

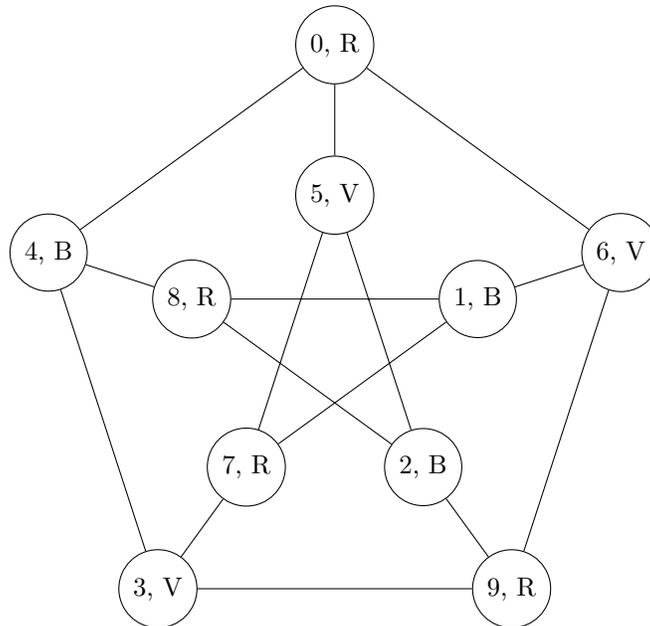
I Coloriage

Question 1 Le premier graphe admet deux sommets adjacents qui ont la même couleur ; l'étiquetage proposé n'est pas un coloriage. Cependant il admet un 3-coloriage avec l'étiquetage du second graphe donc il est colorié au sens du sujet.

Le second est colorié car l'étiquetage proposé est un 3-coloriage.

Question 2 Le graphe contient des cycles de longueur impaire, par exemple (0, 4, 3, 9, 6) : il n'est pas 2-coloriable. En effet il faudrait que les couleurs soient alternées dans le cycle et on arriverait à deux couleurs égales pour les sommets 0 et 6 qui sont reliés.

Par contre il est 3-coloriable :



Question 3

```

let est_col gphe etiq =
  let n = Array.length gphe in
  let p = Array.length etiq in
  let coloriable = ref true in
  if n <> p
  then coloriable := false
  else for i = 0 to (n-2) do
    for j = (i+1) to (n-1) do
      if gphe.(i).(j) && etiq.(i) = etiq.(j)
      then coloriable := false
    done
  done;
!coloriable
;;

```

Question 4 Un graphe de n sommet possède un n -coloriage, il suffit de donner une couleur distincte à chaque sommet.

Pour tester son nombre chromatique il suffit de tester, pour k variant de 2 à $n - 1$ s'il admet un coloriage à k couleurs.

Pour cela on peut tester tous les coloriage : il y en a k^n .

La complexité est alors majorée par

$$An^2 \left(\sum_{k=2}^{n-1} k^n \right) \leq An^2 . n . n^n = A2^{(n+3) \log_2(n)} \leq B2^{n^2}$$

la complexité est exponentielle.

II 2-coloriage

Question 5 Si G est biparti, alors ses sommets peuvent se diviser en deux sous-ensembles T et U . On attribue alors la couleur 1 aux sommets de T , et 2 aux sommets de U . La propriété de coloration est alors instantanément vérifiée.

Inversement, si G possède une 2-coloration, alors on peut appeler T les sommets recevant la couleur 1 et U les sommets recevant la couleur 2. Aucune arête ne peut relier deux sommets de couleur 1 (ou 2), et les arêtes vont donc d'un sommet de T vers un sommet de U .

Question 6 Le sous-programme `explo i k` réalise le parcours en profondeur à partir du sommet `i`, en fixant sa couleur à `k`.

```

let deux_col gphe =
  let n = Array.length gphe in
  let etiq = Array.make n (-1) in
  let rec explo i k =
    etiq.(i) <- k;
    for j = 0 to n-1 do
      if gphe.(i).(j) && etiq.(j) = -1 then
        explo j (1-k)
    done
  in
  for i = 0 to n-1 do
    if etiq.(i) = -1 then
      explo i 0 ;
  done;
  etiq
;;

```

Ici, si le graphe n'est pas 2-coloriable, alors le programme renvoie une coloration fausse (on ne détecte pas les erreurs si le sommet j est déjà colorié).

