

Université Mohammed V
Ecole Nationale Supérieure de l'Informatique et de l'Analyse des Systèmes
Département Génie Logiciel

Algorithmes avancés en langage C

Support de TD/TP

Langage C (M—)

Semestre 1 – 2026–2027

3

Tome 3

Algorithmes : optimiser, rechercher et décider

Pr. Youness BOUTYOUR

youness.boutyour@um5.ac.ma

Table des matières

Objectifs	4
Consignes de travail	4
Partie 1 : Tris et analyse de complexité	7
Exercice 1 – Sélection et invariants	7
Exercice 2 – Insertion stable	7
Exercice 3 – Tri à bulles optimisé	7
Exercice 4 – Tri par comptage	7
Exercice 5 – Tri de chaînes	7
Exercice 6 – Tri de structures	7
Exercice 7 – Fusion de deux tableaux	7
Exercice 8 – Tri fusion récursif	8
Exercice 9 – Tri fusion sans fuite	8
Exercice 10 – Partition de Hoare	8
Exercice 11 – Tri rapide	8
Exercice 12 – Pivot médian de trois	8
Exercice 13 – Tri par tas	8
Exercice 14 – Comparer les tris	9
Exercice 15 – Tri externe par fichiers	9
Partie 2 : Recherches et diviser pour régner	12
Exercice 16 – Recherche linéaire instrumentée	12
Exercice 17 – Recherche dichotomique itérative	12
Exercice 18 – Recherche dichotomique récursive	12
Exercice 19 – Première occurrence	12
Exercice 20 – Dernière occurrence	12
Exercice 21 – Insertion dans un tableau trié	12
Exercice 22 – Recherche exponentielle	13
Exercice 23 – Recherche par interpolation	13
Exercice 24 – Racine entière	13
Exercice 25 – Puissance rapide	13
Exercice 26 – Maximum par diviser pour régner	13
Exercice 27 – Inversion d'un tableau	13

Exercice 28 – Compter les inversions	13
Exercice 29 – Sélection du k-ième élément	14
Exercice 30 – Médiane en flux	14

Partie 3 : Glouton et programmation dynamique 17

Exercice 31 – Rendu de monnaie glouton	17
Exercice 32 – Contre-exemple au glouton	17
Exercice 33 – Rendu de monnaie dynamique	17
Exercice 34 – Fibonacci mémorisé	17
Exercice 35 – Escalier	17
Exercice 36 – Sous-suite croissante	17
Exercice 37 – Distance d'édition	17
Exercice 38 – Sac à dos 0/1	18
Exercice 39 – Sac fractionnaire	18
Exercice 40 – Activités compatibles	18
Exercice 41 – Ordonnancement à une machine	18
Exercice 42 – Chemin de somme maximale	18
Exercice 43 – Parenthésage optimal	19
Exercice 44 – Produit de mots	19
Exercice 45 – Projet de planification	19

Partie 4 : Graphes et algorithmes avancés 22

Exercice 46 – Parcours en profondeur	22
Exercice 47 – Composantes connexes	22
Exercice 48 – Détection de cycle	22
Exercice 49 – Tri topologique	22
Exercice 50 – Plus court chemin non pondéré	22
Exercice 51 – Dijkstra	22
Exercice 52 – Dijkstra avec tas	22
Exercice 53 – Bellman-Ford	23
Exercice 54 – Floyd-Warshall	23
Exercice 55 – Arbre couvrant minimal	23
Exercice 56 – Prim	23
Exercice 57 – Réseau de transport	23
Exercice 58 – Coloration gloutonne	23

Exercice 59 – Chemin critique	24
Exercice 60 – Projet de réseau	24

Objectifs

- Choisir et analyser un algorithme de tri ou de recherche.
- Estimer le coût en temps et en mémoire d'un programme C.
- Utiliser la récursivité, le diviser pour régner et la programmation dynamique.
- Construire des algorithmes gloutons et des parcours avancés de graphes.

