TP 4

IUT de Lens - Université d'Artois

1 Modélisation du problème

Cet exercice s'inspire de **math en jean** pour la classe de terminale, lycée Jean Racine, Paris 8-ième. Considérons quatre cubes C_1, C_2, C_3 et C_4 dont les faces sont coloriées de quatre couleurs, rouge, vert, bleu et jaune.
Nous considérons ici que chaque cube contient au moins une fois chaque couleur. Pour chaque cube, nous aurons deux représentations :

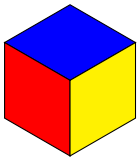


FIGURE 1: Cube en 3 dimensions.

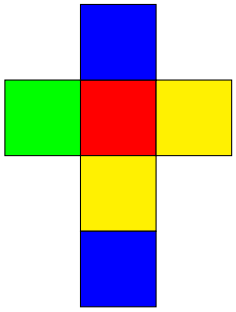
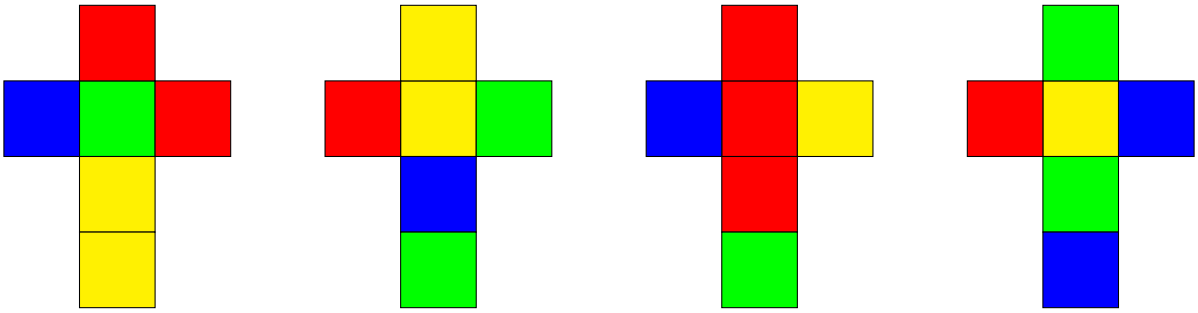
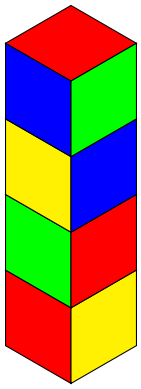


FIGURE 2: Cube développé.

Voici un exemple de 4 cubes valides :



Les cubes conviennent bien puisqu'ils contiennent toutes les couleurs. Le but de cet exercice est d'empiler les quatre cubes de manière à ce que sur chaque côté de la pile apparaissent les quatre couleurs. La figure suivante nous montre deux cotés de la pile. Les quatre couleurs apparaissent à chaque fois.

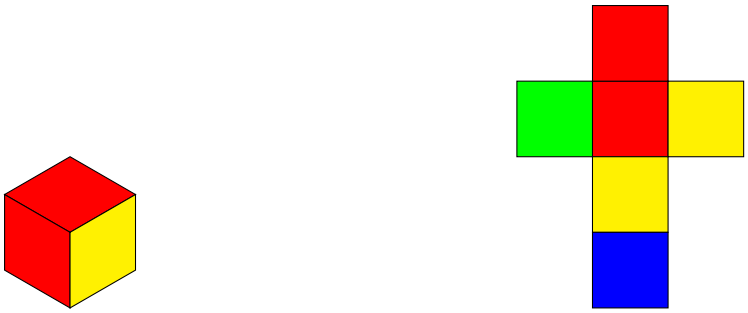


En `Python`, un cube sera représenté par un dictionnaire du type :

