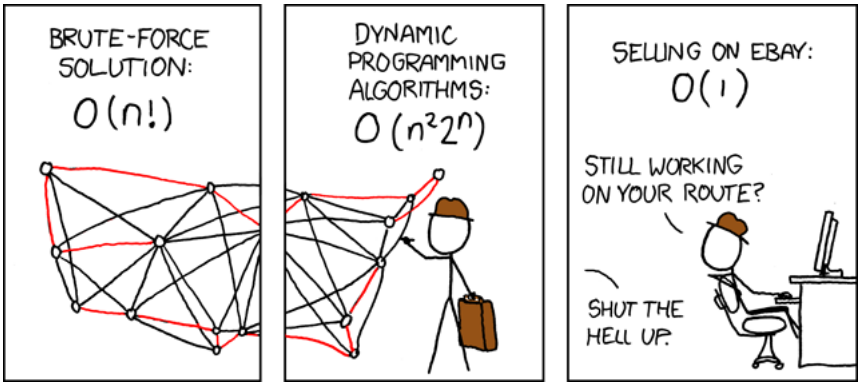
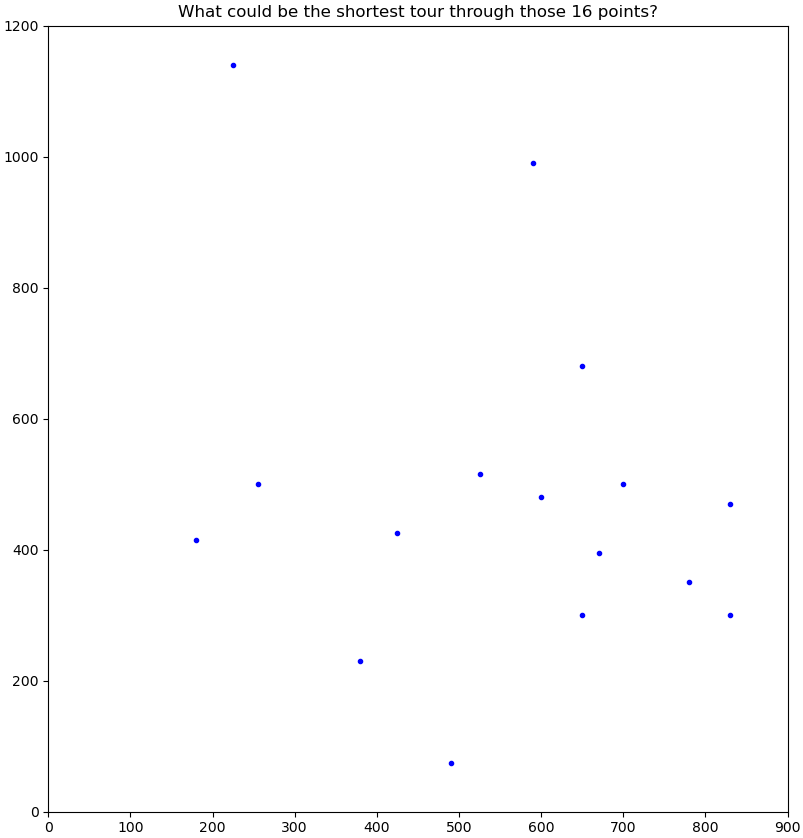
TP #3 — Problème du voyageur de commerce

