Initiation à l'informatique
Année 2007–2008

10 décembre 2007

Table des matières

1 Premiers pas en Python **3**
- 1.1 Affectation et expressions . . . 3
- 1.2 Définition de fonction . . . 4
- 1.3 Conditionnelle . . . 5
- 1.4 Boucle Pour . . . 6
- 1.5 Boucles Pour et Tant que . . . 7
- 1.6 Temps de calcul . . . 8

2 Exploration d'un graphe à l'aide des fonctions de la bibliothèque **9**
- 2.1 Les voisins d'un sommet . . . 9
- 2.2 Marquer, colorier les sommets d'un graphe . . . 10
- 2.3 Les boucles . . . 11

3 Degré d'un sommet **13**
- 3.1 Degré d'un sommet (max, min, sommet isolé) . . . 13
- 3.2 Applications du théorème sur la somme des degrés des sommets d'un graphe . . . 14

4 Les chaînes, la connexité. **15**
- 4.1 Les chaines. La connexité . . . 15
- 4.2 Accessibilité, connexité : les algorithmes du cours . . . 16

5 Graphes Eulériens **17**

6 Coloration d'un graphe. Graphes 2-coloriables. Graphes planaires **19**
- 6.1 Coloration d'un graphe. Nombre chromatique . . . 19
- 6.2 Graphes 2-coloriables . . . 19
- 6.3 Graphes planaires . . . 20

7 Les graphes comme outil de modélisation **21**
- 7.1 Modélisation de problèmes à l'aide de graphes orientés . . . 21
- 7.2 Modélisation et coloration de graphes . . . 22

Chapitre 1

Premiers pas en Python

Affectation, expressions, alternative, boucles. Algorithmes simples, fonctions, première approche de la complexité (nombre d'opérations arithmétiques que nécessite l'exécution d'une fonction).

1.1 Affectation et expressions

Python permet tout d'abord de faire des calculs. On peut évaluer des expressions (arithmétiques ou Booléennes, par exemple). Il faut pour cela respecter la syntaxe du langage. On peut sauvegarder des valeurs dans des variables. Chaque variable a un nom. Par exemple, pour sauvegarder la valeur 123×45 dans une variable qui s'appelle `x`, on tape l'instruction

```
x = 123 * 45
```

On peut ensuite réutiliser `x`, par exemple calculer x^x en tapant `x**x`. Contrairement à sa signification mathématique, le symbole = signifie *« calculer la valeur à droite du signe = et la mémoriser dans la variable dont le nom est à gauche du signe = »*.

Python différencie les calculs n'utilisant que des nombres entiers des calculs utilisant des nombres flottants (avec virgule, qu'on écrit sous forme d'un point). Par exemple, `5/2` se calcule comme 2 (le reste de la division par `2` est oublié), mais `5.0/2` vaut bien `2.5`.

Exercice 1.1.1 Que contiennent les variables `x`, `y`, `z` après les instructions suivantes ?

```
x = 6
y = x + 3
z = 2 * y - 7
```

Exercice 1.1.2 L'instruction `i = i + 1` a-t-elle un sens ; si oui lequel ? Et `i + 1 = i` ?

Exercice 1.1.3 Que contiennent les variables `x`, `y`, `z` après les instructions suivantes ?

```
x = 6
y = 7
z = x
z = z + y
```

Exercice 1.1.4 Écrire une suite d'instructions permettant d'échanger le contenu de deux variables `a` et `b`.

Exercice 1.1.5 Écrire une instruction qui affecte à la variable `m` le résultat du calcul de la moyenne de deux réels `a` et `b` pondérés par les coefficients respectifs `coef1` et `coef2`.

Exercice 1.1.6 Écrire une expression qui vaut `True` (vrai) si `x` appartient à $[0, 5[$ et `False` (faux) dans le cas contraire.

Exercice 1.1.7 Écrire une expression qui vaut `True` (vrai) si l'entier `n` est pair et `False` (faux) dans le cas contraire.

Premiers pas avec l'interpréteur Python.

Exercice 1.1.8

1. Taper les expressions suivantes et expliquer précisément les résultats obtenus. Pour quitter l'interpréteur taper `C-d` (appuyer simultanément sur les touches Ctrl et d).

```
11*34
13.4-6 # ceci est un commentaire
13/4
13.0/4
13 % 2
14 % 3
i = 5
i = i + 4
i
j = 0
k
i < j
i != 9
i == 9
```

2. Tester les affichages suivants :

```
x = 3
print x
print "x"
```

3. Tester les affichages suivants. Quelle différence remarquez-vous ?

```
print "x = " print x

print "x = ", x
```

1.2 Définition de fonction

On peut enregistrer une suite d'instructions qu'on souhaite effectuer fréquemment en utilisant des fonctions. Une fonction travaille sur un certain nombre de paramètres. Il faut distinguer :

- la *définition* de la fonction, où le programmeur décrit ce que sa fonction doit faire. Une définition commence par le mot-clé `def`, suivi du nom de la fonction, et est en général faite une fois pour toutes. On donne ensuite, entre parenthèses, les noms des paramètres sur lesquels elle travaillera (séparés par des virgules s'il y a plusieurs paramètres). Enfin, on écrit les différentes instructions que la fonction devra exécuter à partir de ces paramètres. Ces instructions sont indentées (décalées) par rapport à la ligne `def`. L'instruction `return` permet d'indiquer ce que la fonction calcule.
- l'*utilisation* de la fonction. On peut utiliser une fonction plusieurs fois. On le fait en précisant son nom, ainsi qu'une valeur pour chaque argument.

Par exemple, les lignes suivantes définissent la fonction $f : x \mapsto x^2 + x + 1$:

```
def f(x):
    return x*x + x + 1
```

Pour l'utiliser, on écrit par exemple `f(5)`, qui calcule $5^2 + 5 + 1$, soit 31. Pour afficher $f(5)$, on écrit l'instruction `print f(5)`.

Exercice 1.2.1 Écrire une fonction `moyenne(a, b)` qui renvoie la moyenne de deux réels `a` et `b`.

Exercice 1.2.2 Écrire une fonction `affiche_moyenne(a, b)` qui affiche la moyenne des réels `a` et `b`.

Exercice 1.2.3 Écrire une fonction `est_pair(n)` qui renvoie `True` si l'entier `n` est pair et `False` sinon.

