ENSAM – Casablanca **Année Universitaire 2025-2026**
Département Génie Informatique **Pr. Soukaina Bouhsissin**

Intelligence Artificielle
TP 4 : Espace d'États & Recherche
BFS / DFS - Algorithmes de Parcours

Objectifs du TP

- Comprendre la notion d'espace d'états pour résoudre des problèmes
- Modéliser un problème classique (Fermier-Loup-Chèvre-Laitue)
- Implémenter l'algorithme BFS (Breadth-First Search / Parcours en largeur)
- Implémenter l'algorithme DFS (Depth-First Search / Parcours en profondeur)
- Comparer les performances et caractéristiques des deux approches

Problème Étudié

Le Fermier, le Loup, la Chèvre et la Laitue

Énoncé :
Un fermier doit traverser une rivière avec :

- Un loup (L)
- Une chèvre (C)
- Une laitue (T pour "Turnip")

Règles :

- Le bateau peut transporter le fermier + un seul objet à la fois
- La présence du fermier empêche tout danger

Contraintes :

- Le loup ne peut pas rester seul avec la chèvre (sinon il la mange)
- La chèvre ne peut pas rester seule avec la laitue (sinon elle la mange)

Objectif : Faire traverser tout le monde de l'ouest vers l'est sans violer les contraintes.

Partie 1 : Modélisation du Problème

Dans cette partie, vous allez modéliser le problème en définissant les états, les transitions et les contraintes.

Question 1 : États de référence et affichage

Un état est représenté par un tuple de 4 valeurs (F, L, C, T) où chaque valeur vaut 0 (ouest) ou 1 (est).

A) Définissez les constantes et états initiaux :

```
OUEST, EST =
INITIAL =          # Tous à l'ouest
GOAL =             # Tous à l'est
NOMS =
```

B) Implémentez la fonction affiche_etat(s) :
Cette fonction doit afficher un état sous la forme : Ouest: [F, L] | Est: [C, T]

Votre code Python :

Question 2 : Test d'état interdit

Implémentez la fonction est_interdit(s) qui retourne True si l'état viole une contrainte
Exemples de tests :

(0, 0, 0, 0) -> False

(1, 0, 0, 0) -> True

(1, 1, 0, 0) -> True

(1, 0, 1, 0) -> False

(0, 1, 0, 1) -> False

Votre code Python :

Question 3 : Opérateurs (actions)

Implémentez la fonction appliquer_action(s, avec=None) qui simule une traversée :

- Le fermier traverse toujours (change de côté)
- Si avec='L', 'C' ou 'T', l'objet correspondant traverse aussi (s'il est du même côté que le fermier)
- Si avec=None, le fermier traverse seul
- Retourner None si l'action est impossible (objet pas du même côté)

Exemples :

appliquer_action((0,0,0,0), None) → (1,0,0,0)

appliquer_action((0,0,0,0), 'C') → (1,0,1,0)

appliquer_action((0,1,0,0), 'L') → None (L pas du même côté que F)

Votre code Python :

Question 4 : Successeurs valides

Implémentez la fonction successeurs(s) qui retourne la liste de tous les états successeurs valides depuis l'état s :

1. Essayer toutes les actions possibles : None, 'L', 'C', 'T'
2. Appliquer chaque action avec appliquer_action
3. Ne garder que les états non-None, non-interdits, et non-dupliqués

Votre code Python :

Testez avec l'état initial et affichez les successeurs :

Résultats de vos tests :

Partie 2 : Algorithmes BFS & DFS

Vous allez maintenant implémenter deux algorithmes classiques de parcours de graphe pour résoudre le problème.

Question 5 : Implémentation de BFS (Breadth-First Search)

Principe : BFS explore les états niveau par niveau, garantissant de trouver le chemin le plus court (en nombre d'étapes).

Implémentez la fonction bfs(start) qui retourne (chemin, nb_explorations) :

Votre code Python :

Question 6 : Implémentation de DFS (Depth-First Search)

Principe : DFS explore en profondeur, ne garantit pas le chemin le plus court mais peut être plus rapide en mémoire.

Implémentez la fonction dfs(start, limit=None) qui retourne (chemin, nb_explorations) :

Votre code Python :

Question 7 : Exécution et comparaison

Exécutez les deux algorithmes sur le problème et remplissez le tableau suivant :

Critère	BFS	DFS
Longueur du chemin		
Nombre de traversées		

Chemin BFS :
Chemin DFS :