

# PROJET DE GROUPE

## Système Intelligent de Routage et d'Analyse de Réseaux

*NetFlow Optimizer & Security Analyzer*

ALC2101 - Algorithmique et Complexité

Master Informatique - Année 2025-2026

Université Virtuelle de Côte d'Ivoire (UVCI)

### Spécialités concernées

Cybersécurité & IoT	•	Blockchain	•
Big Data Analytics	•	Traitement d'Images	•
Animation Audiovisuelle			

Durée du projet : 3 semaines

Travail en groupe : 7 étudiants

Langage imposé : C

# Table des matières

<b>1</b>	<b>Contexte et Motivation</b>	<b>3</b>
1.1	Problématique . . . . .	3
1.2	Défi Technique . . . . .	3
<b>2</b>	<b>Objectifs Pédagogiques</b>	<b>4</b>
<b>3</b>	<b>Description du Projet</b>	<b>4</b>
3.1	Vue d'Ensemble . . . . .	4
3.2	Fonctionnalités Principales . . . . .	4
3.2.1	Module 1 : Modélisation du Réseau (Graphes) . . . . .	4
3.2.2	Module 2 : Algorithmes de Routage Optimal . . . . .	5
3.2.3	Module 3 : Détection d'Anomalies et Analyse de Sécurité . . . . .	5
3.2.4	Module 4 : Gestion des Files de Paquets (Listes Chaînées) . . . . .	6
3.3	Extensions Innovantes (Bonus) . . . . .	6
<b>4</b>	<b>Spécifications Techniques</b>	<b>6</b>
4.1	Structures de Données à Implémenter . . . . .	6
4.2	Algorithmes Clés à Implémenter . . . . .	7
<b>5</b>	<b>Livrables Attendus</b>	<b>8</b>
5.1	Code Source (Format : Archive .zip) . . . . .	8
5.2	Rapport Technique (Max 30 pages) . . . . .	8
5.3	Présentation PowerPoint/Beamer . . . . .	10
5.4	Vidéo de Démonstration . . . . .	10
<b>6</b>	<b>Critères d'Évaluation</b>	<b>11</b>
<b>7</b>	<b>Planning Suggéré (3 Semaines)</b>	<b>12</b>
7.1	Semaine 1 : Conception et Fondations . . . . .	12
7.2	Semaine 2 : Algorithmes Avancés et Optimisations . . . . .	12
7.3	Semaine 3 : Finalisation et Préparation . . . . .	13
<b>8</b>	<b>Répartition des Tâches (Groupe de 7)</b>	<b>13</b>
<b>9</b>	<b>Ressources et Références</b>	<b>13</b>
9.1	Bibliographie Recommandée . . . . .	13
9.2	Outils de Développement . . . . .	14
9.3	Jeux de Données Suggérés . . . . .	14
<b>10</b>	<b>Conseils et Bonnes Pratiques</b>	<b>14</b>
10.1	Organisation du Travail . . . . .	14
10.2	Qualité du Code . . . . .	15
10.3	Préparation de la Soutenance . . . . .	15
<b>11</b>	<b>FAQ (Foire Aux Questions)</b>	<b>15</b>

# 1 Contexte et Motivation

## 1.1 Problématique

Dans l'ère du numérique, la gestion efficace des réseaux informatiques est cruciale pour garantir la performance, la sécurité et la fiabilité des systèmes distribués. Que ce soit pour :

- **Cybersécurité & IoT** : Détection de chemins d'intrusion, sécurisation des communications entre capteurs IoT
- **Blockchain** : Optimisation de la propagation des transactions et blocs dans un réseau P2P
- **Big Data** : Routage optimal des flux de données entre datacenters
- **Traitement d'Images/Audio** : Distribution de charge pour le traitement parallèle

La nécessité de concevoir des algorithmes performants pour analyser, optimiser et sécuriser les réseaux devient primordiale.

