Corrigé du contrôle : coloration d'un graphe

Partie I. Détermination des voisins des sommets

Question 1.

```
let rec insere lst s = match lst with
  | [] -> [s]
  | t::q when s < t -> s::lst
  | t::q when s = t -> lst
  | t::q            -> t::(insere q s) ;;
```

Question 2. Lorsque s est supérieur à tous les éléments de la liste L, cette dernière doit être parcourue dans son entier ; si n désigne la longueur de L, la complexité de cette fonction est un $O(n)$.

Question 3.

```
let voisins graph s =
  let rec aux lst = function
    | [] -> lst
    | arete::q when arete.a = s -> aux (insere lst arete.b) q
    | arete::q when arete.b = s -> aux (insere lst arete.a) q
    | arete::q                  -> aux lst q
  in aux [] graph.A ;;
```

Partie II. Un algorithme de bonne coloration d'un graphe

Question 4. L'algorithme décrit dans l'énoncé va colorer les sommets de *Gex2* de la façon suivante :

- le sommet 0 se voit attribuer la couleur 1 ;
- le sommet 1 se voit attribuer la couleur 1 ;
- le sommet 2 se voit attribuer la couleur 2 ;
- le sommet 3 se voit attribuer la couleur 2 ;
- le sommet 4 se voit attribuer la couleur 3 ;
- le sommet 5 se voit attribuer la couleur 3.

Cette coloration n'est pas optimale, car deux couleurs suffisent si on attribue aux sommets 0, 2 et 4 la couleur 1 et aux sommets 1, 3 et 5 le couleur 2.

Question 5. La fonction suivante utilise une fonction auxiliaire `couleur` qui détermine la première couleur disponible pour colorer un sommet.

```
let coloration graph =
  let c = make_vect graph.n 0 in
  let couleur s =
    let vois = voisins graph s in
    let rec aux k = function
      | []                -> k
      | t::q when c.(t) = k -> aux (k+1) vois
      | t::q                -> aux k q
    in aux 1 vois
  in
  for s = 0 to graph.n - 1 do c.(s) <- couleur s done ;
  c ;;
```

Partie III. Définition du nombre chromatique de **G**

Question 6. Un graphe ayant n sommets possède à l'évidence une bonne n-coloration : il suffit d'attribuer au sommet k la couleur $k+1$. Ceci prouve que $\mathrm{EC}(\mathrm{G}) \neq \emptyset$ et donc que cet ensemble possède un plus petit élément $\mathrm{nbc}(\mathrm{G})$.
Il reste à prouver que pour tout $p \geqslant \mathrm{nbc}(\mathrm{G})$ il existe une bonne p-coloration de G : c'est évident si on observe qu'une bonne p-coloration est aussi une bonne q-coloration pour tout $q \geqslant p$.

Question 7. Un graphe G n'ayant aucune arête peut être coloré à l'aide d'une seule couleur, donc $\mathrm{nbc}(\mathrm{G}) = 1$. De plus, toute coloration est une bonne coloration, donc une bonne p-coloration est une application quelconque de $[\![0, n(\mathrm{G})-1]\!]$ vers $[\![1, p]\!]$. Ainsi, $\mathrm{fc}(\mathrm{G}, p) = p^{n(\mathrm{G})}$.

Question 8. Dans le cas d'un graphe complet, tout sommet doit être d'une couleur différente de tous les autres, ce qui nécessite $n(\mathrm{G})$ couleurs. Ainsi, $\mathrm{nbc}(\mathrm{G}) = n(\mathrm{G})$.
Lorsque $p < n(\mathrm{G})$, on a donc $\mathrm{fc}(\mathrm{G}, p) = 0$; lorsque $p \geqslant n(\mathrm{G})$, une bonne p-coloration est une application injective de $[\![0, n(\mathrm{G})-1]\!]$ vers $[\![1, p]\!]$ et ainsi $\mathrm{fc}(\mathrm{G}, p) = \dfrac{p!}{(p-n(\mathrm{G}))!}$.

Question 9. Les sommets 0, 3 et 4 sont tous trois voisins donc doivent posséder une couleur différente ; ainsi $\mathrm{nbc}(Gex1) \geqslant 3$.
Mais si on attribue aux sommets 0, 1 et 2 la couleur 1, au sommet 3 la couleur 2 et au sommet 4 la couleur 3, on obtient une bonne coloration de *Gex1*, donc $\mathrm{nbc}(Gex1) = 3$.
Si $p < 3$, on a $\mathrm{fc}(Gex1, p) = 0$; si $p \geqslant 3$ il faut choisir 3 couleurs distinctes pour colorer les sommets 0, 3 et 4, puis choisir une couleur différente de celle du sommet 3 pour colorer le sommet 1, et enfin colorer d'une couleur quelconque le sommet 2, ce qui donne : $\mathrm{fc}(Gex1, p) = p(p-1)(p-2) \times (p-1) \times p = p^2(p-1)^2(p-2)$.

