7 *Écrire la fonction `longueur(chemin:list, tab:list) -> float` où `tab` est un tableau de distances tel que défini précédemment.*

Question 8 *Proposer une version récursive de cet algorithme.*

Dans le cadre de l'algorithme glouton, on considère qu'on a atteint le point k. On va alors rechercher le point le plus proche parmi les points non visités. On va donc définir la liste des points disponibles. Pour cela, on introduit une liste de booléens `dispo` de taille $n + 1$. Initialement, aucun point n'a été visité, `dispo` est donc constituée de $n + 1$ `True`. Lorsqu'un point a été visité, l'index du point visité passe à `False`.

Question 9 *On considère que `dispo=[True, True, False, False, True, False]`. Combien existerait-il de points dans la variable `pts` définie précédemment ? Combien de points ont été parcouru ? Citer les points visités.*

Pour un point d'indice `i` on va rechercher l'indice du point le plus proche parmi les points qui n'ont pas été visités.



Question 10 *Dans la figure ci-dessus, expliquer pourquoi il y a une case à ne pas visiter ?*

Question 11 *Écrire une fonction `indice(i:int, dist:list, dispo:list)->int` permettant de déterminer l'index du point le plus proche du point d'index `i`, parmi les points disponibles. `dist` désigne la liste des distances auparavant crée avec la fonction `distances`.*

Question 12 *Écrire la fonction `plus_court_chemin(dist:list)->list` permettant de construire la chemin le plus court. Le chemin sera constitué de la liste des index des points. On initialisera donc le chemin avec le plus grand index de `dist`. On initialisera la liste `dispo` des points disponibles. Á chaque itération, on ajoutera* *à* ***chemin*** *le point le plus proche puis on mettra à jour la variable* ***dispo****.*

Représentation du chemin

Question 13 *Commenter chacune des lignes suivantes (on pourra faire des regroupements de lignes pour les commentaires).*

```python
import matplotlib.pyplot as plt

x = [u[0] for u in pts]
y = [u[1] for u in pts]
plt.plot(x, y, "x")
plt.plot(depart[0], depart[1], "+")
plt.show()

tableau = distances(pts, depart)
ch = plus_court(tableau)
print(longueur(ch, tableau))

xliste = [depart[0]] + [pts[k][0] for k in ch]
yliste = [depart[1]] + [pts[k][1] for k in ch]
plt.plot(x, y, "x") # affichage des points
plt.plot(depart[0], depart[1], "+")
plt.plot(xliste, yliste) # affichage du chemin
plt.show()
```