## 1.2 Défi Technique

Ce projet vous place face à un défi réel : **concevoir un système complet** capable de :

1. Modéliser un réseau complexe sous forme de graphe
2. Trouver les chemins optimaux avec contraintes multiples
3. Détecter des anomalies et vulnérabilités
4. Gérer dynamiquement les files d'attente de paquets/données
5. Optimiser l'allocation des ressources

## 2 Objectifs Pédagogiques

### Objectifs Pédagogiques

À l'issue de ce projet, vous serez capable de :

1. **Maîtriser les structures de données avancées :**

- Implémenter et manipuler des graphes (listes d'adjacence, matrices)
- Concevoir des listes chaînées optimisées pour la gestion de files de priorité
- Utiliser des arbres binaires de recherche pour l'indexation

2. **Appliquer les techniques de conception d'algorithmes :**

- *Diviser pour régner* : Décomposition du graphe, recherche parallèle
- *Backtracking* : Exploration exhaustive de chemins avec contraintes
- Programmation dynamique pour l'optimisation

3. **Analyser rigoureusement l'efficacité :**

- Calculer la complexité temporelle et spatiale (notation Big-O)
- Comparer empiriquement différentes approches
- Prouver la correction des algorithmes

4. **Développer des compétences professionnelles :**

- Travail collaboratif sur un projet d'envergure
- Documentation technique rigoureuse
- Présentation orale de résultats scientifiques

## 3 Description du Projet

### 3.1 Vue d'Ensemble

Vous devez concevoir et implémenter **NetFlow Optimizer & Security Analyzer**, un système logiciel en langage C permettant d'analyser et d'optimiser un réseau informatique modélisé sous forme de graphe pondéré.

### 3.2 Fonctionnalités Principales

#### 3.2.1 Module 1 : Modélisation du Réseau (Graphes)

- **Représentation** : Implémenter un graphe pondéré orienté/non-orienté
  - Matrice d'adjacence (pour les graphes denses)
  - Listes d'adjacence avec listes chaînées (pour les graphes creux)
- **Pondérations multiples** : Chaque arête possède :
  - Latence (délai en ms)
  - Bande passante (Mbps)
  - Coût (unité monétaire)
  - Niveau de sécurité (0-10)

— **Opérations :**

- Ajout/suppression dynamique de nuds et arêtes
- Chargement depuis un fichier (format texte structuré)
- Sauvegarde de la topologie

### 3.2.2 Module 2 : Algorithmes de Routage Optimal

1. **Plus Court Chemin Simple**

- Implémenter Dijkstra avec file de priorité (liste chaînée)
- Implémenter Bellman-Ford (pour graphes avec poids négatifs)
- Analyse comparative de complexité

2. **Plus Court Chemin avec Contraintes (Backtracking)**

- Trouver un chemin minimisant la latence AVEC :
  - Bande passante minimale garantie
  - Coût maximal autorisé
  - Passage obligatoire par certains nuds
  - Exclusion de nuds non-sécurisés
- Utilisation du backtracking pour explorer les solutions
- Élagage des branches non-prometteuses

3. **K Plus Courts Chemins (Diviser pour Régner)**

- Trouver les K meilleurs chemins alternatifs
- Décomposition récursive du problème
- Analyse de la redondance du réseau

### 3.2.3 Module 3 : Détection d'Anomalies et Analyse de Sécurité

- **Détection de Cycles (DFS/BFS)**
  - Identifier les boucles de routage
  - Détection de cycles suspects (attaques par redirection)
- **Points de Vulnérabilité**
  - Calcul des points d'articulation (nuds critiques)
  - Identification des ponts (arêtes critiques)
  - Analyse de la connectivité
- **Composantes Fortement Connexes**
  - Algorithme de Tarjan ou Kosaraju
  - Identification de sous-réseaux isolés

### 3.2.4 Module 4 : Gestion des Files de Paquets (Listes Chaînées)

- Implémenter une **file de priorité** avec liste chaînée doublement chaînée
- Opérations :
  - **enqueue** : Insertion avec priorité ( $O(n)$  ou  $O(\log n)$  si optimisé)
  - **dequeue** : Extraction du paquet prioritaire ( $O(1)$ )
  - **peek** : Consultation sans suppression
- Simulation de flux de paquets avec statistiques :
  - Temps moyen d'attente
  - Taux de perte (si capacité limitée)
  - Débit effectif

### 3.3 Extensions Innovantes (Bonus)

Pour maximiser la note d'innovation (20%), proposez au moins UNE extension :