Consignes de travail

Chaque exercice doit être réalisé dans un fichier C indépendant. Compiler avec `gcc -std=c11 -Wall -Wextra -Wpedantic exercice.c -lm -o exercice`. Pour chaque algorithme, indiquer l'invariant, tester une limite et donner la complexité en temps et en mémoire. Les solutions complètes se trouvent dans la version avec corrections.

Décrire les données, l'invariant, le coût asymptotique, puis vérifier le résultat sur un cas ordinaire, un cas limite et un cas défavorable.

1 PARTIE

Tris et analyse de complexité

NOTIONS

tris par sélection, insertion, bulles, fusion, rapide et tas

COMPÉTENCES

implémenter, comparer et justifier le coût des tris

PROGRESSION

Application → maîtrise

Langage C | CHEATSHEET DE LA PARTIE 1

Tris et analyse de complexité

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Un tri réordonne les éléments sans en perdre ni en créer.
Un tri est stable s'il conserve l'ordre relatif des éléments de même clé.

Exemple : `int a[] = {4, 1, 3};` contient trois valeurs ;
après un tri croissant, on attend {1, 3, 4}.

Exemples C commentés

Noter la taille n , compter les comparaisons et les échanges, puis exprimer le coût en $O(\dots)$. Tester un tableau vide, déjà trié, inversé et rempli de doublons.

Patron de sélection : rechercher le minimum de la zone non triée, puis l'échanger avec sa première case.

Code :

```
for (size_t i = 0; i + 1 < n; ++i) {  
    size_t min = i; // invariant : [0,i) est trié  
    for (size_t j = i + 1; j < n; ++j)  
        if (a[j] < a[min]) min = j;
```

Partie 1 : Tris et analyse de complexité



Exercice 1 – Sélection et invariants

Un service de facturation reçoit des montants non triés. Trier le tableau par sélection ; après chaque itération, afficher la partie déjà triée et expliquer l'invariant. Donner les coûts dans les cas meilleur, moyen et pire.



Exercice 2 – Insertion stable

Un tableau de tickets est presque trié par heure d'arrivée. Implémenter le tri par insertion stable, compter les décalages et comparer un flux déjà trié à un flux inversé.



Exercice 3 – Tri à bulles optimisé

Écrire un tri à bulles qui s'arrête lorsqu'aucun échange n'a été réalisé. Mesurer le nombre de passages et expliquer l'intérêt de l'arrêt anticipé.



Exercice 4 – Tri par comptage

Pour des entiers compris entre 0 et 100, utiliser un tableau de fréquences puis reconstruire le tableau trié. Préciser le coût en fonction de n et de la plage des valeurs.



Exercice 5 – Tri de chaînes

Un moteur de recherche doit afficher ses mots-clés par ordre lexicographique. Trier un tableau de mots avec un tri stable, comparer avec `strcmp` et préserver l'ordre des doublons.



Exercice 6 – Tri de structures

Un service RH exporte des candidats. Trier des structures par note décroissante puis par nom croissant en cas d'égalité. Séparer la comparaison de l'algorithme de tri.



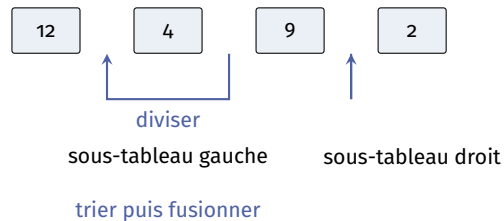
Exercice 7 – Fusion de deux tableaux

Fusionner deux tableaux déjà triés dans un troisième tableau trié. Compter les comparaisons et conserver les doublons.



Exercice 8 – Tri fusion récursif

Une chaîne de traitement reçoit des lots de mesures. Implémenter le tri fusion avec allocation d'un tableau temporaire, traiter les erreurs d'allocation et donner la complexité.



