Algorithmique 2
Coloration et Ordonnancement

Axel Abels (aabels@ulb.ac.be) Jean Cardinal (jcardin@ulb.ac.be)

Université libre de Bruxelles (ULB)
Avril 2021

Coloration de cartes

Il est toujours possible de colorer les pays d'une carte géographique avec au plus quatre couleurs, de façon que deux pays ayant une frontière commune n'aient pas la même couleur. Ce théorème, dit "Théorème des 4 couleurs" n'a été complètement démontré que relativement récemment. Ce n'est en effet qu'en 1976 qu'une démonstration a été produite par Kenneth Appel et Wolfgang Haken à l'aide d'un ordinateur. Celle-ci a été simplifiée, puis complètement vérifiée respectivement en 1997 et en 2005. La figure 1 donne un exemple de telle coloration.

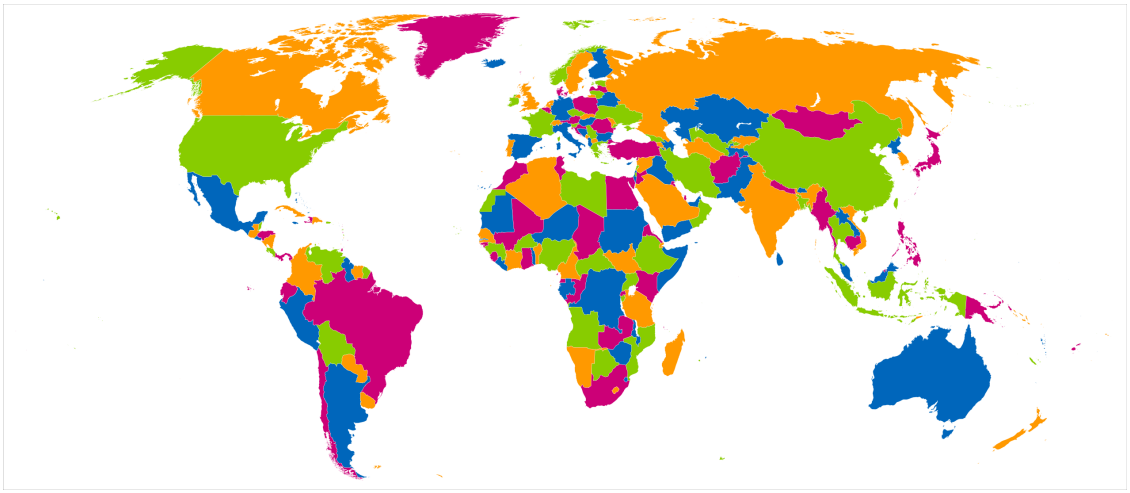

Figure 1: Une carte du monde colorée avec quatre couleurs (Wikimedia commons).

Coloration de graphes

Étant donné un graphe G (simple, non-dirigé), une *coloration propre* de G est une assignation de couleurs à ses sommets, telle que deux sommets adjacents n'aient pas la même couleur. Le nombre minimum de couleurs dans une coloration propre de G est appelé *nombre chromatique* de G, et

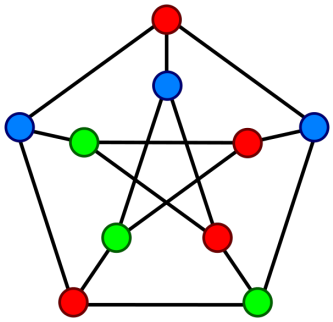

Figure 2: Le nombre chromatique du graphe de Petersen est égal à trois (Wikimedia commons).

noté $\chi(G)$. Le Théorème des 4 couleurs est un résultat de théorie des graphes, qui peut être énoncé de la manière suivante : Tout graphe planaire a un nombre chromatique au plus 4. La figure 2 montre un graphe (le *Graphe de Petersen*) coloré avec trois couleurs.

Votre première tâche consiste à décrire un algorithme permettant de décider, étant donné un graphe G et un entier positif k en entrée, si le nombre chromatique de G est inférieur ou égal à k. Si c'est le cas, l'algorithme devra en plus fournir une coloration propre de G avec au plus k couleurs.

Code

Nous vous demandons de programmer cet algorithme en Java. Votre programme devra lire le graphe à partir d'un fichier dont la structure est reprise dans le Tableau 1.