Exercice 1.2.4 Écrire une fonction qui calcule $f(x) = 2x^2 + 3x - 35$. Affichez le résultat de cette fonction pour $x = 10$, $x = 20$, $x = 12.7$.

1.3 Conditionnelle

L'instruction `if ... elif ... else` permet de tester des conditions pour exécuter ou non certaines instructions.

Exercice 1.3.1 Tester la fonction `f` définie par :

```
def f(n) :
  if n < 10:
    print n, " est plus petit que 10"
  elif n > 10:
    print n, " est plus grand que 10"
  else:
    print n, " est \'egale \‘a 10"
```

pour $n = 4$, $n = 10$ et $n = 12$.

Exercice 1.3.2 Écrire une fonction `max(x, y)` qui renvoie le maximum de deux nombres `x` et `y`.

Exercice 1.3.3 Écrire une fonction `max3(x, y, z)` qui renvoie le maximum de trois nombres `x`, `y`, `z`. Donner plusieurs versions de cette fonction dont une utilise la fonction de l'exercice 1.3.2.

Exercice 1.3.4 Écrire une fonction `median` qui retourne l'élément médian de 3 nombres passés en paramètres.

Exercice 1.3.5 Écrire une fonction `une_minute_en_plus` qui affiche l'heure une minute après celle passée en paramètre sous forme de deux entiers que l'on suppose cohérents. Exemples :

- `une_minute_en_plus(14,32)` affiche `Il est 14 h 33 min`.
- `une_minute_en_plus(14,59)` affiche `Il est 15 h 0 min`.
- `une_minute_en_plus(23,59)` affiche `Il est 0 h 0 min`.

Exercice 1.3.6 Le service de reprographie propose les photocopies avec le tarif suivant : les 10 premières coûtent 20 cents l'unité, les 20 suivantes coûtent 15 cents l'unité et au-delà de 30 le coût est de 10 cents. Écrire une fonction `cout_photocop(n)` qui renvoie le prix à payer pour n photocopies.

1.4 Boucle Pour

La boucle `for` permet d'appliquer un ensemble d'instructions sur chaque éléments d'une liste.

Exercice 1.4.1 Qu'obtient-on lorsqu'on exécute les instructions suivantes ?

```
for k in range(11) :
  print 7, " * ", k, " = ", 7*k
```

Exercice 1.4.2 Écrire les instructions Python permettant d'afficher les carrés des entiers de 1 à 20.

Exercice 1.4.3 On considère l'algorithme suivant :

```
fonction f(n):
    s = 0
    pour i allant de 1 \‘a n faire
        s = s + i
    renvoyer s
```

Quel est le résultat de `f(3)` ? de `f(5)` ? de `f(n)` ? Écrire la fonction en Python.

Exercice 1.4.4 Écrire une fonction `somme_carres(n)` qui renvoie $\sum_{i=1}^{n} i^2$.

Exercice 1.4.5 Écrire une fonction qui calcule $f(x) = 3x^2 + 2x - 7$. Afficher le résultat de cette fonction pour tous les entiers de 1 à 10.

Exercice 1.4.6 La fonction factorielle peut être définie de la manière suivante :

$$\begin{cases} 0! & = 1, \\ n! & = 1 \times 2 \times \cdots \times n = \prod_{k=1}^{n} k \text{ pour } n \geqslant 1. \end{cases}$$

Écrire une fonction `factorielle(n)` qui utilise cette définition pour calculer `n!`. Ajouter les instructions nécessaires pour afficher les factorielles de 0 à 10. Tester cette fonction pour de grandes valeurs de n.

Exercice 1.4.7 Écrire une fonction `diviseurs` qui prend en paramètre un entier n et qui affiche la liste de ses diviseurs. Testez cette fonction pour $n = 6$, $n = 17$, et $n = 36$.

Exercice 1.4.8 (Premiers pas avec l'interpréteur Python) Taper les expressions suivantes et expliquer précisément les résultats obtenus. Pour quitter l'interpréteur taper `C-d` (appuyer simultanément sur les touches Ctrl et d).

```
range(1, 10)
range(10)
range(2, 20, 3)
for i in range(0, 10):
  print "Bonjour"
for i in range(0, 10):
  print i
```

```
l = [2,6,1,10,7]
for x in l :
  print x
semaine = ["lundi","mardi","mercredi","jeudi","vendredi"]
for j in semaine :
  print j
```

1.5 Boucles Pour et Tant que

La boucle `while` (tant que) permet de répéter une partie de programme, tant qu'une condition est vraie.

Exercice 1.5.1 (Premiers pas avec l'interpréteur Python) Taper les expressions suivantes, et expliquer précisément les résultats obtenus. Pour quitter l'interpréteur taper `C-d` (appuyer simultanément sur les touches Ctrl et d).

```
k = 10
while k >= 0:
   print "k = ", k
   k = k-1
print k

k = 10
while k >= 0:
   k = k-1
   print "k = ", k
print k
```

Exercice 1.5.2 Qu'obtient-on en exécutant les lignes suivantes ?

```
for i in range(1,6) :
   for j in range(i) :
      print "*",
   print
```

Exercice 1.5.3 Écrire une fonction `puissance(a, n)` qui calcule a^n pour a réel et n entier positif. Quelles sont les instructions nécessaires pour afficher les puissances de 3, de l'exposant 1 jusqu'à l'exposant 10 ?

Exercice 1.5.4 Écrire une fonction `plus_petit_entier(a, n)` qui retourne le plus petit entier k tel que $a^k > n$ (on supposera $a > 1$).

Exercice 1.5.5 (Exercice sur les nombres premiers) L'objectif est d'écrire un algorithme qui retourne `True` (vrai) si un entier naturel non nul n est premier et `False` (faux) sinon.

1. Sachant qu'un nombre est premier s'il admet exactement deux diviseurs (1 et lui-même), une première approche consiste à compter les diviseurs de n ; écrire l'algorithme correspondant puis le traduire en Python.
2. En observant, d'une part que l'on peut conclure dès qu'un diviseur autre que 1 ou n est trouvé, d'autre part qu'il est inutile de rechercher des diviseurs au-delà de $\sqrt{n}$, écrire un deuxième algorithme s'appuyant sur ces remarques, puis sa traduction en Python.
3. Tester cette fonction en affichant tous les nombres premiers inférieurs à 1000.