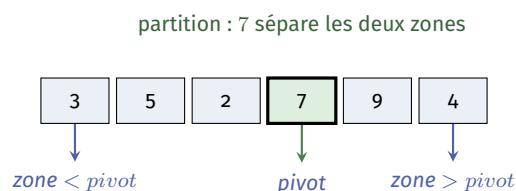
Exercice 9 – Tri fusion sans fuite

Réécrire le tri fusion en réutilisant un seul tampon alloué au début. Vérifier par un compteur que chaque allocation est libérée une seule fois.



Exercice 10 – Partition de Hoare

Implémenter une partition autour d'un pivot puis afficher les deux sous-tableaux obtenus. Énoncer la propriété vérifiée après la partition.



Exercice 11 – Tri rapide

Écrire le tri rapide récursif avec un pivot choisi au milieu. Comparer ses performances sur un tableau trié et un tableau aléatoire.



Exercice 12 – Pivot médian de trois

Améliorer le tri rapide en choisissant la médiane du premier, du milieu et du dernier élément. Mesurer l'effet sur des tableaux presque triés.



Exercice 13 – Tri par tas

Construire un tas max dans un tableau puis utiliser le tri par tas. Vérifier la propriété de tas après chaque construction.



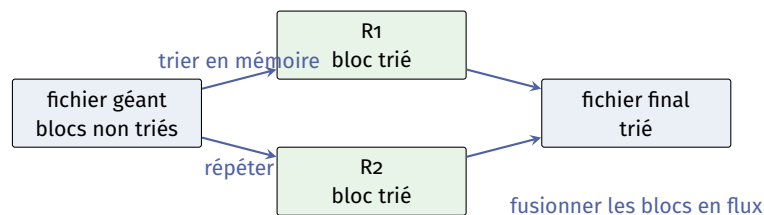
Exercice 14 – Comparer les tris

Un responsable qualité veut choisir un tri pour une plateforme de télémétrie. Implémenter sélection, insertion, fusion et rapide ; générer des tableaux de tailles croissantes et produire un rapport de comparaisons, déplacements et temps.



Exercice 15 – Tri externe par fichiers

Un serveur d'archives ne peut pas charger un journal de plusieurs gigaoctets en mémoire. Trier le fichier par blocs puis fusionner les fichiers temporaires ; documenter le protocole et chaque erreur d'entrée-sortie.



2

PARTIE

Recherches et diviser pour régner

NOTIONS

recherche linéaire, dichotomique, récursivité, sélection et tableaux triés

COMPÉTENCES

choisir une recherche adaptée et prouver la réduction du problème

PROGRESSION

Application → approfondissement

Langage C | CHEATSHEET DE LA PARTIE 2

Recherches et diviser pour régner

Notions de la partie, code C commenté et tests repères.

Notions essentielles

La recherche linéaire examine successivement les éléments. La recherche dichotomique exige un tableau trié et conserve un intervalle candidat.

Exemple : dans {12, 18, 23, 31, 42}, rechercher 31 élimine d'abord la moitié gauche ou droite selon la comparaison.

Exemples C commentés

Pour une récursion, préciser le cas de base et la réduction. Pour chaque recherche, annoncer le coût, le nombre de comparaisons et les préconditions.

Milieu sûr : $\text{milieu} = \text{gauche} + (\text{droite} - \text{gauche}) / 2;$

Boucle : `while (gauche <= droite) { ... }`

// Après comparaison, réduire l'intervalle candidat.

Partie 2 : Recherches et diviser pour régner



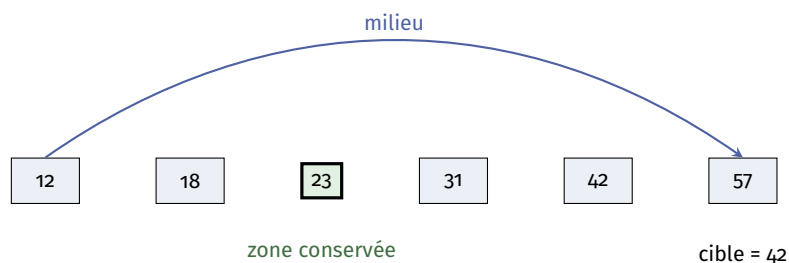
Exercice 16 – Recherche linéaire instrumentée

Rechercher une valeur dans un tableau non trié et retourner sa première position. Compter les comparaisons et traiter le tableau vide.