1. **Visualisation graphique** (avec bibliothèque comme Graphviz ou SDL)
2. **Parallélisation** de l'algorithme de recherche de chemins (threads POSIX)
3. **Interface interactive** en ligne de commande (menu dynamique)
4. **Analyse de réseaux réels** (topologies de datacenter, réseau IoT simulé)
5. **Algorithme génétique** pour optimisation multi-objectifs
6. **Prédiction de charge** avec historique (structure de données temporelle)

## 4 Spécifications Techniques

### 4.1 Structures de Données à Implémenter

#### Important

**OBLIGATOIRE** : Vous devez implémenter FROM SCRATCH (sans bibliothèques externes pour les structures de base) :

1. **Graphe** (au moins 2 représentations)

```
typedef struct Arete {
    int destination;
    float latence;
    float bande_passante;
    float cout;
    int securite;
    struct Arete* suivant; // Liste chaînée
} Arete;

typedef struct Noeud {
    int id;
    char nom[50];
```

```

    Arete* aretes; // Liste d'adjacence
} Noeud;

typedef struct Graphe {
    int nb_noeuds;
    Noeud* noeuds;
    float** matrice_adjacence; // Représentation alternative
} Graphe;

```

## 2. Liste Chaînée Double (pour file de priorité)

```

typedef struct Paquet {
    int id;
    int priorite;
    float taille_Mo;
    int source;
    int destination;
    struct Paquet* precedent;
    struct Paquet* suivant;
} Paquet;

typedef struct FileAttente {
    Paquet* tete;
    Paquet* queue;
    int taille_actuelle;
    int capacite_max;
} FileAttente;

```

## 3. Pile (pour DFS et backtracking)

## 4. File (pour BFS)

## 4.2 Algorithmes Clés à Implémenter

Algorithme	Technique	Complexité
Dijkstra	Glouton + File priorité	$O((V + E) \log V)$
Bellman-Ford	Programmation dynamique	$O(VE)$
DFS/BFS	Parcours	$O(V + E)$
Chemin contraint	Backtracking	$O(b^d)$ exponentiel
K plus courts chemins	Diviser pour régner	À analyser
Points d'articulation	DFS modifié	$O(V + E)$
Tarjan (SCC)	DFS + Pile	$O(V + E)$

TABLE 1 – Algorithmes et complexités théoriques

## 5 Livrables Attendus

### Important

**Date limite :** Fin de la semaine 3

**Modalités :** Dépôt sur plateforme + Soutenance

### 5.1 Code Source (Format : Archive .zip)

— Organisation :

```
ProjetNetFlow/  
  src/  
    graphe.c / graphe.h  
    liste_chaine.c / liste_chaine.h  
    dijkstra.c / dijkstra.h  
    backtracking.c / backtracking.h  
    securite.c / securite.h  
    utils.c / utils.h  
    main.c  
  data/  
    reseau_test1.txt  
    reseau_test2.txt  
    reseau_reel.txt  
  tests/  
    tests_unitaires.c  
  docs/  
    README.md  
  Makefile
```

— Qualité du code :

- Commentaires clairs et exhaustifs (en français)
- Respect des conventions de nommage
- Gestion rigoureuse de la mémoire (pas de fuites - vérifier avec `valgrind`)
- Modularité et réutilisabilité

— Compilation : Makefile avec options :

```
make          # Compilation normale  
make debug    # Avec symboles de débogage  
make clean    # Nettoyage  
make test     # Exécution des tests
```

### 5.2 Rapport Technique (Max 30 pages)

Structure imposée :

1. Page de garde (1 page)

- Titre du projet
- Noms et spécialités des membres
- Répartition des tâches (tableau)
- 2. **Introduction** (2 pages)
  - Contexte et motivation
  - Objectifs du projet
  - Organisation du rapport
- 3. **État de l'art** (3 pages)
  - Algorithmes de routage existants
  - Comparaison des approches
  - Justification de vos choix
- 4. **Conception** (8 pages)
  - Architecture globale (diagrammes)
  - Structures de données détaillées
  - Algorithmes (pseudo-code commenté)
  - Justification des choix de conception
- 5. **Analyse Théorique de Complexité** (6 pages) **CRUCIAL!**
  - Preuve formelle de complexité pour CHAQUE algorithme
  - Notation Big-O, Omega, Theta
  - Analyse du meilleur/moyen/pire cas
  - Comparaison théorique des approches
- 6. **Implémentation** (3 pages)
  - Choix techniques (structures de données spécifiques)
  - Difficultés rencontrées et solutions
  - Optimisations appliquées
- 7. **Résultats Expérimentaux** (5 pages) **CRUCIAL!**
  - Protocole de tests (jeux de données)
  - Mesures de performance (graphiques)
  - Validation de la complexité théorique
  - Comparaison des algorithmes :
    - Dijkstra vs Bellman-Ford
    - Backtracking avec/sans élagage
    - Matrice vs Listes d'adjacence
- 8. **Innovation** (2 pages)
  - Description des extensions réalisées
  - Apport par rapport à l'existant
  - Perspectives d'amélioration
- 9. **Conclusion** (1 page)

