CCINP TSI 2024

Partie I – Des algorithmes pour colorier un graphe

I.1 Introduction sur un exemple

Q.1 Pour construire la matrice d'adjacence, on procède ligne par ligne. Par exemple, la première ligne (correspondant au sommet 0), on met 1 zéro à la première colonne (le sommet 0 n'a pas d'arête commune avec le sommet 0), puis un 1 à la deuxième colonne (il y a une arête entre le sommet 0 et le sommet 1), puis 0 (pas d'arête de 0 à 2) et 1 (arête entre 0 et 3). En procédant de même à chaque ligne, on obtient la matrice d'adjacence suivante :

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Q.2

$$\text{LA} = \big[[1,3,4,6,7],[0,2,3],[1,3],[0,1,2,4],[0,3,5,6,7],[4,6,7],[0,4,5,7],[0,4,5,6]\big]$$

Q.3 Un avantage de la matrice d'adjacence est qu'il est facile de savoir si deux sommets i et j sont reliés : il suffit d'avoir l'élément de matrice M_{ij}. Cette recherche se fait en temps constant.
L'inconvénient est que cette matrice demande un grand espace mémoire (s'il y a n sommets, il faut n^2 coefficients), ce qui peut être inutile si le graphe présente peu d'arêtes (le nombre de coefficients non nuls est égal au double du nombre d'arêtes).
Un avantage de la liste d'adjacence est son efficacité pour stocker les informations (seules les arêtes comptent), mais pour savoir si i et j sont reliés, il faut explorer toute la liste LA$[i]$ pour vérifier la présence de j.

Q.4

Sommet	0	1	2	3	4	5	6	7
Degré	5	3	2	4	5	3	4	4

I.2 Tester si une coloration est valide

Q.5 Il y avait plusieurs possibilités pour cette fonction, faites simple !

```
def voisins(i,j,LA) :
    return j in LA[i]
```

Q.6 A priori l'énoncé voulait qu'on écrive la fonction suivante :

```
def coloration_valide(LA, C) :
    for i in range(len(LA)) :
        for j in range(len(LA)) :  # la boucle peut aussi commencer à i+1
            if voisins(i, j, LA) and C[i] == C[j] :
                return False
    return True
```

mais on pouvait aussi proposer :

```
def coloration_valide(LA ,C) :
    for i in range(len(LA)) :
        for j in LA[i] :
            if C[i] == C[j] :
               return False
    return True
```

Q.7 Dans le pire des cas, la coloration est valide et tous les sommets sont reliés entre eux deux à deux. Dans ce cas, les $\frac{n(n-1)}{2}$ paires de villes sont testées, donc la complexité temporelle est quadratique.