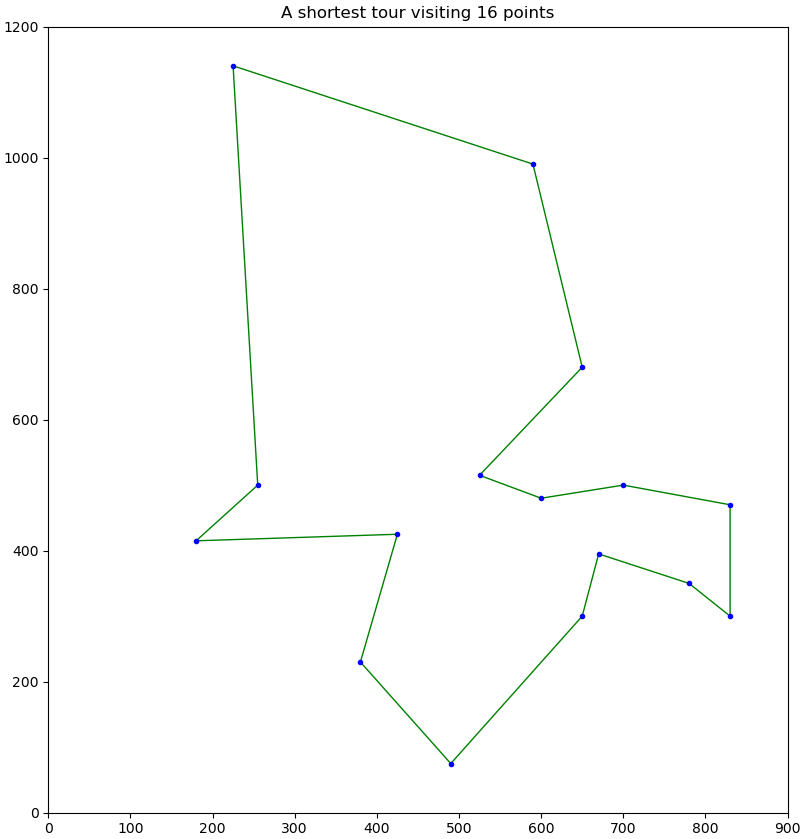


https://xkcd.com/399/

Énoncé du problème

Le but de cet exercice est de résoudre le *problème du voyageur de commerce* à vol d'oiseau (et sans vent) : étant donné $P = \{p_0, \ldots, p_{N-1}\}$ un ensemble de N points du plan, on cherche à trouver le cycle le plus court (pour la distance euclidienne) passant exactement une fois par tous les points (c'est donc un *problème d'optimisation*). Les figures ci-dessous illustrent une instance d'un tel problème, ainsi qu'une solution.


What could be the shortest tour through those 16 points?
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900


A shortest tour visiting 16 points
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900

Fonctions préliminaires

1. Écrivez une fonction Python :
   `genInstance(N:int, gridmax:int) -> list`
   qui renvoie (si cela est possible) une `list` de `N` points distincts de coordonnées entières tirées uniformément et indépendamment au hasard dans l'intervalle $[\![0, \texttt{gridmax}-1]\!]$, individuellement stockés sous forme de couples `(x,y)`.
   Rappel : il existe une fonction `randint` dans le module `random` Python.
   Conseil : vous pouvez par exemple utiliser un dictionnaire pour facilement (et efficacement) être sûr de générer exactement N points distincts (quand cela est possible).
2. Écrivez une fonction Python
   `delta(P:list, i:int, j:int) -> float`
   qui prend en entrée une `list` de points `P` (comme pouvant être générée par la fonction `genInstance`) et renvoie la distance euclidienne entre les points d'indice `i` et `j` dans `P`.
3. Écrivez une fonction Python :
   `lengthPath(P:list, Pt:tuple) -> float`
   qui prend en entrée une `list` de points `P` et un `tuple` `Pt` d'indices (de points dans `P`) et renvoie la longueur (pour la distance euclidienne) du chemin représenté par `Pt`.
   Par exemple, si `Pt` vaut `(0, 1, 5)`, cette fonction doit renvoyer la somme de la distance entre les points de `P` d'indice `0` et `1` avec celle d'entre les points d'indices `1` et `5`.
4. Écrivez une fonction Python :
   `lengthCycle(P:list, C:tuple) -> float`
   qui prend en entrée une `list` de points `P` et un `tuple` `C` et renvoie la longueur (pour la distance euclidienne) du cycle **débutant au point d'indice 0** et parcourant **tous** les autres points de `P` dans l'ordre des indices donnés par `C`, où `C` est une permutation des indices $1, \dots N-1$.
   Par exemple, si `C` vaut `(2, 1, 3)`, le cycle à considérer est $p_0 \to p_2 \to p_1 \to p_3 \to p_0$.
5. Pourquoi peut-on considérer comme dans la question précédente que le cycle commence arbitrairement au point p_0 ? Combien y a-t'il de permutations de N points comportant un unique cycle ?

