Durée prévue : 2 séances d'1h30.

Les objectifs de ce sujet de TP sont :

Révisions de C allocation dynamique et gestion de la mémoire ;

Algorithmique calcul de la fermeture transitive d'un graphe et recherche des composantes connexes.

Les fonctions à implémenter (ou qui le sont déjà) sont documentées à l'adresse
https://www.math.univ-paris13.fr/~rousselin/graphe-2/

Certains exemples peuvent également vous éclairer sur l'interface que l'on veut obtenir.

1 Assassinat de la constante `GRAPHEORDREMAX`

Jusqu'ici nous avions implémenté les matrices d'adjacences en utilisant des tableaux statiques à deux dimensions, de taille `GRAPHEORDREMAX` par `GRAPHEORDREMAX`. Ceci a plusieurs inconvénients :

— si cette constante est grande mais l'ordre du graphe petit, beaucoup de place est perdue en mémoire ;
— il faut modifier cette constante (et donc recompiler la bibliothèque) chaque fois que l'on veut changer la valeur de la constante `GRAPHEORDREMAX` ;
— les tableaux statiques étant dans la pile d'exécution, on risque un dépassement de la pile si la constante `GRAPHEORDREMAX` est trop grande (sur la machine que j'utilise en ce moment, la valeur limite se situe quelque part entre 1000 et 2000 sommets).

On va donc passer des tableaux statiques aux tableaux dynamiques *en changeant le moins possible l'interface de la bibliothèque*.

Le champ `adj` de la structure `graphe` est donc maintenant un *pointeur* vers un tableau dynamique (à une dimension) de n^2 entiers.

1.1 Mise en place

Question 1: Créer un répertoire dédié à ce sujet de TP, télécharger l'archive correspondante sur l'ENT et l'extraire dans ce répertoire. Vous devriez voir les fichiers suivants :

`graphe-2.h`	nouvelle version de l'interface de la bibliothèque graphe
`graphe-2.c`	fichier source correspondant avec quelques fonctions à réécrire ou modifier
`mat_bool.h`	fichier d'entête donnant l'interface du type `mat_bool.h` (matrices booléennes)
`mat_bool.c`	fichier source correspondant, qu'on remplira dans la partie 2
`connexite.h`	fichier d'en-tête déclarant 4 fonctions ayant un lien avec la connexité
`connexite.c`	fichier source correspondant, à remplir à partir de la fin de la partie 2
`main-test-1.c`	fonction `main` de test pour la première partie
`main-test-2.c`	fonction `main` de test pour la deuxième partie
`main-comp-conn.c`	fonction `main` de test pour la conclusion de ce sujet (composantes connexes)
`Makefile`	pour compiler tout ce petit monde sans effort.

Taper la commande `make`, pour vérifier qu'il n'y a pas d'erreur de compilation (bien qu'aucun des programmes produits ne soit pour l'instant intéressant).

--- * ---

1.2 Fonction `graphe_stable`

Question 2: Réécrire entièrement la fonction `graphe_stable` de façon à ce qu'elle réponde à la spécification donnée dans la documentation (ou dans `graphe-2.h`). On utilisera, pour l'allocation dynamique l'une des deux fonctions suivantes :

```
void *malloc(size_t nombre_octets);
void *calloc(size_t nombre_elts, size_t taille_elt);
```

La première réserve `nombre_octets` octets consécutifs en mémoire et retourne un pointeur vers l'adresse du premier de ces octets.

La seconde réserve `nombre_elts * taille_elt` octets consécutifs en mémoire, *initialisés à 0* et retourne un pointeur vers l'adresse du premier de ces octets.

Il est courant, lorsqu'on utilise ces fonctions, d'utiliser l'opérateur `sizeof`, qui, appliqué à un type, retourne la taille en nombre d'octets d'une variable de ce type (par exemple `sizeof(int)` est, sur les machines que vous utilisez, transformé en `4` à la compilation).