Le programme `explo` ne peut être lancé qu'une seule fois par sommet au maximum, et sa complexité est en $O(n)$. On arrive donc à une complexité en $O(n^2)$ dans le pire des cas.

III Algorithmes gloutons

Question 7 Avec le premier ordre, on trouve comme colorations pour les sommets : (0; 0; 0; 0; 1; 1; 1; 2; 2; 2), donc trois couleurs.

Avec le second, on trouve comme colorations : (0; 3; 0; 2; 1; 1; 1; 0; 2; 3), donc quatre couleurs.

Question 8

```

let min_couleur_possible gphe etiq s =
  let n = Array.length gphe in
  let coul = Array.make n false in
  for i = 0 to (n-1) do
    if gphe.(s).(i) && etiq.(i) <> -1
    then coul.(etiq.(i)) <- true
  done;
  let c = ref 0 in
  while coul.(!c) do c := !c + 1 done;
  !c
;;

```

Question 9

```

let glouton gphe num =
  let n = Array.length gphe in
  let coul = Array.make n (-1) in
  for i = 0 to (n-1) do
    let k = num.(i) in
    coul.(k) <- min_couleur_possible gphe coul k
  done;
  coul
;;

```

Question 10 La fonction `min_couleur_possible` renvoie une couleur qui n'a pas été affectée aux voisins du sommet passé en paramètre. Lorsqu'une couleur est affectée à un sommet elle ne pourra plus être affectée aux voisins qui suivent dans l'ordre de numération. De plus chaque sommet reçoit une couleur lorsque `num` est un ordre de numération.

On a bien construit un coloriage du graphe.

Dans l'algorithme glouton chaque sommet est colorié par une couleur non employée par ses voisins déjà coloriés, comme il admet au plus $d(G)$ voisins, la couleur choisie est la plus petite parmi une ensemble d'au plus $d(G)$ entiers positif : elle est donc majorée par $d(G)$.

Ainsi la coloration construite admet au plus $d(G) + 1$ couleurs.

Question 11 Soit L un coloriage de G . On considère une numération des sommets qui les classe par numéro de couleur croissante.

Comme deux sommets de même couleur ne sont pas adjacents, lors de l'appel de `min_couleur_possible` les seuls sommets voisins de s qui seront considérés auront une couleur strictement inférieure à celle de s , $L(s)$. La couleur choisie, $L'(s)$, ne pourra donc pas être strictement supérieure à $L(s)$: on a $L'(s) \leq L(s)$ pour tout s .

Si on part d'un coloriage optimal pour construire la numération des sommets on aboutit donc à un coloriage dont le maximum est majoré par celui du coloriage optimal : ce sera donc aussi un coloriage optimal.

Question 12 Comme l'algorithme `glouton` est de complexité quadratique on peut choisir un algorithme lui-même quadratique pour construire une numération des sommets sans augmenter la complexité. On commence par construire un tableau avec les degrés.

```

let degres gphe =
  let n = Array.length gphe in
  let deg = Array.make n 0 in
  for i = 0 to (n-1) do
    for j = 0 to (n-1) do
      if gphe.(i).(j)
      then deg.(i) <- deg.(i) + 1
    done
  done;
deg
;;

```

Les outils utilisés ensuite (dont `range` pour ceux qui veulent tester le code)

```

let echanger tab i j =
  let temp = tab.(i) in
  tab.(i) <- tab.(j);
  tab.(j) <- temp
;;

let range n =
  Array.init n (fun x -> x)
;;

```

La création de la numération suit la méthode du tri par sélection

```

let tri_degre gphe =
  let n = Array.length gphe in
  let deg = degres gphe in
  let num = range n in
  for i = 0 to (n-2) do
    let max = ref i in
    let deg_max = ref deg.(num.(i)) in
    for j = (i+1) to (n-1) do
      let d = deg.(num.(j)) in
      if d > !deg_max
      then (max := j; deg_max := d)
    done;
    echanger num i !max
  done;
  num
;;

```

Il suffit alors de tout rassembler.

```

let welsh_powell gphe =
  glouton gphe (tri_degre gphe)
;;

```

IV Algorithme de Widgerson

Question 13 Supposons que le graphe G est $(k+1)$ -coloriable, et soit s un de ses sommets. On considère une $(k+1)$ -coloration de G . Alors aucun sommet de $V(s)$ n'est de la même couleur que s (ce serait contradictoire). Sur le sous-graphe induit par $V(s)$, cette coloration reste valide, et n'utilise donc que k couleurs au maximum (celle de s n'est pas employée).

