Agent de résolution de problèmes

Aziz KHAMJANE

Algorithmes de recherche en IA

- Introduction
- Agents de résolution de problème
- Exemples de problèmes
- Algorithmes de recherche de base (recherche non informée)

Introduction

De nombreux problèmes peuvent être définis comme des problèmes de recherche.

Agent de résolution de problème

On modélise le processus de résolution de problème comme étant le parcours d'un espace d'états. La tâche de l'agent est de trouver la séquence d'actions qui mènent vers un état but.

Ceci suppose que l'environnement a les propriétés suivantes:

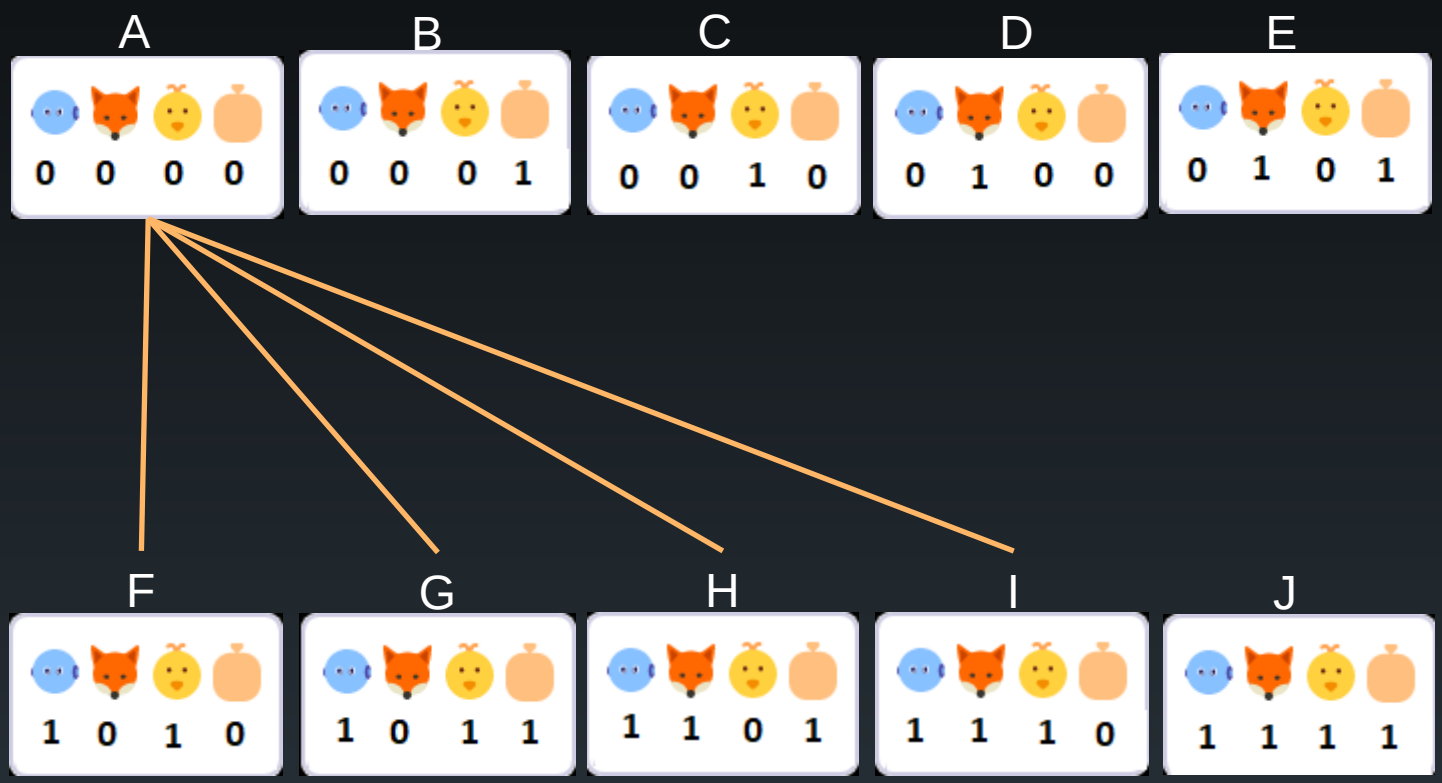
- ✓ Observable: l'agent connait, en permanence, l'état actuel;
- ✓ Discret: pour n'importe quel état, donné, il y a un nombre fini d'actions à choisir;
- ✓ Déterministe: chaque action a une seule issue.

Introduction

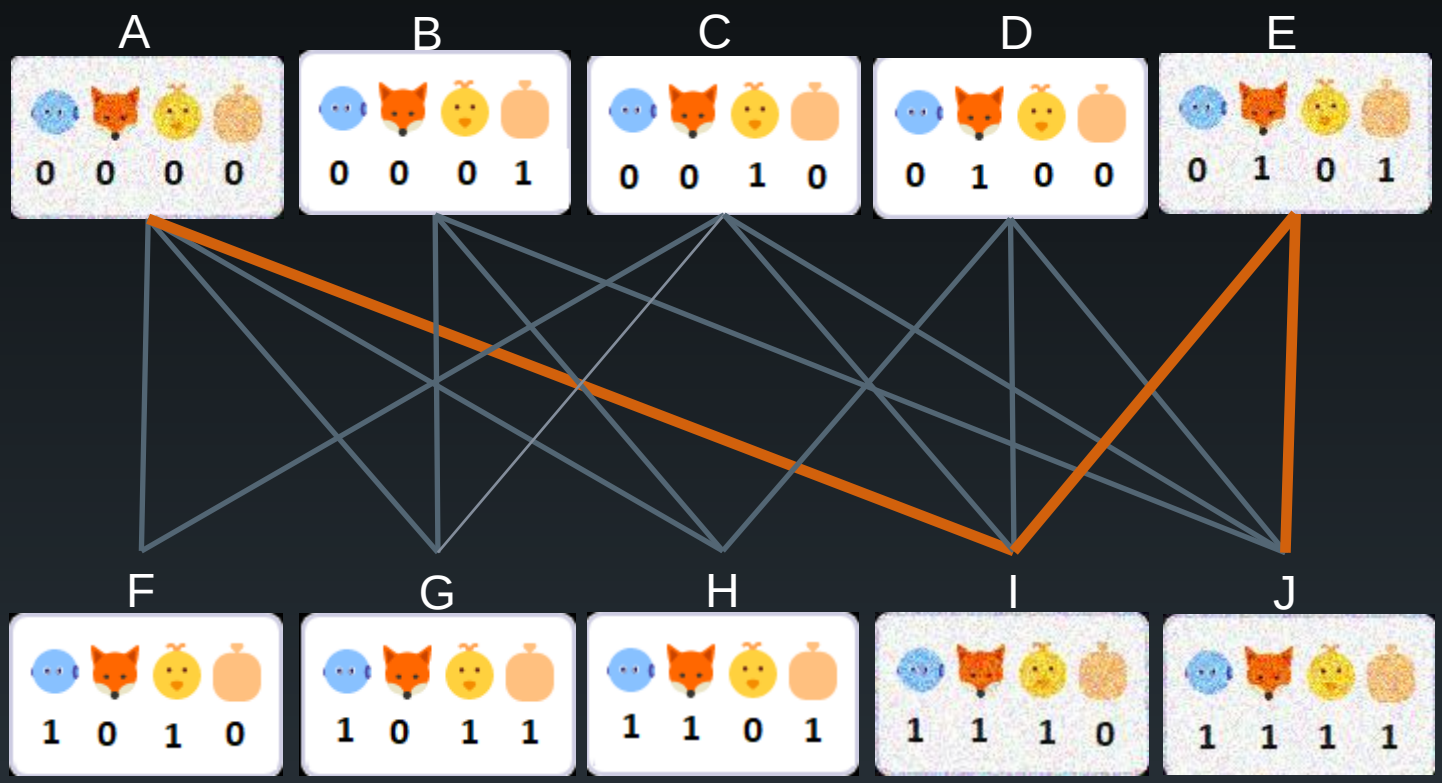
A	B	C	D	E
0 0 0 0	0 0 0 1	0 0 1 0	0 1 0 0	0 1 0 1

F	G	H	I	J
1 0 1 0	1 0 1 1	1 1 0 1	1 1 1 0	1 1 1 1

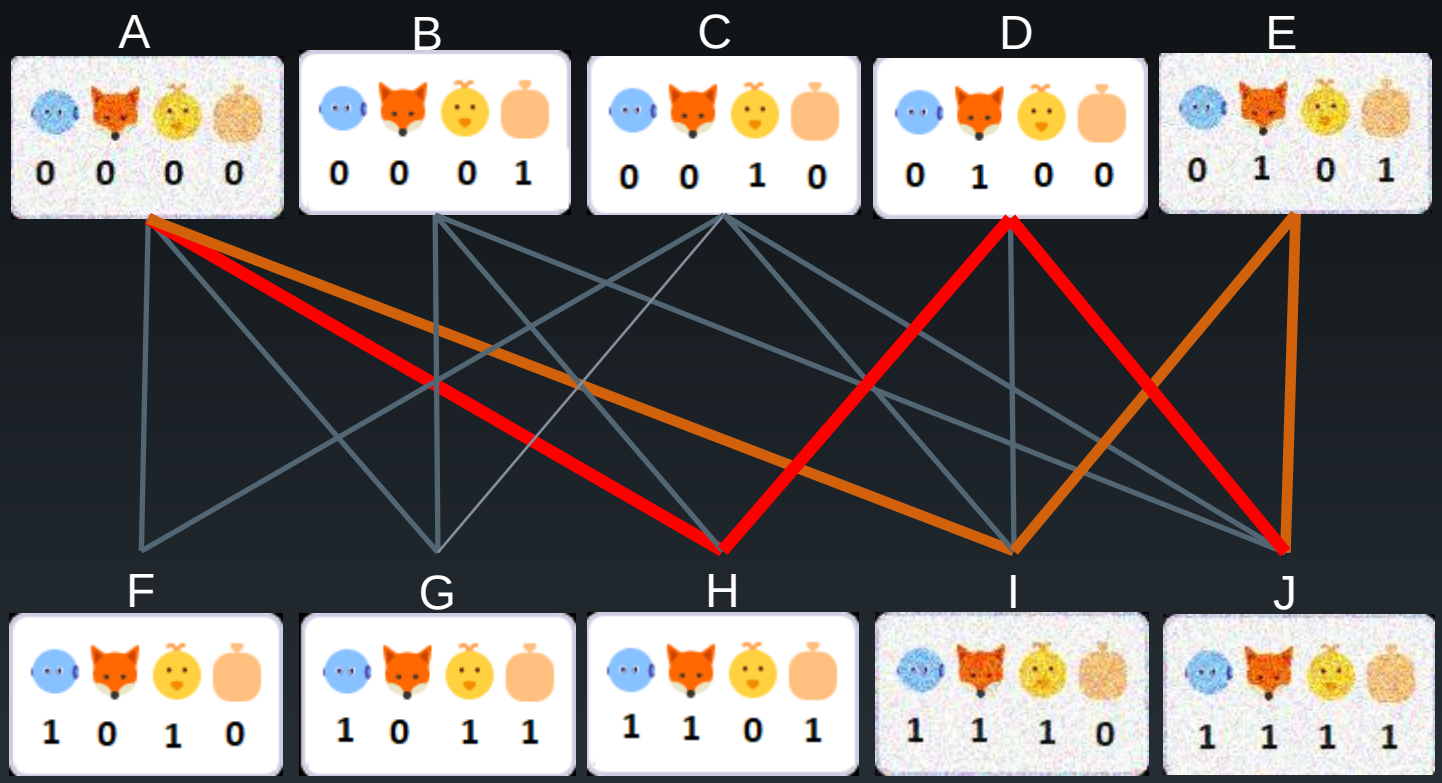
Introduction



Introduction



Introduction



L'espace d‘états

- Le monde réel est trop complexe pour être modélisé
  - L'espace de recherche modélise une vue abstraite et simplifiée du monde réel
- Un état abstrait représente un ensemble d‘états réels
- Une action abstraite représente une combinaison complexe d'actions réelles
  - par exemple, « ville1 → ville2 » représente un ensemble de routes possibles, de détours, d’arrêts, etc.
- Une solution abstraite correspond à un ensemble de chemins qui sont solutions dans le monde réel.

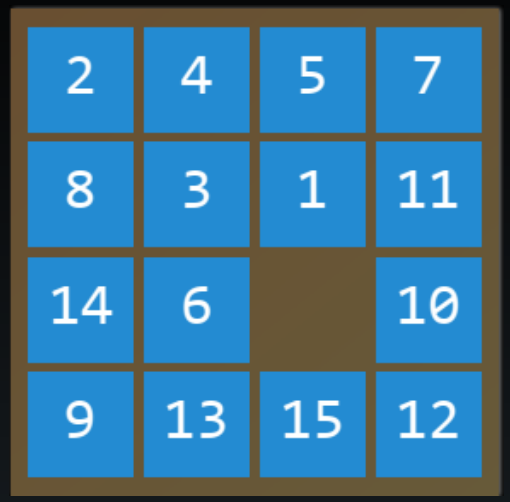
Formulation de problème

Un problème peut être défini par :

- Un Etat initial **:** c'est l'état dans lequel commence l'agent.
- Des Actions **:** les actions possibles dont l'agent se dispose**.** Etant donné un état s, la fonction Action(s) retourne l'ensemble des actions qui peuvent être exécutées dans s.
- Un Modèle de transition**:** c'est une description de ce que chaque action réalise; la fonction Résultat(s,a) renvoie l'état résultant de l'exécution de l'action a dans s.
- Un Test de but: il détermine si un état donné est un but.
- Un coût : fonction numérique qui attribue un coût à chaque chemin. c(s,a,s') est le coût de l'étape qui consiste à entreprendre l'action a dans l'état s pour atteindre s'.

Solution : un ensemble d'actions qui mène de l'état initial vers l'état but

Exemple 2 : 8-puzzle

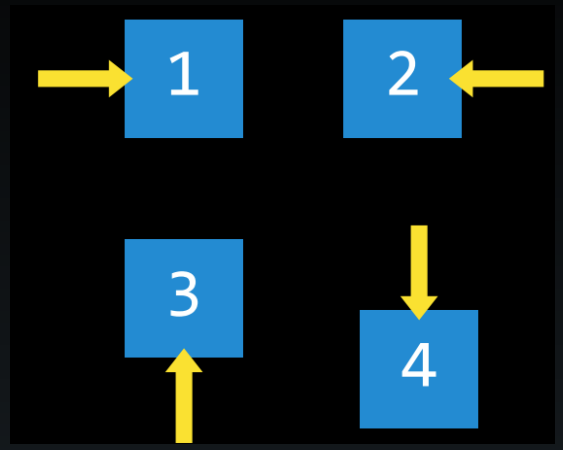


Etat initial

- **Etat**?
- configuration des pièces
- **Actions**?
- Déplacer la pièce adjacente à la case vide vers la gauche, la droite, le haut, le bas.
- **Test du but**?
- Les pièces sont ordonnées
- **Coût du chemin**?
- 1 par action

Exemple 2 : 8-puzzle

Actions(s)

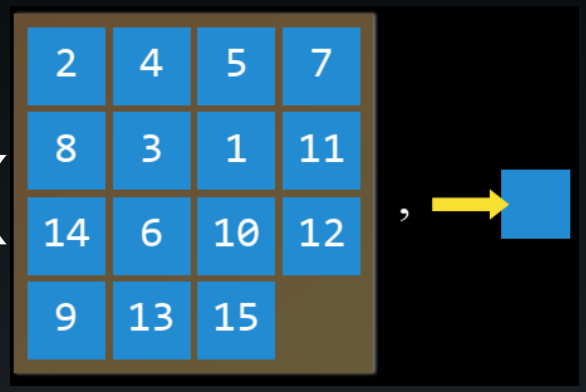


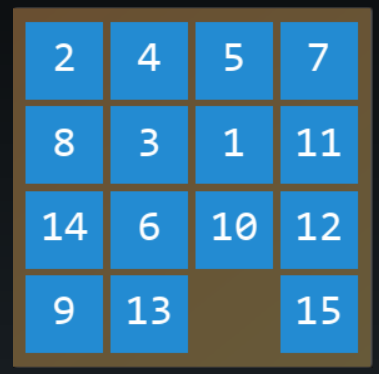
2	4	5	7
8	3	1	11
14	6	10	12
9	13		15

Exemple 2 : 8-puzzle

Modèle de transition

Résultat(s,a)