Table 1: Format des fichiers graphe

```
n
n k
...
j a
```

Table 2: graphe1.txt

```
3
1 2
2 3
1 3
```

Où la première ligne contient n, le nombre de sommets, et les lignes suivantes reprennent les paires de sommets adjacents. Le Tableau 2 représente donc un graphe complet à 3 sommets.

Votre code devra pouvoir être exécuté à l'aide de la commande suivante:

```
java coloration.java fichier_graphe k
```

où `fichier_graphe` contiendra donc la description du graphe G comme indiqué ci-dessus, et k est l'entier positif à tester. Votre programme n'imprimera rien si le nombre chromatique de G est supérieur à k, sinon il imprimera la liste de sommets avec leurs couleurs correspondantes.

Afin de faciliter la vérification du programme il est **impératif** *que le fichier lu soit celui donné comme argument à votre programme. N'encodez donc pas de noms de fichiers en dur! Des exemples de*

graphes sont joints à cet énoncé, vous êtes cependant encouragés à penser à tout autre cas de figure qui ne serait pas couvert par ces exemples.

L'exécution du programme sur l'exemple du Tableau 2 pourrait imprimer:

```
# java coloration.java graphe1.txt 3
1 3
2 1
3 2
```

La première valeur sur chaque ligne est l'indice de chaque sommet, la deuxième valeur est l'indice de sa couleur. Notez qu'il s'agit là d'un exemple d'une coloration propre de ce graphe, il en existe d'autres qui seront tout aussi acceptables.

Analyse

Nous vous demandons en outre de répondre aux questions suivantes (en ignorant dans votre analyse la lecture du fichier ainsi que l'impression des résultats) :

1. Donner une borne supérieure, sous forme de $O(.)$, de la complexité de votre algorithme.
2. Quelle est la complexité du problème dans le cas particulier où $k = 2$?
3. Pensez-vous qu'il existe un algorithme pour résoudre ce problème dans le cas particulier où $k = 3$, et dont la complexité est bornée supérieurement par un polynôme de degré constant en la taille du graphe (par exemple $O(n^2)$, ou $O(n^5)$, où n est le nombre de sommets du graphe)?

Ordonnancement sur plusieurs machines et coloration d'intervalles

Dans une seconde partie de votre travail, nous allons considérer un problème de coloration particulier.

Supposons que nous soyons dans la situation d'un planificateur qui dispose d'une collection de k machines et doit distribuer des tâches à effectuer par ces machines. Les tâches sont identifiées par deux nombres, leurs dates de début et de fin, définissant un intervalle de temps. Une machine ne peut effectuer qu'une seule tâche à la fois : il est donc impossible d'assigner à la même machine deux tâches dont les intervalles de temps s'intersectent.

On vous demande d'écrire un algorithme permettant au planificateur de décider, étant donnée une collections de n tâches et un nombre de machines k, s'il est possible de mener toutes les tâches à bien. Le cas échéant, l'algorithme devra en outre donner une assignation des tâches aux machines. Un exemple est donné à la figure 3.

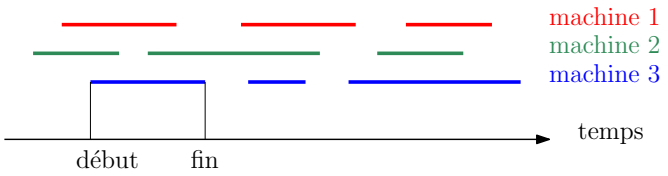

Figure 3: Un exemple de solution avec $k = 3$ pour une collection de $n = 9$ tâches.

Code

Nous vous demandons de programmer cet algorithme en Java. Votre programme devra lire le graphe à partir d'un fichier dont la structure est reprise dans le Tableau 3.

Table 3: Format des fichiers d'ordonnancement

```
n k
1 d1 f1
...
n dn fn
```

Table 4: ord1.txt

```
3
1 0 1
2 0.5 2
3 1 2
```

Où la première ligne contient n, le nombre de tâches. Les lignes suivantes reprennent les indices des tâches ainsi que leur temps de début et de fin. Le Tableau 4 représente donc un exemple à 3 tâches. On supposera qu'il ne faut pas de temps d'attente entre tâches sur une même machine. Les taches 1 et 3 de l'exemple `ord1.txt` peuvent donc être placées sur la même machine.

Votre code devra pouvoir être exécuté à l'aide de la commande suivante:

```
java ordonnancement.java fichier_ord k
```

où `fichier_ord` contiendra donc la description du problème comme indiqué ci-dessus, et k est le nombre de machines. Votre programme n'imprimera rien s'il n'est pas possible de mener à bien toutes les tâches, sinon il imprimera la liste de tâches avec leurs machines correspondantes.