Exercice 17 – Recherche dichotomique itérative

Un catalogue de produits est indexé par identifiant croissant. Implémenter la recherche dichotomique itérative, éviter le débordement dans le calcul du milieu et retourner l'index ou -1.



Exercice 18 – Recherche dichotomique récursive

Écrire la version récursive de la recherche dichotomique. Dessiner les intervalles successifs pour une recherche réussie et une recherche absente.



Exercice 19 – Première occurrence

Adapter la recherche dichotomique pour retourner la première position d'une valeur répétée. Justifier pourquoi la recherche continue à gauche après une égalité.



Exercice 20 – Dernière occurrence

Adapter la recherche dichotomique pour retourner la dernière position d'une valeur répétée. Tester un tableau contenant une seule valeur répétée.



Exercice 21 – Insertion dans un tableau trié

Rechercher la position d'insertion d'une nouvelle valeur puis décaler les éléments. Maintenir le tableau trié et signaler une capacité insuffisante.



Exercice 22 – Recherche exponentielle

Un fichier d'archives est trié mais sa taille n'est pas connue avant lecture. Doubler progressivement la borne supérieure puis effectuer une recherche dichotomique sur la zone utile.



Exercice 23 – Recherche par interpolation

Implémenter une recherche par interpolation pour des entiers triés et régulièrement répartis. Refuser une division par zéro et comparer avec la dichotomie.



Exercice 24 – Racine entière

Calculer la partie entière de la racine carrée d'un entier par recherche dichotomique, sans appeler `sqrt`.



Exercice 25 – Puissance rapide

Calculer a^n par exponentiation rapide récursive puis itérative. Comparer le nombre de multiplications avec la méthode naïve.



Exercice 26 – Maximum par diviser pour régner

Trouver le maximum d'un tableau en divisant l'intervalle en deux. Retourner aussi son premier indice et analyser la complexité.



Exercice 27 – Inversion d'un tableau

Inverser un tableau par échanges des extrémités dans une fonction récursive. Vérifier le cas de taille paire et impaire.



Exercice 28 – Compter les inversions

Compter les couples (i, j) tels que $i < j$ et $a[i] > a[j]$. Donner une version naïve puis une version fondée sur la fusion.



Exercice 29 – Sélection du k-ième élément

Un tableau de latences contient des valeurs sensibles aux valeurs extrêmes. Trouver le k-ième plus petit élément avec quickselect, vérifier les bornes de k et discuter le cas défavorable.



Exercice 30 – Médiane en flux

Un tableau de bord reçoit des mesures en continu. Afficher la médiane après chaque nouvelle valeur et justifier le choix de deux tas pour éviter de retrier tout l'historique.

3 PARTIE

Glouton et programmation dynamique

NOTIONS

sous-problèmes, mémorisation, sacs, chemins, intervalles et ordonnancement

COMPÉTENCES

identifier une récurrence et comparer glouton, récursif et dynamique

PROGRESSION

Approfondissement → maîtrise

Langage C | CHEATSHEET DE LA PARTIE 3

Glouton et programmation dynamique

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Une solution gloutonne prend une décision locale sans revenir en arrière. La programmation dynamique mémorise les solutions de sous-problèmes qui se recouvrent et construit une solution optimale.

Exemple : pour l'escalier, $dp[i] = dp[i-1] + dp[i-2]$; réutilise les deux résultats précédents.

