

# ALGORITHMIQUE AVANCE AVEC PYTHON

## Présentation Générale

Ce mini-projet a été conçu pour nous permettre de comprendre et d'implémenter des structures d'arbres binaires en Python. Ils impliquent des concepts d'algorithme avancée, notamment la manipulation des arbres, l'optimisation de parcours et l'application à des contextes pratiques comme la compilation et les réseaux. Une extension est ajoutée pour inclure la notion d'arbre binaire de recherche (ABR), permettant d'optimiser certaines opérations de recherche et d'organisation des données

## Consignes Générales

- Chaque projet doit être réalisé en binôme ou individuellement.
- L'utilisation de bibliothèques externes est interdite : tout doit être codé à la main.
- Une bonne structuration du code est attendue : des fonctions claires et bien commentées.
- Un rapport expliquant la logique adoptée doit accompagner le code source.

## Modalités d'Évaluation

- Date limite : 23/03/2025 à 23h59 sur MyGes

Instructions : inscrivez le contexte du cahier des charges

## Exigences

L'utilisation de bibliothèques externes est interdite : tout doit être codé à la main.	Une bonne structuration du code est attendue : des fonctions claires et bien commentées.
Un rapport expliquant la logique adoptée doit accompagner le code source.	Modéliser une architecture réseau
Date limite : 23/03/2025 à 23h59 sur MyGes	Optimiser le routage des paquets en utilisant des arbres binaires pour structurer une table de routage.

Instructions : inscrivez les exigences.

## Points d'attentions

Complicuer d'afficher le chemin le plus cours en fonction du temps d'exécution de la méthode de recherche la plus rapide, sans la lib time	Utilisation de la méthode BFS pour l'affichage du chemin de la racine vers le nœud "réseau" rechercher
Ajout d'un fichier cache pour stocker les recherches et optimiser la recherche de l'utilisateur	Utilisation de deux paths pour la recherche avec les trois méthodes et pour le chemin de la racine jusqu'au nœud recherché
l'arbre binaire représente le réseau informatique d'une entreprise.	On considère qu'il n'y a aucune redondance, et que le fichier cache a été réaliser afin d'apprendre à extraire des données, les formater et de les réutiliser.

Instructions : inscrivez les points d'actions à prendre en compte.

## Tâches

Modules	BinTree	Classe d'arbre binaire	RoutingManager	Gestionnaire de route	PacketSender	Fonction d'envoi de paquet	TreeVisualizer	Visualisation d'arbre binaire	TreeBuilder	Construction de l'arbre binaire de routage	NetworkManager	Algorithmes de recherche	Queue	Classe de file d'attente (Queue)	Main	Programme principal qui orchestre l'exécution et gère les points de blocage.	CacheManager	Gestionnaire de cache
Contenus	Class BinTree	Classe d'arbre binaire	bfs_mark_unreachable	Méthode pour marquer un nœud comme inaccessible pendant un parcours BFS.	send_packet	Envoyer l'envoi d'un paquet à un nœud cible.	print_tree	Afficher l'arbre binaire de routage.	Build_tree	Construire l'arbre binaire de routage à partir d'une liste de nœuds et de liens.	search_graph	Fonction de recherche dans le graphe de routage.	__init__	Initialise une file vide.	main	Exécute le programme principal.	save_cache	Enregistre les données de routage dans le cache.
		Gestion de la taille de l'arbre	delete_node	Supprime un nœud et ses sous-arbres de l'arbre de routage.	Latency	Choisir une latence minimale dans une liste de 20.					search_path	Fonction de recherche de chemin dans le graphe de routage.	enqueue	Ajoute un élément à la file.	is_network_free	Vérifie si un réseau est libre.	load_cache	Charge les données de routage du cache.
		Gestion de la hauteur de l'arbre					colors	Définit des couleurs pour l'affichage de l'arbre de routage.			search_info	Fonction de recherche d'informations dans le graphe de routage.	dequeue	Retire un élément de la file.	handle_network_down	Gère la suppression d'un réseau indisponible.	load_cache	Charge les données de routage du cache.
											search_info	Fonction de recherche d'informations dans le graphe de routage.	peek	Récupère l'élément en tête de file sans le retirer.	search_node	Recherche un nœud dans l'arbre.		
											find_fastest_path	Fonction pour trouver le chemin le plus rapide entre deux nœuds.	is_empty	Vérifie si la file est vide.	handle_packet_sending	Gère l'envoi d'un paquet à un nœud.		
											get_path_from_root	Fonction pour obtenir le chemin à partir de la racine.						
											dfs	Fonction de parcours en profondeur (DFS) sur l'arbre de routage.						