Résultat(

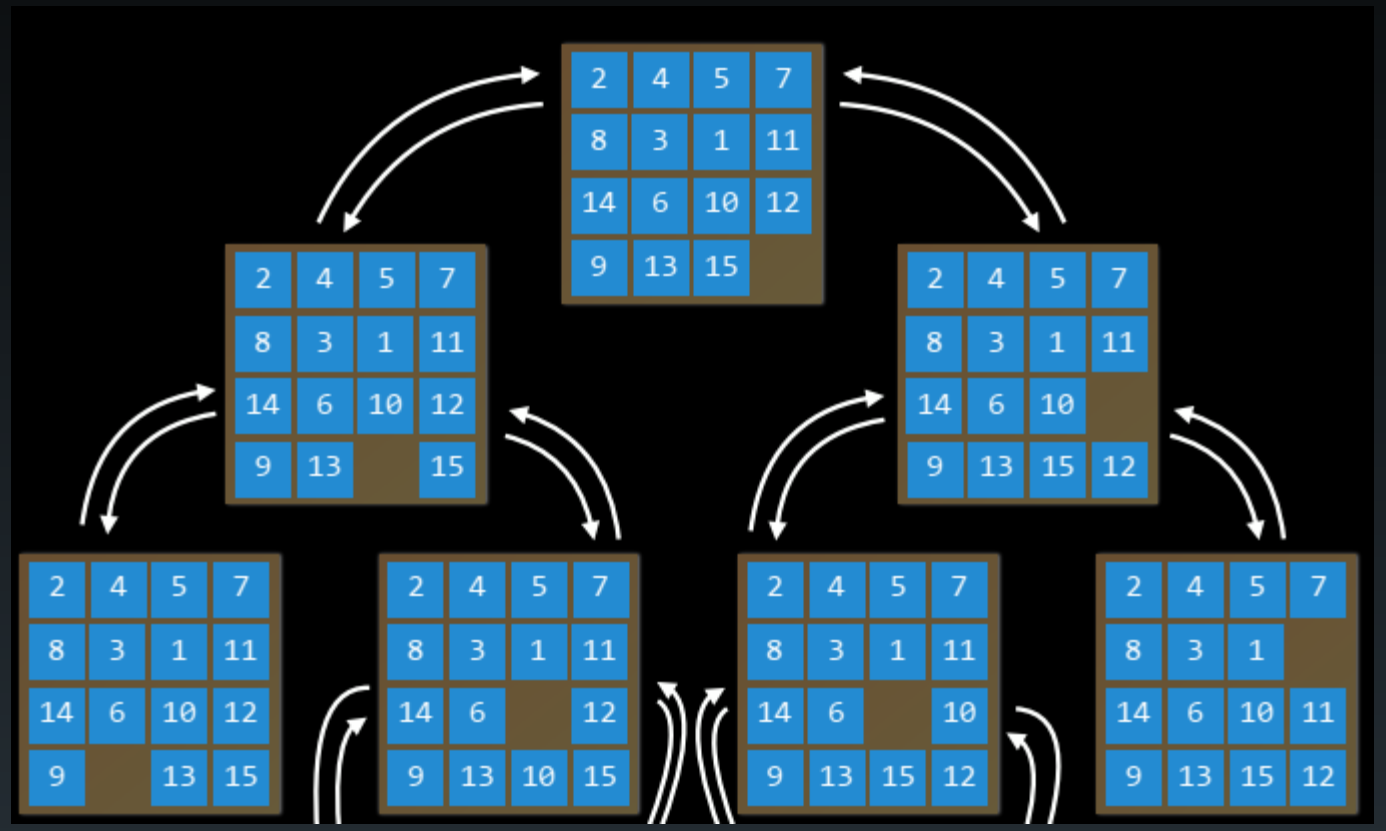
, → ■)= 

Résultat(

, ↓ ■)= 

Exemple 2 : 8-puzzle

L’espace d’états : est l’ensembles des états qu’on peut atteindre à partir de l’état initial par n’importe quelle séquence d’actions



Exemples de problèmes réels

- Recherche de parcours:
  Itinéraires automatiques, guidage routier, planification de routes aériennes, routage sur les réseaux informatiques, ...
- Robotique
  Assemblage automatique, navigation autonome, ...
- Planification et ordonnancement
  Horaires, organisation de tâches, allocation de ressources, ...

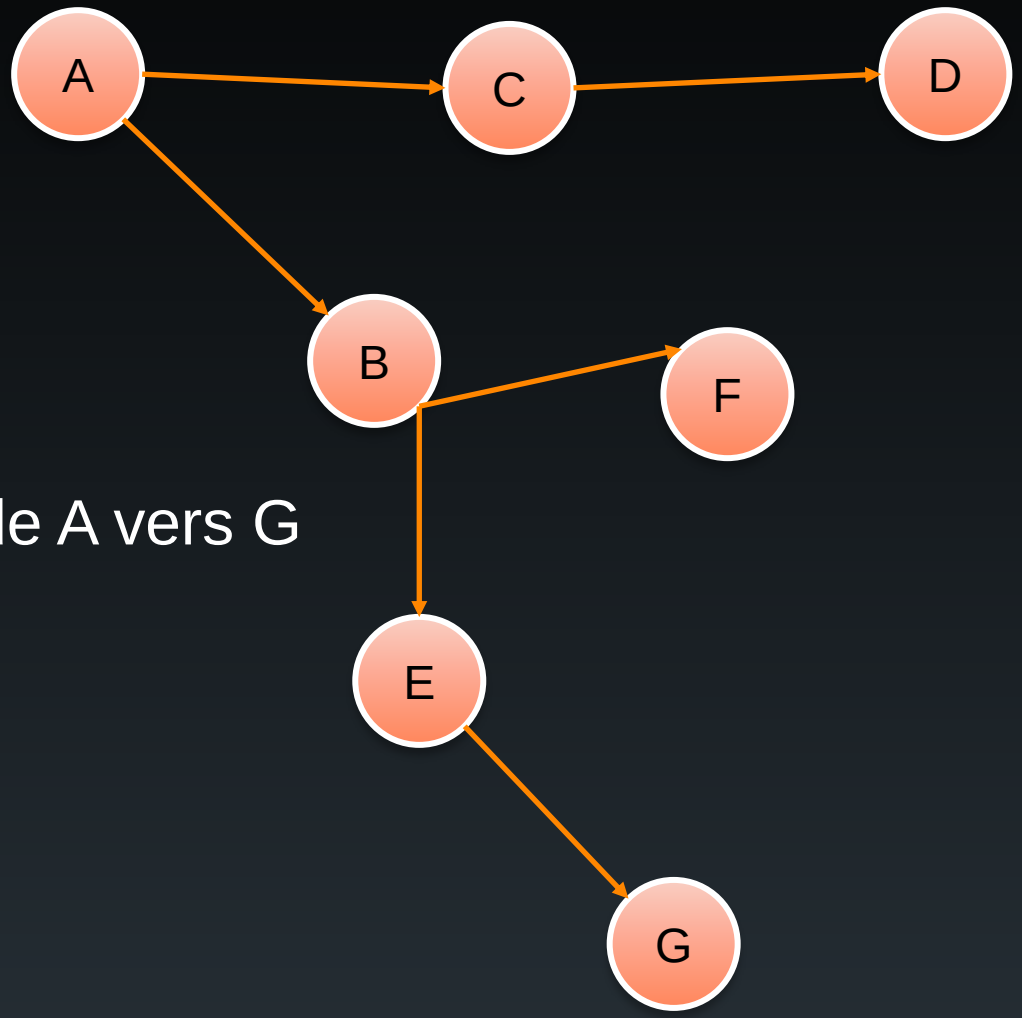
Implémentation des algorithmes de recherche

- Nœud : une Structure de données qui contient
  - L'état
  - Parent (le nœud qui a généré ce nœud)
  - cout du chemin
  - Fils (nœuds fils)

Approche générale de la recherche

- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale

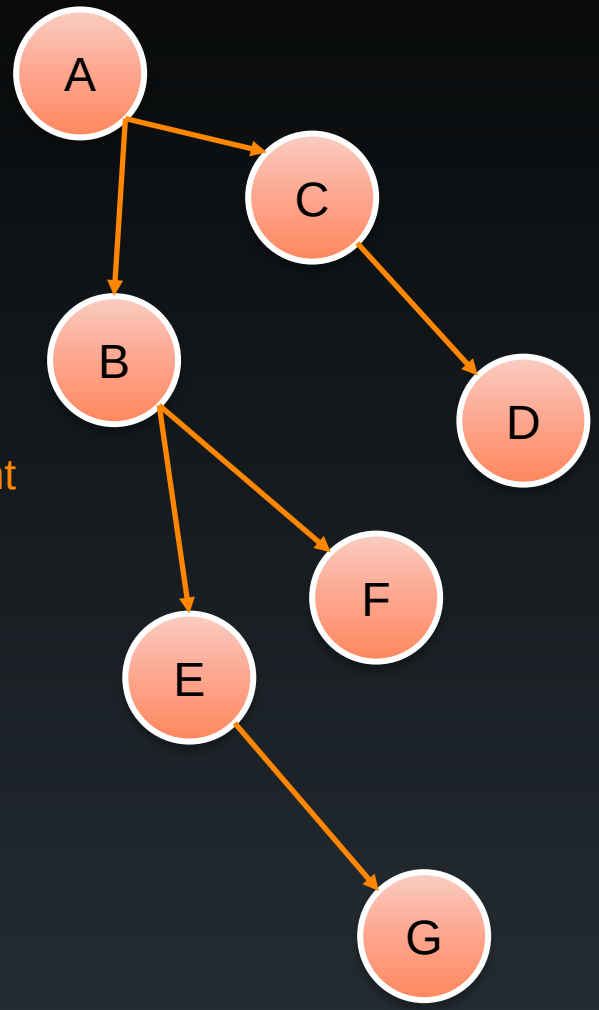


Trouver le chemin de A vers G

Approche générale

FRONTIERE

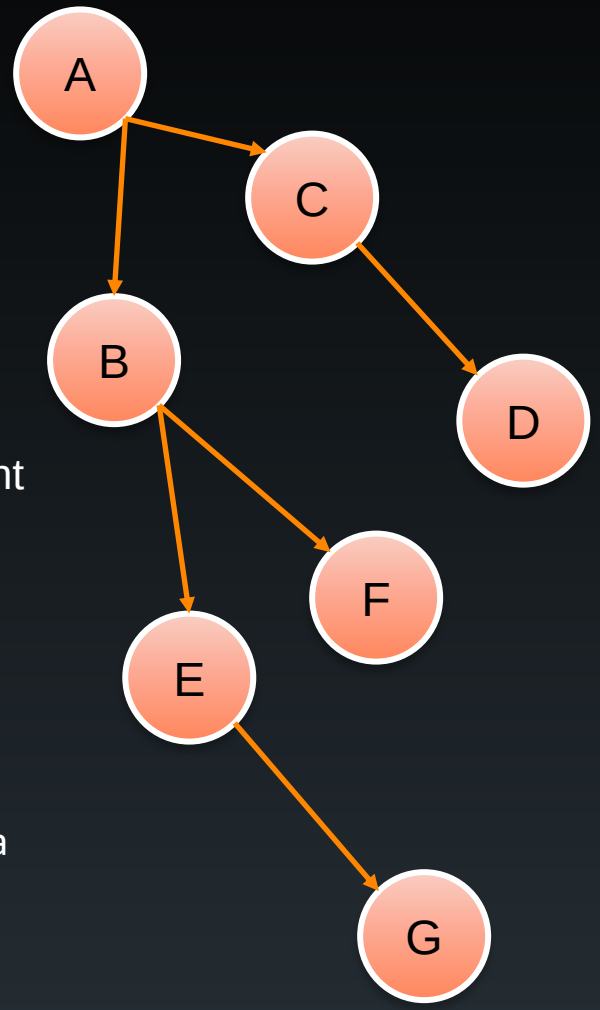
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE



Approche générale

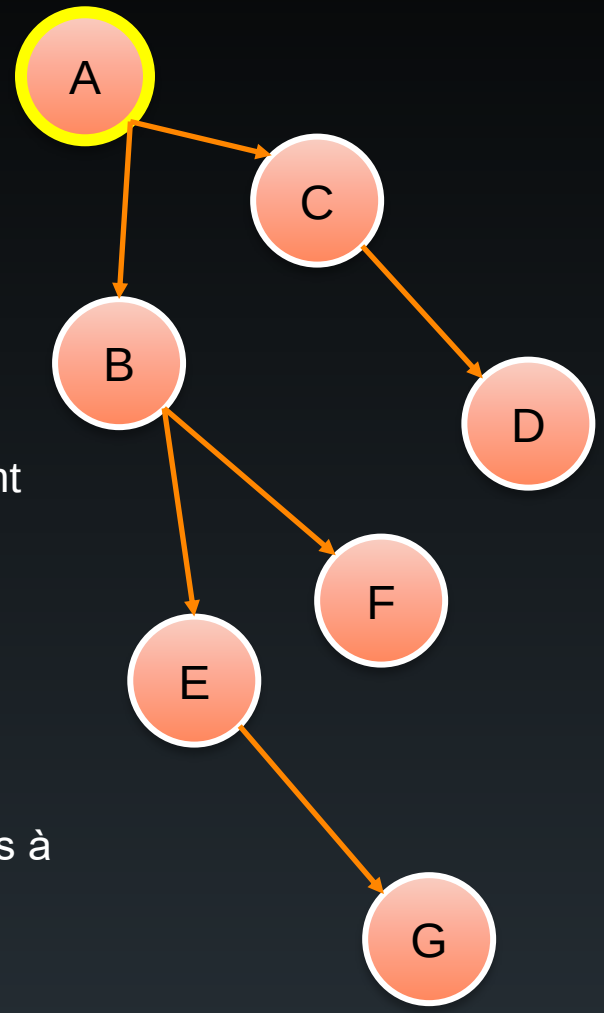
FRONTIERE

- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste OPEN.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste OPEN



Approche générale

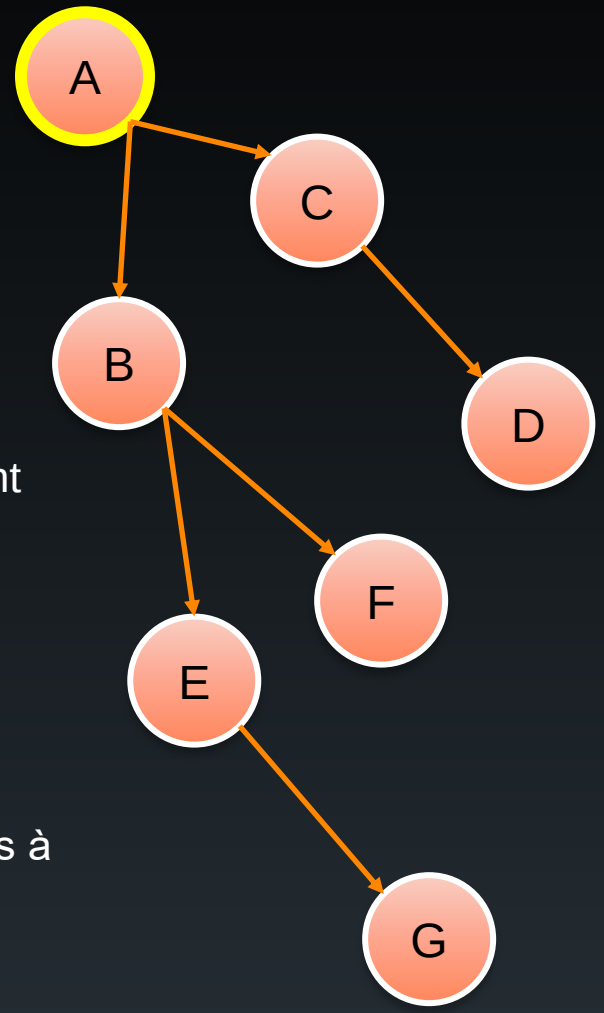
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