Exemples C commentés

Définir l'état, les transitions, les cas initiaux et l'ordre de calcul. Comparer le coût mémoire et temporel avec la récursion naïve et rechercher un contre-exemple lorsqu'une stratégie gloutonne n'est pas optimale.

Mémoïsation : tester $memo[n] \neq -1$ avant de recalculer un sous-problème.

```
Code : if (memo[n] != -1) return memo[n];  
memo[n] = fib(n - 1, memo) + fib(n - 2, memo);  
// Chaque sous-problème est résolu une seule fois.
```

Partie 3 : Glouton et programmation dynamique



Exercice 31 – Rendu de monnaie glouton

Pour des pièces 1, 2, 5, 10 et 20, rendre une somme avec un nombre minimal de pièces par stratégie gloutonne. Compter les pièces et vérifier les entrées.



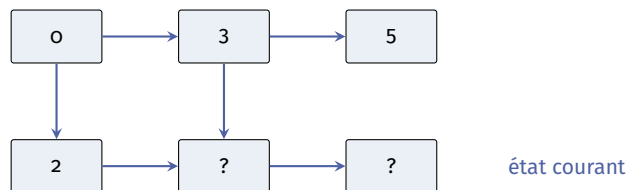
Exercice 32 – Contre-exemple au glouton

Construire un système de pièces pour lequel le rendu glouton n'est pas optimal. Comparer avec une recherche exhaustive sur les petites sommes.



Exercice 33 – Rendu de monnaie dynamique

Une caisse automatique doit rendre la monnaie avec un stock limité de pièces. Calculer le minimum de pièces, reconstruire la solution, décrémenter le stock et traiter une somme impossible.



Exercice 34 – Fibonacci mémorisé

Implémenter Fibonacci avec mémorisation puis avec un tableau itératif. Compter les appels récurrents dans la version naïve.



Exercice 35 – Escalier

Calculer le nombre de façons de monter un escalier de n marches en avançant d'une ou deux marches. Fournir les versions récursive, mémorisée et itérative.



Exercice 36 – Sous-suite croissante

Calculer la longueur de la plus longue sous-suite strictement croissante en $O(n^2)$ et reconstruire une solution.



Exercice 37 – Distance d'édition

Un outil de support rapproche deux noms saisis avec des fautes de frappe. Calculer la distance de Levenshtein, afficher la matrice et expliquer le coût d'une insertion, suppression ou substitution.

		a	b
	0	1	2
a	1	1	2
b	2	2	1

état courant
min(insertion, suppression, substitution)



Exercice 38 – Sac à dos 0/1

Choisir des objets de poids et valeurs donnés sans dépasser une capacité. Calculer la valeur maximale et reconstruire les objets retenus.



Exercice 39 – Sac fractionnaire

Résoudre la version fractionnaire du sac par ratio valeur/poids. Trier les objets et expliquer pourquoi la stratégie gloutonne est optimale ici.



Exercice 40 – Activités compatibles

Sélectionner le plus grand nombre d'activités non chevauchantes en choisissant toujours la date de fin la plus proche. Vérifier l'ordre de tri.



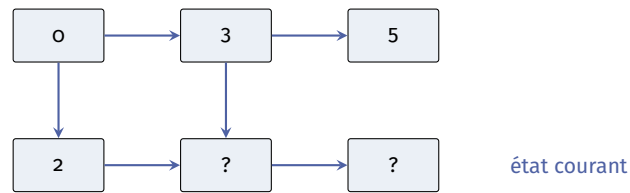
Exercice 41 – Ordonnancement à une machine

Un atelier ne possède qu'une machine de test. Ordonner des tâches avec durées et dates limites afin de maximiser le nombre livré à temps ; comparer une heuristique gloutonne et une solution exhaustive.



Exercice 42 – Chemin de somme maximale

Un robot inspecte une grille où chaque case contient un gain ou un coût. Calculer la somme maximale du coin supérieur gauche au coin inférieur droit, avec déplacements à droite ou vers le bas, puis reconstruire le trajet.