1.6 Temps de calcul

Exercice 1.6.1 On considère l'algorithme suivant :

```
fonction f(n)
  s = 0
  Pour i allant de 1 \‘a n faire
    s = s+i
  retourner s
```

1. Quel est le résultat de $f(3)$, $f(5)$, $f(n)$?
2. En mathématique on peut démontrer que l'algorithme :

   ```
   fonction g(n)
      s = n*(n+1)/2
      retourner s
   ```

   donne exactement le même résultat que `f(n)`. Combien d'opérations sont nécessaires au calcul de s dans les deux algorithmes ?
3. Que devient le temps de calcul de chaque algorithme si la valeur de n est multipliée par 10 ?

Exercice 1.6.2
1. Écrire un algorithme qui calcule a^n pour un nombre réel a et un entier positif n.
2. Écrire un algorithme qui calcule $\sum_{i=0}^{n} a^i$.
3. A-t-on intérêt à utiliser la fonction précédente ? Sur machine, on pourra tester différentes versions pour $a = 0.5$ et de grandes valeurs de n.

Exercice 1.6.3
1. Écrire une fonction qui calcule le $n^{\text{ième}}$ terme de la suite définie par $u_0 = 2$, et $u_{n+1} = 4u_n + 3$.
2. Afficher les 10 premiers termes de cette suite.
3. Écrire une fonction qui affiche tous les termes de la suite de l'indice 0 à l'indice n.

Exercice 1.6.4 (La suite de Syracuse) Cette suite est définie par récurrence par

$$\begin{cases} u_0 & = a \\ u_{n+1} & = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases} \end{cases}$$

où a est un entier naturel strictement positif.

1. Ecrire une fonction `syracuse(a, n)` qui calcule le terme de rang n de cette suite lorsque le premier terme u_0 est égal à a.
2. Afficher les 10 premiers termes de la suite avec successivement $a = 1$, et $a = 3$; que constatez-vous ?
3. Que se passe-t-il si pour une valeur de n, u_n est égal à 1 ?
4. Écrire une fonction `premier_terme_egal_1(a)` qui renvoie la première valeur de n telle que $u_n = 1$ lorsque le premier terme u_0 vaut a.
5. Vérifier la conjecture « quel que soit a, il existe un rang n tel que $u_n = 1$ » pour tous les entiers a de 1 à 100.

Chapitre 2

Exploration d'un graphe à l'aide des fonctions de la bibliothèque

Les voisins d'un sommet. Marquer, colorier. Les boucles.

Deux sommets `s` et `t` sont appelés *voisins* s'il existe une arête $\{\texttt{s}, \texttt{t}\}$, c'est-à-dire ayant `s` et `t` comme extrémités. Par exemple, les sommets `'A'` et `'B'` du graphe `gr4_2` de la figure 2.1 sont voisins, mais pas les sommets `'A'` et `'C'`. Si `s` et `t` sont le même sommet, l'arête se note simplement $\{\texttt{s}\}$, et est appelée une *boucle*. Sur le graphe `gr4_2`, il y a une boucle autour du sommet `'B'` et deux boucles autour du sommet `'D'`. Chaque boucle contribue pour deux unités au degré du sommet autour duquel elle se trouve.

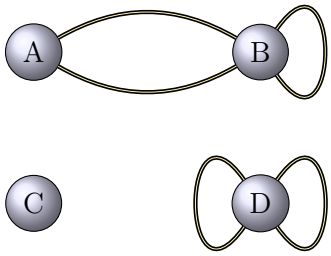


FIGURE 2.1: Graphe `gr4_2`

Lorsqu'on utilise la fonction `liste_voisins(s)`, où `s` est un sommet, la liste retournée contient `degre(s)` sommets. Un même sommet peut se retrouver plusieurs fois dans cette liste, s'il est voisin de `s` par plusieurs arêtes différentes. De même, s'il existe k boucles autour de `s`, alors `s` apparaît $2k$ fois dans la liste de ses voisins. Par exemple, `'D'` est 4 fois son propre voisin, car il est trouvé en suivant depuis `'D'` chacune des deux boucles, dans chaque sens.

2.1 Les voisins d'un sommet

Exercice 2.1.1 On se propose de travailler sur le graphe `tgv2005` de la figure 2.2 ci-contre. Comment interpréter la boucle suivante ?

```
for s in liste_sommets(tgv2005) :
   print nom_sommet(s)
```

Expliquer pour quoi on obtient le même résultat avec :

```
for k in range(nb_sommets(tgv2005)) :
   s = sommet(G, k)
   print nom_sommet(s)
```

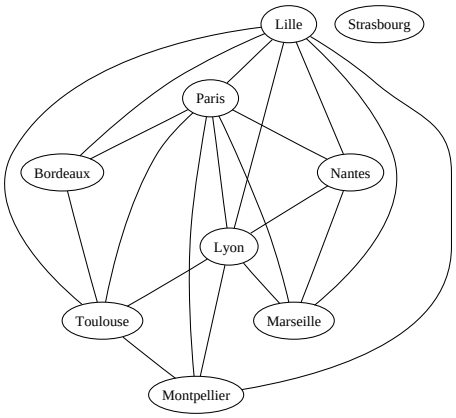


FIGURE 2.2: Le graphe tgv2005

Exercice 2.1.2 (a) Pour un sommet `s` quelconque du graphe `tgv2005`, comment interpréter l'instruction `print degre(s)` ?

(b) À quel résultat doit-on s'attendre si on l'applique au sommet `s` de nom `"Paris"` ?

(c) Comment modifier l'intruction du (a) pour obtenir le résultat de la question (b) ?

Exercice 2.1.3 Quelles fonctions utiliser pour colorier en vert le sommet de nom `"Paris"` ?

Exercice 2.1.4 Pour un sommet `s` quelconque du graphe `tgv2005`, comment interpréter la boucle

```
for v in liste_voisins(s) :
   print nom_sommet(v)
```

Expliquer pourquoi on obtient le même résultat avec :

```
for i in range(degre(s)) :
   v = voisin(s, i)
   print nom_sommet(v)
```

Par quoi remplacer `s` si on souhaite appliquer ce qui précède au sommet dont le nom est `"Lyon"` ?