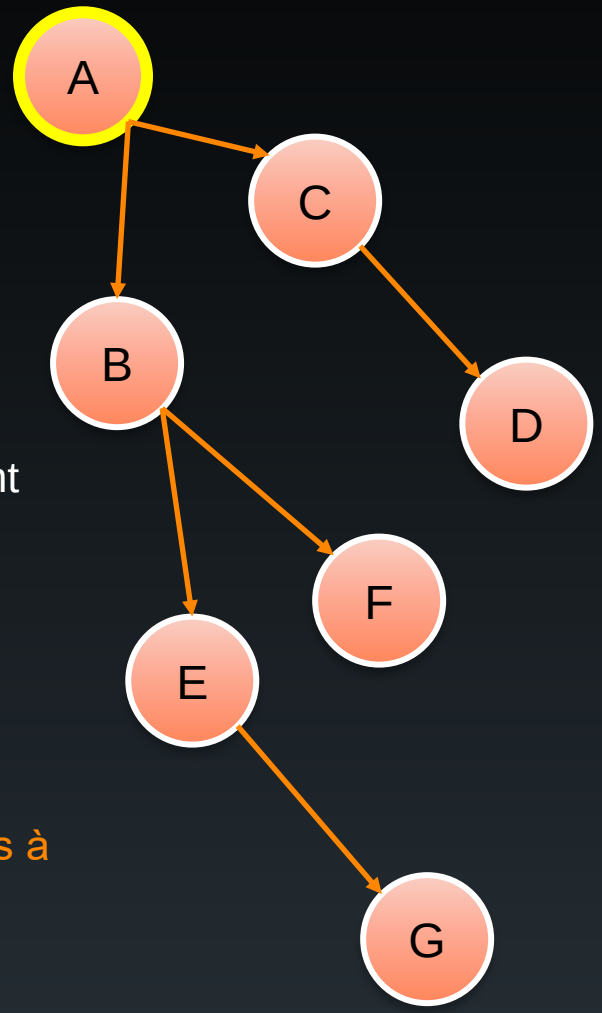
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

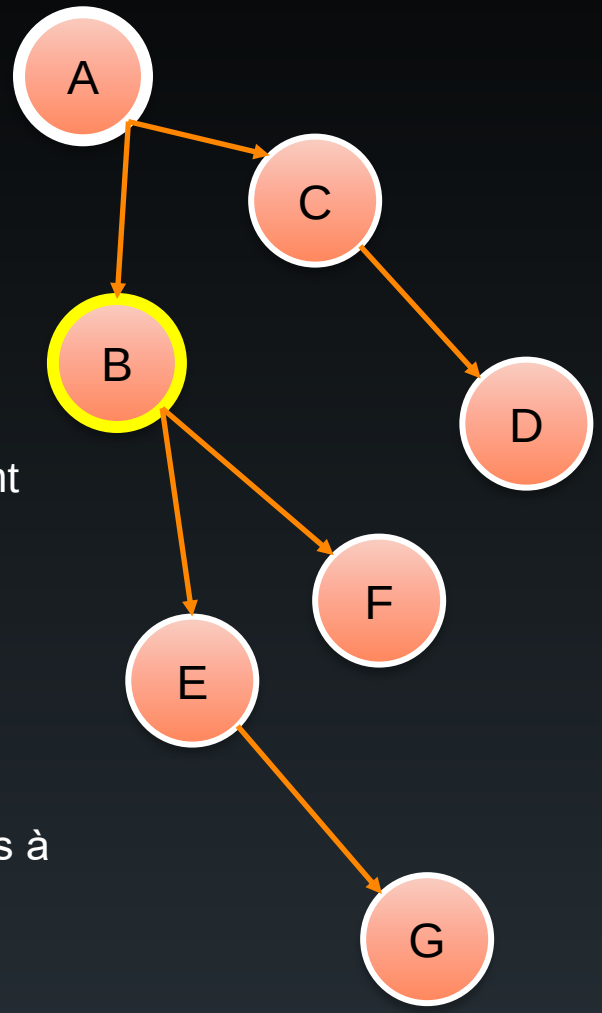
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

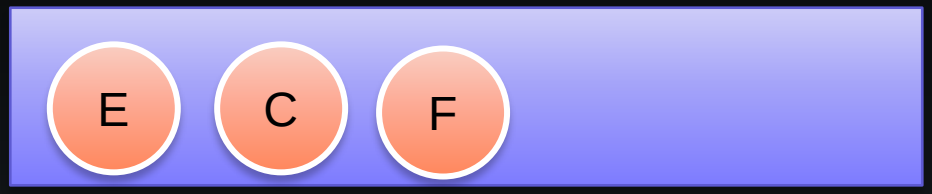
FRONTIERE

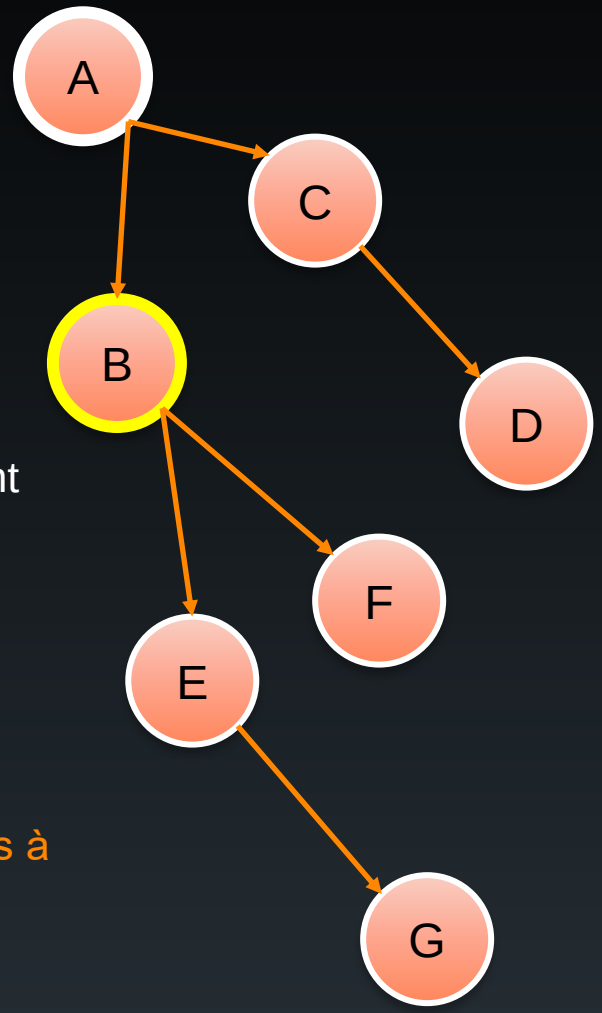


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

FRONTIERE

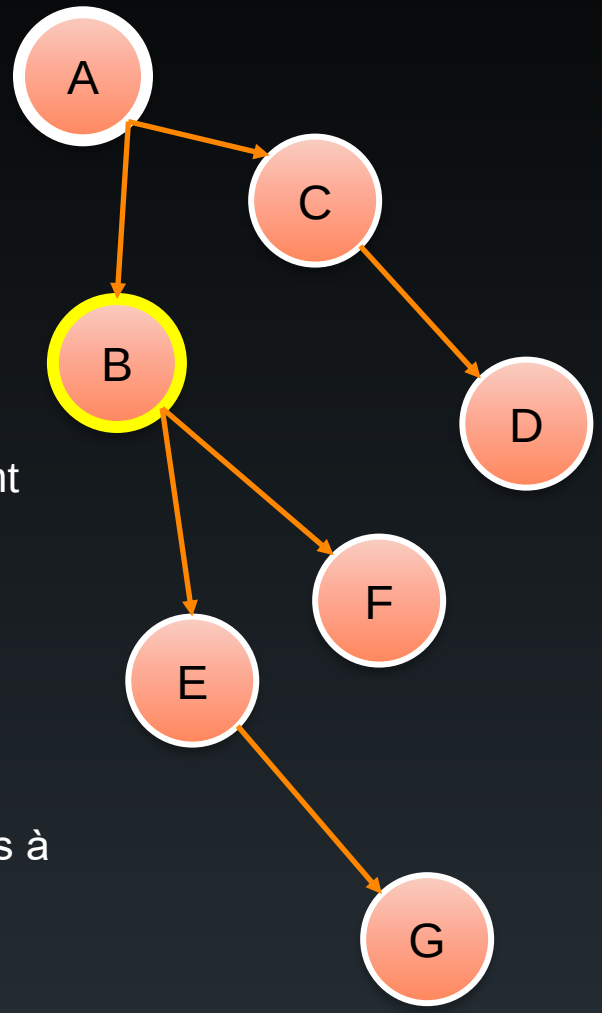




- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

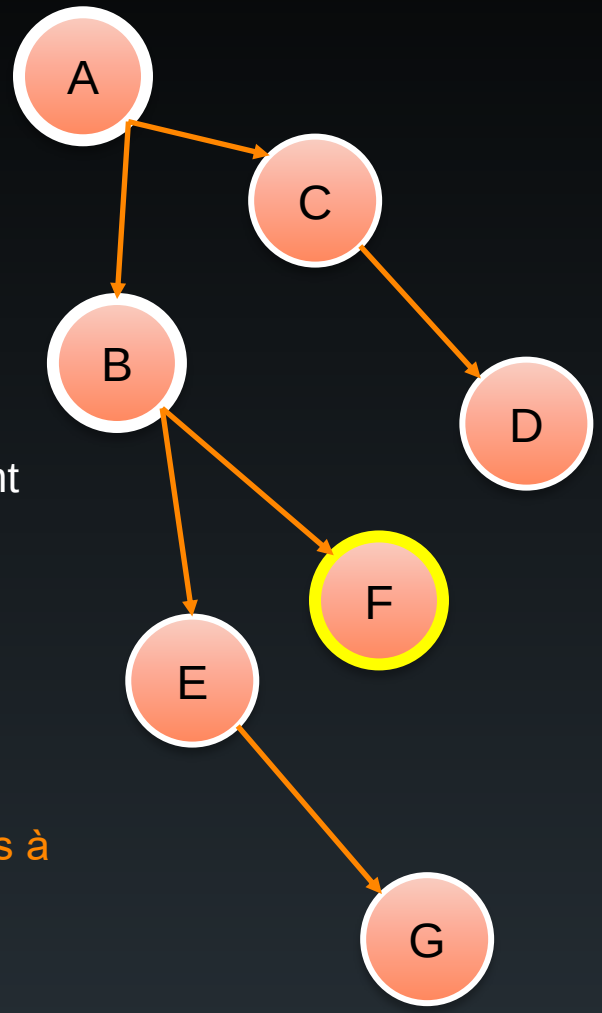
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

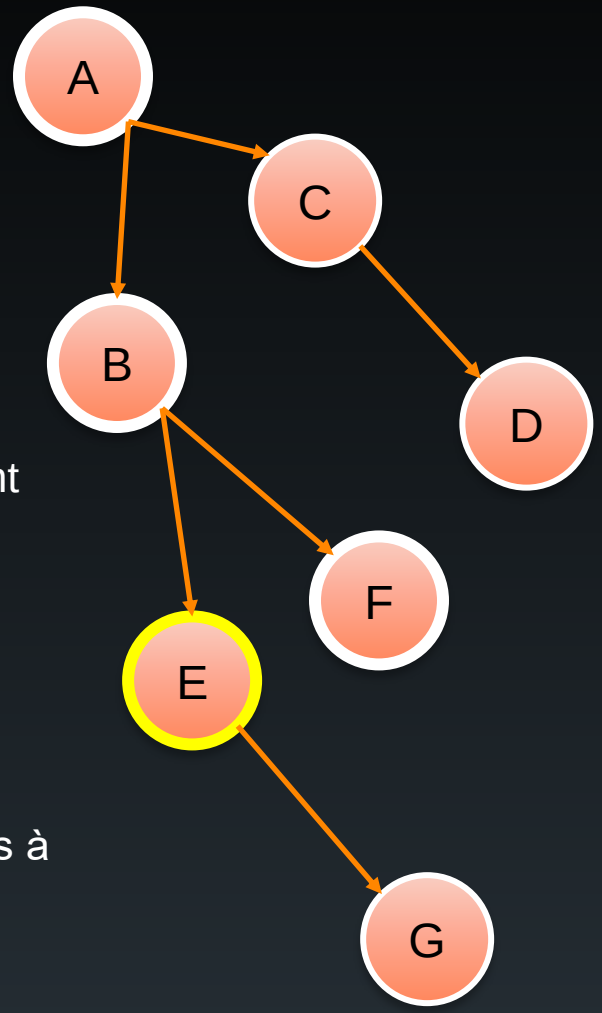
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

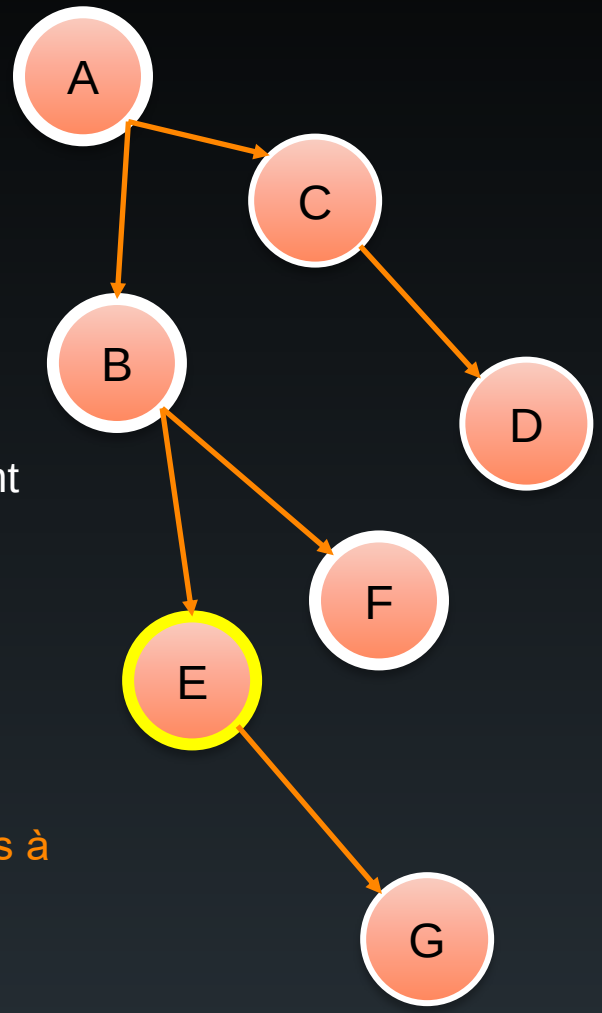
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE

Approche générale

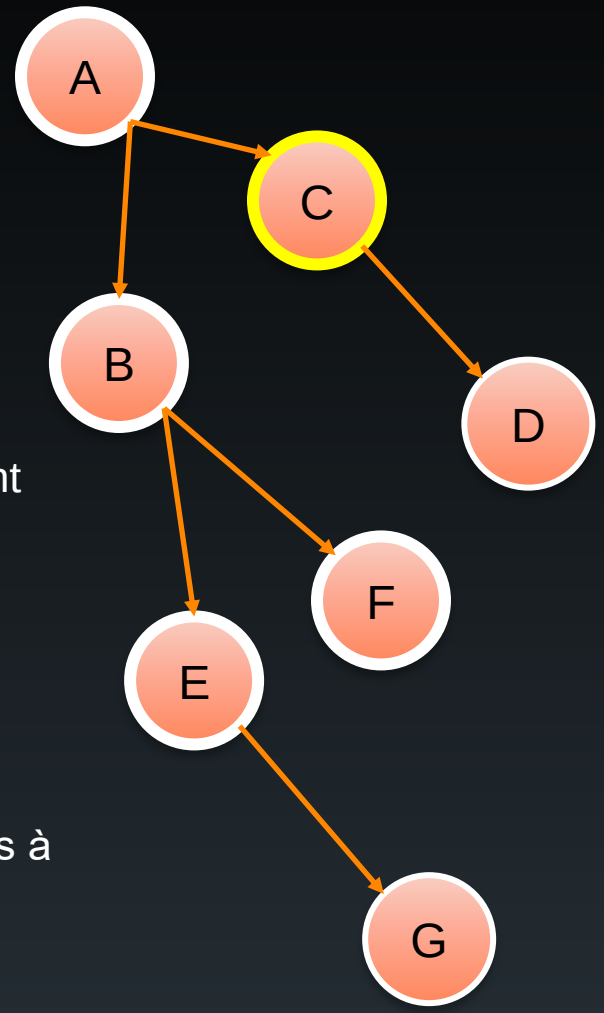
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale

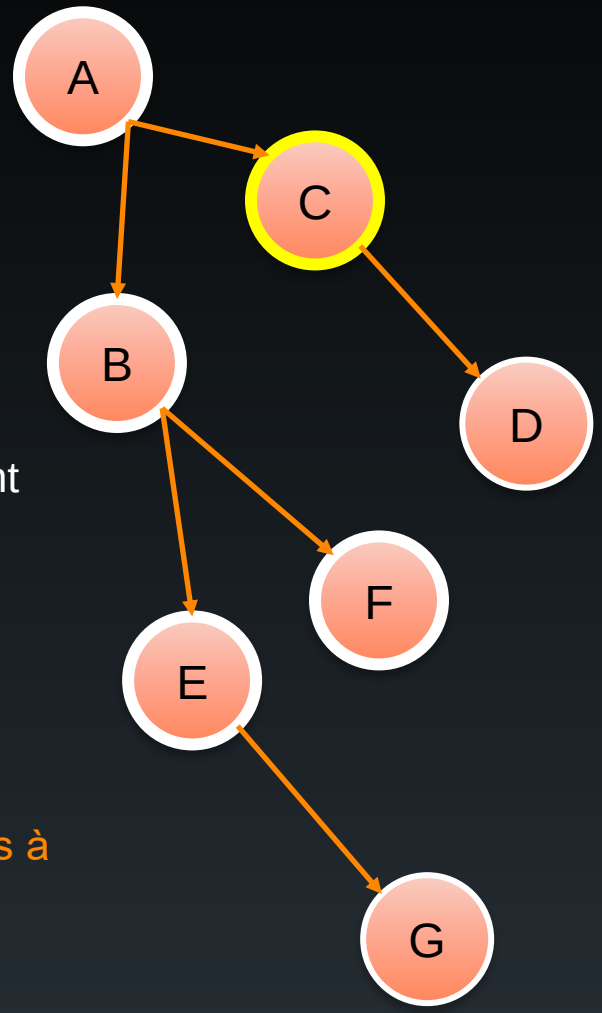
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale

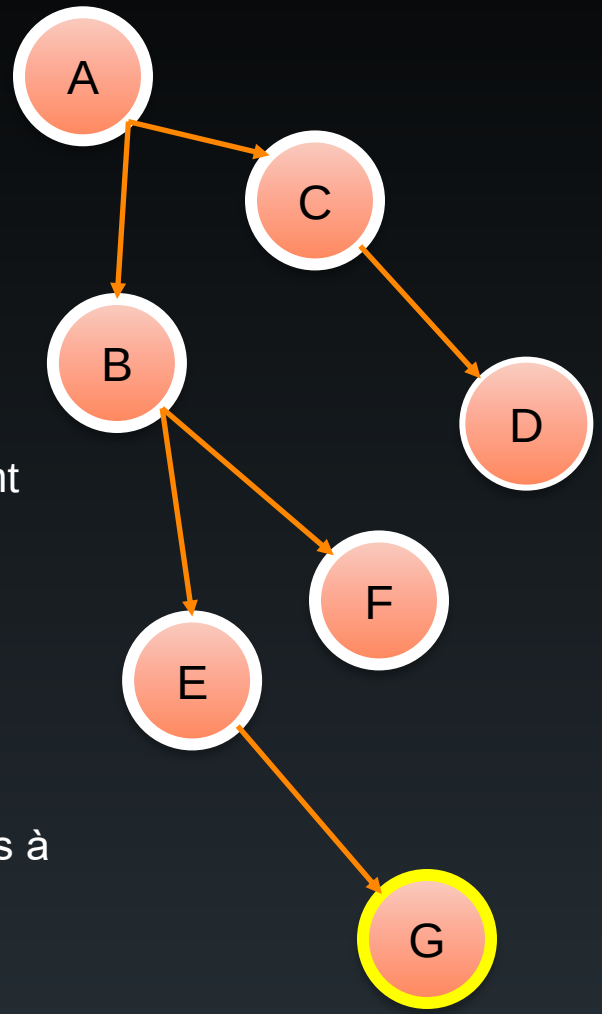
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale

FRONTIERE

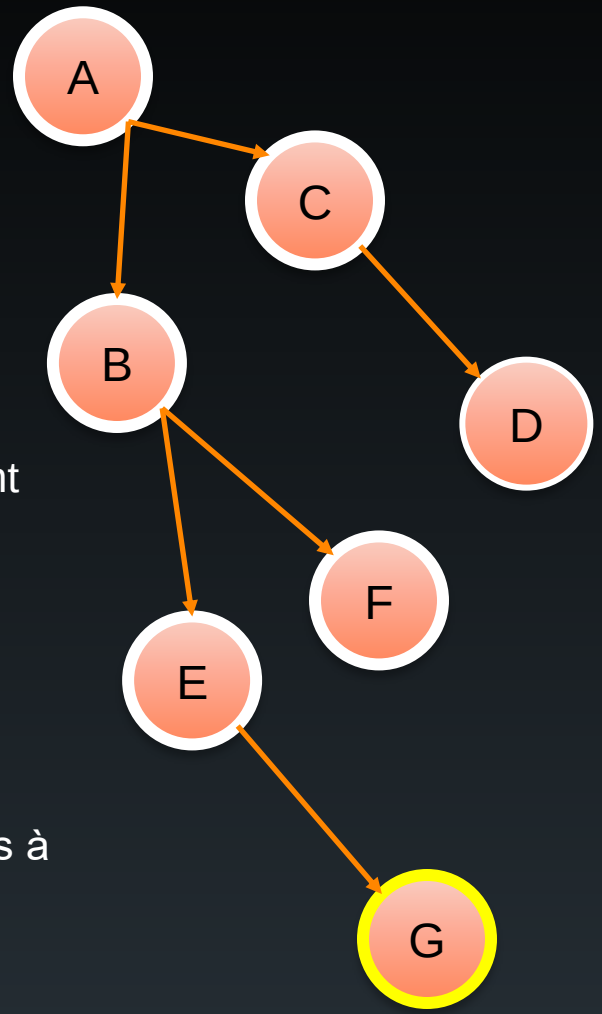


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale

FRONTIERE

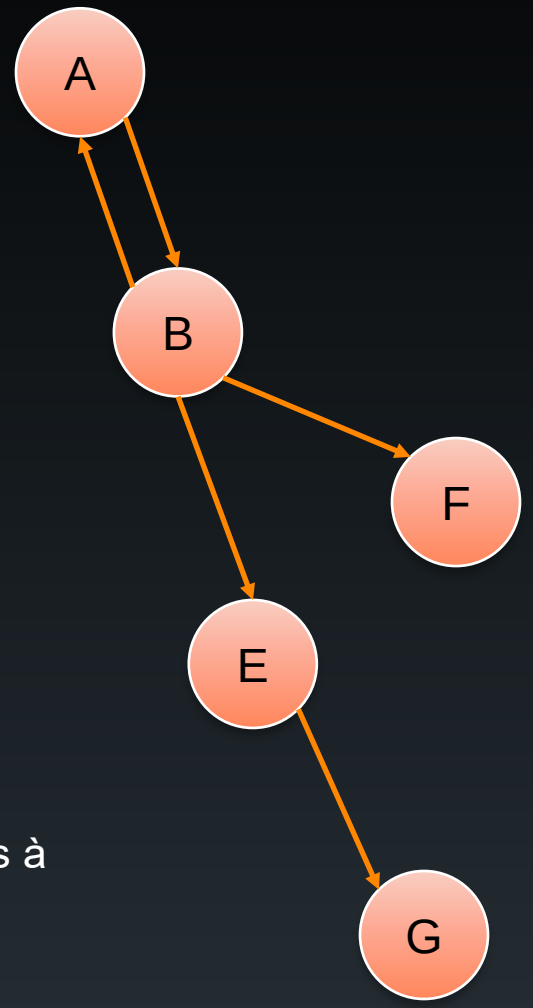
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.



Approche générale (problème)

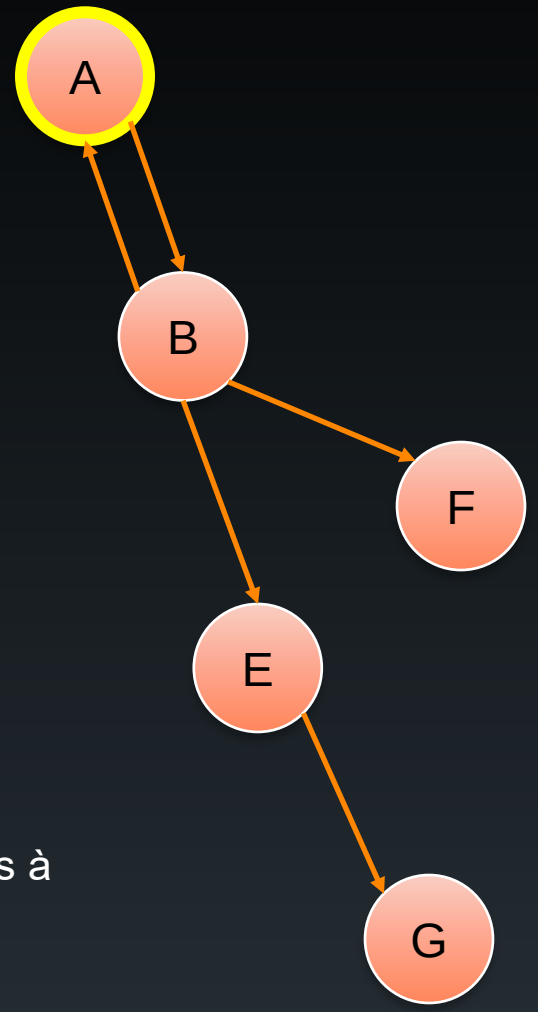
FRONTIERE

- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.



Approche générale (problème)

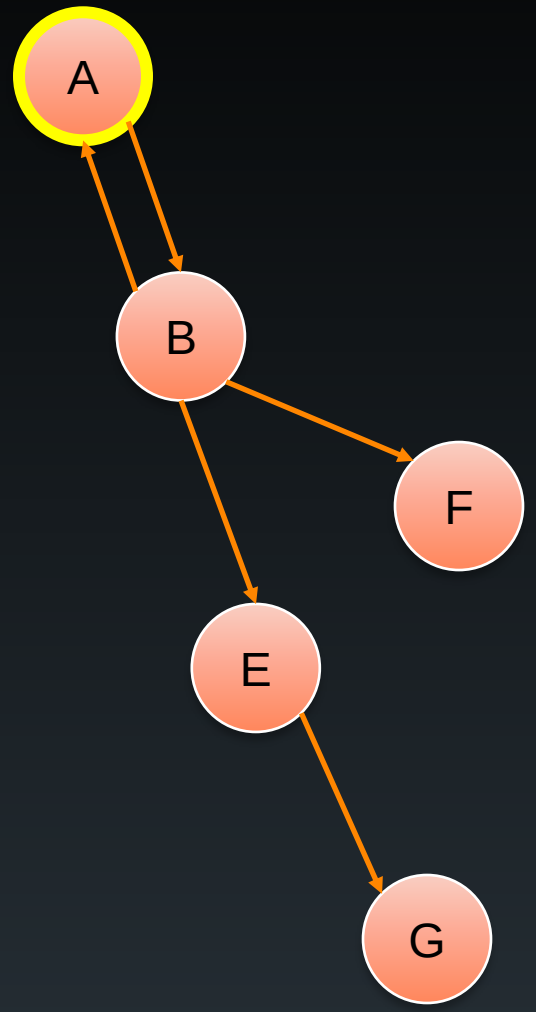
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale (problème)

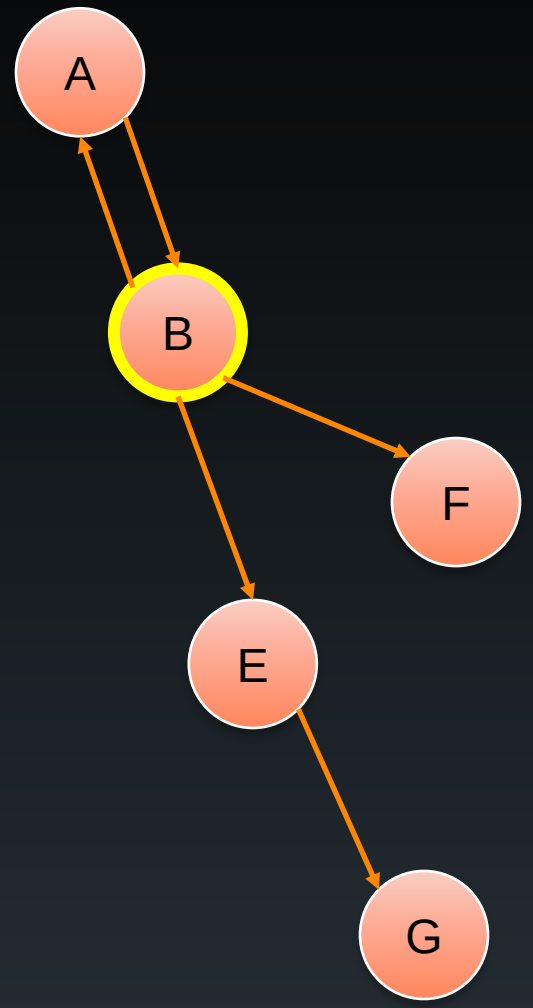
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale (problème)

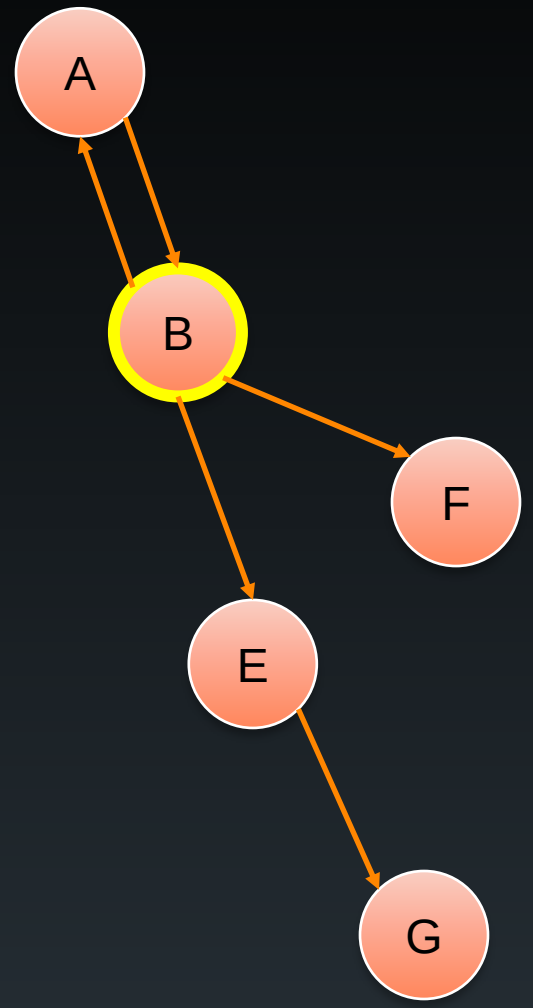
FRONTIERE



- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale (problème)

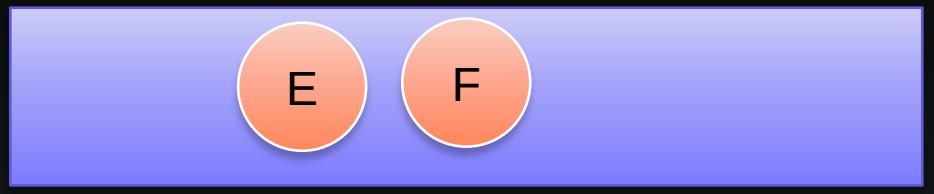
FRONTIERE

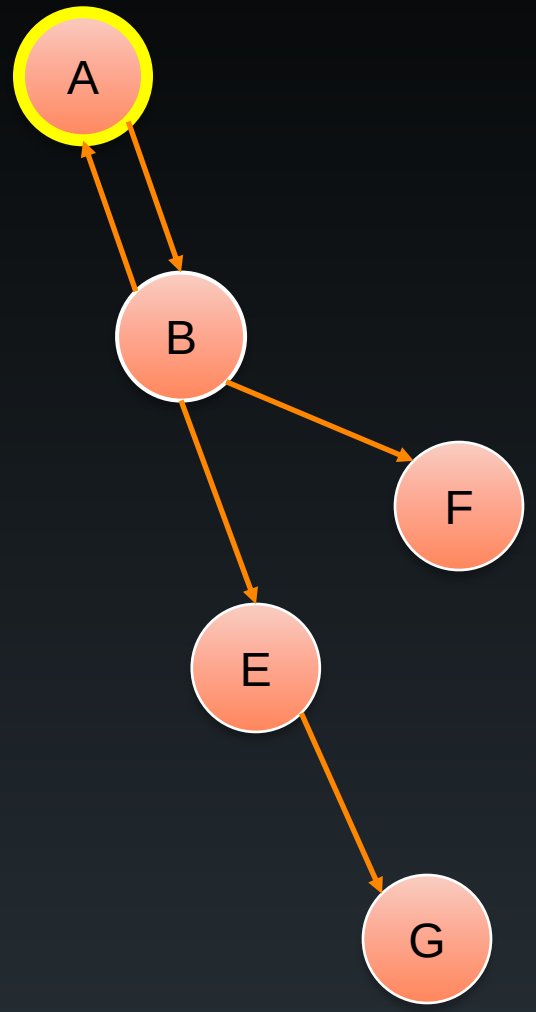


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale (problème)