## Instructions

- Étape 1 :** Ajouter les modules à créer sur les autocollants jaunes.
- Étape 2 :** Ajouter le sujet de chaque module sur les autocollants gris.
- Étape 3 :** Ajouter les fonctionnalités de chaque module sur les autocollants verts.
- Étape 4 :** Ajouter les fonctionnalités supplémentaires sur les autocollants cyans.
- Étape 5 :** Ajouter les Ressources documentaires à créer (descriptions/documentations).

**Définir une hiérarchisation des tâches :** les tâches prioritaires en haut de la liste.

**Tâche fini :** ajouter le symbole pour valider la tâche.

**Bloquant :** changer la couleur des autocollants en rouge si vous rencontrez un point bloquant.

## Release Legend

Modules	Fonctions	Modules supplémentaires	points bloquant
Sujets		Ressources documentaires	

# ALGORITHMIQUE AVANCE AVEC PYTHON

## Présentation Générale

Ce mini-projet a été conçu pour nous permettre de comprendre et d'implémenter des structures d'arbres binaires en Python. Ils impliquent des concepts d'algorithmie avancée, notamment la manipulation des arbres, l'optimisation de parcours et l'application à des contextes pratiques comme la compilation et les réseaux. Une extension est ajoutée pour inclure la notion d'arbre binaire de recherche (ABR), permettant d'optimiser certaines opérations de recherche et d'organisation des données

## Consignes Générales

- Chaque projet doit être réalisé en binôme ou individuellement.
- L'utilisation de bibliothèques externes est interdite : tout doit être codé à la main.
- Une bonne structuration du code est attendue : des fonctions claires et bien commentées.
- Un rapport expliquant la logique adoptée doit accompagner le code source.

## Modalités d'Évaluation

- Date limite : 23/03/2025 à 23h59 sur MyGes

Instructions : inscrivez le contexte du cahier des charges

# Exigences

L'utilisation de bibliothèques externes est interdite : tout doit être codé à la main.

Une bonne structuration du code est attendue : des fonctions claires et bien commentées.

Un rapport expliquant la logique adoptée doit accompagner le code source.

Modéliser une architecture réseau

Date limite : 23/03/2025 à 23h59 sur MyGes

Optimiser le routage des paquets en utilisant des arbres binaires pour structurer une table de routage.

Instructions : inscrivez les exigences.

## Points d'attentions

Complicquer d'afficher le chemin le plus cours en fonction du temps d'exécution de la méthode de recherche la plus rapide, sans la lib time

Utilisation de la méthode BFS pour l'affichage du chemin de la racine vers le nœud "réseau" rechercher

Ajout d'un fichier cache pour stocker les recherches et optimiser la recherche de l'utilisateur

Utilisation de deux paths pour la recherche avec les trois méthodes et pour le chemin de la racine jusqu'au nœud recherché

l'arbre binaire représente le réseau informatique d'une entreprise.

On considère qu'il n'y a aucune redondance, et que le fichier cache a été réaliser afin d'apprendre à extraire des données, les formater et de les réutiliser.

Instructions : inscrivez les points d'actions à prendre en compte.