Exercice 2.1.5 Écrire une fonction `affiche_degres_sommets(G)` qui, pour chaque sommet du graphe `G` affiche son degré.

Exercice 2.1.6 Écrire une fonction `sont_voisins(s1, s2)` qui renvoie `True` si les sommets `s1` et `s2` sont voisins et `False` dans le cas contraire.

2.2 Marquer, colorier les sommets d'un graphe

Exercice 2.2.1 Écrire une fonction `tout_demarquer(G)` qui démarque tous les sommets d'un graphe `G`.

Exercice 2.2.2 Écrire une fonction `tout_en_rouge(G)` qui colorie tous les sommets d'un graphe `G` en rouge. Écrire une fonction `tout_colorier(G, c)` qui colorie tous les sommets de `G` de la couleur `c`.

Exercice 2.2.3 Écrire une fonction `colorier_voisins(s, c)` qui colorie tous les voisins du sommet `s` avec la couleur `c`. Écrire ensuite une fonction `affiche_sommets_colores(G,c)` qui affiche les noms des sommets du graphe `G` dont la couleur est `c`.

Exercice 2.2.4 Écrire une fonction `existe_sommet_rouge(G)` qui renvoie `True` (vrai) s'il existe au moins un sommet colorié en rouge dans le graphe `G` et `False` (faux) sinon.

Exercice 2.2.5 Écrire une fonction `nb_sommets_rouges(G)` qui renvoie le nombre de sommets du graphe `G` qui ont la couleur rouge. Écrire une fonction qui compte les sommets colorés (c'est-à-dire qui ont une couleur différente de « blanc »).

Exercice 2.2.6 Écrire une fonction `marquer_voisins(s)` qui marque tous les voisins d'un sommet `s`. Écrire ensuite une fonction `affiche_sommets_marques(G)` qui affiche les noms des sommets du graphe `G` qui sont marqués. Attention, lorsqu'on utilise le marquage des sommets, il faut auparavant démarquer tous les sommets (cf. exercice 2.2.1).

Exercice 2.2.7 (plus difficile)

1. En vous inspirant de l'exercice 2.2.6, écrire une fonction qui affiche les voisins communs à deux sommets `s1` et `s2`.
2. Écrire une autre fonction pour déterminer les voisins communs à deux sommets, sans utiliser de marques (ni de couleurs).

2.3 Les boucles

Exercice 2.3.1 Écrire une fonction `y_a_boucle(s)` qui renvoie `True` si une des arêtes incidentes au sommet `s` (c'est-à-dire dont l'une des extrémités "touche" `s`) est une boucle, et `False` dans le cas contraire.

Par exemple sur le graphe de la figure 2.1, la fonction doit retourner `True` pour les sommets `'B'` et `'D'`, et `False` pour les sommets `'A'` et `'C'`.

Exercice 2.3.2 Écrire une fonction qui compte combien un graphe `G` a de sommets ayant au moins une boucle incidente.

Exercice 2.3.3 Écrire une fonction qui teste si un graphe est sans boucle.

Chapitre 3

Degré d'un sommet

À une arête d'un graphe (non orienté), on associe le ou les sommets aux extrémités de cette arête. On dit que l'arête est *incidente* à ce(s) sommet(s). Si l'arête est extrémité d'un seul sommet, on dit que c'est une *boucle* autour du sommet. Par exemple, sur le graphe `gr4_2` de la figure 3.1, il y a 3 boucles : `e3` autour du sommet `B`, `e4` et `e5` autour du sommet `D`. Le *degré d'un sommet* est le nombre d'extrémités d'arêtes qui sont incidentes à ce sommet. Par exemple, dans `gr4_2`

- le degré de `A` est 2, car il est extrémité de `e1` et `e2`,
- le degré de `B` est 4, car il est extrémité de `e1`, `e2` et `e3` (qui contribue pour 2 unités à son degré, car c'est une boucle autour de `B`),
- le degré de `C` est 0,
- le degré de `D` est 4.

3.1 Degré d'un sommet (max, min, sommet isolé)

Exercice 3.1.1 1. Que renvoie la fonction suivante si `G` est le graphe `gr4_2` ci-dessous ?

2. Que calcule la fonction mystere dans le cas général ?
3. Pourquoi a-t-on initialisé `x` avec la valeur `0.0` et non `0` ?

```
def mystere(G):
    n = nb_sommets(G)
    x = 0.0
    for s in liste_sommets(G) :
      x = x+degre(s)
    return x/n
```

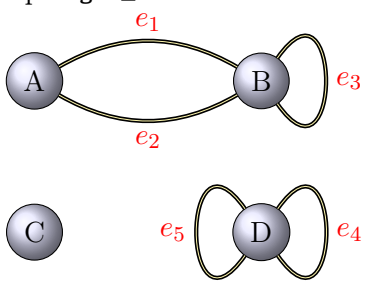


FIGURE 3.1: Graphe `gr4_2`

Exercice 3.1.2 Écrire une fonction `degre_max(G)` qui calcule le degré maximum des sommets d'un graphe `G`. Même question pour le degré minimum.

Exercice 3.1.3 Écrire une fonction `nb_sommets_degre(G, d)` qui renvoie le nombre de sommets du graphe `G` ayant pour degré `d`.

Exercice 3.1.4 On dit qu'un sommet est isolé s'il n'a aucune arête incidente. Écrire une fonction `a_un_sommet_isolé(G)` qui teste si un graphe `G` a au moins un sommet isolé.

Exercice 3.1.5 Un graphe est dit *cubique* si tous ses sommets sont de degrés 3. Écrire une fonction `est_cubique(G)` qui teste si le graphe `G` est cubique.

Exercice 3.1.6 Un graphe est appelé *simple* s'il n'a ni boucle, ni arête multiple. Autrement dit, s'il n'existe aucune arête dont les deux extrémités sont le même sommet, et si, pour deux sommets s et t distincts, il existe au plus une arête incidente à s et t.

