

# 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 `graph_part2` (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 : run_tarjan(), run_hasse().	Implémenter TarjanMeta, pile (push/pop/top/empty) et utilitaires.	Définir build_links(), removeTransitiveLinks(), hasse_export_mermaid().	Définir Class{verts,size}, Partition{classes,count,v2c}; partition_add_class(), free_partition().
3. Lecture du graphe (réutilisation )	Réutiliser adj_read_file() de la Partie 1 pour charger l'adjacence.	—	—	—
4. Lancement de Tarjan	Initialiser structures, appeler tarjan_run(graph,&partition);, gérer affichage.	Implémenter DFS : index/lowlink, pile, détection composantes, ajout à partition.	—	Créer et remplir les Class dans Partition (ajout ordonné des sommets).
5. Affichage des classes	Afficher : Component Ck: {...} (ordre indifférent, contenu exact requis).	Vérifier que chaque sommet appartient à une seule classe.	—	Gérer numérotation C1, C2... et mapping vertex→class (v2c).
6. Construction du Hasse	Appeler build_links(graph, partition) et stocker les liens.	—	Pour chaque arête (u→v), déterminer (Ci→Cj) si Ci≠Cj, éviter doublons.	Fournir v2c pour la détection des liens entre classes.
7. Nettoyage des liens (optionnel)	—	—	removeTransitiveLinks(&links;) pour supprimer les liens redondants.	—
8. Export Mermaid	Appeler hasse_export_mermaid(partition, links) → classes.mmd.	—	Générer nodes C1..Cn et arêtes Cx --> Cy (directes seulement).	—
9. Analyse des propriétés	Appeler analyze_classes(partition, links) 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).
----------------------------	---	----------------------------------	--	--