Afin de faciliter la vérification du programme il est **impératif** *que le fichier lu soit celui donné comme argument à votre programme. N'encodez donc pas de noms de fichiers en dur! Des exemples de tâches sont joints à cet énoncé, vous êtes cependant encouragés à penser à tout autre cas de figure qui ne serait pas couvert par ces exemples.*

L'exécution du programme sur l'exemple `ord2.txt` qui reprend les tâches de la Figure 3 pourrait par exemple imprimer:

```
# java ordonnancement.java ord2.txt 3
1 2
2 1
3 3
4 2
5 1
6 3
7 3
8 2
9 1
```

La première valeur sur chaque ligne est l'indice de chaque tâche, la deuxième valeur est la machine à laquelle la tâche a été attribuée. Notez qu'il s'agit là d'un exemple d'ordonnancement valide, il en existe d'autres qui seront tout aussi acceptables.

Analyse

Nous vous demandons de plus de (en ignorant à nouveau la lecture du fichier et l'impression des résultats) :

1. Montrer que ce problème est un cas particulier du problème précédent, mais n'est pas équivalent au problème précédent.
2. Donner une borne supérieure, sous forme de $O(.)$, de la complexité de votre algorithme.

Consignes

Votre travail sera composé d'un rapport, de deux fichiers en Java (`coloration.java` et `ordonnancement.java`), et d'un fichier `README`.

Notez que le rapport, et donc votre analyse, est tout aussi important que votre code.

Le rapport sera composé des parties suivantes, chacune faisant l'objet d'une évaluation :

1. Pour chaque partie, une explication détaillée des algorithmes mis en œuvre et les réponses aux questions posées.
2. Une description synthétique de l'organisation de vos programmes et un commentaire détaillé des parties les plus importantes.
3. des exemples d'exécutions de vos programmes.

Le rapport devra être au format `pdf`, rédigé en français correct, mis en page avec le logiciel LaTeX, et devra avoir la forme d'un rapport scientifique. Les programmes doivent être dûment commentés et structurés, et respecter les bonnes pratiques enseignées dans les cours de programmation : noms de variables, découpe en fonctions et classes, etc. Le fichier `README` est un fichier texte devant contenir les instructions permettant au correcteur d'exécuter le code.

Les fichiers sont à remettre dans une archive au format `zip` sur l'Université Virtuelle (UV).

Ressources externes

Rédaction scientifique. Des conseils généraux sur la rédaction d'un rapport scientifique se trouvent dans le document "Éléments de rédaction scientifique en informatique", rédigé par Hadrien Mélot (UMons) et disponible à l'adresse suivante : `http://informatique.umons.ac.be/algo/redacSci.pdf`.

Ressources bibliographiques. Vous êtes encouragé.e.s, pour répondre aux questions, à consulter la littérature scientifique. Les documents utilisés doivent être clairement mentionnés dans le texte et figurer dans une liste de références bibliographiques. Les règles concernant la présentation de cette liste sont données dans un document publié par les bibliothèques de l'ULB (`https://bib.ulb.be/medias/fichier/regles-de-citation-2020_1598363674697-pdf`). Nous vous encourageons en outre à utiliser le logiciel BibTeX (`https://www.bibtex.com/`).

Code. Vous êtes autorisé·e·s à réutiliser, en le mentionnant clairement, les classes de la bibliothèque Java `algs4.jar`, disponible à l'adresse suivante : `https://algs4.cs.princeton.edu/code/`.

Échanges et plagiat

Vous êtes autorisé.e.s à discuter des problèmes et de vos solutions éventuelles avec d'autres étudiant.e.s. Si un point de votre rapport est le fruit de ces discussions, vous êtes invité.e.s à le mentionner explicitement. La rédaction du rapport et des programmes doit cependant être individuelle. Tout emprunt non mentionné explicitement est un plagiat, et constitue une fraude. Les étudiant·e·s reconnu·e·s coupables de fraude ou de tentative de fraude s'exposent à des sanctions disciplinaires comprenant l'annulation de la session d'examens et l'interdiction de s'inscrire à la session d'examens suivante.

`https://bib.ulb.be/fr/support/boite-a-outils/evitez-le-plagiat`

Date de remise

30 avril 2021, avant minuit.