Travaux pratiques

Coloration d'un graphe

On considère un graphe non orienté $G = (V, E)$, et on appelle *k-coloration* de G une application $c : V \to [\![0, k-1]\!]$ telle que $(a, b) \in E \Longrightarrow c(a) \neq c(b)$. Le problème de la coloration d'un graphe consiste à trouver une k-coloration de G pour laquelle la valeur de k est la moins élevée possible.

Les sommets d'un graphe G d'ordre n seront désignés par les entiers $0, 1, \dots, n-1$ et G sera représenté par les listes d'adjacence de ses sommets ; autrement dit, nous définissons les types :

```
type voisinage == int list ;;
type graphe == voisinage vect ;;
```

Une k-coloration c sera représentée par un vecteur **`couleur`** de type *int vect* défini par **`couleur.(i)`** $= c(i)$, $0 \leqslant i \leqslant n-1$.

Question 1. Rédiger une fonction **`coloration_valide`** qui prend en argument un graphe G et un vecteur **`couleur`** et qui retourne **`true`** si **`couleur`** est une coloration de G, et **`false`** sinon.

```
coloration_valide : graphe -> int vect -> bool
```

Indication. On pourra utiliser la fonctionnelle **`for_all`** de type *('a -> bool) -> 'a list -> bool*.

Exprimer en fonction de $n = |V|$ et $p = |E|$ le coût de cette fonction.

Graphe biparti

Un graphe qui possède une 2-coloration est dit *biparti* : il existe une partition de V en deux ensembles A et B telle que toute arête de E relie un sommet de A à un sommet de B.

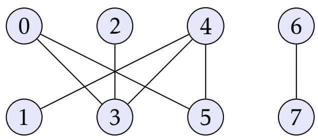

Figure 1 – Un exemple de graphe biparti.

Question 2. Rédiger une fonction **`biparti`** qui prend en argument un graphe *supposé biparti* et retourne une 2-coloration de ce dernier.

```
biparti : graphe -> int vect
```

Évaluer en fonction de n et p le coût de cet algorithme.

Un algorithme glouton

La question précédente a montré que le problème de la 2-coloration possède une solution en temps polynomial en n. Ce n'est malheureusement pas le cas du problème général : tous les algorithmes connus garantissant une coloration optimale sont de complexité exponentielle. C'est la raison pour laquelle nous allons nous intéresser à des solutions gloutonnes qui, à défaut de garantir une solution minimale, fournissent des colorations acceptables.

Question 3. L'algorithme glouton consiste à parcourir les sommets par ordre croissant d'index, en attribuant à chaque sommet la plus petite couleur disponible (c'est-à-dire non déjà donnée à un de ses voisins). Rédiger la fonction **`glouton`** correspondante.

```
glouton : graphe -> int vect
```

Cette fonction retourne-t-elle une coloration optimale pour le graphe biparti présenté figure 1 ?

Graphes d'intervalles

À un ensemble fini d'intervalles $[a_i, b_i[$, $0 \leqslant i \leqslant n-1$ on associe un graphe $G = (V, E)$ pour lequel les sommets de V sont les intervalles et dont les arêtes relient les couples d'intervalles dont l'intersection est non vide. Un tel graphe est appelé *graphe d'intervalle*.

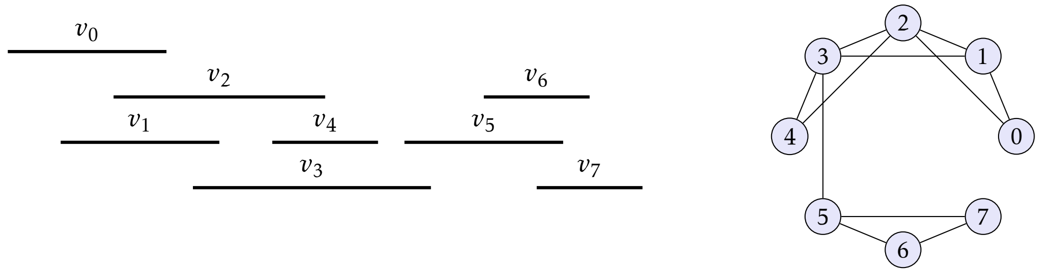

Figure 2 – Un exemple de graphe d'intervalles.

Question 4.

a) Montrer que pour un graphe d'intervalles, l'algorithme **`glouton`** fournit une coloration minimale lorsque les sommets sont ordonnés par valeurs de a_i croissantes.

b) Montrer que pour un graphe G quelconque il existe toujours un ordonnancement des sommets pour lequel l'algorithme glouton fournit un résultat optimal.

Algorithme DSATUR

Cet algorithme est une variante de l'algorithme glouton, dans lequel on essaie de choisir « au mieux » le prochain sommet à devoir être coloré. Pour cela on introduit la notion de *degré de saturation* : en cours d'exécution, et pour tout $v \in V$, $d_s(v)$ est le nombre de couleurs distinctes d'ors et déjà utilisées pour colorer les voisins de v.
À chaque étape, l'algorithme DSATUR attribue au sommet vierge de degré de saturation maximal la plus petite couleur disponible. En cas d'égalité, c'est le sommet de degré maximal qui est choisi.

Question 5. On suppose désormais les sommets triés par degré décroissant.
Rédiger une fonction **`ds`** qui prend en arguments un graphe G en cours de coloriage, le tableau **`couleur`** des couleurs déjà attribuées et un sommet v et qui retourne le degré de saturation de v.

```
ds : graphe -> int vect -> int -> int
```

Rédiger une fonction **`satmax`** qui prend en arguments un graphe G en cours de coloriage et le tableau **`couleur`** des couleurs déjà attribuées et qui retourne le sommet vierge de saturation maximale (et en cas d'égalité, celui de degré maximal).

```
satmax : graphe -> int vect -> int
```

En déduire une fonction **`dsatur`** qui prend en argument un graphe trié par ordre décroissant de degré et retourne la coloration de ce dernier obtenue par application de l'algorithme DSATUR.

```
dsatur : graphe -> int vect
```

Question 6.

a) Montrer qu'appliqué à un graphe biparti, l'algorithme DSATUR retourne une 2-coloration.

b) Appliquer cet algorithme au graphe ci-dessous. La coloration obtenue est-elle optimale ?