Corrigé. D.2./D.S.1.

I - Détermination des voisins des sommets.

1. ```
   let rec insere l s = match l with
       |[] -> [s]
       |t::q when  s < t -> s::l
       |t::q when s = t -> l
       |t::q -> t::(insere q s) ;;
   ```

2. Le pire des cas se produit lorsque s majore la liste l. Dans ce cas, on effectue $O(|l|)$ opérations `::` (où $|l|$ désigne la longueur de l).

3. On écrit une fonction auxiliaire qui stocke dans un accumulateur la deuxième extrémité de chacune des arêtes dont une extrémité est s.

   ```
   let voisins g s =
     let rec aux accu arliste = match arliste with
       |[] -> accu
       |t::q when t.a = s -> aux (insere accu t.b) q
       |t::q when t.b = s -> aux (insere accu t.a) q
       |_::q -> aux accu q
   in aux [] g.al ;;
   ```

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

4. 0 est coloré 1, 1 est coloré 1, 2 est coloré 2, 3 est coloré 2, 4 est coloré 3, 5 est coloré 3.

   Mais il existe une bonne coloration avec seulement deux couleurs : la couleur 1 pour les sommets pairs et la couleur 2 pour les sommets impairs.

5. On écrit une fonction auxiliaire `plus_petits` qui étant donnés un entier i et une liste croissante d'entiers ℓ renvoie la sous-liste de ℓ constituée des entiers de ℓ inférieurs strictement à i.

   La deuxième fonction auxiliaire `premier_choix` est telle que `premier_choix 1 q` renvoie le premier numéro de couleur qui ne fait pas partie de la liste `q`.

   La troisième fonction auxiliaire, étant donné une fonction f et une liste ℓ retourne la liste des images par f des éléments de ℓ : autrement dit, elle applique la fonction f aux éléments de ℓ.

   ```
   let rec plus_petits  i l = match l with
     |t::q when t < i -> t::(plus_petits i q)
     |_ -> [] ;;

   let rec premier_choix i l = match l with
      |q  when List.mem i q ->  premier_choix (i+1) q
      |_ -> i;;
   ```
   ```

```
let rec applique f l = match l with
   |[] -> []
   |t::q -> (f t)::(applique f q);;

let coloration g = let couleurs = Array.make g.n 0 in
   for i=0 to g.n - 1 do
      let vois = voisins g i in
         let v = plus_petits i vois in
            let f x = couleurs.(x) in
               let c = applique f v in
                  couleurs.(i) <- premier_choix 1 c
   done ;
couleurs ;;
```

Dans la fonction `coloration`, `c` est construit, à partir de la liste des voisins de `i` de numéro strictement inférieur, en remplaçant chacun des voisins par son numéro de couleur, de façon à choisir la première couleur disponible pour l'attribuer à `i`.

III - Définition du nombre chromatique de G.

6. L'ensemble $E_c(G)$ est non vide : il existe en effet une bonne coloration à $n(G)$ couleurs en donnant au sommet i la couleur $i+1$.

   Notons $\text{nbc}\,(G) = \min E_c(G)$. Il reste à montrer que $E_c(G) = \{p \mid p \geq \text{nbc}\,(G)\}$.

   Par définition de $\text{nbc}\,(G)$, toute bonne coloration de G utilise au moins $\text{nbc}\,(G)$ couleurs donc $E_c(G) \subset \{p \mid p \geq \text{nbc}\,(G)\}$.

   L'inclusion réciproque résulte du fait que si $q \geq p$, toute bonne p-coloration de G est aussi une bonne q coloration de G dans laquelle les couleurs $p+1, p+2, \ldots q$ ne sont pas utilisées.

7. $\boxed{\text{nbc}\,(G) = 1}$ puisqu'on peut donner la même couleur à tous les sommets.

   $f_c(G,p)$ est le nombre de façon d'affecter arbitrairement une des couleurs à chacun des sommets, donc le nombre d'applications de $[|1, n(G)|]$ dans $[|1,p|]$ soit $\boxed{p^{n(G)}}$.

8. $\boxed{\text{nbc}\,(G) = n(G)}$ car les couleurs des sommets doivent être deux à deux distinctes et pour $\boxed{p < n(G),\ f_c(G,p) = 0}$.

   $\boxed{\text{Pour } p \geq n(G) :\ f_c(G,p) = p(p-1)\ldots(p-n(G)+1) = \dfrac{p!}{(p-n(G))!}}$ car on choisit la couleur du premier sommet (p choix) puis celle du deuxième parmi les $p-1$ couleurs restantes ...

   La première formule est la mieux adaptée à la suite du problème, et reste vraie pour $p < n(G)$.

9. $\boxed{\text{nbc}\,(Gex_1) = 3}$ car 0, 3 et 4 doivent avoir trois couleurs différentes, et 1 et 2 peuvent être de la couleur de 0.

   Pour $p < 3$: $f_c(Gex_1, p) = 0$.

   Pour $p \geq 3$: 3 et 2 peuvent être de n'importe quelle couleur : p choix chacun. 0 et 1 doivent être d'une couleur différente de celle de 3 : $p-1$ choix. 4 est de toute couleur sauf celles de 0 et 3 : $p-2$ choix.

   Ainsi : $\boxed{f_c(Gex_1, p) = p^2(p-1)^2(p-2)}$ pour $p \geq 3$ et cette formule reste valable pour $p < 3$.

IV - Les applications H et K.

10. `let prem_voisin g s = List.hd (voisins g s) ;;`

    puisque `voisins g s` est rangé par ordre croissant.

11.
```
let prem_ni g =
   let rec non_isole s = if (voisins g s =[]) then non_isole (s+1) else s
in non_isole 0 ;;
```

12. On écrit une fonction auxiliaire `filtre` qui prend un prédicat sous forme d'une fonction `p : 'a -> bool` et une liste, et renvoie la sous-liste constituée des éléments qui ne vérifient pas le prédicat. On l'utilise ensuite dans `h` pour éliminer les arêtes entre s_1 et s_2 (avec `p = f`).

```
let rec filtre p = function
  |[] -> []
  |t::q when p t -> filtre p q
  |t::q -> t :: (filtre p q) ;;

let h g = let s1 = prem_ni g in let s2 = prem_voisin g s1 in
   let f {a = x ; b = y} = (x=s1 && y=s2) || (x=s2 && y=s1) in
     let aprime = filtre f g.al in {n = g.n ; al = aprime} ;;
```

13. La fonction auxiliaire `g` renumérote correctement les états.

```
let g s1 s2 = function
    |s when s < s2 -> s
    |s when s = s2 -> s1 ;
    |s -> s-1 ;;

let k g = let s1 = prem_ni g in let s2 = prem_voisin g s1 in
  let f {a = x; b = y} = {a = g s1 s2 x ; b = g s1 s2 y} in
     let gprime = h g in {n = g.n - 1 ; al = applique f gprime.al} ;;
```