- Bilan des objectifs atteints
- Compétences acquises
- Difficultés majeures
- Perspectives

10. **Bibliographie** (1 page minimum)

- Articles scientifiques
- Ouvrages de référence
- Documentation technique

### 5.3 Présentation PowerPoint/Beamer

- **Format** : 15-20 diapositives maximum
- **Durée** : 7 minutes présentation + 15 min questions)
- **Structure** :
  1. Introduction et contexte (2 slides)
  2. Problématique technique (1 slide)
  3. Architecture et conception (3 slides)
  4. Algorithmes clés (3 slides avec pseudo-code)
  5. Analyse de complexité (2 slides avec preuves)
  6. Résultats expérimentaux (3 slides avec graphiques)
  7. Démo (1 slide d'annonce)
  8. Conclusion et perspectives (1 slide)
- **Conseils** :
  - Slides aérées et lisibles
  - Graphiques et schémas de qualité
  - Animations sobres
  - Pas de texte dense (bullet points)

### 5.4 Vidéo de Démonstration

- **Durée** : 3-5 minutes
- **Contenu** :
  1. Introduction rapide (20 sec)
  2. Compilation du projet (30 sec)
  3. Chargement d'un réseau (30 sec)
  4. Démonstration de fonctionnalités clés :
    - Calcul plus court chemin (30 sec)
    - Recherche avec contraintes (45 sec)
    - Détection d'anomalies (30 sec)
    - Simulation de flux (45 sec)

- 5. Innovation (si applicable) (30 sec)
- 6. Conclusion (20 sec)
- **Qualité :**
  - Résolution HD (720p minimum)
  - Voix-off claire (français)
  - Montage soigné
  - Sous-titres optionnels mais appréciés
- **Format :** MP4, AVI ou MOV
- **Hébergement :** Lien YouTube/Drive à fournir

## 6 Critères d'Évaluation

Critère	Points	Détails
<b>Présentation Orale &amp; Défense (50%)</b>		
Clarté de l'exposé	15	Structure, élocution, timing
Maîtrise technique	15	Capacité à expliquer les algorithmes
Réponses aux questions	10	Pertinence et précision
Démonstration live	10	Fluidité, robustesse du code
<b>Innovation &amp; Originalité (20%)</b>		
Extensions réalisées	10	Qualité et pertinence
Créativité	5	Approches originales
Ambition technique	5	Complexité des fonctionnalités
<b>Implémentation (15%)</b>		
Correction du code	5	Absence de bugs majeurs
Qualité du code	5	Commentaires, modularité
Performance	5	Optimisations, gestion mémoire
<b>Analyse Théorique (15%)</b>		
Preuves de complexité	8	Rigueur mathématique
Comparaisons	4	Pertinence des analyses
Correction des calculs	3	Justesse des résultats
<b>TOTAL</b>	<b>100</b>	

TABLE 2 – Grille d'évaluation détaillée

### Information

#### Bonus :

- Tests unitaires exhaustifs : +2 points
- Documentation Doxygen : +1 point
- Interface graphique : +3 points
- Analyse de réseau réel : +2 points

**Note maximale cumulée :** 108/100 (ramené à 20/20)

## 7 Planning Suggéré (3 Semaines)

### 7.1 Semaine 1 : Conception et Fondations

Jours	Tâches
1-2	<ul style="list-style-type: none"> <li>— Brainstorming et répartition des rôles</li> <li>— Étude de l'état de l'art</li> <li>— Conception de l'architecture</li> <li>— Définition des structures de données</li> </ul>
3-4	<ul style="list-style-type: none"> <li>— Implémentation du graphe (2 représentations)</li> <li>— Implémentation des listes chaînées</li> <li>— Tests unitaires des structures de base</li> </ul>
5-7	<ul style="list-style-type: none"> <li>— Implémentation Dijkstra</li> <li>— Implémentation DFS/BFS</li> <li>— Chargement/sauvegarde de graphes</li> <li>— <b>Livrable intermédiaire</b> : Code compilable avec fonctionnalités de base</li> </ul>

