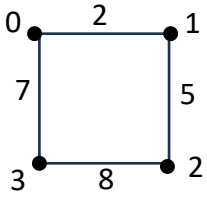
Université MB de M'sila | **Formation master 1 IDO S1 2023/2024**
Faculté de MI | **Module OC1**
Département d'informatique | **INTERROGATION ECRITE (durée 40')**

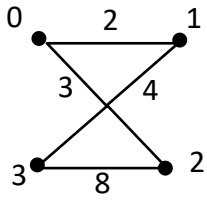
Considérons le problème du voyageur de commerce symétrique de n villes et soit d[][] la matrice des distances. Une solution (un tour) est représentée par une permutation des entiers de 0 à n-1.

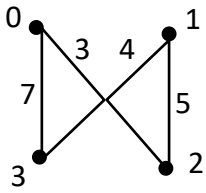
1. Rappeler la taille de l'espace de recherche en fonction de n.
2. Prenons n = 4 et d[][] = $\begin{bmatrix} 0 & 2 & 3 & 7 \\ 2 & 0 & 5 & 4 \\ 3 & 5 & 0 & 8 \\ 7 & 4 & 8 & 0 \end{bmatrix}$
   Calculer toutes les solutions possibles, leurs longueurs respectives et les représenter graphiquement.
3. Ecrire l'algorithme permettant de calculer la longueur (total des distances) d'une solution x[].
4. L'heuristique du plus proche voisin consiste à calculer une solution en commençant par une ville quelconque (aléatoire) puis aller à la ville la plus proche et ainsi de suite.
   a) Résoudre manuellement l'instance ci-dessus en utilisant cette heuristique (commencer par la ville 0).
   b) Qu'appelle-t-on ce paradigme d'algorithmique.
   c) La solution obtenue est-elle une borne inférieure ou supérieure de la solution optimale. Justifier.
   d) Ecrire l'algorithme correspondant et évaluer sa complexité.

Réponses :

1) (n-1)!/2……..0.5
2) En y a 3 : 0123 ; 0132 ; 0312 de longueurs respectives : 2+5+8+7=22 ; **2+4+8+3=<u>17</u>** ; 7+4+5+3=19…..4.5







3) Int lenght(int[] x) {
```
    s = d[ x[0] ][ x[1] ] ;
    for(i=1 ; i<n-1 ; i++)
        s += d[ x[i] ][ x[i+1] ] ;
    return s += d[ x[n-1] ][ x[0] ] ; }
```
……1

4) –
   a) On prend x[0]=0 ;
      Le minimum de la ligne 0 est 2 de la colonne 1 donc x[1]=1 ;
      Le minimum de la ligne 1 est 4 de la colonne 3 donc x[2]=3 ; ainsi x[3]=2 et x =[0132] ……1
   b) Algorithme glouton. …….0.5
   c) Une borne supérieure puisque c'est une solution <u>faisable</u> d'un problème de <u>minimisation</u>. ……..…0.5
   d) Int NearestNeighbor (int n; int[][] d) {
```
    x[0]=0 ;  // x est la solution à calculer.
    visited[0] = 1 ; // visited est un flag binaire pour marquer les villes visitées.
    for(i=1 ; i<n-1 ; i++)
        visited[i] = 0 ;
    for(i=1 ; i<n-1 ; i++) {
        min = 0 ;  // calcul du plus proche voisin non encore visité.
        for(j = 1 ; j<n-1 ; j++)
            if (! visited[j] and d[i][j] < min)
                min = d[i][j]  ;
        x[i]= j; visited[j] = 1 ; } // stocker la colonne du plus proche voisin et marquer le.
    return x ; }
```
……2
   Complexité = O(n^2). ……0.5