--- * ---

1.3 Fonction `graphe_detruire`

La fonction `graphe_detruire` libère la mémoire occupée par la matrice d'adjacence d'un graphe dont l'adresse est passée en argument.

Question 3: En utilisant la fonction

```
void free(void *ptr);
```

qui libère la mémoire préalablement allouée avec `malloc` ou `calloc` à l'adresse `ptr`, écrire la définition de la fonction `graphe_detruire`.

Remarques :

— `free(NULL)` se contente de *ne rien faire* ;

— Si `ptr` n'est pas la valeur de retour d'un appel précédent de `malloc`, `calloc` (ou `realloc`), l'appel `free(ptr)` aboutit le plus souvent à une erreur de segmentation.

--- * ---

1.4 Trois autres fonctions à modifier

Comme le tableau `adj` n'est plus un tableau statique à deux dimensions, on ne peux plus avoir recours à la syntaxe `adj[v][w]` pour désigner le coefficient de la ligne `v` et colonne `w`.

À la place, on considère que *les lignes de la matrice sont les unes à la suite des autres* dans un grand tableau à une seule dimension.

Ainsi, `adj[0]`, `adj[1]`, ..., `adj[n-1]` désignent les coefficients de la première ligne, `adj[n]`, `adj[n+1]`, ..., `adj[2*n - 1]` ceux de la deuxième, etc.

Question 4: Modifier les fonctions `graphe_ajouter_arete`, `graphe_supprimer_arete` et `graphe_get_multiplicite_arete` de manière à transformer les indices doubles (du type `adj[u][v]`) en indices simples (du type `adj[truc]`).

--- * ---

1.5 Autres fonctions et test

Question 5: Vérifier qu'aucune autre fonction du fichier `graphe-2.c` n'a besoin d'être modifiée.

--- * ---

Question 6: Lire le fichier `main-test-1.c`, compiler, puis exécuter le premier programme de test `./test-1`.

--- * ---

2 Le type `mat_bool` et la matrice d'adjacence booléenne

L'interface du type `mat_bool` est donnée à l'adresse
https://www.math.univ-paris13.fr/~rousselin/graphe-2/mat__bool_8h.html
(ou dans le fichier `mat_bool.h`).

Question 7: Compléter le fichier `mat_bool.c` en implémentant les 6 fonctions manquantes.

On fera attention au fait que la fonction `mat_bool_creer` alloue *à la fois* de la mémoire pour une variable de type `mat_bool` et son tableau `tab`.

La fonction `mat_bool_detruire` devra donc faire deux libérations de mémoire dont l'ordre est important.

--- * ---

La matrice d'adjacence booléenne d'un graphe $G = (V, E)$ est la matrice carrée $B = (b_{u,v})_{u,v\in V}$ définie par

$$b_{u,v} = \begin{cases} 1 & \text{s'il existe une arête entre } u \text{ et } v\,; \\ 0 & \text{sinon.} \end{cases}$$

Question 8: Dans le fichier `connexite.c`, implémenter la fonction `mat_bool_creer_adj` à l'aide des fonctions définies à la question précédente.

--- * ---

Question 9: Lire le fichier `main-test-2.c` pour voir son contenu, compiler, puis exécuter le programme `./test-2` pour tester les fonctions implémentées dans cette partie.

--- * ---

3 Clôture transitive et composantes connexes

La clôture (ou fermeture) transitive d'un graphe G peut être représentée par la matrice $T = (t_{u,v})_{u,v\in V}$ définie par

$$t_{u,v} = \begin{cases} 1 & \text{s'il existe une chaîne entre } u \text{ et } v \text{ dans } G\,; \\ 0 & \text{sinon.} \end{cases}$$

Remarque : il y a toujours une chaîne (de longueur 0) entre un sommet et lui-même, donc $t_{u,u} = 1$ pour tout sommet u.

3.1 Clôture transitive et algorithme de Warshall

L'algorithme de Warshall est un algorithme de programmation dynamique permettant de calculer la clôture transitive d'un graphe, avec une complexité en $O(n^3)$.

