TP n°3 : Graphes: parcours en largeur et plus court chemin - Partie 2.

Exercice 1.

1. Dans le nouveau fichier *matriceadj.c*, écrire une fonction *floydwarshall* implémentant l'algorithme de Floyd-Warshall à partir d'une matrice d'adjacence. On affichera comme pour Dijkstra la distance d'un sommet `src` vers tous les autres sommets. Voici le prototype de la fonction :

```
void floydwarshall(graphe *grph, int src);
```

2. Changer les fonctions *dijkstra_mat*, *dijkstra_mat_prio* et *floydwarshall* de manière à renvoyer le tableau des distances selon le prototype ci-dessous:

```
int * dijkstra_mat(graphe *grph, int src);
```

Tester ensuite le temps de calcul de ces trois fonctions pour trouver les plus courts chemins depuis **tous les sommets** sur des échantillons de graphes avec un nombre de sommets allant de 10 à 200 par pas de 10:

```
int n=10;
clock_t td, ta;
while (n<201){
    graphe *grph=remplir_graphe(n);
    // Test Floyd-Warshall
    td = clock();
    for (int i=0;i<n;i++){
        floydwarshall(grph,i);
    }
    ta=clock();
    printf ("Floyd-Warshall : %d \t \t",(int) (ta-td));
    // Test Dijkstra
    td = clock();
    for (int i=0;i<n;i++){
        dijkstra_mat(grph,i);
    }
```

```
    ta=clock();
    printf ("Dijkstra : %d \t \t",(int) (ta-td));
    // Test Dijkstra avec file de priorité
    td = clock();
    for (int i=0;i<n;i++){
        dijkstra_mat_prio(grph,i);
    }
    ta=clock();
    printf ("Dijkstra prio : %d \n",(int) (ta-td));
    n+=10;
}
```

Exercice 2. *Représentation par triplets.*

1. Dans le fichier *triplets.c*, où est implémentée la structure de graphe avec vecteur de triplets, écrire les fonctions:
   (a) *new_graph* de paramètre entier n, permettant de créer un graphe à n sommets et 0 arêtes,
   (b) *delete_graph*, qui libère un graphe en mémoire,
   (c) *add_edge* de paramètre un graphe `grph`, un entier i, un entier j et un entier `poids`, et ajoutant une arête de i vers j de ce poids dans le vecteur de triplets,
   (d) *print_aretes* prenant en paramètre un graphe, et permettant d'afficher son nombre de sommets et son vecteur de triplets,
   (e) *is_edge_in_graph* de paramètre un graphe `grph`, deux entiers i et j et teste si il y a une arête de i vers j dans `grph`, en renvoyant le poids de l'arête si oui et 0 sinon.
   (f) *remplir_graphe*, en copiant la fonction utilisée pour les listes d'adjacence.
2. En utilisant ces fonctions, implémenter l'algorithme de Dijkstra version triplet en utilisant la fonction *minDistance*.
3. Écrire une fonction *bellman_ford* implémentant l'algorithme de Bellman-Ford pour la représentation par vecteur de triplets.
4. Comparer le temps d'exécution de ces deux algorithmes (Dijkstra et Bellman-Ford) pour trouver les plus courts chemins depuis **tous les sommets** sur des échantillons de graphes avec un nombre de sommets allant de 10 à 200 par pas de 10.