### 7.2 Semaine 2 : Algorithmes Avancés et Optimisations

Jours	Tâches
8-10	<ul style="list-style-type: none"> <li>— Implémentation Bellman-Ford</li> <li>— Implémentation backtracking pour chemins contraints</li> <li>— Tests et comparaisons</li> </ul>
11-12	<ul style="list-style-type: none"> <li>— Algorithmes de sécurité (points d'articulation, SCC)</li> <li>— File de priorité et simulation de flux</li> </ul>
13-14	<ul style="list-style-type: none"> <li>— Extensions innovantes</li> <li>— Optimisations de performance</li> <li>— Début rédaction du rapport (sections conception/implémentation)</li> </ul>

Jours	Tâches
15-16	<ul style="list-style-type: none"> <li>— Mesures de performance exhaustives</li> <li>— Génération de graphiques et tableaux</li> <li>— Analyse comparative approfondie</li> <li>— Rédaction analyse théorique + résultats expérimentaux</li> </ul>
17-18	<ul style="list-style-type: none"> <li>— Finalisation du rapport (relecture collective)</li> <li>— Création de la présentation PowerPoint</li> <li>— Enregistrement de la vidéo de démonstration</li> </ul>
19-20	<ul style="list-style-type: none"> <li>— Répétitions de la soutenance</li> <li>— Préparation aux questions probables</li> <li>— Vérifications finales (compilation, tests)</li> </ul>
21	<b>SOUTENANCE ORALE</b>

### 7.3 Semaine 3 : Finalisation et Préparation

## 8 Répartition des Tâches (Groupe de 7)

#### Important

**Important :** Chaque membre doit contribuer au code ET à la rédaction. La répartition ci-dessus est indicative et peut être adaptée selon les compétences.

## 9 Ressources et Références

### 9.1 Bibliographie Recommandée

#### 1. Ouvrages

- Cormen et al., *Introduction to Algorithms* (4ème édition)
- Sedgewick & Wayne, *Algorithms* (4ème édition)
- Knuth, *The Art of Computer Programming, Vol. 1-3*

#### 2. Articles Scientifiques

- Dijkstra, E. W. (1959). “A note on two problems in connexion with graphs”
- Tarjan, R. (1972). “Depth-first search and linear graph algorithms”
- Yen, J. Y. (1971). “Finding the K shortest loopless paths in a network”

#### 3. Ressources en Ligne

- GeeksforGeeks : <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>
- Visualgo : <https://visualgo.net/> (visualisation d’algorithmes)
- MIT OpenCourseWare : 6.006 Introduction to Algorithms

Rôle	Responsabilité	Tâches Principales
<b>Chef de projet</b>	Coordination	Planification, suivi, intégration des modules, gestion Git
<b>Architecte</b>	Conception	Architecture globale, structures de données, diagrammes
<b>Dev. Graphes</b>	Module 1	Implémentation graphe, matrice/listes, chargement/sauvegarde
<b>Dev. Routage</b>	Module 2	Dijkstra, Bellman-Ford, backtracking, K plus courts chemins
<b>Dev. Sécurité</b>	Module 3	DFS/BFS, points d'articulation, détection de cycles, SCC
<b>Dev. Files</b>	Module 4	Listes chaînées, file de priorité, simulation de flux
<b>Testeur/Doc</b>	Qualité	Tests unitaires, mesures de performance, rapport, présentation

TABLE 3 – Répartition suggérée des rôles

## 9.2 Outils de Développement

- **Compilateur** : GCC (version 9+) avec flags : `-Wall -Wextra -std=c11`
- **Débogueur** : GDB ou LLDB
- **Analyse mémoire** : Valgrind
- **Versioning** : Git + plateforme (GitHub/GitLab)
- **Profiling** : gprof, perf
- **Documentation** : Doxygen (optionnel)
- **Graphiques** : Gnuplot, Python/Matplotlib pour visualisations

