DIU Enseigner l'informatique au lycée
mercredi 8 juillet (matin)

Graphes

Exercice 1 Dessiner tous les graphes non orientés ayant exactement trois sommets. □

Exercice 2 Combien y a-t-il de graphes orientés ayant exactement trois sommets ? On ne demande pas de les dessiner tous, mais seulement de les dénombrer. □

Exercice 3 Ajouter à la classe `Graphe` (version matrice d'adjacence) une méthode `afficher` pour afficher le graphe sous la forme suivante

```
0 -> 1 3
1 -> 2 3
2 -> 3
3 -> 1
```

c'est-à-dire une ligne par sommet, avec pour chacun la liste de ses voisins. □

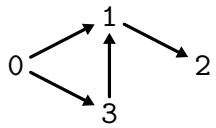
Exercice 4 Ajouter à la classe `Graphe` (dictionnaire adjacence) une méthode `degre(s)` qui donne le nombre d'arcs issus du sommet `s`. On appelle cela le *degré* du sommet `s`. □

Exercice 5 En utilisant l'exercice précédent, ajouter à la classe `Graphe` (dictionnaire adjacence) une méthode `nb_arcs()` qui donne le nombre total d'arcs du graphe. □

Exercice 6 Donner une coloration du graphe des régions qui n'utilise que quatre couleurs, c'est-à-dire une couleur à chaque sommet (région) de façon à ce que deux sommets reliés par un arc n'aient pas la même couleur. □

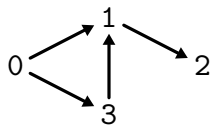
Parcours en profondeur et en largeur

Exercice 7 Dérouler à la main le parcours en profondeur sur le graphe suivant



pour différentes valeurs du sommet de départ. Donner à chaque fois la valeur finale de l'ensemble `vus`, c'est-à-dire l'ensemble des sommets atteints par le parcours. □

Exercice 8 Dérouler à la main le parcours en largeur sur le graphe suivant



pour différentes valeurs du sommet de départ. Donner à chaque fois la valeur finale du dictionnaire `dist`, c'est-à-dire la distance à la source de chaque sommet atteint par le parcours. □

Exercice 9 On peut se servir d'un parcours en profondeur pour déterminer si un graphe *non orienté* est *connexe*, c'est-à-dire si tous ses sommets sont reliés entre eux par des chemins. Pour cela, il suffit de faire un parcours en profondeur à partir d'un sommet quelconque, puis de vérifier que tous les sommets ont été atteints par ce parcours. Écrire une fonction `est_connexe()` qui réalise cet algorithme. On pourra se servir de la méthode `sommets()` de la classe `Graphe` et de la fonction `parcours` du parcours en profondeur. □

Exercice 10 Dans cet exercice, on se propose d'utiliser le parcours en profondeur pour *construire* un chemin entre deux sommets, lorsque c'est possible. On le fait avec deux fonctions, comme dans le cours :

```
def parcours_ch(g, vus, org, s):
    """parcours depuis le sommet s, en venant de org"""
    ...
def chemin(g, u, v):
    """un chemin de u à v, le cas échéant, None sinon"""
    ...
```

L'idée est que l'attribut `vus` n'est plus un ensemble mais un *dictionnaire*, qui associe à chaque sommet visité le sommet qui a permis de l'atteindre pendant le parcours en profondeur. La fonction `parcours_ch` prend un argument supplémentaire, `org` (pour origine), qui est justement le sommet qui a permis d'atteindre `s`, en empruntant l'arc `org` $\to$ `s`. Écrire le code de la fonction `parcours_ch` ; il est très semblable à celui de la fonction `parcours` du parcours en profondeur. Écrire ensuite le code de la fonction `chemin` qui renvoie un chemin entre les sommets `u` et `v`, le cas échéant, et `None` s'il n'existe pas de chemin entre ces deux sommets. Pour cela, lancer un parcours en profondeur à partir du sommet `u`, en donnant à `org` la valeur `None`, puis, si le sommet `v` a été atteint, construire le chemin dans un tableau en« remontant » le dictionnaire `vus` de `v` jusqu'à `u`. □