FRONTIERE

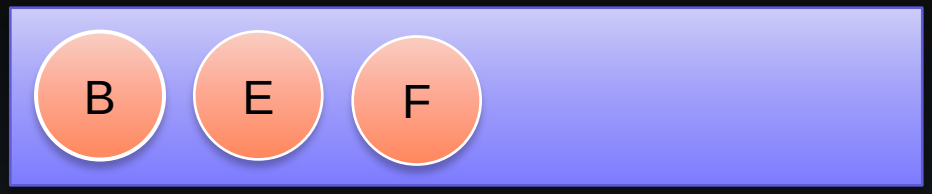


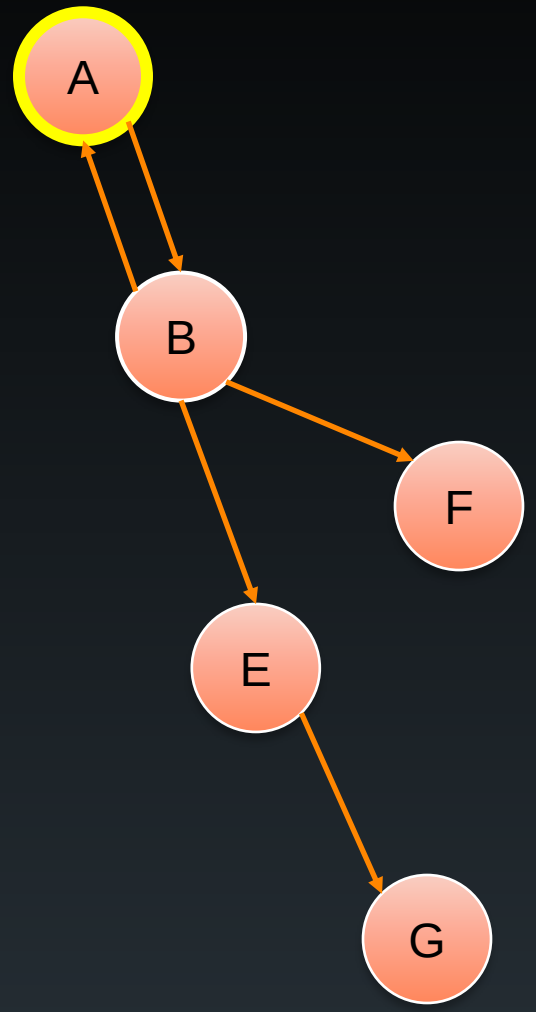


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste OPEN.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale (problème)

FRONTIERE





- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE.

Approche générale révisée

- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Stratégie de recherche

- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer un nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Une stratégie de recherche est définie par l'ordre dans lequel les nœuds sont développés.

Une stratégie s'évalue en fonction de 4 dimensions :

- la complétude : est ce que cette stratégie trouve toujours une solution si elle existe ?
- la complexité en temps : le nombre de nœuds créés
- la complexité en mémoire : le nombre maximum de nœuds en mémoire
- l'optimalité : est ce que la stratégie trouve toujours la solution la moins coûteuse ?

Stratégie de recherche

La complexité en temps et en mémoire se mesure en termes de :

- b : le facteur maximum de branchement de l'arbre de recherche: le nombre maximum de fils des nœuds de l'arbre de recherche
- d : la profondeur de la solution la moins coûteuse
- m : la profondeur maximale de l'arbre de recherche → attention peut être ∞.

Stratégies de recherche non informés

- Les stratégies de recherche non informées utilisent seulement les informations disponibles dans le problème.
- Il existe plusieurs stratégies :
  - Recherche en largeur d'abord
  - Recherche en profondeur d'abord
  - Recherche en profondeur limitée
  - Recherche en profondeur itérative
  - Recherche à coût uniforme

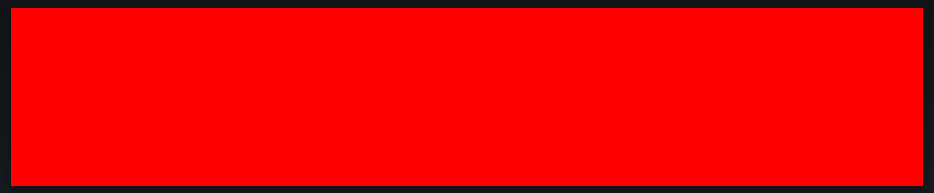
Recherche en largeur d'abord

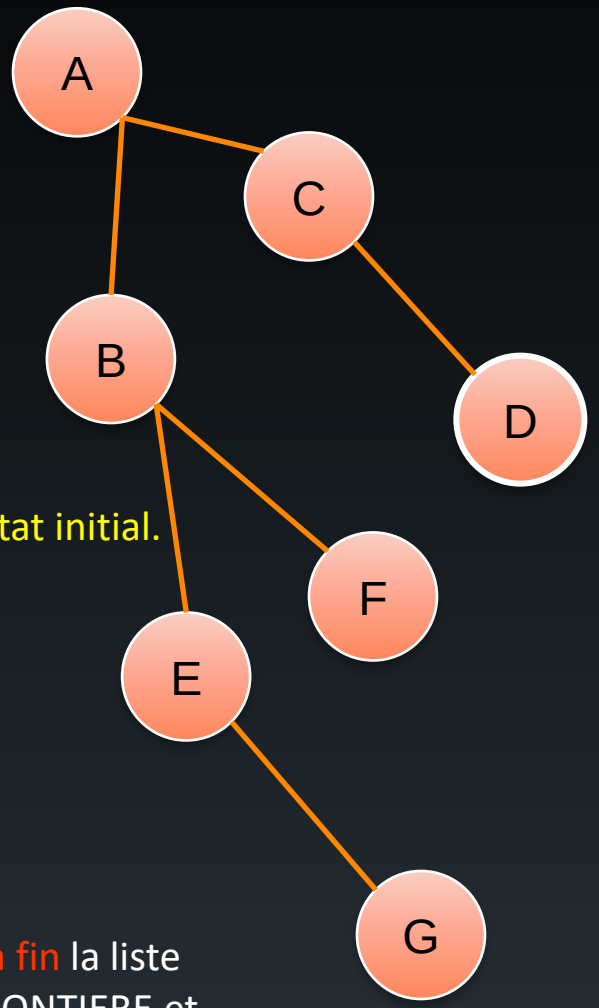
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE



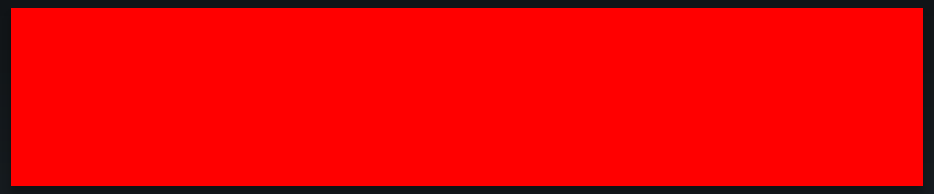


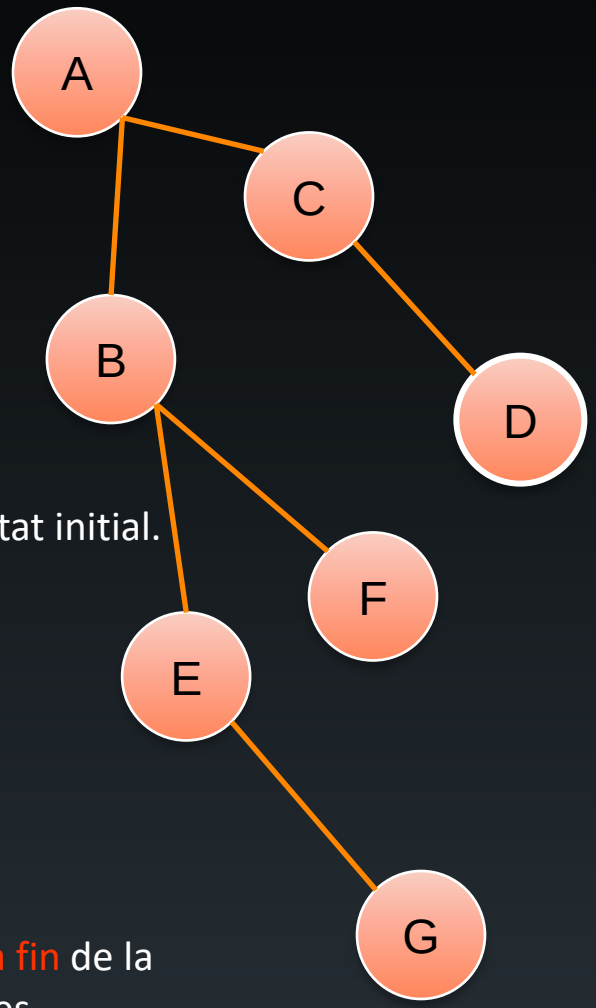
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE



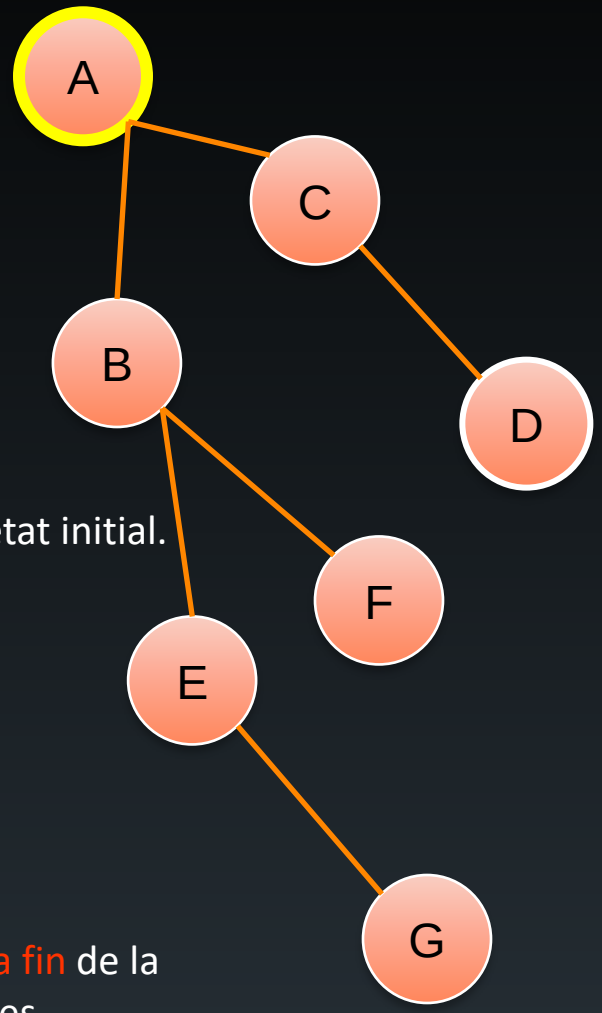


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE



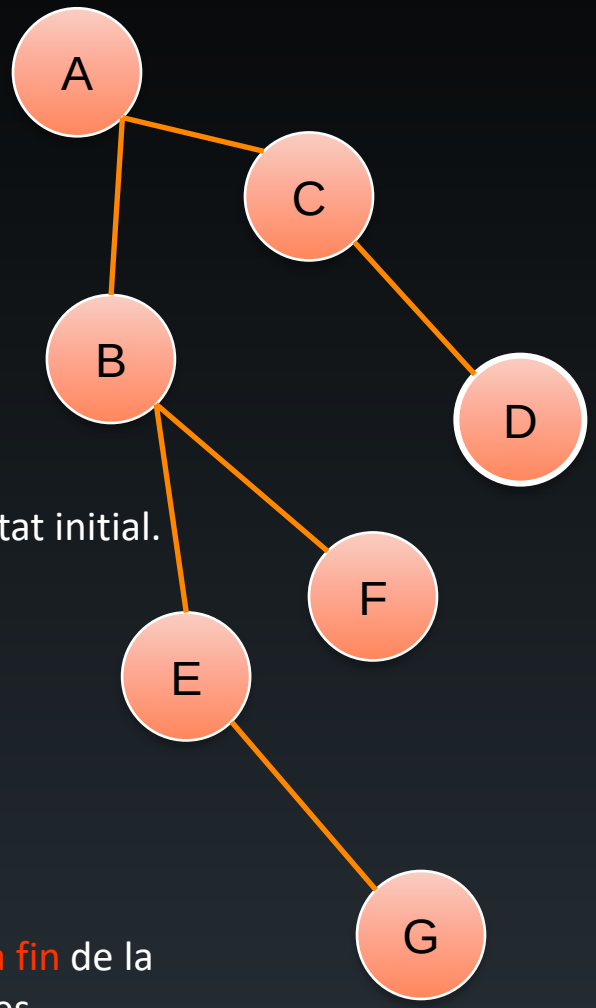
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE

A

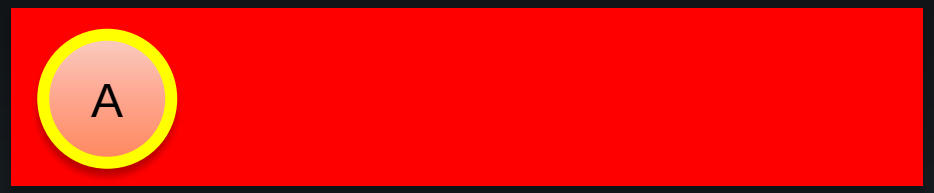


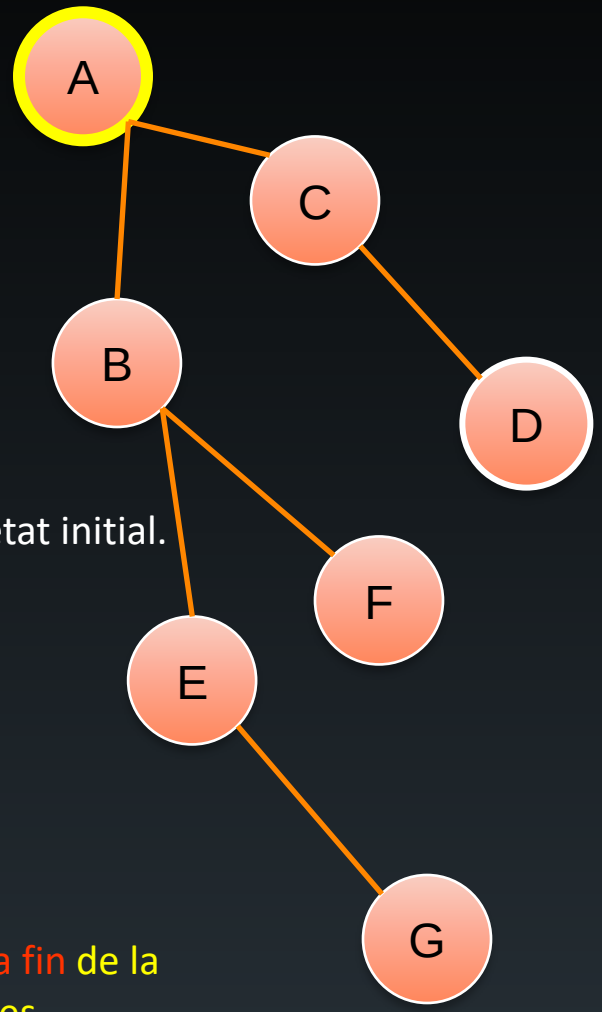
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE



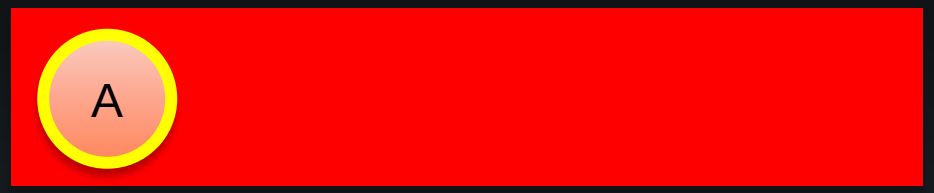


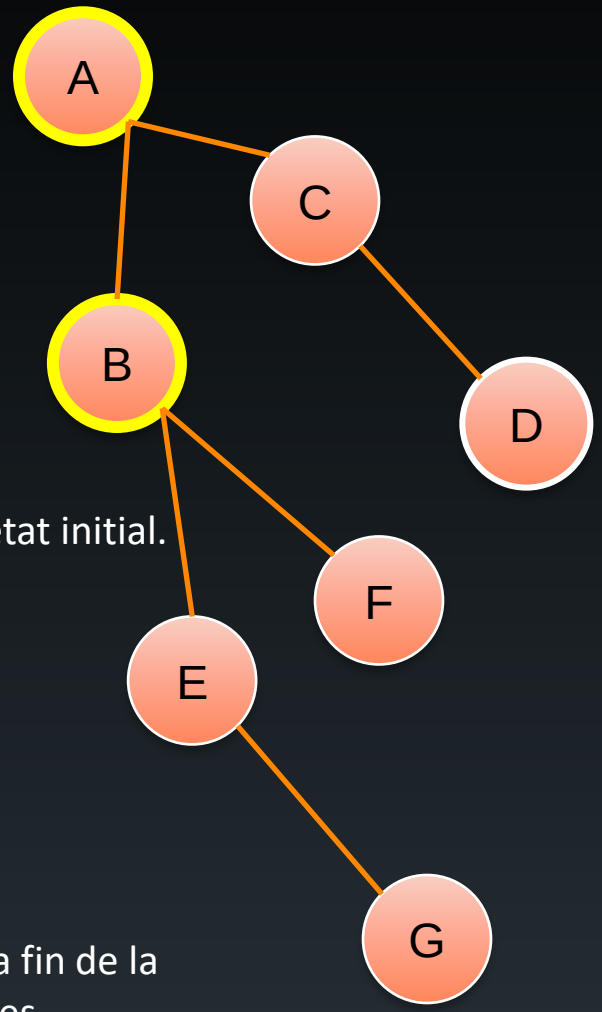
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE



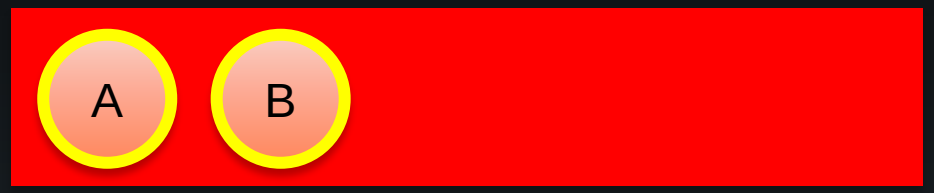


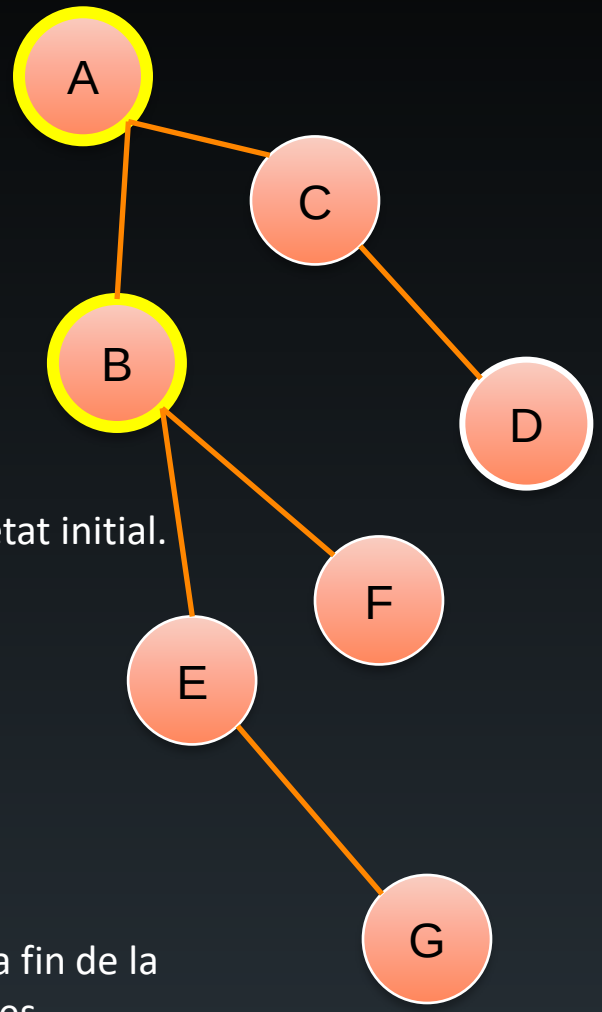
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE

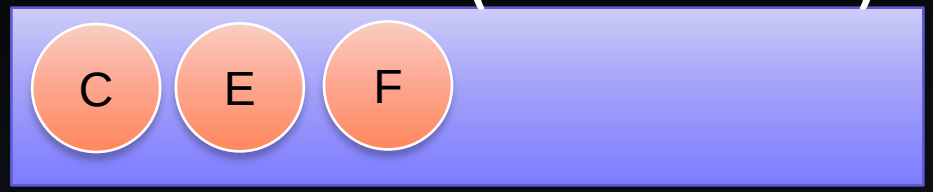




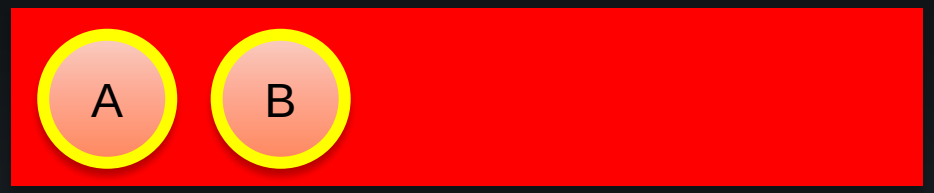
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

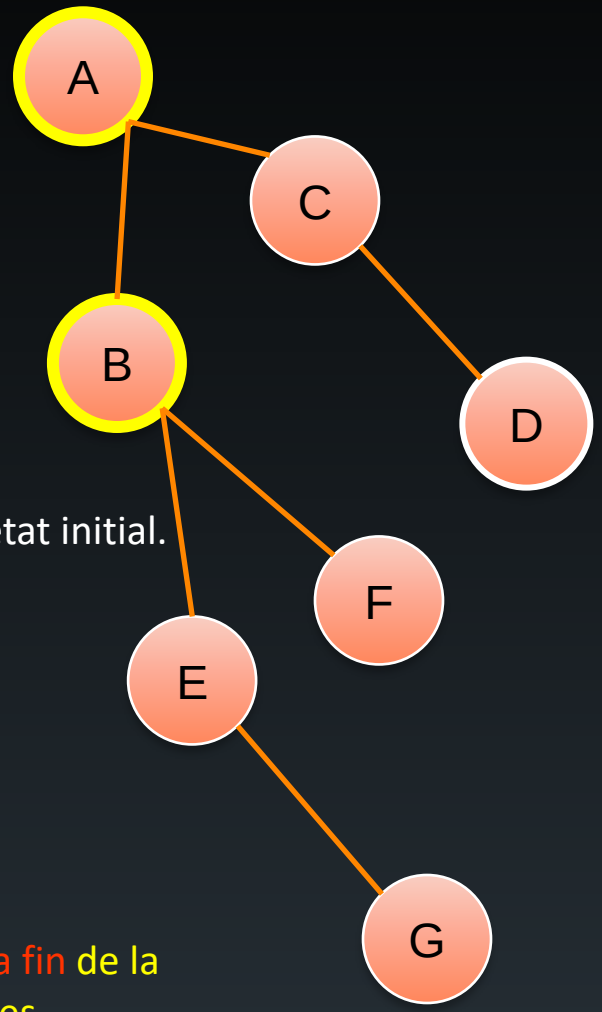
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE



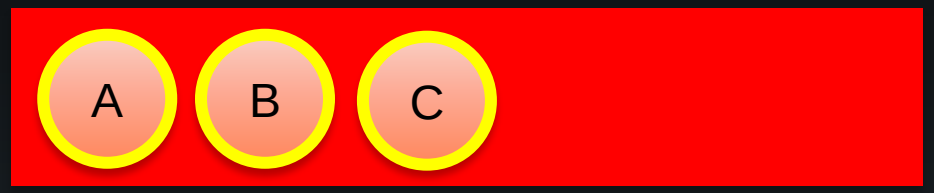


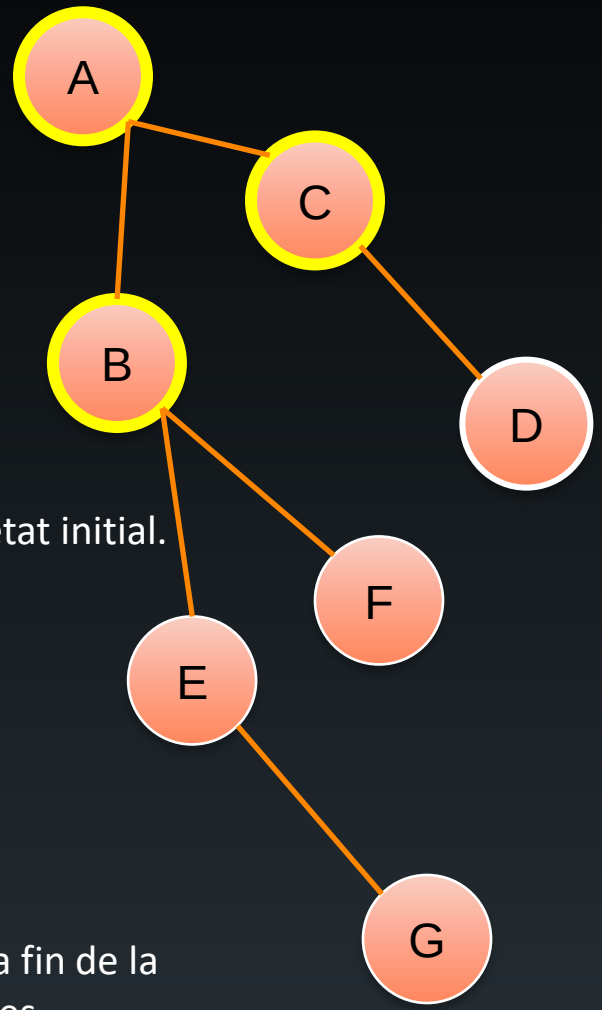
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE

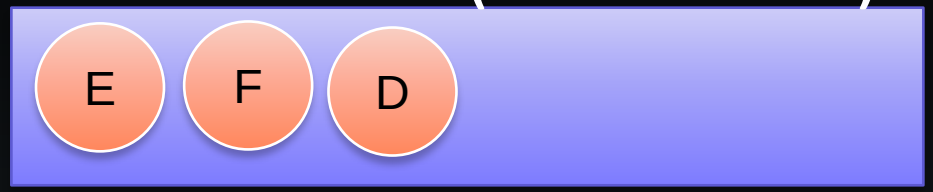




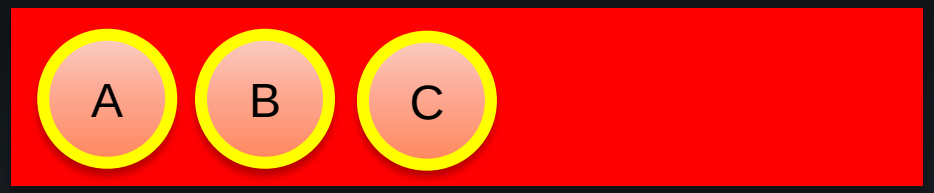
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

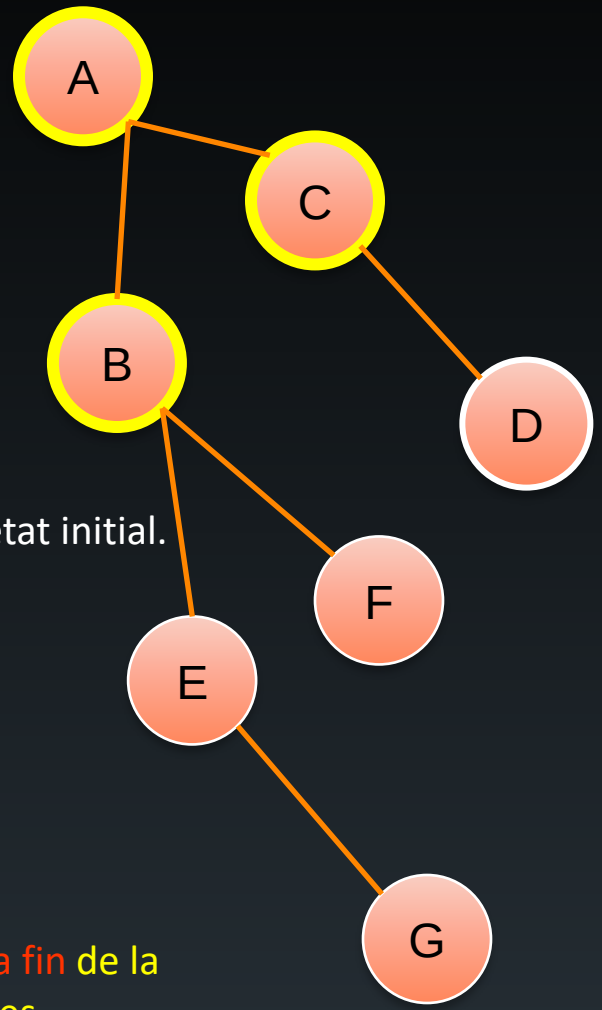
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE

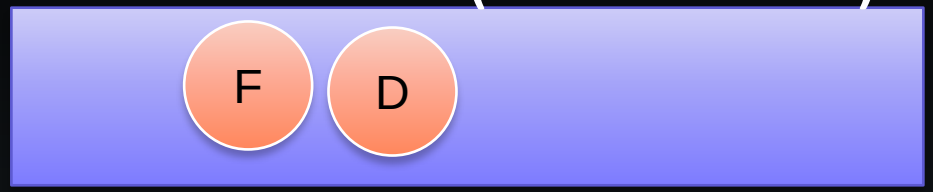




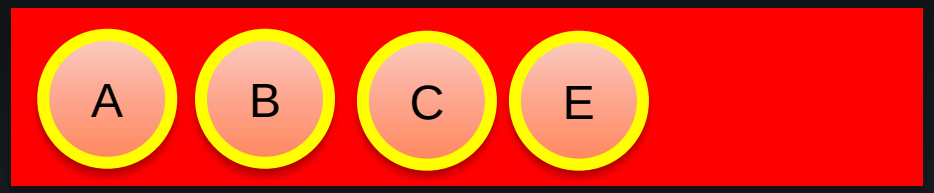
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

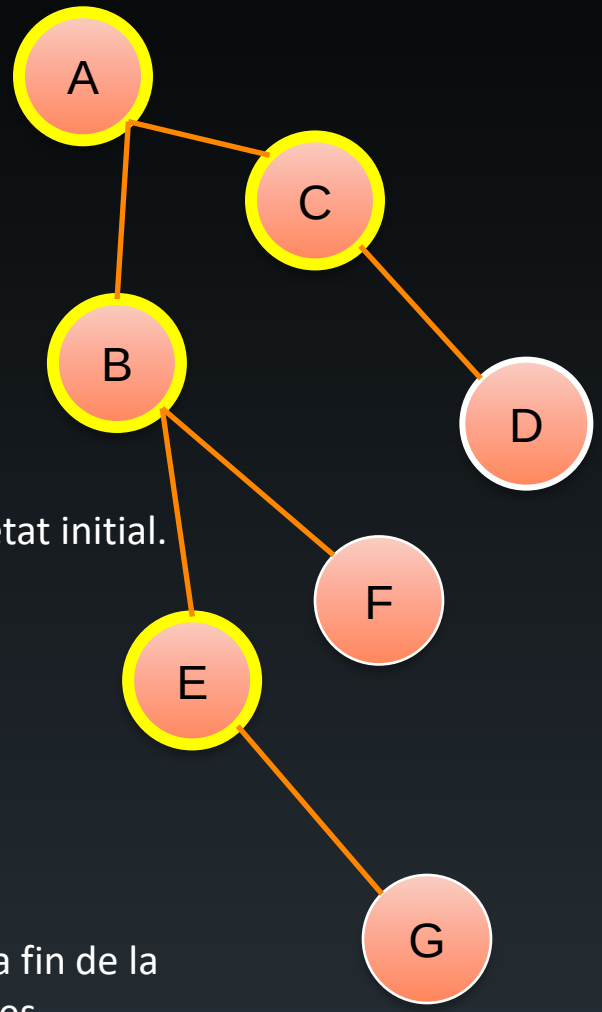
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE



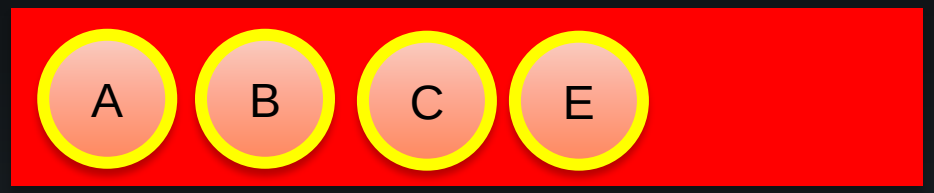


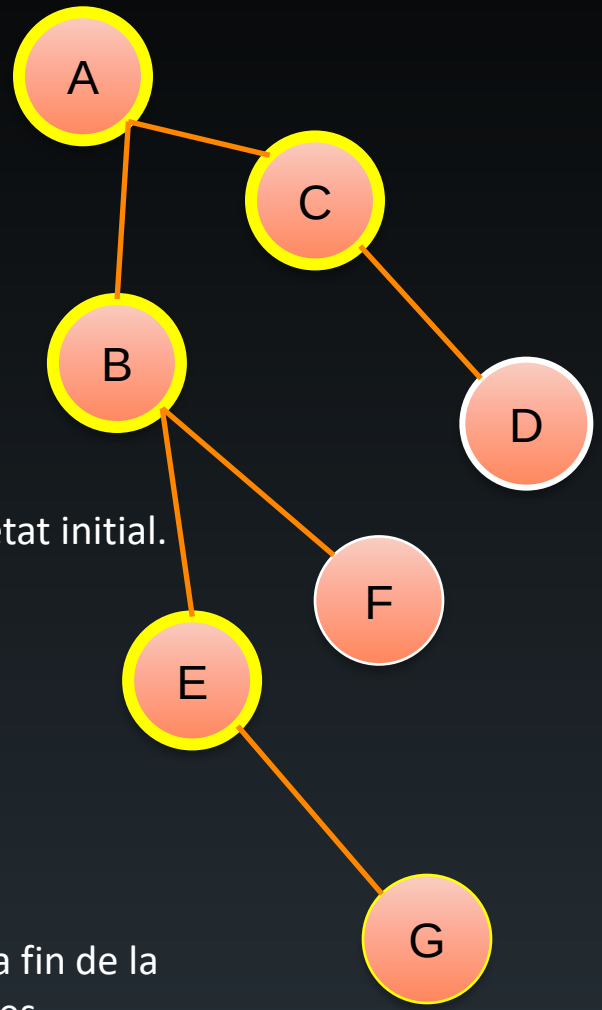
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

EXPLOREE

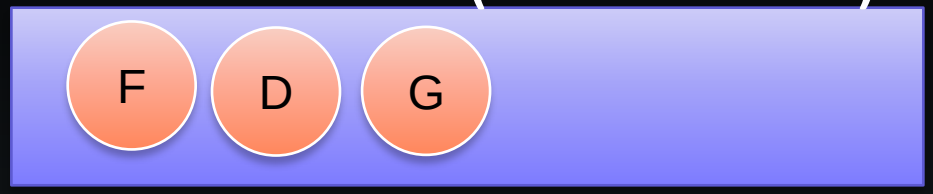




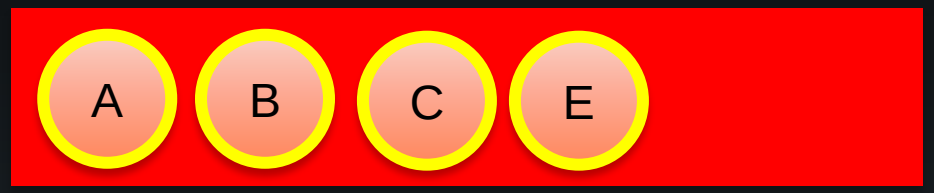
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

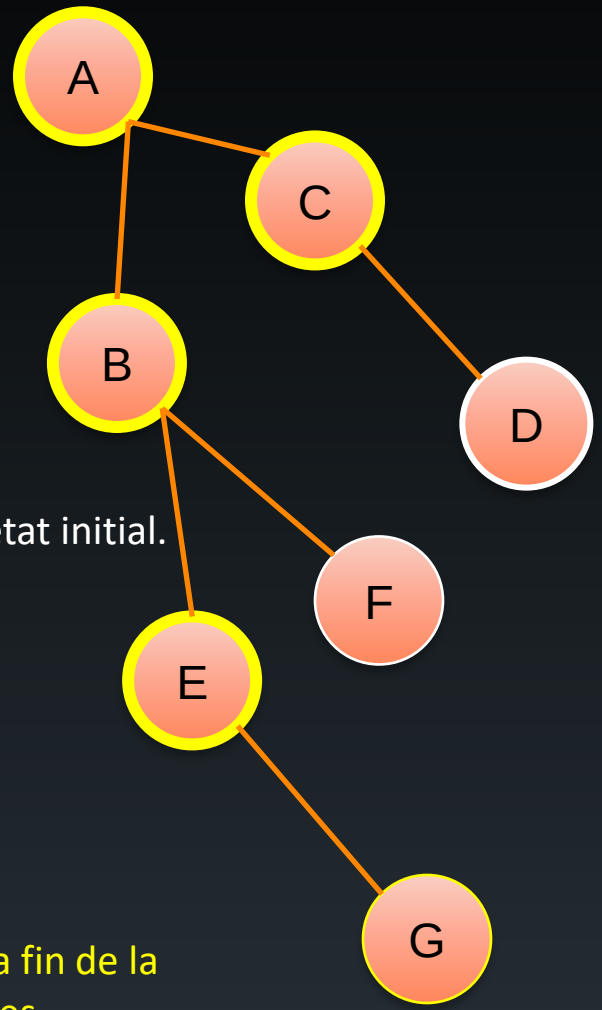
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE

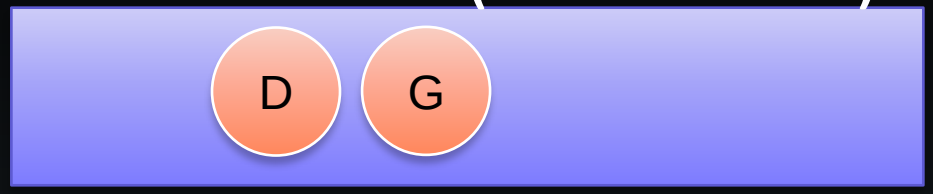




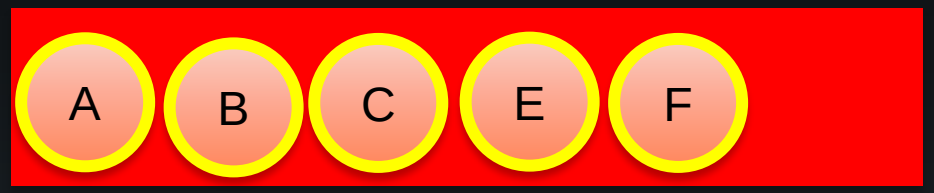
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

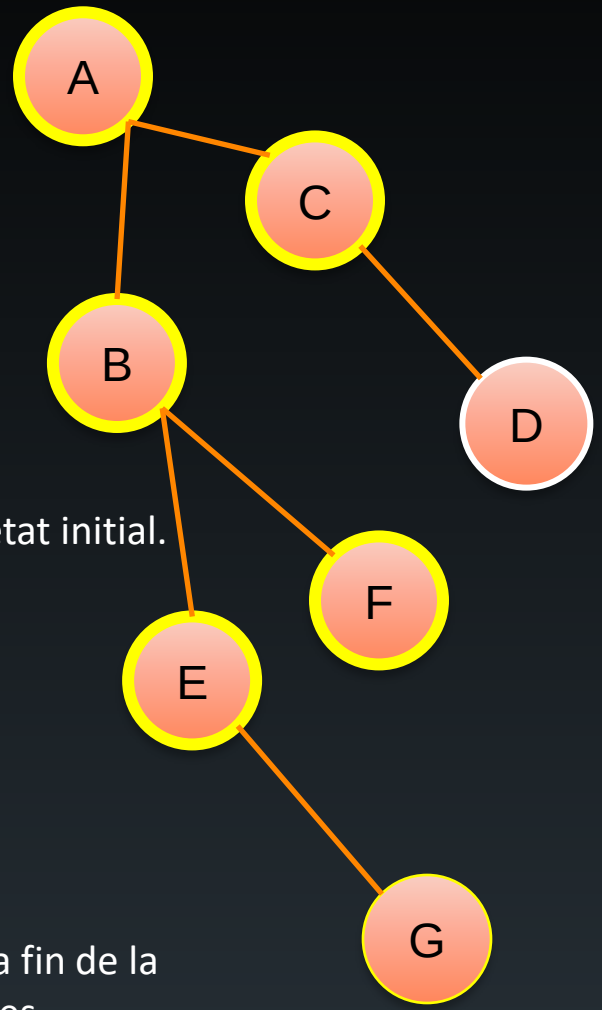
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE

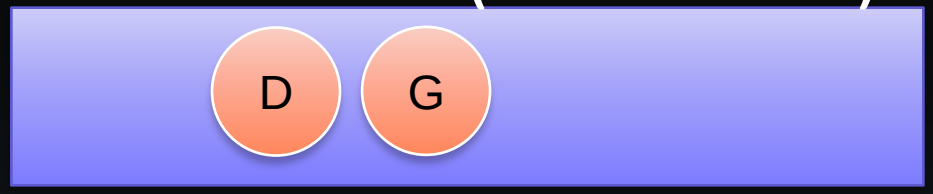




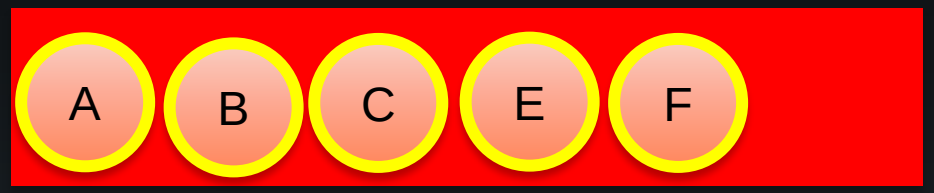
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

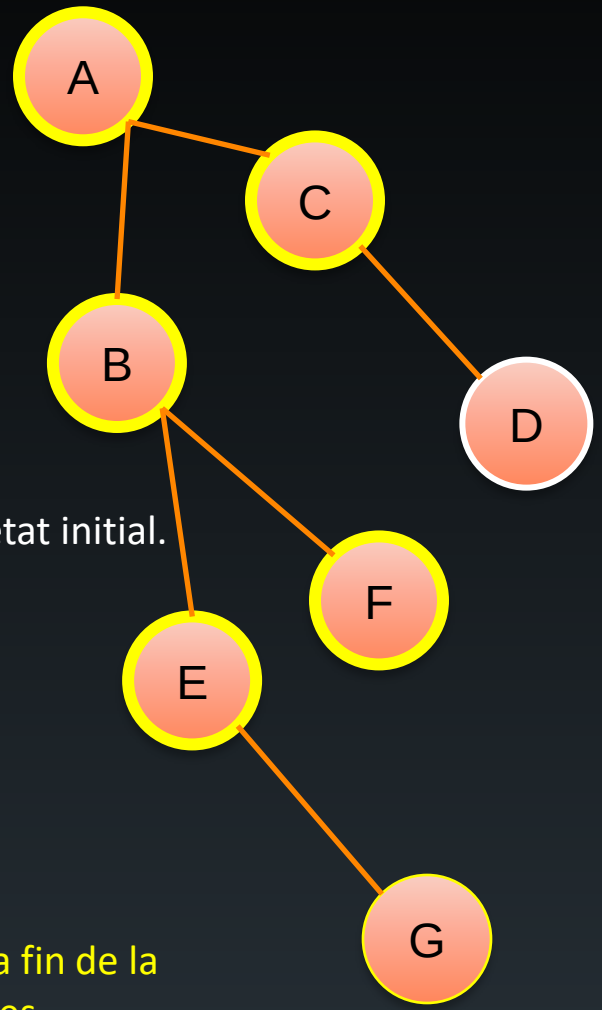
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE

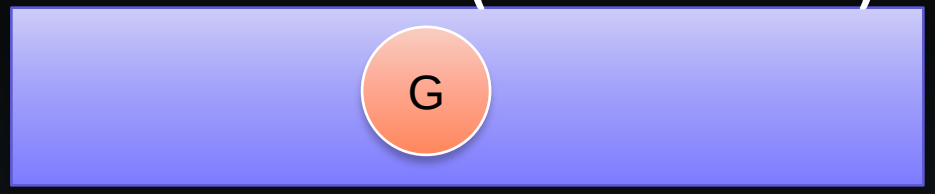




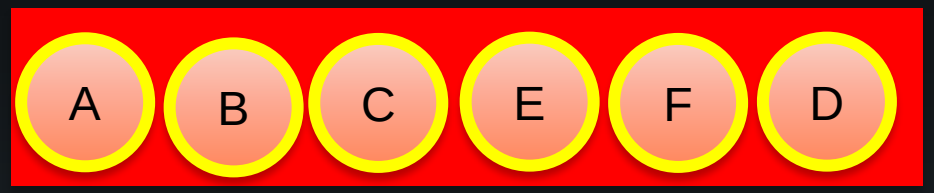
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

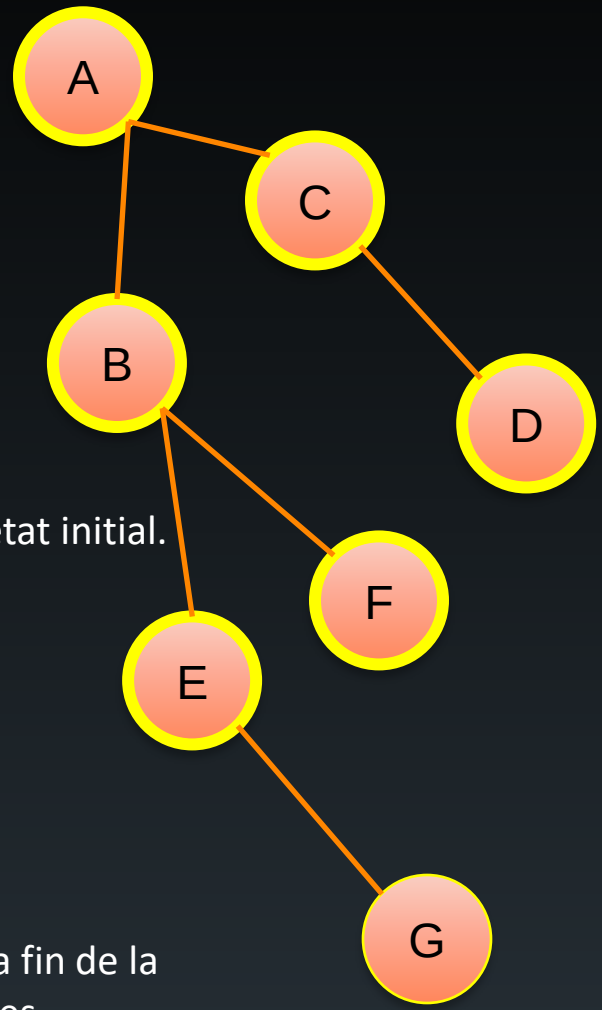
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE

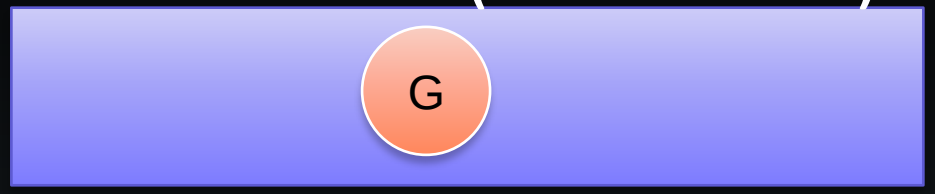




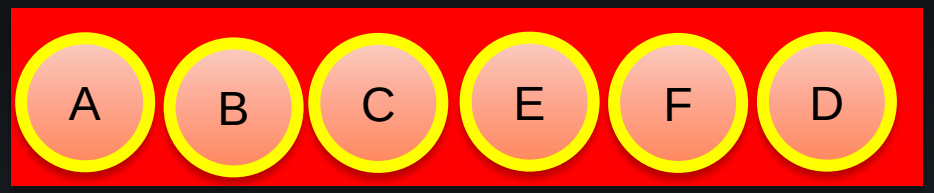
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