Notons au passage qu'on a forcément $s \notin V(S)$, sans quoi aucune coloration ne serait possible.

Question 14 Fixons les notations.

On pose $G_0 = G$ de taille n .

L'étape (a) se décompose en étapes.

1. Si G_i admet un sommet de degré supérieur à \sqrt{n} ,
2. on choisit un tel sommet, s_i , par exemple celui de degré maximum,
3. on colorie avec les couleurs $2i$ et $2i + 1$ $V(s_i)$, ce qui est possible d'après la question précédente. En effet G_i est induit d'un graphe 3-coloriable donc est 3-coloriable d'où $V(s_i)$ est 2-coloriable.
4. On définit G_{i+1} comme le graphe induit de G_i en enlevant les sommets appartenant à $V(s_i)$.

Après au plus \sqrt{n} itérations G_p n'admet plus de sommet de degré supérieur à \sqrt{n} et on le colorie par l'algorithme glouton.

On a ainsi partitionné l'ensemble des sommets : $S = S_0 \cup S_1 \cup \dots \cup S_p$,

où $S_i = V(s_i)$ pour $i < p$ et S_p est l'ensemble des sommets de G_p .

Chaque S_i admet un coloriage :

1. pour $i < p$, $V(s_i)$ est colorié par $2i$ et $2i + 1$,
2. G_p est colorié par q couleurs qui sont supérieures ou égales à $2p$. De plus, d'après la question 10, on a $q \leq \sqrt{n} + 1$.

Comme chaque partie est coloriée par des couleurs distinctes on obtient un coloriage de G avec $2p + q$ couleurs qui vérifient $2p + q \leq 2\sqrt{n} + \sqrt{n} + 1 = \mathcal{O}(\sqrt{n})$.

Question 15

```

let sous_graphe gphe sg =
  let p = Array.length sg in
  let ss_gphe = Array.make_matrix p p false in
  for i = 0 to (p-1) do
    let si = sg.(i) in
    for j = 0 to (p-1) do
      let sj = sg.(j) in
      ss_gphe.(i).(j) <- gphe.(si).(sj)
    done
  done;
  ss_gphe
;;

```

Question 16

```

let voisins_non_colories gphe etiq s =
  let n = Array.length gphe in
  let vnc = ref [] in
  for j = 0 to (n-1) do
    if gphe.(s).(j) && etiq.(j) = -1
    then vnc := j :: !vnc
  done;
  !vnc
;;

let degre_non_colories gphe etiq s =
  List.length (voisins_non_colories gphe etiq s)
;;

```

Question 17 A-t-on besoin du graphe ?

```

let non_colories gphe etiq =
  let n = Array.length gphe in

```

```

let nc = ref [] in
for i = 0 to (n-1) do
  if etiq.(i) = -1
  then nc := i :: !nc
done;
!nc
;;

```

Question 18 Pour ne pas écrire un programme démesurément long, je sors quelques fonctions. La première détermine le degré en sommets non coloriés maximal. Le test de majoration de \sqrt{n} est renvoyé par le type optionnel : la fonction renvoie `None` s'il n'existe pas de sommets avec suffisamment de voisins non coloriés et sinon elle renvoie un sommet k vérifiant cette propriété sous la forme `Some k`.

```

let sous_degre_maximum gphe etiq =
  let n = Array.length gphe in
  let r = int_of_float (sqrt (float_of_int n)) in
  let deg_max = ref (degre_non_colories gphe etiq 0) in
  let ind_max = ref 0 in
  for i = 1 to (n-1) do
    let d = degre_non_colories gphe etiq i in
    if d > !deg_max
    then (deg_max := d; ind_max := i)
  done;
  if !deg_max > r then Some !ind_max else None
;;

```

La seconde modifie le tableau des couleurs (`etiq`) à partir des couleurs calculées (`ss_etiq`) pour un sous-graphe (`sg`)

```

let ajouter_couleurs etiq sg ss_etiq =
  let p = Array.length sg in
  for i = 0 to (p-1) do
    etiq.(sg.(i)) <- ss_etiq.(i)
  done
;;