Partie IV. Les applications **H** et **K**

Question 10. On peut remarquer que le premier voisin de s est le premier élément de la liste triée de ses voisins ; d'où :

```
let prem_voisin graph s = hd (voisins graph s) ;;
```

Dans le cas d'un sommet isolé, cette fonction déclenchera l'exception `Failure "hd"`.

Question 11.

```
let prem_ni graph =
  let rec aux = function
    | s when voisins graph s = [] -> aux (s+1)
    | s                           -> s
  in aux 0 ;;
```

Question 12.

```
let h graph =
  let s1 = prem_ni graph in
  let s2 = prem_voisin graph s1 in
  let rec aux lst = function
    | [] -> {n = graph.n; A = lst}
    | t::q when t.a = s1 && t.b = s2 -> aux lst q
    | t::q when t.b = s1 && t.a = s2 -> aux lst q
    | t::q -> aux (t::lst) q
  in aux [] graph.A ;;
```

Question 13.

```
let k graph =
  let s1 = prem_ni graph in
  let s2 = prem_voisin graph s1 in
  let renumerote = function
    | s when s < s2 -> s
    | s when s = s2 -> s1
```

```
    | s            -> s - 1 in
  let f {a = x; b = y} = {a = renumerote x ; b = renumerote y} in
  {n = graph.n - 1; A = map f (h graph).A} ;;
```

Partie V. Fonction $f_C(G, p)$ et polynôme chromatique

Question 14. Pour passer d'un graphe G au graphe H(G), on se contente d'ôter des arêtes, donc toute bonne coloration de G est aussi une bonne coloration de H(G). D'où : $\mathrm{BC}(G, p) \subset \mathrm{BC}(H(G), p)$.

Question 15. $\mathrm{BC}(H(G), p) \setminus \mathrm{BC}(G, p)$ est l'ensemble des bonnes colorations de H(G) qui ne sont pas des colorations de G, donc des colorations pour lesquelles les sommets s_1 et s_2 ont la même couleur. Ils sont donc en bijection avec les bonnes colorations de K(G) :

$$\operatorname{card} \mathrm{BC}(K(G), p) = \operatorname{card} \mathrm{BC}(H(G), p) \setminus \mathrm{BC}(G, p).$$

Puisque $\mathrm{BC}(G, p) \subset \mathrm{BC}(H(G), p)$, $\operatorname{card} \mathrm{BC}(H(G), p) \setminus \mathrm{BC}(G, p) = \operatorname{card} \mathrm{BC}(H(G), p) - \operatorname{card} \mathrm{BC}(G, p)$ et l'égalité s'écrit :

$$f_C(G, p) = f_C(H(G), p) - f_C(K(G), p).$$

Question 16. Pour un graphe sans arêtes nous savons que $f_C(G, p) = p^{n(G)}$; pour un graphe avec des arêtes on applique la formule précédente. Cet algorithme se termine car les graphes H(G) et K(G) ont un nombre d'arêtes strictement inférieur à celui de G.

Question 17. Raisonnons par récurrence sur le nombre d'arêtes d'un graphe :

- lorsque G n'a pas d'arêtes, $f_C(G, p) = p^{n(G)}$ est bien un polynôme en p de degré $n(G)$;
- lorsque G a des arêtes, $f_C(G, p) = f_C(H(G), p) - f_C(K(G), p)$ et par hypothèse de récurrence, $f_C(H(G), p)$ est un polynôme en p de degré $n(G)$ et $f_C(K(G), p)$ un polynôme en p de degré $n(G) - 1$, donc $f_C(G, p)$ est la restriction d'un polynôme en p de degré $n(G)$.

Partie VI. Calcul du polynôme $P_C(G, p)$ et de $\mathrm{nbc}(G)$

Question 18. On se contente de traiter le cas où $\deg Q < \deg P$:

```
let difference p q =
  let r = copy_vect p in
  for i = 0 to vect_length q - 1 do
    r.(i) <- r.(i) - q.(i)
  done ;
  r ;;
```

Question 19. Commençons par rédiger une fonction définissant le polynôme X^n :

```
let monome n =
  let p = make_vect (n + 1) 0 in
  p.(n) <- 1 ;
  p ;;
```

On définit ensuite :

```
let rec pc graph = match graph.A with
  | [] -> monome (graph.n)
  | _  -> difference (pc (h graph)) (pc (k graph)) ;;
```

Question 20. Il suffit d'appliquer la méthode de Horner :

```
let eval p x =
  let n = vect_length p - 1 in
  let rec aux acc = function
    | (-1) -> acc
    | k    -> aux (x * acc + p.(k)) (k-1)
  in aux 0 n ;;
```

Question 21. Le nombre chromatique d'un graphe est le plus petit entier n'étant pas racine du polynôme $P_C(G)$. Ceci conduit à la définition suivante :

```
let nbc graph =
  let p = pc graph in
  let rec aux = function
    | k when eval p k = 0 -> aux (k + 1)
    | k                   -> k
  in aux 1 ;;
```

♦ ♦
♦