Exercice 43 – Parenthésage optimal

Déterminer le nombre minimal de multiplications scalaires pour multiplier une chaîne de matrices. Afficher les coupures retenues.



Exercice 44 – Produit de mots

Trouver le plus long préfixe d'un mot qui est aussi un suffixe, en utilisant une table de préfixes. Comparer avec la recherche naïve.



Exercice 45 – Projet de planification

Une équipe DevOps prépare une mise en production. Construire un planificateur de tâches avec dépendances, durées et ressources ; produire un ordre valide, détecter un cycle et signaler les tâches bloquantes.

4 PARTIE

Graphes et algorithmes avancés

NOTIONS

BFS, DFS, composantes, plus courts chemins, arbres couvrants et flots

COMPÉTENCES

modéliser un réseau et choisir un parcours ou une optimisation

PROGRESSION

Maîtrise → projet

Graphes et algorithmes avancés

Notions de la partie, code C commenté et tests repères.

Notions essentielles

Un graphe peut être représenté par une matrice ou des listes d'adjacence. BFS explore par niveaux ; DFS explore en profondeur et utilise une récursion ou une pile.

Exemple : une liste d'adjacence peut représenter $A \rightarrow B$ et $A \rightarrow C$ par deux voisins rattachés au sommet A .

Exemples C commentés

Marquer un sommet au moment où il est découvert. Pour les poids, distinguer les poids non négatifs de ceux qui peuvent être négatifs. Vérifier les sommets isolés, les cycles, les graphes non connexes et les chemins inexistantes.

BFS : enfiler la source, la marquer, puis enfiler chaque voisin encore non visité.

DFS : `visite[s] = 1;`

```
for (Voisin *p = g[s]; p != NULL; p = p->suivant)
    if (!visite[p->sommet]) dfs(g, p->sommet);
```

// Marquer à la découverte évite les revisites et les cycles.

Partie 4 : Graphes et algorithmes avancés



Exercice 46 – Parcours en profondeur

Représenter un graphe par listes d'adjacence et implémenter DFS récursif. Afficher les temps de découverte et de fin.



Exercice 47 – Composantes connexes

Utiliser DFS ou BFS pour compter les composantes connexes d'un graphe non orienté et afficher les sommets de chaque composante.



Exercice 48 – Détection de cycle

Détecter un cycle dans un graphe non orienté avec DFS en tenant compte du parent du sommet courant.



Exercice 49 – Tri topologique

Pour un graphe orienté acyclique, calculer un ordre topologique par degrés entrants. Signaler l'existence d'un cycle.



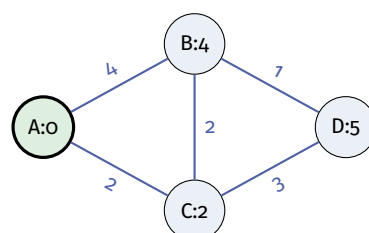
Exercice 50 – Plus court chemin non pondéré

Dans un réseau social, chaque arête représente une relation directe. Calculer avec BFS le nombre minimal de relations entre deux personnes et reconstruire le chemin demandé.



Exercice 51 – Dijkstra

Implémenter Dijkstra avec un tableau pour un graphe pondéré à poids non négatifs. Refuser ou signaler un poids négatif.



étiquette
= meilleure distance connue



Exercice 52 – Dijkstra avec tas

Un service de cartographie doit répondre rapidement à des milliers de requêtes. Remplacer la sélection linéaire par un tas min, comparer les coûts théoriques et mesurer le nombre d'extractions.



Exercice 53 – Bellman-Ford

Calculer les plus courts chemins depuis une source avec Bellman-Ford et détecter un cycle de poids négatif accessible.



Exercice 54 – Floyd-Warshall

Calculer toutes les distances minimales entre paires de sommets avec Floyd-Warshall et reconstruire un chemin.