```

Le programme consiste alors à répéter la recherche de sommets avec suffisamment de voisins non coloriés tant qu'il en existe.

```

let wigderson gphe =
  let n = Array.length gphe in
  let etiq = Array.make n (-1) in
  let rec aux () =
    match sous_degre_maximum gphe etiq with
    |None -> let sg = Array.of_list (non_colories gphe etiq) in
              let ss_gphe = sous_graphe gphe sg in
              let ss_etiq = glouton ss_gphe (range (Array.length
              ss_gphe)) in
              ajouter_couleurs etiq sg ss_etiq
    |Some i -> let sg = Array.of_list (voisins_non_colories gphe
              etiq i) in
               let ss_gphe = sous_graphe gphe sg in
               let ss_etiq = deux_col ss_gphe in
               ajouter_couleurs etiq sg ss_etiq;
               aux ()
  in aux (); etiq
;;

```

Toutes les fonctions auxiliaires sont de complexité polynomiale en la taille du graphe et elles sont appelées au plus \sqrt{n} fois : la complexité est polynomiale.

Je ne comprends pas la demande de justification des propriétés, c'est ce qui a été fait à la question 14.

Question 19 Comme on sait définir un coloriage dans le cas d'un graphe 3-coloriable on peut poursuivre le raisonnement de manière semblable.

Si un graphe est 4-coloriable, les voisins d'un sommets sont 3-coloriables ; on va donc pouvoir définir un coloriage pour les "gros" voisinages puis conclure par un algorithme glouton.

On considère la propriété $\mathcal{P}(k)$: on peut définir un coloriage de $A_k.n^{a_k}$ couleurs d'un graphe k -coloriable avec $a_k < 1$.

$\mathcal{P}(2)$ est vraie avec $a_2 = 1$ et $A_2 = 2$.

$\mathcal{P}(3)$ est vraie avec $a_3 = \frac{1}{2}$ et $A_3 = 3$.

On suppose $\mathcal{P}(k)$ vraie. G est un graphe $k + 1$ -coloriable.

Pour chaque sommet s de G ayant au moins n^r voisins pas encore coloriés on colorie ceux-ci avec au plus $A_k.n^{a_k}$ couleurs non encore utilisées. On colorie le reste avec au plus $n^r + 1$ couleurs non utilisées.

Le nombre de sommets avec plus de n^r voisins est au plus $\frac{n}{n^r}$ donc on a utilisé au plus $n^{1-r}.A_k.n^{a_k} + n^r$ couleurs.

On peut choisir r tel que l'expression ci-dessus devienne homogène : $1 - r + a_k = r$ donc $r = \frac{1+a_k}{2}$.

On obtient ainsi un coloriage avec au plus $(A_k + 1).n^r$ couleurs.

La récurrence $a_{k+1} = \frac{1+a_k}{2}$ avec $a_2 = 0$ donne $a_k = 1 - 2^{2-k}$. donc la propriété $\mathcal{P}(k)$ est donc vraie avec $A_k = k$ et $a_k = 1 - 2^{2-k}$.

Dans le calcul ci-dessus on n'a pas tenu compte du fait qu'on appliquait $\mathcal{P}(k)$ à des graphes de tailles inférieures à n .

Si les tailles auxquelles on applique $\mathcal{P}(k)$ sont m_1, m_2, \dots, m_p , le nombre de couleurs employées est en fait

majoré par $\sum_{i=1}^p A_k.m_i^{a_k} + n^r$

La fonction $x \mapsto n^{a_k}$ est concave donc $\frac{\sum_{i=1}^p m_i^{a_k}}{p} \leq \left(\frac{\sum_{i=1}^p m_i}{p} \right)^{a_k}$.

Ainsi $\sum_{i=1}^p A_k.m_i^{a_k} + n^r \leq p^{1-a_k} A_k \left(\sum_{i=1}^p m_i \right)^{a_k} + n^r$.

Comme on a $\sum_{i=1}^p m_i \leq n$ et $p \leq n^{1-r}$, le nombre de couleurs utilisées est majoré par $n^{(1-r)(1-a_k)} A_k n^{a_k} + n^r$.

On choisit $a_{k+1} = r$ tel que $(1-r)(1-a_k) + a_k = r$ d'où $r = \frac{1}{2-a_k}$.

$a_2 = 0$ donne bien $a_3 = \frac{1}{2}$.

On calcule ensuite $a_4 = \frac{2}{3}$, $a_5 = \frac{3}{4}$ et, par récurrence, $a_k = \frac{k-2}{k-1} = 1 - \frac{1}{k-1}$ qui donne moins de couleurs que la valeurs $1 - 2^{2-k}$.