1. Essayez de construire un graphe simple ayant 4 sommets, et tel que les degrés de sommets soient tous distincts. Qu'en déduisez-vous ?
2. On se propose de démontrer par l'absurde qu'il n'existe pas de graphe simple de n sommets ($n \geqslant 2$) tel que tous ses sommets soient de degrés distincts. Supposons qu'un tel graphe `G` existe.
   (*a*) Montrer que `G` ne peut comporter de sommet de degré supérieur ou égal à n.
   (*b*) En déduire les degrés possibles pour les n sommets.
   (*c*) Montrer que ceci entraîne une absurdité (existence simultanée d'un sommet de degré 0 et d'un sommet de degré $n-1$).
3. Application : peut-on dire que dans un groupe de n personnes, il y en a toujours deux qui ont le même nombre d'amis présents ? Justifiez votre réponse.

3.2 Applications du théorème sur la somme des degrés des sommets d'un graphe

Exercice 3.2.1 Un graphe est dit *cubique* si tous ses sommets sont de degrés 3.

1. Dessinez un graphe cubique ayant 4 sommets ; même question avec 3 sommets, 5 sommets. Que constatez-vous ?
2. Quel est le nombre d'arêtes d'un graphe cubique de n sommets ?
3. En déduire qu'un graphe cubique a un nombre pair de sommets.
4. Utiliser la question précédente pour améliorer l'efficacité de la fonction écrite dans l'exercice 3.1.5.

Exercice 3.2.2 On dispose de 235 machines, qu'on souhaite relier en réseau. Est-il possible de relier chaque machine à exactement 17 autres machines ? Justifier la réponse, soit en expliquant comment relier les machines, soit en expliquant pourquoi ce n'est pas possible.

Exercice 3.2.3 Dessinez un graphe à 6 sommets tel que la liste des degrés des sommets soit `[2, 1, 4, 0, 2, 1]`. Même question avec `[2, 1, 3, 0, 2, 1]`.

Exercice 3.2.4 Écrire une fonction `nb_aretes(G)` qui renvoie le nombre d'arêtes du graphe `G`.

Exercice 3.2.5 Dans un groupe il y a 16 anglais et n espagnols. Chaque anglais connaît exactement 5 espagnols et chaque espagnol connaît exactement 8 anglais. On considère un graphe dont les sommets sont les membres du groupe et les arêtes la relation « se connaître ». On admet que cette relation ne s'applique pas entre des individus de même nationalité.

1. Quel est, en fonction de n, le nombre de sommets ?
2. Quel est le degré d'un sommet de type anglais ? En déduire le nombre d'arêtes.
3. Quel est le degré d'un sommet de type espagnol ? En déduire une autre expression du nombre d'arêtes.
4. Déduire des questions précédentes la valeur de n.

Chapitre 4

Les chaînes, la connexité.

4.1 Les chaines. La connexité

Exercice 4.1.1 Dans le graphe g_1 ci-contre :

(*a*) Donner un exemple de chaîne simple entre S_0 et S_5,

(*b*) Donner un exemple de chaîne non simple entre S_0 et S_5.

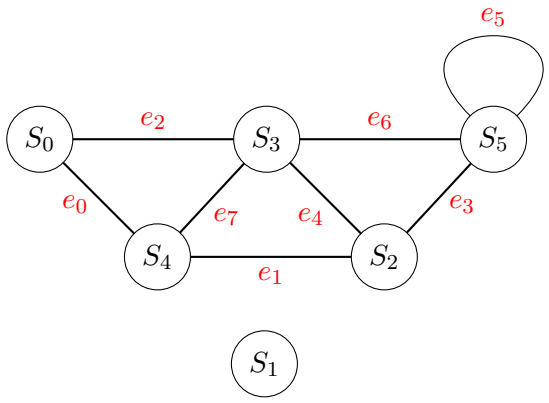
(*c*) La chaîne

$$S_3,\ e_2,\ S_0,\ e_0,\ S_4,\ e_1,\ S_2,\ e_3,\ S_5$$

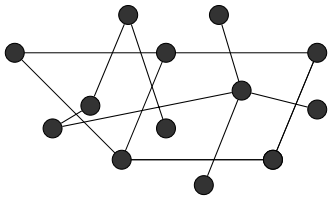
est-elle une chaîne simple entre S_3 et S_5 ?

(*d*) Le graphe g_1 est-il connexe ? Justifier.

(*e*) Donner les cycles de longueur 4 de g_1.



Exercice 4.1.2 Le graphe suivant est-il connexe ?



Exercice 4.1.3 Un peu de logique... Indiquer si les propositions suivantes sont vraies ou fausses en justifiant votre réponse :

(*a*) Si un graphe n'a pas de sommet isolé, alors il est connexe.

(*b*) Si un graphe a au moins un point isolé, alors il n'est pas connexe.

(*c*) Pour qu'un graphe soit connexe il est nécessaire que tous ses sommets soient de degré supérieur ou égal à 1.

(*d*) Si un graphe n'est pas connexe, alors il existe au moins un sommet dont le degré est nul.

(*e*) Pour qu'un graphe soit connexe, il suffit qu'il n'ait aucun sommet isolé.

(*f*) Pour qu'un graphe ne soit pas connexe, il suffit qu'il ait au moins un point isolé.

Exercice 4.1.4 On admet le théorème suivant : « Un graphe connexe de n sommets a au moins $n-1$ arêtes. »

1. Peut-on en déduire les propositions suivantes ?
   (*a*) Si un graphe de n sommets a plus de n arêtes, alors il n'est pas connexe.
   (*b*) Si un graphe de n sommets a plus de n arêtes, alors il est connexe.
   (*c*) Si un graphe de n sommets a $n-2$ arêtes, alors il n'est pas connexe.
2. Dessiner deux exemples de graphes connexes de 5 sommets ayant 4 arêtes.

Exercice 4.1.5 (Plus difficile) On se propose de montrer le théorème énoncé dans l'exercice 4.1.4, par induction sur le nombre de sommets.

1. Vérifier que le théorème est vrai pour tout graphe à un sommet.
2. Induction : partant d'un graphe à $n+1$ sommets, on construit un ou plusieurs graphe(s) à moins de n sommets pour appliquer l'hypothèse d'induction. On propose deux méthodes :
   a) 1ère méthode : supprimer un sommet et toutes les arêtes qui lui sont incidentes. Dans ce cas, le graphe obtenu n'est plus nécessairement connexe. Donner un exemple où cela se produit. Appliquer l'hypothèse d'induction sur chacune des composantes connexes du graphe obtenu pour conclure.
   b) 2ème méthode : utiliser la contraction d'une arête. Vérifier que le graphe obtenu reste connexe, et calculer son nombre de sommets et son nombre d'arêtes pour conclure.

