Le Problème du voyageur de commerce

Carto

Cette fonction génère et retourne une matrice aléatoire à diagonale vide des distances entre n villes. Pour cet exemple, le résultat de Carto(5) sera stocké dans une variable « distance ».

```
Carto :
[  0. 454. 186.  13. 143.]
[454.   0.  87. 484. 548.]
[186.  87.   0.  97. 881.]
[ 13. 484.  97.   0.  42.]
[143. 548. 881.  42.   0.]
```

distance = Carto(5) ← on génère une matrice 5*5

Populat

Cette fonction prenant comme paramètres d'entrée le nombre n de villes et le nombre 2m de solutions (individus) que l'on souhaite générer et qui renvoie une matrice P de solutions aléatoires. Pour cet exemple, le résultat de Populat(5,6) sera stocké dans une variable « pop » pour représenter la population de départ.

```
Populat :  [[1, 5, 3, 4, 2], [1, 2, 4, 5, 3], [1, 3, 4, 5, 2], [1, 2, 3, 4, 5], [1, 2, 3, 5, 4], [1, 4, 5, 3, 2]]
```

pop = Populat(5, 6) ← population de 6 individus avec 5 villes

CalculAdapt

Cette fonction prend comme paramètre d'entrée une solution (ici, la première instance de pop) et la matrice des distances entre les villes (ici, distance) et qui renvoie la distance totale parcourue si l'on suit ce trajet.

```
CalculAdapt :  2059.0
```

CalculAdapt(pop[0], distance) ← calcul de la distance du 1er individu avec la matrice carte

SelectElit

Cette fonction prend en paramètre une population P de 2m individus (ici, pop et sa longueur) et la matrice des distances entre les villes (ici, distance) et qui renvoie les m meilleures solutions. Il s'agit d'une sélection élitiste, où la meilleure moitié est prise.

```
SelectElit :  [[1, 2, 3, 4, 5], [1, 3, 4, 5, 2], [1, 2, 3, 5, 4]]
```

SelectElit(pop, distance) ← ici 2m vaut 6 donc on aura une sélection de 3 individus

SelectTourn

Cette fonction est similaire a SelectElit() a l'exception de la méthode de sélection qui ici est une sélection par tournoi. On prend 2 solutions au hasard de la sélection et on sélectionne le meilleur des deux.

```
SelectTourn :  [[1, 4, 5, 3, 2], [1, 3, 4, 5, 2], [1, 2, 3, 4, 5]]
```

SelectTourn(pop, distance)

Croisement & CroisementBis

La fonction Croisement() prend en paramètre deux individus (ici, les deux premiers instances de pop) et deux indices, la valeur de début i de croisement et la largeur j du croisement. Elle fait appel a la fonction CroisementBis() 2 fois pour les deux listes en inversant l'ordre des paramètres. Cette fonction CroisementBis() prend en paramètre deux individus et deux indices, la valeur de début i de croisement et la largeur j du croisement. Les j valeurs de p2 d'indice i seront croisés avec la liste p1.

```
Croisement et CroisementBis :  ([1, 2, 4, 5, 3], [1, 5, 3, 4, 2])
```

Croisement(pop[0], pop[1], 2, 3) ← on croise les 2 premiers parents de la population à partir du 3^{e} numéro et sur une longueur de 3

Mutation

Cette fonction prend en paramètre un individu (ici, la première instance de pop) et retourne ce même individu avec deux indices permuté aléatoirement.

```
Mutation :  [1, 2, 5, 4, 3]
```

Mutation(pop[0]) ← on échange 2 numéros dans le 1er individu

Genetiq

Cette fonction prend en entrée un entier n (nombre de villes), le nombre d'individus, m, dans la population initiale, le taux t, de la population subissant une mutation à chaque itération, la méthode de sélection c choisie (élitiste ou par tournoi) et le nombre iters d'itérations. Cette fonction donne en sortie la meilleure des solutions par une dernière sélection élitiste.

```
élitiste :  [1, 5, 2, 4, 3]
par tournoi :  [1, 4, 3, 2, 5]
```

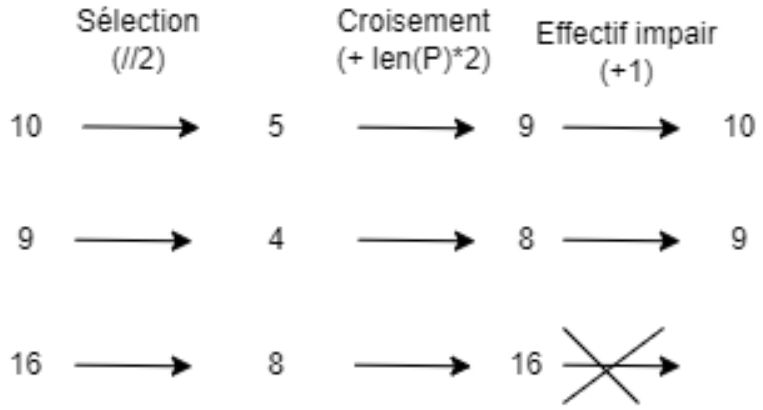
Genetiq(5,10,50,"élitiste",10)
Genetiq(5,10,50,"par tournoi",10)

Remarque :

Lors du croisement il faut gérer le cas d'une population à effectif impair. En effet, on croise par couple dans la sélection, donc dans le cas d'un effectif impair, l'individu du milieu ne sera pas reproduit et il manquerait un fils. On a donc la condition suivante :

```
if((m%2==0 and len(P)%2!=0) or (m%2!=0 and len(P)%2==0)):
```

où m est le nombre d'individu et len(P) est la longueur de la sélection courante.
On peut représenter les différents cas par le schéma suivant :



On voit ici qu'on a besoin d'ajouter un dernier fils uniquement quand l'effectif de départ est pair et que l'effectif après sélection est impair OU quand l'effectif de départ est impair et que l'effectif après sélection est pair.

Un effectif de départ pair qui après être divisé par 2 est toujours pair n'a pas besoin de rajouter un dernier fils et un effectif de départ impair sera toujours pair car on effectue une division euclidienne (//2).