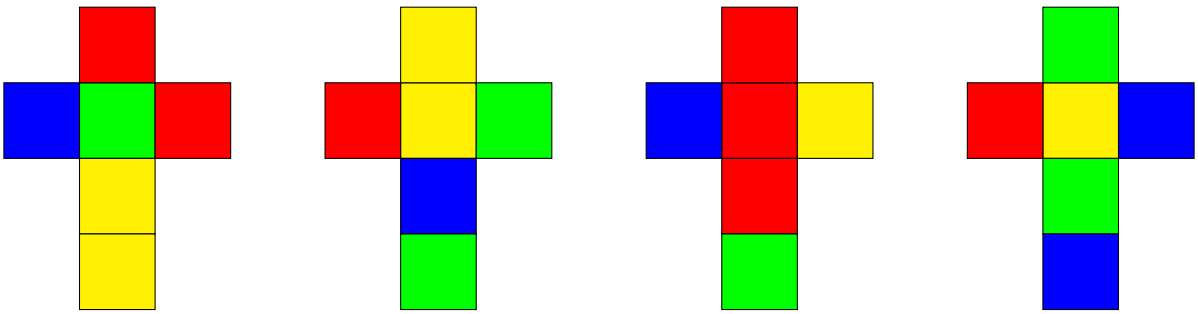
```
1 {"avant":"R","arriere":"B","gauche":"V","droite":"J","dessus":"R","dessous":"J"}
```

Les lettres `R, B, V,` et `J` représentent respectivement le rouge, le bleu, le vert et le jaune. Bien sûr, la notion de "avant" n'a pas vraiment de sens mais cela fixe les idées.

Le dictionnaire précédent donne un cube du type :

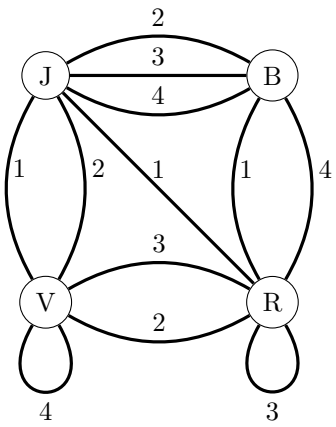


Pour résoudre ce problème à l'aide des graphes, nous allons créer un graphe dont les sommets sont `R, B, V,` et `J`.
Chaque paire de faces opposées d'un cube donne une arête du graphe. Par exemple, si le cube 3 a une face avant rouge et une face arrière verte, cela donne une arête entre le sommet `R` et le sommet `V` qui portera une étiquette 3 (numéro du cube). Par exemple, les cubes



Cube 1 **Cube 2** **Cube 3** **Cube 4**

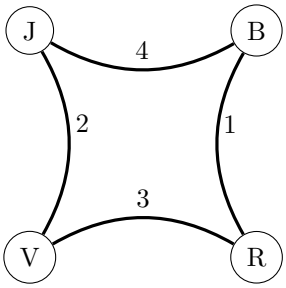
donnent le graphe

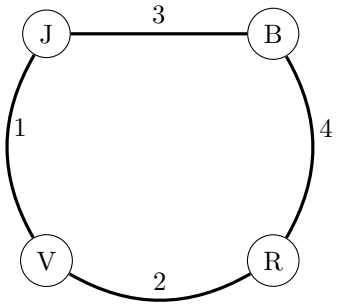


Ce graphe sera appelé **graphe des possibilités**.

2 Solution théorique du problème

Pour trouver la solution de notre problème, il suffira alors de trouver deux graphes partiels, l'un représentant les faces avant-arrière et l'autre correspondant aux faces gauche-droite. Chaque graphe partiel devra contenir une unique arête de chaque étiquette et les arêtes des deux graphes doivent être disjointes. Par exemple, dans le graphe précédent, nous avons deux graphes partiels :



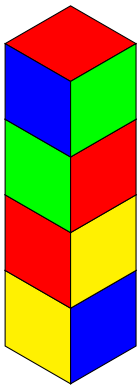


Les deux graphes partiels contiennent exactement les étiquettes `1, 2, 3` et `4` et il n'y a pas d'arêtes communes entre les deux graphes.
Ainsi, il faut placer les cubes pour les côtés avant-arrière suivant :

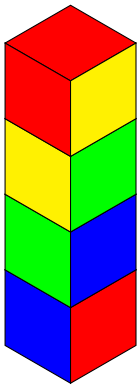
- cube 1 : B – R
- cube 2 : V – J
- cube 3 : R – V
- cube 4 : J – B

et pour les côtés gauche-droite :

- cube 1 : J – V
- cube 2 : V – R
- cube 3 : B – J
- cube 4 : R – B



Faces avant et droite.