Pour une chaîne

$$(v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1}),$$

de longueur $k \geq 2$, on dit que les sommets v_2, ..., v_k en sont les *sommets intermédiaires*. Une chaîne de longueur 0 ou 1, n'a pas de sommet intermédiaire.

L'algorithme de Warshall consiste à calculer successivement les matrices booléennes $C^{(p)}$ pour $p = 0, 1, \ldots, n-1$, où $C^{(p)} = (c^{(p)}_{u,v})_{u,v\in V}$ est défini par

$$c^{(p)}_{u,v} = \begin{cases} 1 & \text{s'il existe une chaîne entre } u \text{ et } v \text{ dont tous les sommets intermédiaires sont dans } \{0, 1, \ldots, p\}\,; \\ 0 & \text{sinon.} \end{cases}$$

La matrice de départ est $C^{(-1)}$, définie par

$$c^{(-1)}_{u,v} = \begin{cases} 1 & \text{s'il existe une chaîne entre } u \text{ et } v \text{ sans sommet intermédiaire}\,; \\ 0 & \text{sinon.} \end{cases}$$

Question 10: Pour $p = 0, 1, \ldots, n-1$, exprimer les coefficients de la matrice $C^{(p)}$ en fonction de ceux de $C^{(p-1)}$.

Justifier que pour tout u, v dans V, $c^{(p)}_{u,p} = c^{(p-1)}_{u,p}$ et $c^{(p)}_{p,v} = c^{(p-1)}_{p,v}$.

En déduire l'algorithme de Warshall « en place », permettant de calculer la fermeture transitive d'un graphe n'utilisant en mémoire qu'une seule matrice booléenne.

--- * ---

Question 11: Définir dans `connexite.c` la fonction `mat_bool_creer_clot_trans`, respectant la spécification donnée dans la documentation.

Tester votre fonction en modifiant le fichier `main-test-2.c`.

--- * ---

3.2 Recherche et affichage des composantes connexes

Question 12: En utilisant la fonction `mat_bool_creer_clot_trans`, écrire dans le fichier `connexite.c` la définition de la fonction `graphe_creer_comp_conn` qui retourne un pointeur vers un tableau dynamique de n entiers tel que pour chaque indice i entre 0 et $n-1$, l'élément d'indice i du tableau est le plus petit représentant de la composante connexe de i dans le graphe pointé par `g`.

--- * ---

Question 13: En utilisant la fonction `graphe_creer_comp_conn`, écrire dans le fichier `connexite.c` la définition de la fonction `graphe_afficher_comp_conn` qui affiche sur la sortie standard les composantes connexes du graphe dont l'adresse est passée en argument.

Indice : avec le choix fait à la question précédente, quand est-ce qu'un sommet représente sa propre composante connexe ?

--- * ---

3.3 Bonus : calcul de la clôture transitive par l'algorithme « naïf »

L'algorithme de Warshall est une amélioration par rapport à l'algorithme « naïf » (qui ne l'est pas tant que cela) suivant.

Pour $k = 0, 1, \ldots$, on définit la matrice $B^{(k)} = (b^{(k)}_{u,v})_{u,v \in V}$, par

$$b^{(k)}_{u,v} = \begin{cases} 1 & \text{s'il existe un chemin de longueur } k \text{ entre } u \text{ et } v\,; \\ 0 & \text{sinon.} \end{cases}$$

Question 14: Que sont les matrices $B^{(0)}$ et $B^{(1)}$? Exprimer une relation de récurrence entre les coefficients de $B^{(k+1)}$ et ceux de $B^{(k)}$, pour $k \geq 0$.

En déduire un algorithme pour calculer la clôture transitive d'un graphe, en utilisant les matrices $B^{(k)}$.

--- * ---

Question 15: Implémenter cet algorithme. Comparer le temps d'exécution avec la fonction utilisant l'algorithme de Warshall pour des graphes de taille importante.

--- * ---