

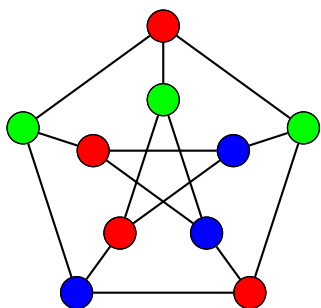
## I Coloration de graphe par algorithme glouton

Soit  $G = (V, E)$  un graphe. Une  $k$ -coloration de  $G$  est une fonction  $c : V \mapsto \{0, \dots, k-1\}$  vérifiant :

$$\forall \{u, v\} \in E, c(u) \neq c(v)$$

1. Trouver une 3-coloration du graphe de Petersen :

**Solution :**



2. Montrer que ce graphe n'est pas 2-coloriable.

**Solution :** Le graphe de Petersen contient plusieurs cycles de longueur 5. On remarque facilement qu'il n'est pas possible de colorier un tel cycle avec 2 couleurs (ni d'ailleurs n'importe quel cycle de longueur impaire).

On appelle le **nombre chromatique**  $\chi(G)$  le plus petit  $k$  tel que  $G$  soit  $k$ -coloriable. Par exemple, le nombre chromatique du graphe de Petersen est 3.

On considère un algorithme glouton de coloriage :

$C \leftarrow \emptyset$

Pour chaque sommet  $v$  (dans un ordre quelconque) :

Si une couleur de  $C$  n'est utilisée par aucun voisin de  $v$

:

Donner à  $v$  cette couleur

Sinon :

Ajouter une nouvelle couleur à  $C$  et l'utiliser pour  $v$

3. Montrer sur un exemple simple que le coloriage obtenu n'est pas forcément optimal.

**Solution :** Si on colorie un chemin à 4 sommets en commençant par les deux extrémités, l'algorithme glouton va leur donner la même couleur. Il va ensuite être obligé de donner 2 nouvelles couleurs aux deux autres sommets, ce qui donne 4 couleurs qui n'est pas l'optimum (on peut le colorier avec 2 couleurs).



On souhaite créer un emploi du temps pour une journée : chaque cours possède une heure de début et de fin et doit être assigné à une salle. Il ne peut pas y avoir 2 cours en même temps dans la même salle. L'objectif est de minimiser le nombre de salles à utiliser.

4. Modéliser ce problème sous forme de coloriage de graphe.

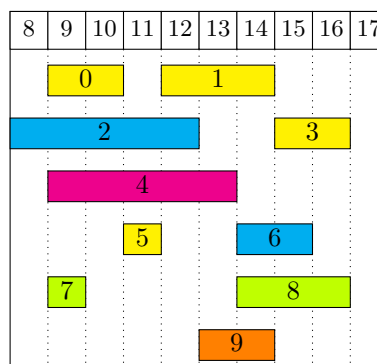
**Solution :** On définit  $G = (V, E)$  où

- $V$  est l'ensemble des cours (chaque sommet est un intervalle de temps)
- $E$  représente les chevauchements de cours (deux sommets sont reliés si les intervalles de temps s'intersectent)

Dans l'algorithme glouton précédent, on considère maintenant les sommets par **ordre de début de cours croissant** (on regarde d'abord le cours qui se termine le plus tôt).

5. Appliquer ce nouvel algorithme glouton sur le diagramme de Gantt suivant :

**Solution :** On trouve qu'un a besoin de 4 salles différentes.



6. Soit  $k$  le nombre maximum de cours se déroulant pendant la même heure. Montrer que le nombre chromatique du graphe de la question 4 est au moins  $k$ .

**Solution :** Il faut colorier chacun de ces cours avec une couleur différente.

7. Montrer que l'algorithme glouton donne un coloriage à  $k$  couleurs et est donc optimal.

**Solution :** Supposons par l'absurde que l'algorithme glouton utilise au moins  $k+1$  couleurs et considérons le premier intervalle  $I$  pour lequel l'algorithme utilise  $k+1$  couleurs.

Soit  $i$  l'heure de début de  $I$ . Comme l'algorithme utilise  $k+1$  couleurs pour  $I$ , il existe  $k$  intervalles déjà coloriés qui intersectent  $I$ . De plus ces intervalles commencent avec  $i$  car ils ont été considérés avant dans l'algorithme. Donc il y a  $k+1$  intervalles qui contiennent  $i$ .