Faces arrière et gauche.

3 Questions

1. Écrire un programme permettant de générer aléatoirement un cube valide.
   Pour cela, vous pourrez :
   - Créer une liste avec toutes les couleurs possibles et y ajouter deux autres couleurs aléatoirement avec la fonction `randint` du paquet `random`
   - Mélanger cette liste avec la fonction `shuffle` du paquet `random`
   - Affecter les valeurs de la liste dans un dictionnaire représentant un cube.

```
>>> import random
>>> liste = ['un', 'deux', 'trois', 'quatre']
>>> random.shuffle(liste)
>>> print liste
['quatre', 'trois', 'deux', 'un']
>>> random.randint(0,3)
2
```

2. Écrire un programme qui prend quatre cubes en entrée et qui retourne le graphe des possibilités.
   Attention, `Graph()` et `DiGraph()` ne permettent pas les arêtes multiples (comme c'est le cas pour notre graphe). Pour cela, vous devrez utiliser la commande `MultiGraph()`. L'utilisation de cet objet est identique aux deux autres.

```
>>> import networkx as nx
>>> G = nx.MultiGraph()
>>> G.add_edge(1,2,keys="cube1")
>>> G.add_edge(1,2,keys="cube2")
>>> G.add_edge(1,2,keys="cube3")
>>> G.edges(key=True)
[(1, 2,'cube1'), (1, 2,'cube2'), (1, 2,'cube3')]
```

L'option `key` permet de distinguer les arêtes.
Pour travailler sur les arêtes, il est toujours possible de faire un `for` sur les arêtes avec la commande

```
for e in G.edges(key=True):
```

3. Écrire une fonction permettant d'afficher les arêtes (avec la référence au cube) du graphe des possibilités.

   La fonction retournera des lignes du type :

```
R - J - cube1
J - B - cube2
...
```

4. Écrire les quatre cubes mis en exemple dans le sujet sous forme de dictionnaire et vérifier avec la fonction de la question précédente que le graphe des possibilités est le bon.
5. Écrire un programme permettant de transformer une solution en graphe.

   Dans la solution, un cube sera donné par un 4-uplet (avant, arrière, gauche et droite). La solution sera donc un tableau contenant 4 cubes `[cube 1,cube 2, cube3, cube 4]`. Par exemple, la solution de la fin de la section 2 est le tableau :

```
[("B","R","J","V"),("V","J","V","R"),("R","V","B","J"),("J","B","R","B")]
```

   La fonction retournera deux graphes partiels "solutions", comme expliqué précédemment.

6. Les solutions sont toujours deux graphes partiels du graphe des possibilités dont les sommets sont `R, V, B` et `J` et dont tous les sommets sont de degré 2.
   Écrire un programme qui permet de vérifier si un tableau présenté comme dans la question précédente est un tableau solution pour un graphe G de possibilités donné.
   Pour vérifier qu'il s'agit de graphes partiels, nous pourrons utiliser la méthode `has_edge` qui permet de savoir si une arête appartient à un graphe.

```
>>> G.has_edge(1,2,key="cube1")
True
```

   La méthode `degree` permettra d'obtenir le degré de chaque sommet.

```
>>> G.degree(1)
3
```

7. Vérifier que les tableaux solutions donnés dans le sujet passent bien le test de la fonction définie dans la question précédente.

Bonus Réaliser une interface graphique sur `Tkinter` permettant de visualiser les cubes, les manipuler et qui montre la solution.