# Tâches

## Modules

## Contenus

BinTree	Classe d'arbre binaire	RouteManager	Gestionnaire de route	PacketSender	Fonction d'envoi de paquet	TreeVisualizer	Visualisation d'arbre binaire	TreeBuilder	Construction de l'arbre binaire de routage	SearchAlgorithms	Algorithmes de recherche	Queue	Classe de file d'attente (Queue)	Main	Programme principal pour la découverte d'arbre et la visualisation de routage de paquets.	CacheManager	Gestionnaire de cache
Class BinTree	Classe d'arbre binaire	bfs_mark_unreachable	Marque un réseau et tous ses sous-réseaux comme inaccessibles en utilisant un parcours BFS.	send_packet	Envoie l'envoi d'un paquet en utilisant un chemin précalculé et le label d'adresse dans un range de 1 à 255.	print_tree	Affiche l'arbre binaire de routage à l'écran en utilisant des symboles pour représenter la structure de l'arbre.	Build_tree	Cette fonction crée un arbre binaire de routage avec des nœuds représentant des sous-réseaux et des liens connectés.	search_prefix	Fonction de recherche de type "prefix" qui explore d'abord la gauche, puis la droite (Breadth First Search).	_init__	Initialise une file vide.	main	Fonction principale qui orchestre le programme.	save_search_cache	Sauvergarde les informations de recherche dans un fichier cache "search_cache.txt".
Add : sous-arbres de sous-arbres	Gestion de la taille de l'arbre	delete_route	Supprime un réseau et ses sous-réseaux de l'arbre de routage.	Latency	Choisir une latence aléatoire dans le range de 1 à 29.		Définir des couleurs pour l'affichage dans le terminal.			search_infix	Fonction de recherche de type "infix" qui explore gauche, puis droite, puis gauche.	enqueue	Ajoute un élément à la file (FIFO).	is_network_alive	Vérifie si un réseau existe dans l'arbre avant suppression.	load_search_cache	Écrit le cache mis à jour dans le fichier.
	Gestion de la hauteur de l'arbre					colors	Blanc : couleur pour les nœuds non recherchés Bleu : couleur pour le nœud recherché			search_suffix	Fonction de recherche de type "suffix" qui explore gauche, puis droite, puis gauche.	dequeue	Retire et retourne l'élément en tête de la file.	handle_network_down	Gère la suppression d'un réseau injoignable.	load_search_cache	Charge les données de recherche à partir du fichier cache.
										bfs_search	Recherche un nœud dans l'arbre en utilisant un parcours en largeur (BFS).	peek	Retourne l'élément en tête de file sans le retirer.	search_route	Recherche une route dans l'arbre.		
										find_fastest_path	Fonction pour trouver la méthode de recherche la plus rapide.	is_empty	Vérifie si la file est vide.	handle_packet_sending	Gère l'envoi d'un paquet si demandé.		
										get_path_from_root	Trouve le chemin de la racine à un nœud spécifique dans l'arbre.						
										dfs	Parcours en profondeur d'abord (DFS) pour trouver un nœud dans l'arbre.						

## Instructions

- Étape 1 :** Ajouter les modules à créer sur les autocollants jaunes.
- Étape 2 :** Ajouter le sujet de chaque module sur les autocollants gris.
- Étape 3 :** Ajouter les fonctionnalités de chaque module sur les autocollants verts.
- Étape 4 :** Ajouter les fonctionnalités supplémentaires sur les autocollants cyans.
- Étape 5 :** Ajouter les Ressources documentaires à créer (descriptions/documentations).

- Définir une hiérarchisation des tâches :** les tâches prioritaires en haut de la liste.
- Tâche fini :** ajouter le symbole pour valider la tâche.
- Bloquante :** changer la couleur des autocollants en rouge si vous rencontrez un point bloquant.

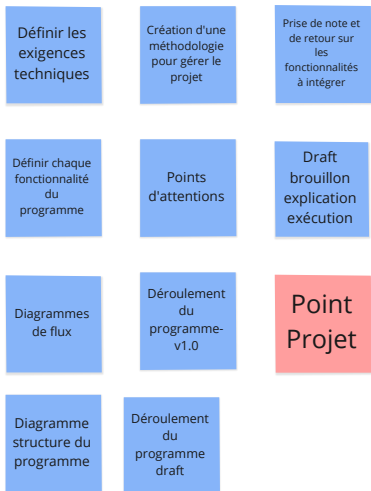
## Release Legend



# Off-boarding plan for Maxence

## Headline 1

### Initialisation



## Headline 2

### Exécution / Contrôle



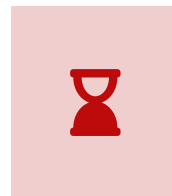
## Headline 3

### Rédaction



## Headline 4

### Clôture



17/ 02 / 2025

22/ 02 / 2025

15/ 03 / 2025

23/ 03 / 2025

# Explication du programme

Ce diagramme représente la structure du programme et les interactions entre les différents fichiers

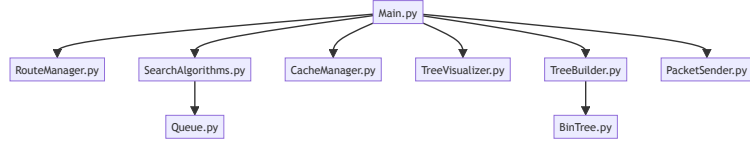
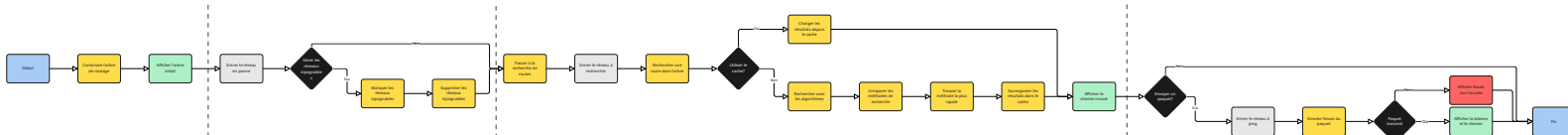


Diagramme de flux qui illustre le déroulement du programme, depuis la construction de l'arbre jusqu'à l'envoi de paquets.