Résolution par recherche exhaustive

Le module Python itertools contient une fonction permutations permettant d'itérer dans l'ordre lexicographique sur toutes les permutations de son argument :

```
from itertools import permutations
for p in permutations([1,2,3]):
```

```
# p vaudra successivement
# (1,2,3) ; (1,3,2) ; (2,1,3)
# (2,3,1) ; (3,1,2) ; (3,2,1)
```

6. Écrivez une fonction Python `TSP_exh(P:list) -> tuple` qui résout l'instance du problème de voyageur de commerce définie par les points `P`, en utilisant une recherche exhaustive. C'est à dire : énumérez tous les cycles possibles et gardez (et renvoyez) le meilleur !
   Rappel : l'opérateur `+` permet de concaténer deux tuples `t1` et `t2` en un unique tuple constitué des éléments de `t1` suivis de ceux de `t2`. Mais attention, ceci coûte assez cher : le coût est proportionnel à la longueur du résultat.
7. Utilisez votre fonction pour résoudre de petites instances. Quelle est la taille maximale d'instance que vous réussissez à résoudre en temps « raisonnable » ?
8. *(Optionnel.)* Trouvez, prouvez et implémentez un algorithme permettant de générer toutes les permutations d'une liste d'entiers dans l'ordre lexicographique.
   Indice : observez que la dernière permutation dans l'ordre lexicographique est triée dans l'ordre décroissant, et procédez récursivement.

Résolution par programmation dynamique

On souhaite maintenant utiliser une approche par programmation dynamique pour résoudre le problème de façon plus efficace.
On suppose (pour l'instant) donnée une fonction Δ qui pour $p_j \neq p_0$ et $U \subseteq P$ tel que $p_0, p_j \in U$, définit $\Delta(U, p_j)$ comme la longueur du plus court chemin commençant en p_0 et terminant en p_j et passant par tous les points de U.

9. Écrivez un algorithme (simple) qui utilise Δ pour résoudre le problème du voyageur de commerce. (Il n'est pas nécessaire d'écrire du code Python fonctionnel pour cette question, mais vous pouvez le faire néanmoins.)
10. Soit $\widetilde{U} := U \setminus \{p_j\}$, montrez que l'on a :

$$\Delta(U, p_j) = \min_{p_i \in \widetilde{U} \setminus \{p_0\}} \Delta(\widetilde{U}, p_i) + \delta(p_i, p_j)$$

    avec δ la distance euclidienne entre les points.

L'objectif va être d'implémenter une fonction `Delta` calculant Δ récursivement en utilisant la caractérisation ci-dessus, et en mémoïsant les valeurs déjà calculées dans un dictionnaire.

11. En fonction de la représentation utilisée, quel problème double peut poser l'utilisation de U et p_j comme clef d'un dictionnaire Python : 1) en terme d'unicité de représentation ; 2) de typage ?

12. Choisissez une représentation de U et p_j évitant les problèmes ci-dessus. Au besoin, écrivez une fonction `canonizeUj` renvoyant une représentation canonique et `hashable` du couple (U, p_j).

13. Écrivez une fonction Python `Delta(M, P, U, j)` calculant Δ récursivement en utilisant un dictionnaire `M` pour la mémoïsation, et renvoyant une représentation du plus court chemin (par exemple sous forme de liste) en plus de sa longueur.

14. Écrivez une fonction Python `TSP_dp(P:list) -> tuple` qui résout l'instance du problème de voyageur de commerce définie par les points `P`, en utilisant une approche par programmation dynamique.

15. Utilisez votre fonction pour résoudre de petites instances. Quelle est la taille maximale d'instance que vous réussissez à résoudre ? Comparez les résultats (et les performances) avec votre fonction `TSP_exh`.

16. Testez votre fonction sur l'instance donnée par :

```
P = [(225,1140), (590,990), (650,680), (255,500),
     (525,515), (600,480), (700,500), (425,425),
     (180,415), (830,470), (780,350), (650,300),
     (830,300), (380,230), (490,75), (670,395)]
```

Quel ordre trouvez-vous, et pour quel coût ?

17. Quelle est la complexité de la résolution du problème par programmation dynamique, en supposant un coût unitaire pour les accès dictionnaire ?
Indice : Quelle est la taille de l'ensemble des parties de P ?

Représentation des résultats *(optionnel)*

18. Écrivez une fonction `plotCycle` qui trace une image représentant les points d'un ensemble P et (optionnellement) un cycle (optimal).
Vous pouvez par exemple utiliser le module `matplotlib` et vous inspirer de la galerie d'exemples, notamment celui-ci.