TP3 : Coloration

Le but de ce TP est d'évaluer les performances de l'algorithme de coloration DSATUR sur les graphes aléatoires.

Le fichier `tp3.txt` sur le serveur pédagogique contient l'entête ci-dessous que vous pouvez réutiliser ainsi qu'une fonction `colorExact(k)` qui teste, par une recherche exhaustive (donc réservée aux "petits" graphes uniquements), si un graphe admet une k-coloration.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <fstream>
#include <cmath>

using namespace std;

const int n=35; // nombre de sommets
int adj[n][n];  // matrice d'adjacence du graphe
int couleur1[n];  // couleurs des sommets pour l'agorithme exact
bool trouve=false; // permet de stopper l'algorithme exact
                   // quand une coloration  a ete trouvee
```

Exercice 1- *Création du graphe*

On se donne un nombre de sommets n et une probabilité d'arête par le biais d'un entier p (entre 0 et 100) qui correspond au pourcentage de chance d'avoir une arête entre n'importe quelle paire de sommets $\{i, j\}$.

Écrire la fonction `void generegraphe(int n, int p)` qui génère le graphe et remplit la matrice d'adjacence `adj`.

Écrire ensuite une fonction qui affiche le graphe sous forme `sommet i: voisin1 voisin2 ...`

Exercice 2- *Test de la fonction de coloration exacte*

Écrire, en utilisant la fonction `colorExact()`, une fonction `nbChromatique()` qui calcule le nombre chromatique du graphe stocké dans la matrice `adj`.

Tester cette fonction pour différentes valeurs de n et p et vérifier à partir de quel valeur de n le temps de calcul "explose".

Exercice 3- *Algorithme Glouton* Écrire une fonction `int ColorGlouton()` qui va colorier le graphe stocké dans la matrice `adj` en prenant les sommets dans l'ordre de 0 à $n-1$ et en donnant à chaque sommet la plus petite couleur disponible pour ce sommet et retourner le nombre de couleurs utilisées. Pour cela, prendre un deuxième tableau `couleur2` pour stocker les couleurs des sommets obtenues par cet algorithme. On pourra réutiliser la fonction `convient()`.

Exercice 4- *Algorithme DSATUR*

Écrire une fonction `int DSATUR()` sur le modèle de l'algorithme vu en cours qui va colorier le graphe et retourner le nombre de couleurs utilisées. Pour cela, prendre un troisième tableau `couleur3` pour stocker les couleurs des sommets obtenues par cet algorithme. On pourra réutiliser la fonction `convient()` en l'adaptant pour qu'elle teste les couleurs des sommets voisins de i inférieurs **et supérieurs** à i.

Exercice 5- *Comparaison*

Écrire une fonction `int simul(int m)` qui compare le nombre de couleurs utilisées par l'algo exact et par DSATUR pour colorier m graphes aléatoires pour n et p fixés.

Exercice 6- *Pour aller plus loin*

- Implanter une version itérée de l'algorithme Glouton ou de DSATUR, avec échange de deux couleurs, comme vu en cours.
- Dérécursifier l'algorithme exact.