Le diagramme montre les étapes principales et les interactions entre les différents modules.



```

    192.168.1.0/24
    └── 10.0.0.0/8
        ├── 192.168.10.0/24
        ├── 192.168.40.0/24
        ├── 192.168.50.0/24
        ├── 192.168.20.0/24
        ├── 172.18.0.0/16
        └── 192.168.60.0/24
172.16.0.0/16
└── 172.17.0.0/16
    ├── 172.18.20.0/24
    ├── 172.18.30.0/24
    ├── 192.168.30.0/24
    ├── 192.168.45.0/24
    └── 172.19.0.0/16

```

Entrez le réseau en panne (ex: 192.168.50.0/24) : 192.168.50.0/24

Le réseau 192.168.50.0/24 et ses descendants ont été supprimés.

Entrez le réseau en panne (ex: 192.168.50.0/24) :

Attention : le réseau n'existe pas dans la table de routage.

[illegible][illegible][illegible]

- Envoyer vers, envoyer en paquet 7 (oid/oui) : oui  
Envoyer l'adresse du réseau à joindre : 192.168.30.0/24
- Envoyer en transit de 192.168.30.0/24...
- Envoyer en transit de 192.168.1.0/24 à 192.168.30.0/24 (interface 3)
- Envoyer en transit de 192.168.0.0/24 à 192.168.30.0/24 (interface 1)
- Envoyer arrivé à destination 192.168.30.0/24 en 10 ms.

Envoyer vers, envoyer en paquet 7 (oid/oui) : oui  
Envoyer l'adresse du réseau à joindre : 192.168.0.0/24

- Recherche de 192.168.0.0/24 avec la méthode DFS...
- Rechercher vers l'interface 300: 192.168.0.0/24

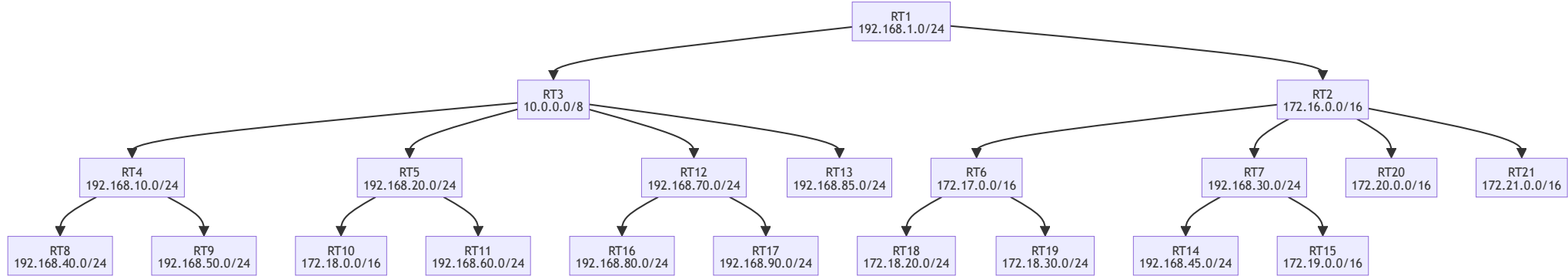
### Explication du Diagramme de Flux

- [illegible]

- **Constituants de l'arbre** - Cliquez par [l'icône d'arbre](#)
- **Génération des réseaux hiérarchiques** - Cliquez par [l'icône d'arbre](#)
- **Recherche de routes** - Cliquez par [l'icône d'arbre](#)
- **Stock de paquets** - Cliquez par [l'icône d'arbre](#)
- **Visualisation de l'arbre** - Cliquez par [l'icône d'arbre](#)

Le diagramme de flux illustre le déroulement logique du programme et montre comment les différents modules interagissent pour offrir les fonctionnalités principales.