Exercice 4.1.6 Démontrer qu'un graphe est connexe si et seulement si il existe une chaîne passant par tous les sommets du graphe (au moins une fois).

4.2 Accessibilité, connexité : les algorithmes du cours

Exercice 4.2.1 Les algorithmes vus en cours utilisent le marquage des sommets du graphe. Il est nécessaire d'avoir à sa disposition les deux fonctions : `tout_demarquer(G)` qui démarque tous les sommets du graphe `G` (cf. exercice 2.2.1), et `sommets_tous_marques(G)` qui teste si tous les sommets du graphe `G` sont marqués. Écrire ces deux fonctions.

Exercice 4.2.2 Le but est d'écrire un programme qui teste s'il existe une chaîne entre les sommets `s` et `t` d'un graphe `G`, autrement dit de tester si le sommet `t` est accessible depuis le sommet `s`. Écrire les fonctions suivantes :

(*a*) `sommet_non_marque_avec_voisin_marque(G)` qui retourne un sommet non marqué ayant un voisin marqué (extrémité non marquée d'une arête « bicolore ») et qui retourne `None` s'il n'existe pas de tel sommet.

(*b*) `accessible(G, nom_s, nom_t)` qui renvoie `True` si, dans le graphe `G`, le sommet dont le nom est `nom_t` est accessible depuis le sommet de nom `nom_s`, et `False` sinon. Tester cette dernière fonction sur le graphe de l'Europe, avec les sommets de noms `"Autriche"` et `"Portugal"`, puis `"France"` et `"Angleterre"`.

Exercice 4.2.3 Le but est d'écrire un programme qui teste si un graphe `G` est connexe. Écrire les fonctions suivantes :

(*a*) `sommet_non_marque_avec_voisin_marque(G)` vue à l'exercice précédent.

(*b*) `propager_marquage(s,G)` qui marque tous les sommets `t` du graphe `G` tels qu'il existe une chaîne entre `s` et `t`.

(*c*) `est_connexe(G)` qui utilise les fonctions précédentes pour tester si le graphe `G` est connexe.

On peut améliorer l'efficacité du marquage en utilisant la fonction `marquer_voisins(s)` vue à l'exercice 2.2.6, et en écrivant une fonction `sommet_marque_avec_voisin_non_marque(G)`.

Chapitre 5

Graphes Eulériens

Exercice 5.1.4 Les propositions suivantes sont-elles vraies ou fausses ?

(*a*) Tous les graphes Eulériens sont connexes.

(*b*) Si un graphe n'est pas Eulérien, alors il n'est pas connexe.

(*c*) Il existe des graphes connexes non Eulériens.

Exercice 5.1.5 (Bac 2006) On considère le graphe de la figure 5.1. Ce graphe admet-il une chaîne Eulérienne ? Si oui, donner une telle chaîne, sinon, justifier. Admet-il une chaîne Eulérienne dont le sommet de départ est aussi le sommet d'arrivée ?

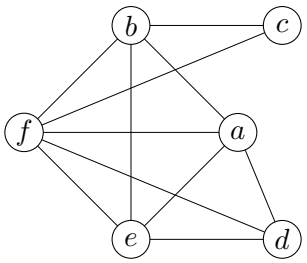


FIGURE 5.1: Graphe G_1

Exercice 5.1.6 (Bac 2006) Un agent de sécurité effectue régulièrement des rondes de surveillance. Le graphe de la figure 5.2 schématise les lieux qu'il doit visiter (les sommets) ainsi que les couloirs qu'il peut emprunter poru se rendre d'un lieu à un autre (les arêtes).

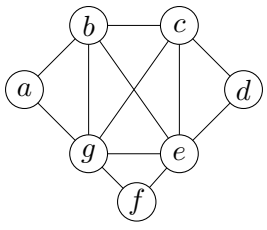
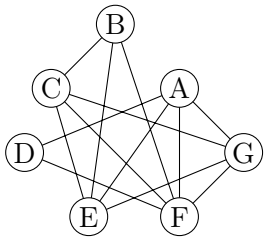


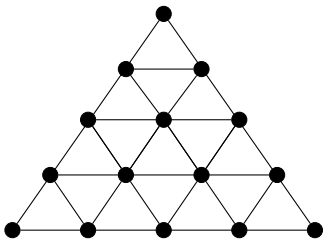
FIGURE 5.2: Graphe G_2

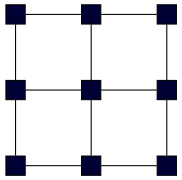
1. En justifiant la réponse, montrer qu'il est possible que l'agent de sécurité passe une fois et une seule par tous les chemins de cette usine. Donner un exemple de trajet.

2. L'agent de sécurité peut-il revenir à son point de départ après avoir parcouru une fois et une seule tous les chemins ? Justifier la réponse.

Exercice 5.1.7 Parmi les 3 graphes suivants, lesquels sont des graphes Eulériens ?







Exercice 5.1.8 Existe-il des graphes non Eulériens n'ayant aucun sommet de degré impair ? (si la réponse est « oui » dessinez-en un, si la réponse est « non » expliquez pourquoi).

Exercice 5.1.9 (Examen 2006) On considère un graphe G Eulérien dont les sommets sont A, B, C, D, E, F. On connaît une chaîne Eulérienne de ce graphe :

$$B, e_3, F, e_0, A, e_1, E, e_2, F, e_4, D, e_6, C, e_7, B, e_5, D.$$

1. Déduire de la chaîne Eulérienne le degré de chaque sommet.
2. Dessiner le graphe en précisant le nom de chaque sommet de sorte qu'aucune arête n'en croise une autre.

Exercice 5.1.10 Un graphe complet est un graphe tel que chaque sommet est relié aux autres sommets par une arête. Pour quelles valeurs de n un graphe complet de n sommets est-il Eulérien ?

Exercice 5.1.11 L'affirmation suivante est-celle correcte ? Justifier ou donner un contre-exemple. *Si un graphe G admet une chaîne Eulérienne qui est un cycle (c'est-à-dire que son sommet de départ est aussi son sommet d'arrivée) alors toutes les chaînes Eulériennes de G sont des cycles.*

Exercice 5.1.12 Écrire une fonction `nb_sommets_impairs(G)` qui renvoie le nombre de sommets impairs du graphe G. Écrire une fonction `est_Eulerien(G)` qui teste si le graphe G est Eulérien. On pourra utiliser la fonction `est_connexe(G)` écrite à l'exercice 4.2.3.

