

Plan de codage — Partie 2 (Répartition par membre)

Tarjan (CFC) + Diagramme de Hasse + Propriétés (transitoire/persistante/absorbante/irréductible)

Étape	Membre 1 — tarjan_main.c	Membre 2 — tarjan.c / tarjan.h	Membre 3 — hasse.c / hasse.h	Membre 4 — partition.c / partition.h
1. Initialisation	Configurer le binaire <code>`graph_part2`</code> (Makefile/CMake). Inclure tarjan.h, hasse.h, partition.h.	Créer tarjan.h avec structures (TarjanMeta, Stack) et prototypes.	Créer hasse.h avec structures (Link, LinkArray) et prototypes.	Créer partition.h avec Class, Partition (v2c) et prototypes.
2. Structures & prototypes	Déclarer les fonctions à appeler depuis main : <code>run_tarjan()</code> , <code>run_hasse()</code> .	Implémenter TarjanMeta, pile (push/pop/top/empty) et utilitaires.	Définir <code>build_links()</code> , <code>removeTransitiveLinks()</code> , <code>hasse_export_mermaid()</code> .	Définir <code>Class{verts,size}</code> , <code>Partition{classes,count,v2c}</code> ; <code>partition_add_class()</code> , <code>free_partition()</code> .
3. Lecture du graphe (réutilisation)	Réutiliser <code>adj_read_file()</code> de la Partie 1 pour charger l'adjacence.	—	—	—
4. Lancement de Tarjan	Initialiser structures, appeler <code>tarjan_run(graph,h,&partition;)</code> , gérer affichage.	Implémenter DFS : <code>index/lowlink</code> , pile, détection composantes, ajout à partition.	—	Créer et remplir les Class dans Partition (ajout ordonné des sommets).
5. Affichage des classes	Afficher : <code>Component Ck: {...}</code> (ordre indifférent, contenu exact requis).	Vérifier que chaque sommet appartient à une seule classe.	—	Gérer numérotation C1, C2... et mapping <code>vertex→class (v2c)</code> .
6. Construction du Hasse	Appeler <code>build_links(graph,partition)</code> et stocker les liens.	—	Pour chaque arête ($u→v$), déterminer ($C_i→C_j$) si $C_i≠C_j$, éviter doublons.	Fournir v2c pour la détection des liens entre classes.
7. Nettoyage des liens (optionnel)	—	—	<code>removeTransitiveLinks(&links;)</code> pour supprimer les liens redondants.	—
8. Export Mermaid	Appeler <code>hasse_export_mermaid(partition, links)→classes.mmd</code> .	—	Générer nodes $C1..Cn$ et arêtes $Cx→Cy$ (directes seulement).	—
9. Analyse des propriétés	Appeler <code>analyze_classes(partition, links)</code> et afficher le statut de chaque classe.	—	—	Implémenter la logique : transitoire (sortie), persistante (aucune sortie), absorbante (persistante et 1 sommet), irréductible (1 seule classe).
10. Tests & validation	Compiler/exécuter sur fichiers d'exemple ; vérifier sorties et .mmd.	Vérifier la détection correcte des CFC (cas borderline).	Vérifier cohérence des liens et rendu Mermaid.	Vérifier propriétés (transitoire/persistante/absorbante/irréductible) et cohérence v2c.

11. Livraison finale	Livrer exécutable graph_part2, README, capture Mermaid.	Code commenté, sans warnings.	Fichier .mmd valide visuellement (mermaid.live).	Libérations mémoire propres (valgrind).
----------------------------	---	----------------------------------	--	--