# TreeBuilder





# Queue

Queue

elements: List

enqueue

dequeue

peek

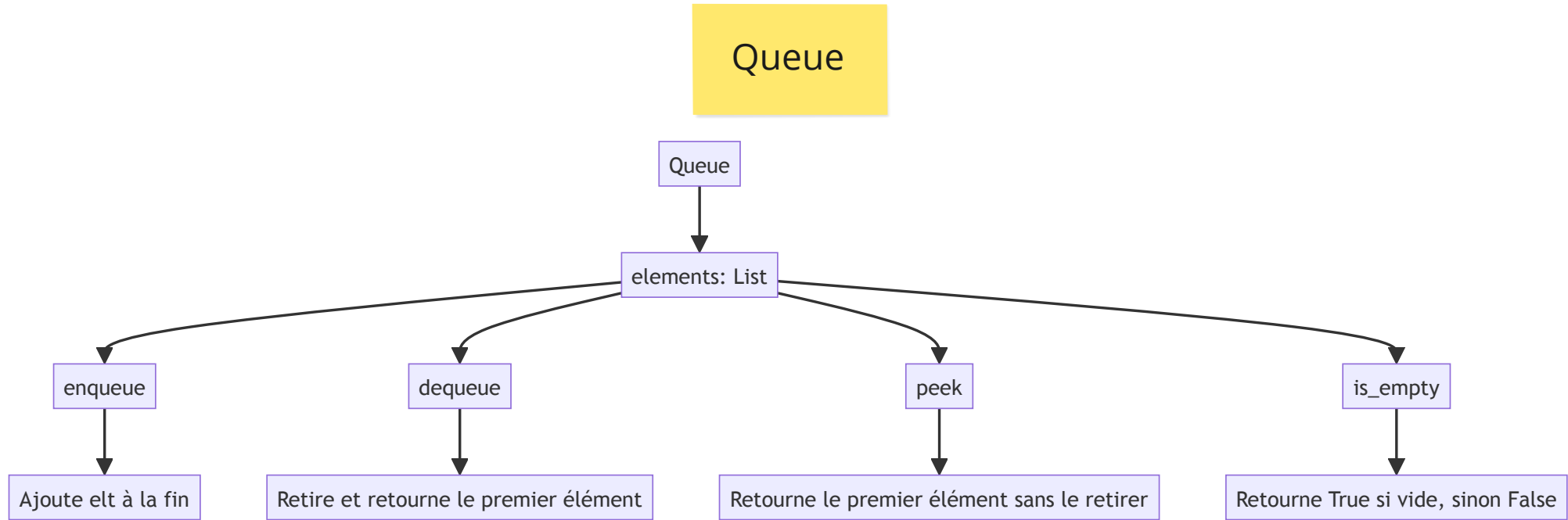
is\_empty

Ajoute elt à la fin