Chapitre 6

Coloration d'un graphe. Graphes 2-coloriables. Graphes planaires

6.1 Coloration d'un graphe. Nombre chromatique

Exercice 6.1.1 Un graphe est dit *bien colorié* si deux sommets voisins sont coloriés de couleurs différentes. Écrire une fonction `bien_colorie(G)` qui teste si un graphe est bien colorié.

Exercice 6.1.2 Un graphe complet d'ordre n est un graphe de n sommets tel que chaque sommet est relié aux autres sommets par une arête. On appelle K_n un tel graphe. Dessiner K_2, K_3, K_4, K_5. De combien de couleur doit-on disposer pour colorier un graphe complet d'ordre n ?

Exercice 6.1.3 Le nombre minimal de couleurs nécessaires pour colorier un graphe G est appelé nombre chromatique de G. En remarquant que le graphe G_1 de l'exercice 5.1.5 (figure 5.1) contient un sous-graphe complet, montrer que son nombre chromatique est supérieur ou égal à 4. Vérifier qu'on peut effectivement le colorier avec 4 couleurs.

6.2 Graphes 2-coloriables

Exercice 6.2.1 Parmi les graphes de l'exercice 5.1.7,

1. préciser lesquels sont 2-coloriables en justifiant votre réponse ;
2. pour les autres, calculer leur nombre chromatique.

Exercice 6.2.2 Le but est d'écrire un programme qui colorie un graphe avec deux couleurs et renvoie `True` s'il est 2-coloriable et renvoie `False` s'il n'est pas 2-coloriable.
Conventions :

- le graphe est supposé connexe.
- un sommet non colorié est un sommet dont la couleur est `"white"`.

1. Écrire une fonction `efface_couleurs(G)` qui colorie tous les sommets en blanc.
2. Écrire une fonction `sommet_blanc_avec_voisin_colorie(G)` qui renvoie un sommet de `G` non colorié ayant au moins un voisin colorié et `None` si un tel sommet n'existe pas.
3. Écrire une fonction `mono_couleur_voisins(s)` qui, si les voisins colorés du sommet `s` sont tous de la même couleur, renvoie cette couleur et qui dans le cas contraire (`s` a deux voisins de couleurs différentes) renvoie `"cycle_impair"`.
4. Écrire une fonction `deux_coloration(G)` qui colorie le graphe G avec les deux couleurs `"red"` et `"green"` et renvoie `True` si `G` est 2-coloriable et, `False` dans le cas contraire.

5. Tester la 2-coloration sur les graphes `tram`, `Petersen`, `arbre3h3`, `grille3x5`, `cercle3`, `cercle4`, `cercle5` et `complet4`. Lesquels de ces graphes ne sont pas 2-coloriables ?

6.3 Graphes planaires

Exercice 6.3.1 Montrer que, K_4, le graphe complet à 4 sommets est planaire.

Exercice 6.3.2 Le graphe G_1 de l'exercice 5.1.5 est-il planaire ? Même question pour le graphe de l'exercice 4.1.2.

Exercice 6.3.3 Avec combien de couleurs minimum peut-on colorier K_7 le graphe complet à 7 sommets ? En déduire qu'il n'est pas planaire.

Exercice 6.3.4 Le graphe K_5 est-il planaire ?

Exercice 6.3.5 (Plus difficile) On se propose d'écrire l'algorithme DEGMIN de 6-coloration d'un graphe planaire vu en cours. Comme on ne dispose pas de moyen de supprimer un sommet dans un graphe, on va marquer les sommets, avec la convention qu'un sommet marqué sera considéré comme étant supprimé. Un sommet non colorié aura la couleur `"white"`.

1. Écrire une fonction `nombre_voisins_non_marques(s)` qui retourne le nombre de voisins du sommet `s` qui ne sont pas marqués.
2. Écrire une fonction `sommet_non_marque_minimal(G)` qui retourne un sommet non marqué du graphe `G` ayant un nombre minimal de voisins non marqués, ou `None` si tous les sommets de `G` sont marqués.
3. Écrire une fonction `efface_couleurs(G)` qui positionne à `"white"` la couleur de chacun des sommets (cf. exercice 6.2.2).
4. Écrire une fonction `demarquer(G)` qui supprime toutes les marques des sommets de `G`.
5. Écrire une fonction `couleur_libre(s)` qui retourne le nom de la première couleur de la liste `["blue", "yellow", "green", "magenta", "cyan", "red"]` différente de la couleur actuelle de chacun des voisins de `s`, ou `"white"`, si les 6 couleurs précédentes sont utilisées par au moins un voisin de `s`.
6. Utiliser les fonctions `sommet_non_marque_minimal` et `couleur_libre` pour écrire une fonction `colorie_DEGMIN(G)` qui colorie un graphe en utilisant l'algorithme DEGMIN vu en cours. Cette fonction
   – supposera que le graphe G a tous ses sommets de couleur `"white"`, et non marqués,
   – pourra s'appeler elle-même, comme dans l'algorithme.
7. Utiliser les fonctions `efface_couleurs`, `demarquer` et `colorie_DEGMIN` pour écrire une fonction qui colorie un graphe selon l'algorithme du cours.
8. Vérifier sur les graphes planaires de la bibliothèque que 6 couleurs au plus sont utilisées.

Note. Une correction de cet exercice est disponible ici et les graphes coloriés obtenus sont là.

Chapitre 7

Les graphes comme outil de modélisation

7.1 Modélisation de problèmes à l'aide de graphes orientés

Exercice 7.1.1 On souhaite prélever 4 litres de liquide dans un tonneau. Pour cela on dispose de deux récipients non gradués, l'un de 5 litres, l'autre de 3 litres, et d'une bassine vide de contenance au plus 6 litres. Comment doit-on procéder pour obtenir 4 litres de liquide dans la bassine ? On pourra modéliser le problème à l'aide d'un graphe orienté.