Ceci contredit la définition de  $k$ .

8. Est-ce que l'algorithme glouton serait optimal si on triait par ordre de **fin croissant**?

On dit qu'un graphe est **biparti** s'il ne possède pas de cycle de longueur impaire.

9. Montrer qu'un graphe biparti est 2-coloriable en donnant un algorithme pour trouver un 2-coloriage.

**Solution :** On fait un parcours en largeur depuis un

sommet de départ  $s$  quelconque, en donnant la couleur 0 à  $s$ , 1 à ses voisins, puis en alternant les couleurs. Par propriété de parcours en largeur, on colorie en 0 tous les sommets à distance paire de  $s$  et en 1 tous les sommets à distance impaire. Ainsi, deux sommets  $u$  et  $v$  coloriés de la même couleur ne peuvent pas être adjacents : sinon, le chemin composé du chemin de  $s$  à  $u$ , puis de  $\{u, v\}$ , puis du chemin de  $v$  à  $s$  serait de longueur impaire.

## II Dichotomie 2D

On définit le type de structure suivante :

```
typedef struct coord {
    int i;
    int j;
} coord;
```

On considère une matrice d'entiers  $m$  dont les éléments sont triés en colonne de haut en bas et de gauche à droite. Par exemple :

$$\begin{pmatrix} 0 & 4 & 8 & 12 & 16 & 20 \\ 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \end{pmatrix}$$

1. Écrire en C une fonction `dicho_matrice(x, m, n, p)` qui prend en argument un entier  $x$  et une matrice  $m$  à  $n$  lignes et  $p$  colonnes d'entiers triés en colonne de haut en bas et de gauche à droite, et qui renvoie :
  - les coordonnées  $(i, j)$  (dans une structure `coord`) minimales pour l'ordre défini plus haut tel que  $x$  se trouve en ligne  $i$  et colonne  $j$ , si l'entier  $x$  est bien présent dans la matrice  $m$
  - $(-1, -1)$  si  $x$  n'est pas présent dans la matrice  $m$ .

Cette fonction devra être de complexité logarithmique en  $\max(n, p)$ .

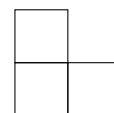
2. On suppose maintenant que chaque ligne de  $m$  et chaque colonne est rangée par ordre croissant (et non plus de haut en bas et de gauche à droite). Par exemple :

$$\begin{pmatrix} 1 & 2 & 6 & 14 \\ 4 & 5 & 7 & 17 \\ 6 & 9 & 12 & 22 \\ 7 & 11 & 13 & 42 \end{pmatrix}$$

Pour simplifier, on supposera que  $m$  est de taille  $n \times n$ , où  $n$  est une puissance de 2. Donner (en français) un algorithme pour savoir si un élément appartient à une telle matrice, en complexité  $O(\log(n))$ .

## III Pavage

On souhaite paver (c'est-à-dire recouvrir sans chevauchements) une grille carré de côté  $2^k$  avec le motif suivant ainsi que ses 3 rotations :



1. Montrer que ce problème de pavage n'est pas résoluble.
2. On choisit une case quelconque de la grille. Montrer qu'il est possible de paver toute la grille à part cette case. Pour ce faire, on pourra décrire un algorithme (en français) permettant de trouver un tel pavage.

## IV Inversions

Étant donné un tableau d'entiers  $a$ , on appelle inversion de  $a$  tout couple  $(i, j)$  tel que  $i < j$  et  $a.(i) > a.(j)$ . Par exemple, `[4; 1; 3; 2]` possède 4 inversions :  $(0, 1)$ ,  $(0, 2)$ ,  $(0, 3)$  et  $(2, 3)$ . On s'intéresse au calcul du nombre d'inversions de  $a$ .

1. Donner un algorithme en complexité quadratique pour trouver le nombre d'inversions dans un tableau de taille  $n$ .
2. Écrire une fonction `inversions_sorted` telle que, si `a1` et `a2` sont deux tableaux triés, `inversions_sorted a1 a2` renvoie le nombre de couples  $(i, j)$  tels que le `a.(i) > a.(j)`.
3. Écrire une fonction `fusion` fusionnant deux tableaux triés en un unique tableau trié.
4. En déduire une fonction `inversions` calculant le nombre d'inversion dans un tableau en  $O(\log(n))$ , en s'inspirant du tri fusion.