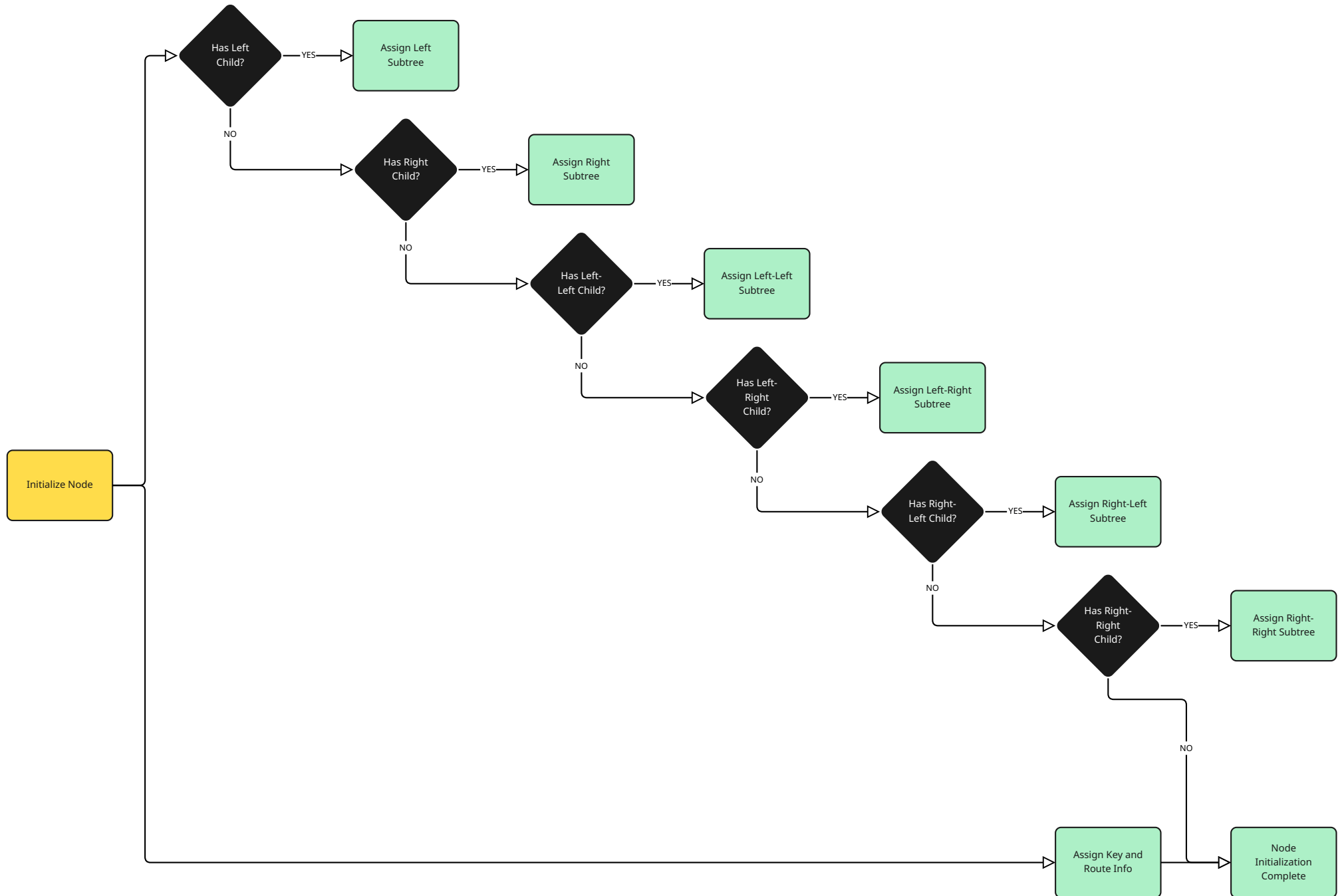
Retire et retourne le premier élément

Retourne le premier élément sans le retirer