Exercice 7.1.2 (Jeu de Chomp) Chomp se joue avec une tablette de chocolat rectangulaire dont le coin supérieur gauche est empoisonné. Chaque joueur choisit à tour de rôle un carré et le croque ainsi que tous les morceaux situés à sa droite et en dessous. Le joueur qui mange le dernier carré, en haut à gauche, a perdu.

On représente les configurations possibles d'un tel jeu à partir d'une configuration initiale par un graphe. Par exemple, partant d'une tablette de taille 2×2, le graphe est le suivant.

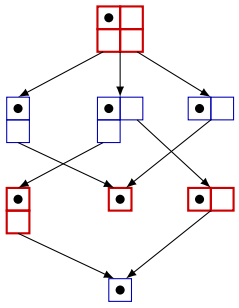


FIGURE 7.1: Jeu de Chomp

Chaque sommet représente une position du jeu (le carré empoisonné est marqué par un $\bullet$), ainsi que le numéro du joueur qui doit jouer sur cette position (en rouge et gras pour le joueur 0, en bleu pour le joueur 1). Le graphe est orienté, et un arc (l'équivalent d'une arête) entre deux sommets représente la possibilité de jouer un coup. On peut donc voir ce jeu comme joué sur un graphe, avec un jeton se trouvant initialement sur la position 2×2 du jeu. Les joueurs jouent à tour de rôle. Un coup consiste à déplacer le jeton en suivant un arc. Ici, l'objectif du joueur 0 (rouge) est d'atteindre le carré bleu, et l'objectif du joueur 1 (bleu) est d'atteindre le carré rouge.

Le joueur 0 a ici une stratégie pour gagner : il lui suffit de choisir l'arête centrale à partir de la position initiale.

1. Dessiner le graphe correspondant à une tablette 4×2.
2. Montrer qu'à partir d'une tablette carrée, le joueur qui joue en premier a une stratégie pour gagner.

Exercice 7.1.3 (Jeu de Fan Tan) Deux joueurs disposent de deux ou plusieurs tas d'allumettes. À tour de rôle chaque joueur peut enlever un certain nombre d'allumettes de l'un des tas (selon la règle choisie). Le joueur qui retire la dernière allumette a perdu.

1. Modéliser ce jeu à l'aide d'un graphe dans le cas où l'on dispose au départ de deux tas d'allumettes contenant chacun trois allumettes, et où un joueur peut enlever une ou deux allumette à chaque fois de l'un des tas.
2. Que doit jouer le premier joueur pour gagner à coup sûr ?

Exercice 7.1.4 Même questions que dans l'exercice 7.1.3, si on dispose initialement de 3 tas, contenant 1, 2, et 4 allumettes, et si chaque joueur peut, à son tour, prendre autant d'allumettes qu'il le veut dans chaque tas (au moins une). On pourra représenter une configuration du jeu par un triplet (x, y, z), où les entiers x, y, z désignent le nombre d'allumettes restant dans chacun des tas.

7.2 Modélisation et coloration de graphes

Exercice 7.2.1 Un aquariophile souhaite acquérir 6 espèces de poissons : A, B, C, D, E, F. Certaines espèces ne peuvent cohabiter dans un même aquarium. La liste des incompatibilités est la suivante : A avec E, F avec D, E avec F, C avec D, B avec C, A avec F, B avec D, B avec F. Il se pose la question de savoir de combien d'aquariums il doit disposer au minimum.

1. Dessiner le graphe représentant les incompatibilités entre les poissons
2. Modéliser le problème posé en un problème sur le graphe précédant.
3. Résoudre le problème posé.

Exercice 7.2.2 Sept élèves, désignés par A, B, C, D, E, F et G se sont rendus à la bibliothèque aujourd'hui. Le tableau suivant précise qui a rencontré qui (la bibliothèque étant petite, deux élèves présents au même moment se rencontrent nécessairement)

1. Modéliser à l'aide d'un graphe la relation « a rencontré ».
2. De combien de places assises doit disposer la bibliothèque pour que chacun ait pu travailler correctement au cours de cette journée ?

Exercice 7.2.3 (Examen 2006) Voici la liste des incompatibilités de six produits chimiques : P1 est incompatible avec P2 et P4 P2 est incompatible avec P1, P3, et P5 P3 est incompatible avec P2 et P4 P4 est incompatible avec P1 et P3 P5 est incompatible avec P2 et P6 P6 est incompatible avec P5.

1. Dessiner le graphe modélisant ces incompatibilités.
2. Sachant que deux produits incompatibles ne peuvent pas voyager dans le même wagon, on cherche le nombre minimum de wagons nécessaires au transport de ces six produits. Exprimer ce problème en terme de coloration de graphe et donner la solution en la justifiant.

Exercice 7.2.4 Un musée est constitué de 9 salles notées A, B, C, D, E, F, G, H et S. Le plan du musée est représenté sur la figure 7.2.

Ainsi, un visiteur qui se trouve dans la salle S peut atteindre directement les salles A, D ou G. S'il se trouve dans la salle C, il peut se rendre directement dans la salle B, mais pas dans les autres salles. On s'intéresse au parcours d'un visiteur dans ce musée. On ne se préoccupe pas de la manière dont le visiteur accède au musée ni comment il en sort. Cette situation peut être modélisée par un graphe, les sommets étant les noms des salles, les arêtes représentant les portes de communication.

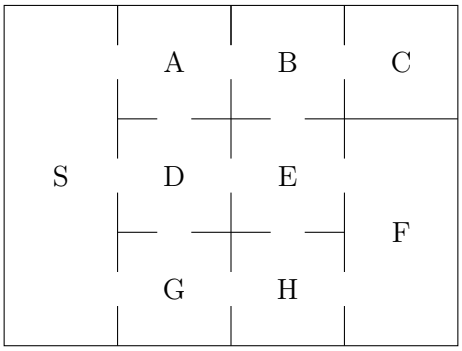


FIGURE 7.2: Plan du musée

(*a*) Dessiner un graphe modélisant la situation décrite.

(*b*) Est-il possible de visiter le musée, en empruntant chaque porte une fois et une seule ? Justifier en utilisant un théorème du cours sur les graphes.

(*c*) Pour rompre une éventuelle monotonie, le conservateur du musée souhaite différencier chaque salle de sa ou des salles voisines (c'est-à-dire accessibles par une porte) par la moquette posée au sol. Quel est le nombre minimum de types de moquettes nécessaires pour répondre à ce souhait ? Justifier.