Exercice 55 – Arbre couvrant minimal

Implémenter Kruskal avec une structure union-find. Trier les arêtes et détecter un graphe non connexe.



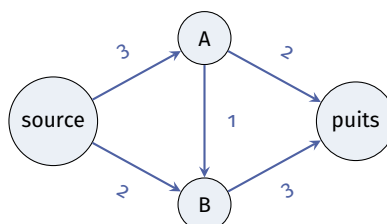
Exercice 56 – Prim

Construire un arbre couvrant minimal avec Prim sur une matrice de poids. Afficher les arêtes sélectionnées et le poids total.



Exercice 57 – Réseau de transport

Un réseau de distribution relie un dépôt à plusieurs magasins avec des capacités de transport. Calculer un flot maximal avec Ford-Fulkerson, vérifier la conservation du flot et identifier le goulot d'étranglement.





Exercice 58 – Coloration gloutonne

Colorer un graphe avec une stratégie gloutonne et afficher le nombre de couleurs. Tester plusieurs ordres de sommets.



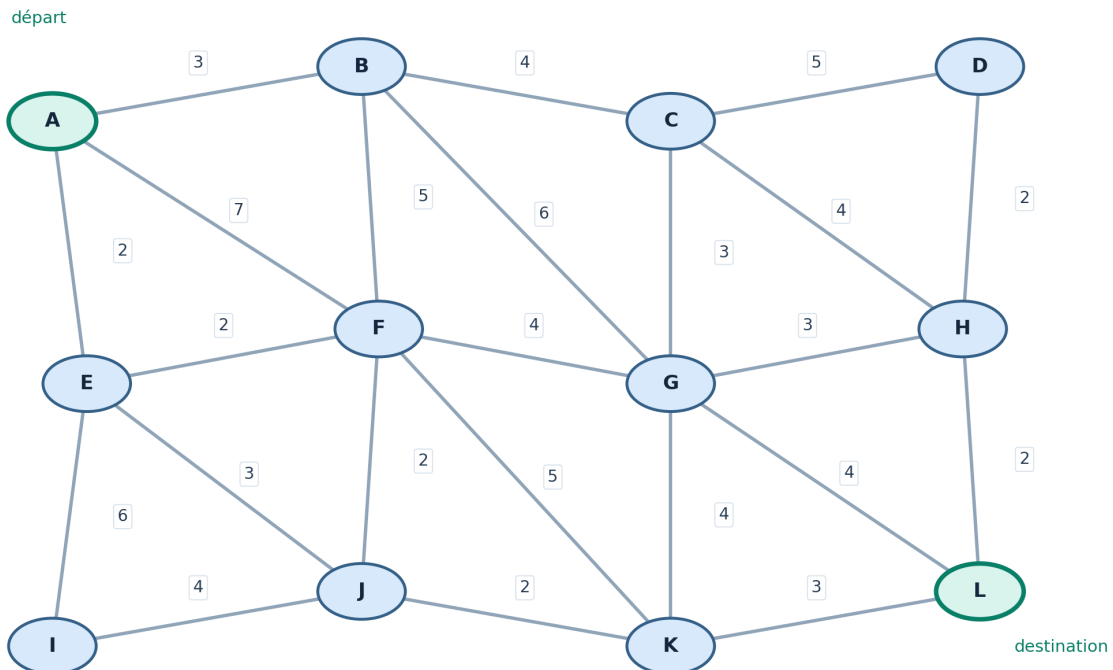
Exercice 59 – Chemin critique

Pour un projet de déploiement logiciel, construire le graphe des dépendances et calculer les dates au plus tôt, au plus tard, les marges et les tâches critiques.



Exercice 60 – Projet de réseau

Développer un analyseur de réseau en C pour une équipe d'exploitation : chargement depuis un fichier, affichage, BFS, Dijkstra, composantes, export d'un rapport et justification des complexités.



Graphe pondéré proposé comme jeu de données de départ : rechercher un chemin de A vers L.