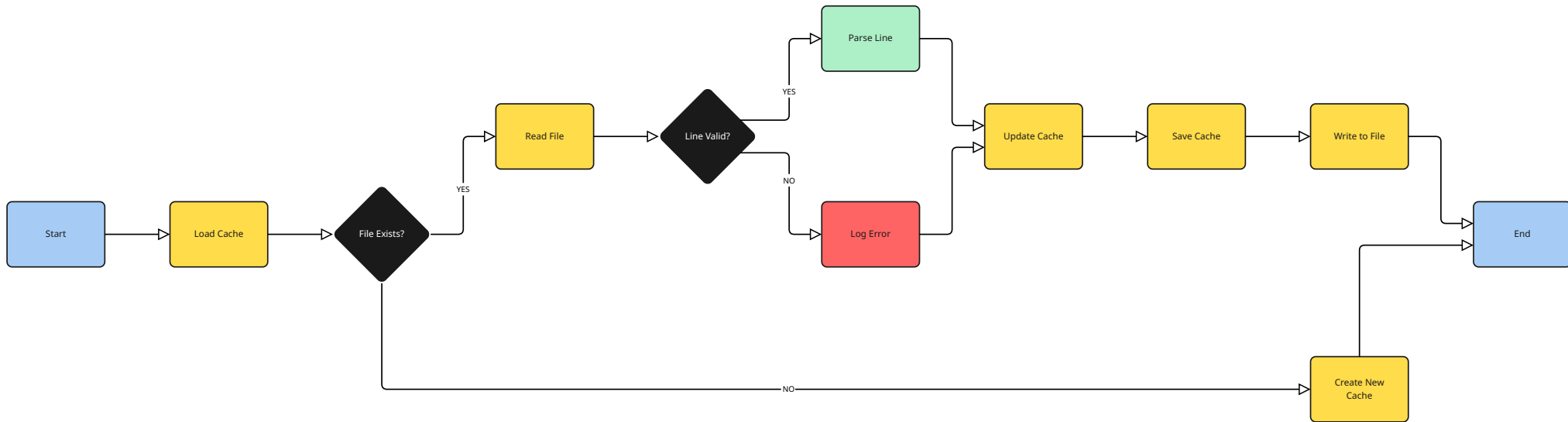
Retourne True si vide, sinon False



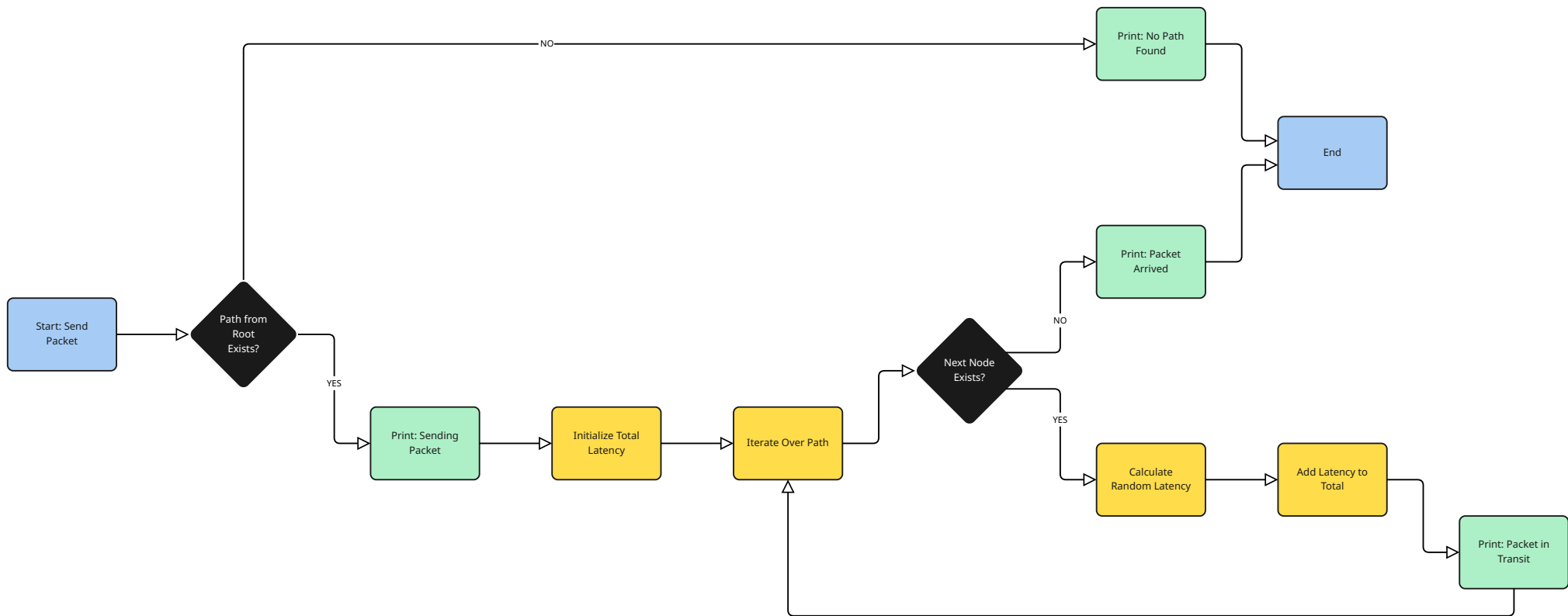
# BinTree



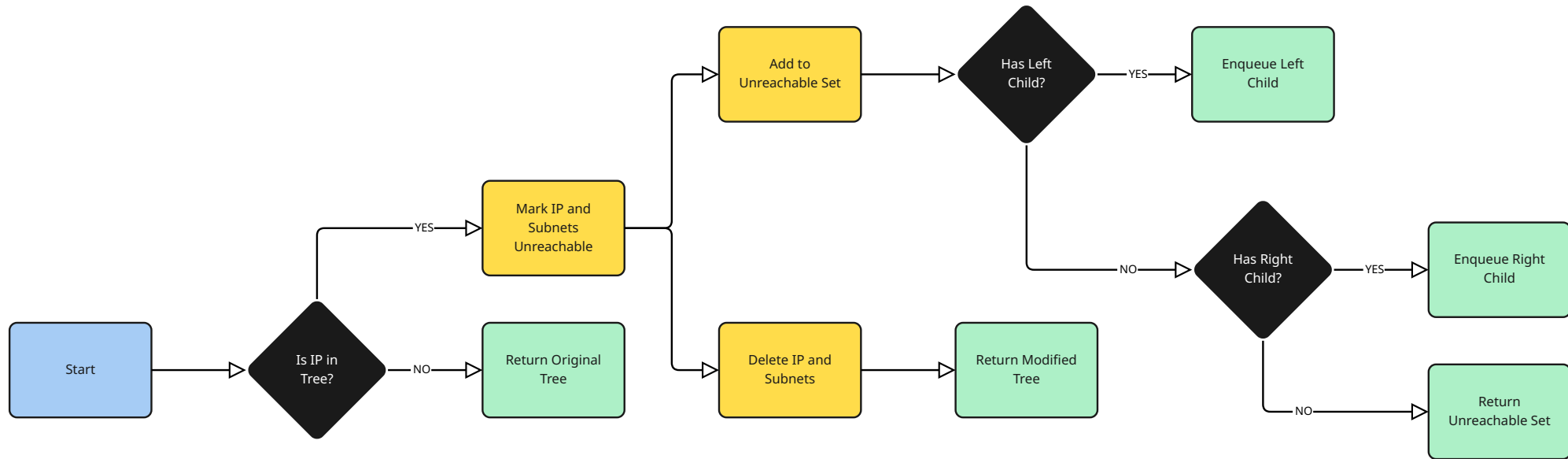