## 9.3 Jeux de Données Suggérés

1. **Graphes de test simples** (5-10 nuds) pour validation
2. **Graphes moyens** (50-100 nuds) pour benchmarks
3. **Graphes complexes** (500+ nuds) pour tests de scalabilité
4. **Topologies réelles** :
  - Réseau de campus universitaire
  - Topologie datacenter (Fat-tree, Clos)
  - Réseau IoT simulé (capteurs distribués)
  - Backbone Internet (AS-level topology)

# 10 Conseils et Bonnes Pratiques

## 10.1 Organisation du Travail

- **Réunions régulières** : Au minimum 2 fois par semaine (30 min)
- **Outil de suivi** : Trello, Notion ou GitHub Projects

- **Communication** : Groupe WhatsApp/Discord/Slack actif
- **Versioning** :
  - Commits fréquents avec messages clairs
  - Branches par fonctionnalité
  - Pull requests avec revue de code

## 10.2 Qualité du Code

- **Conventions de nommage** :
  - Structures : PascalCase (ex: GrapheOriente)
  - Fonctions : snake\_case (ex: trouver\_plus\_court\_chemin)
  - Variables : snake\_case (ex: nombre\_noeuds)
  - Constantes : MAJUSCULES (ex: TAILLE\_MAX\_BUFFER)
- **Gestion de la mémoire** :
  - Toujours libérer (`free`) ce qui a été alloué (`malloc/calloc`)
  - Vérifier les retours de `malloc` (NULL si échec)
  - Utiliser `valgrind` régulièrement
- **Commentaires** :
 

```
/**
 * @brief Calcule le plus court chemin entre deux nuds
 * @param graphe Pointeur vers la structure Graphe
 * @param source ID du nud source
 * @param destination ID du nud destination
 * @return Pointeur vers liste chaînée du chemin (NULL si aucun)
 * @complexity O((V+E) log V) avec file de priorité
 */
Chemin* dijkstra(Graphe* graphe, int source, int destination);
```

## 10.3 Préparation de la Soutenance

- **Répétez plusieurs fois** : Chronométrez-vous !
- **Anticipez les questions**
- **Préparez un plan B** si la démo échoue (captures d'écran, vidéo de secours)
- **Tous les membres doivent parler** : Répartissez équitablement le temps

## 11 FAQ (Foire Aux Questions)

1. **Q** : Peut-on utiliser des bibliothèques externes (ex : glib) ?  
**R** : Non pour les structures de données de base (graphes, listes). Oui pour l'affichage graphique ou la génération aléatoire (optionnel).

2. **Q** : Faut-il implémenter TOUS les algorithmes mentionnés ?  
**R** : Minimum requis : Dijkstra, DFS/BFS, backtracking pour chemins contraints, détection de cycles. Le reste est optionnel mais valorisé.
3. **Q** : Comment prouver rigoureusement la complexité ?  
**R** : Analyse mathématique ligne par ligne du pseudo-code, identification des boucles imbriquées, notation Big-O avec justification. Voir exemples dans le cours.
4. **Q** : Peut-on réutiliser du code trouvé en ligne ?  
**R** : Interdiction stricte du plagiat. Vous pouvez vous inspirer, mais le code doit être réécrit et compris. Citez vos sources dans le rapport.
5. **Q** : Comment gérer un membre inactif ?  
**R** : Contactez l'enseignant rapidement. Le rapport doit clairement indiquer la contribution de chacun.
6. **Q** : Que faire si on bloque sur un algorithme ?  
**R** : Forum de discussion du cours, heures de permanence de l'enseignant, entraide entre groupes (sans partager de code).

## Conclusion

Ce projet constitue une opportunité unique d'appliquer concrètement les concepts théoriques du cours d'Algorithmique et Complexité. Il vous permettra de développer non seulement vos compétences techniques, mais aussi vos capacités de travail en équipe, de gestion de projet et de communication scientifique.

## Bon courage et excellent travail !

*“In theory, there is no difference between theory and practice. In practice, there is.”*  
— Yogi Berra

---

### Contact Enseignant

Dr. ATTA Amanvon Ferdinand  
[amanvon.atta@uvci.edu.ci](mailto:amanvon.atta@uvci.edu.ci)

Université Virtuelle de Côte d'Ivoire (UVCI)  
Année académique 2025-2026