# CacheManager



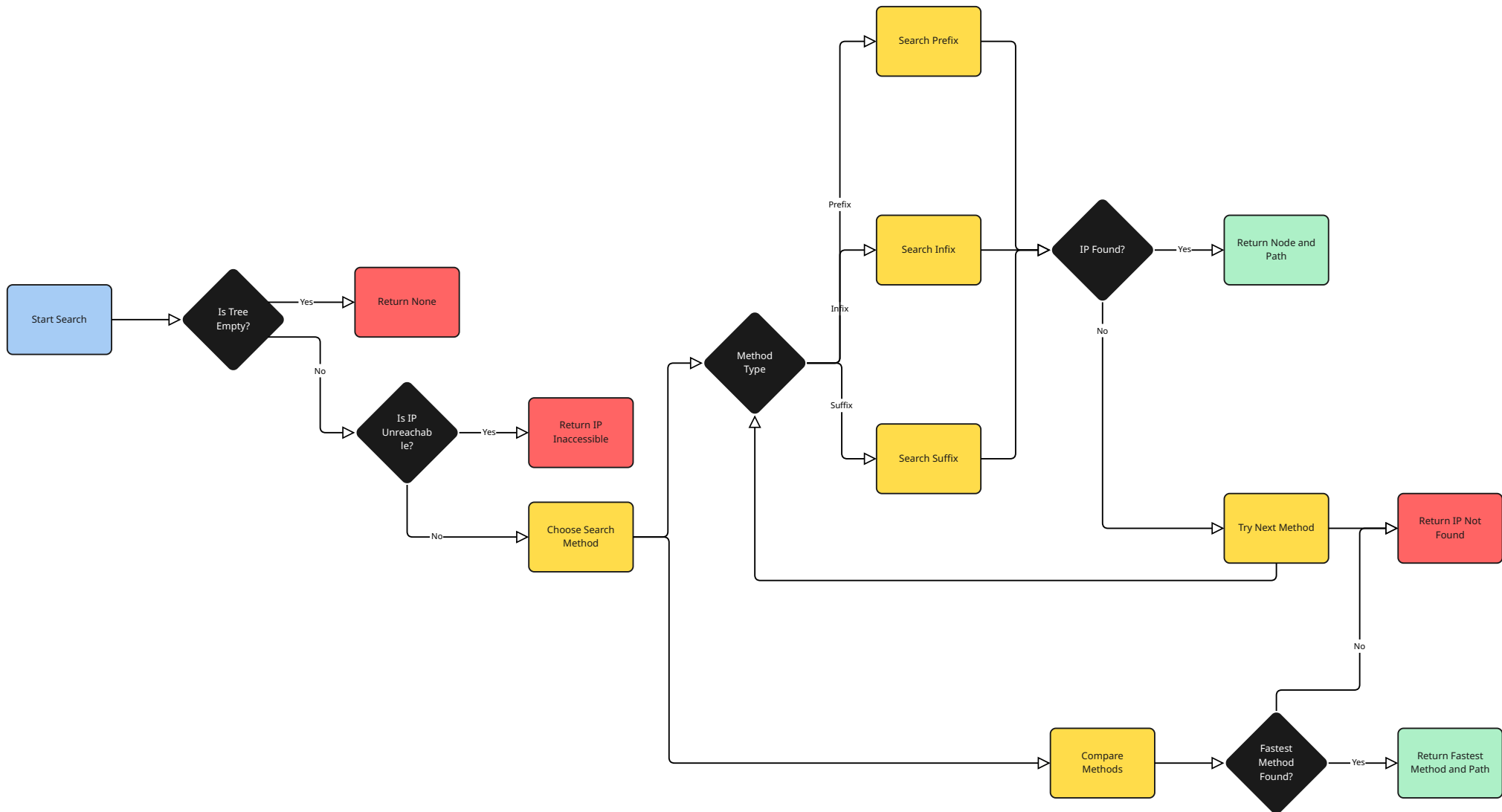
# PacketSender



# RouteManager



# SearchAlgorithms



# TreeVisualizer