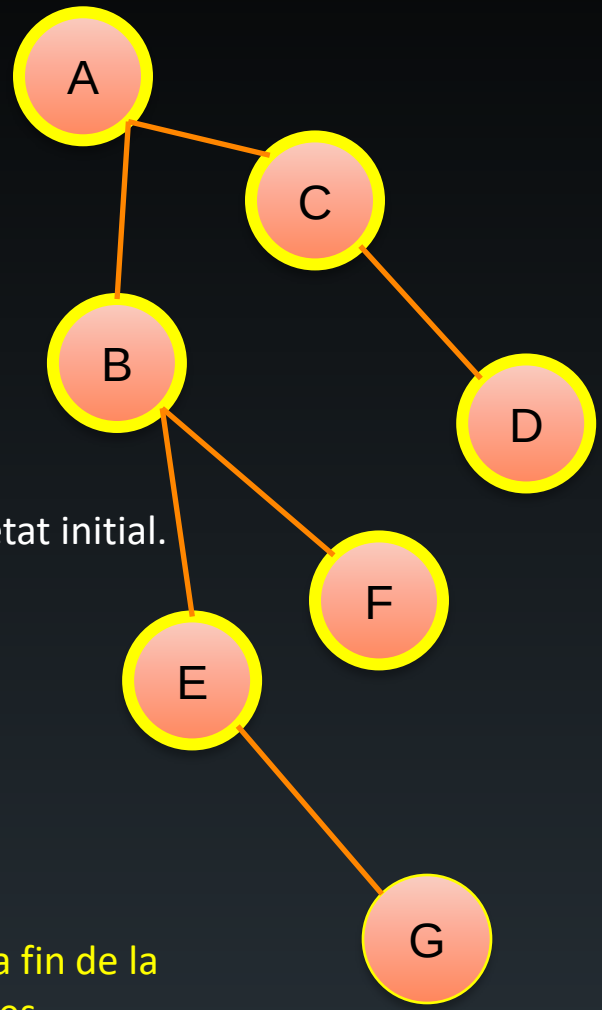
Recherche en largeur d'abord

FRONTIERE (file = FIFO)



EXPLOREE



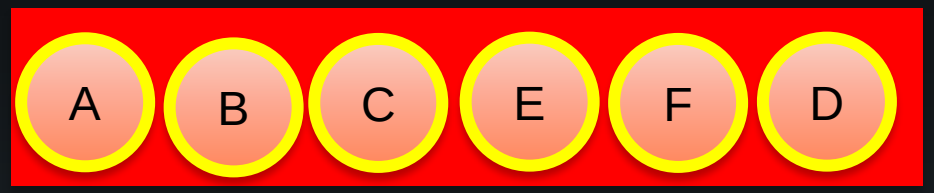


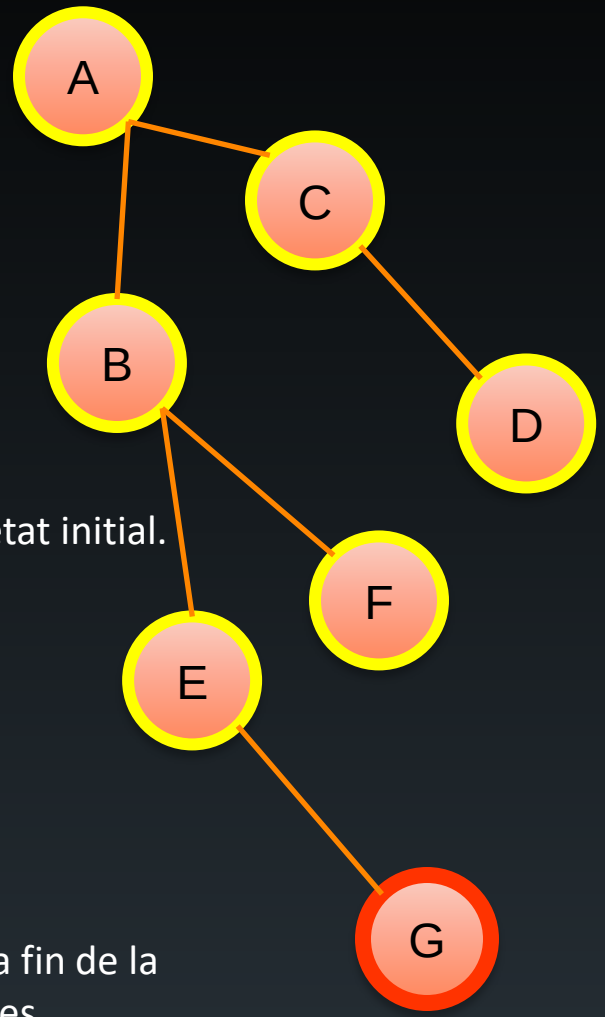
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d'abord

FRONTIERE (file = FIFO)

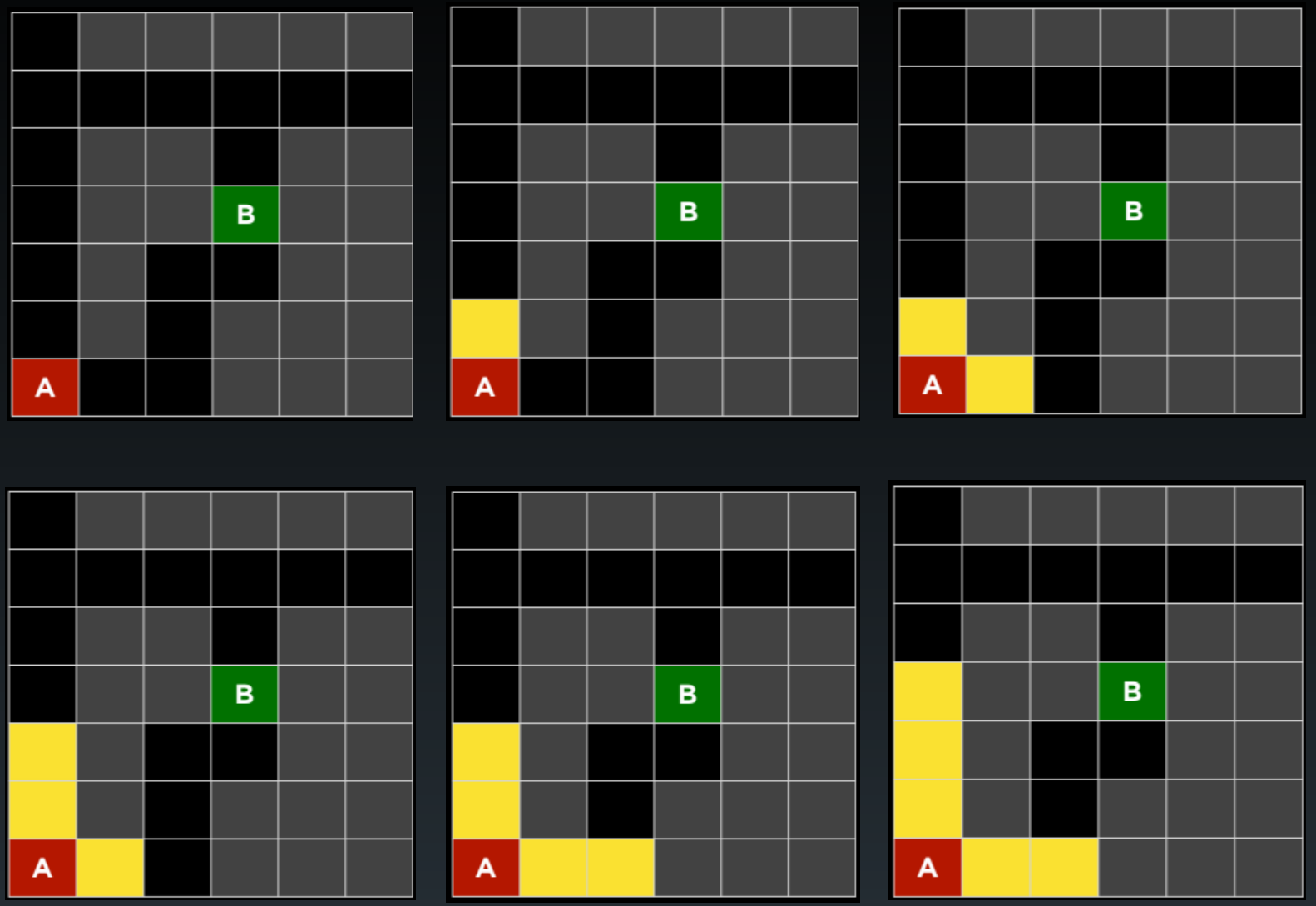
EXPLOREE



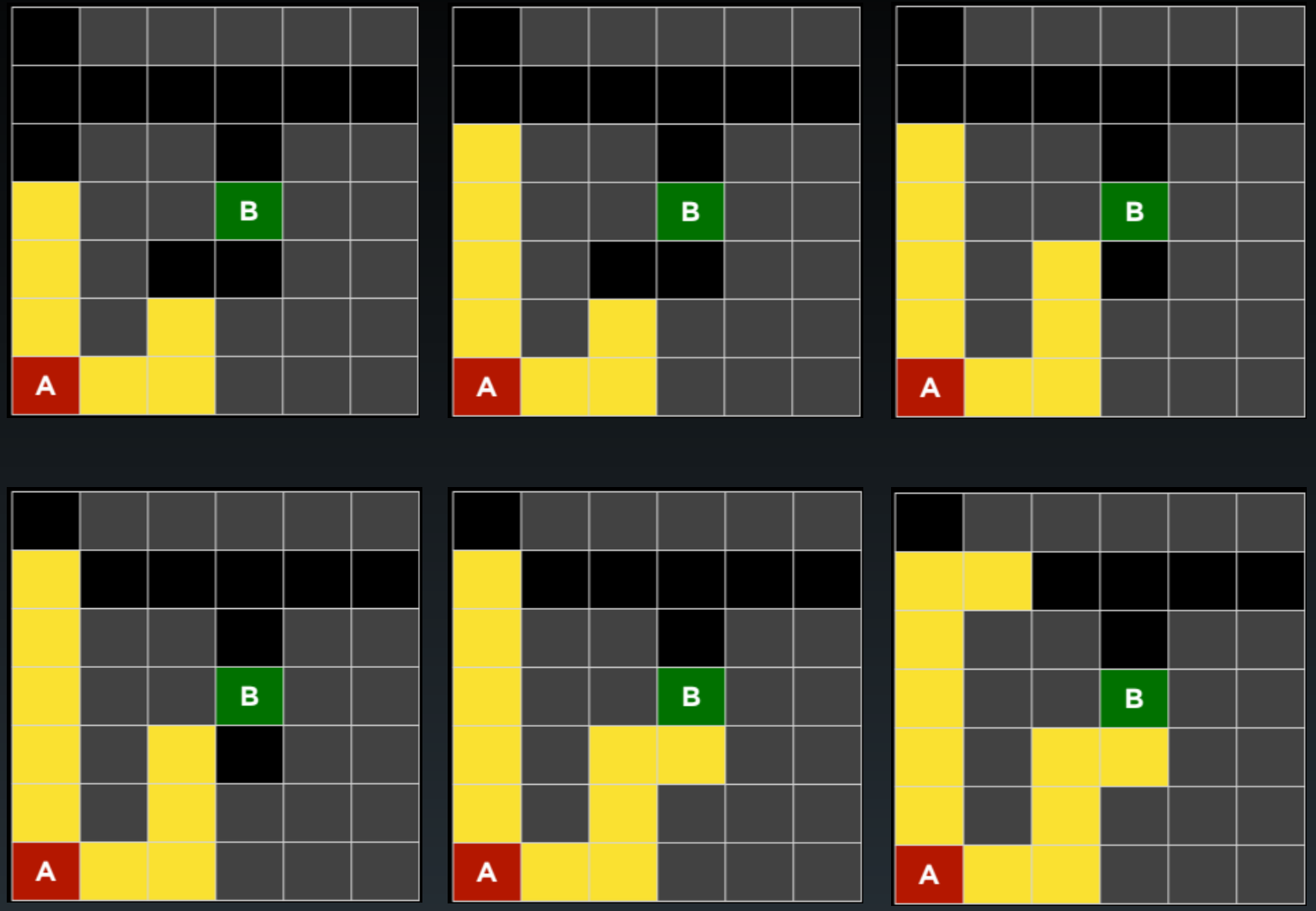


- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants à la fin de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en largeur d’abord



Recherche en largeur d’abord



Recherche en largeur d'abord

- Complete, si b est fini
- Complexité en temps :
  $1 + b + b^2 + b^3 + b^4 + .. + b^d = O(b^d)$
- Complexité en espace :
  $O(b^d)$
- Optimale si coût = 1 pour chaque pas, mais non optimale dans le cas général.

⇒ L'espace est le plus gros problème

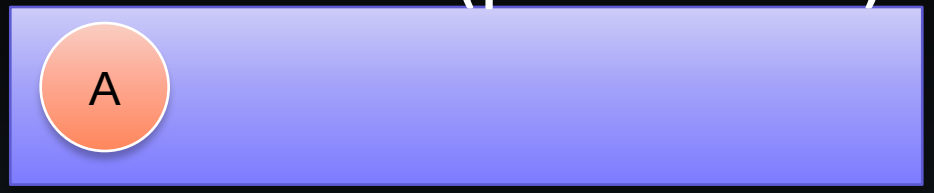
Recherche en largeur d'abord

- b=10
- 1 million de nœuds peuvent être générés dans une seconde
- Un nœud nécessite 1000 octets

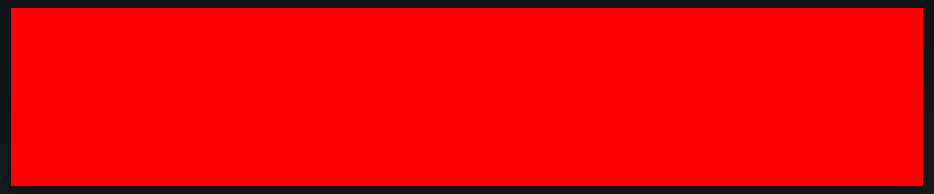
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

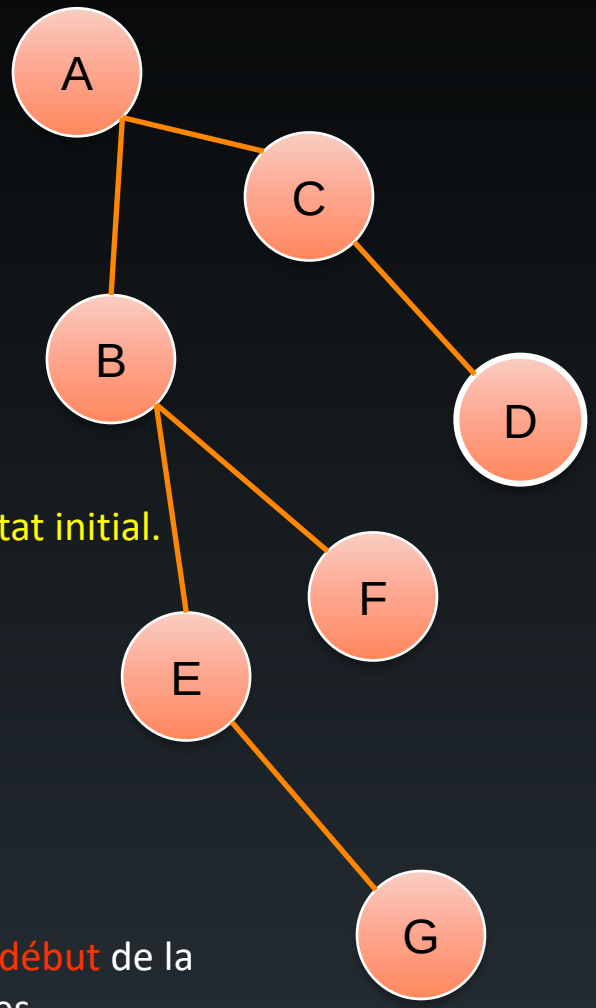
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE





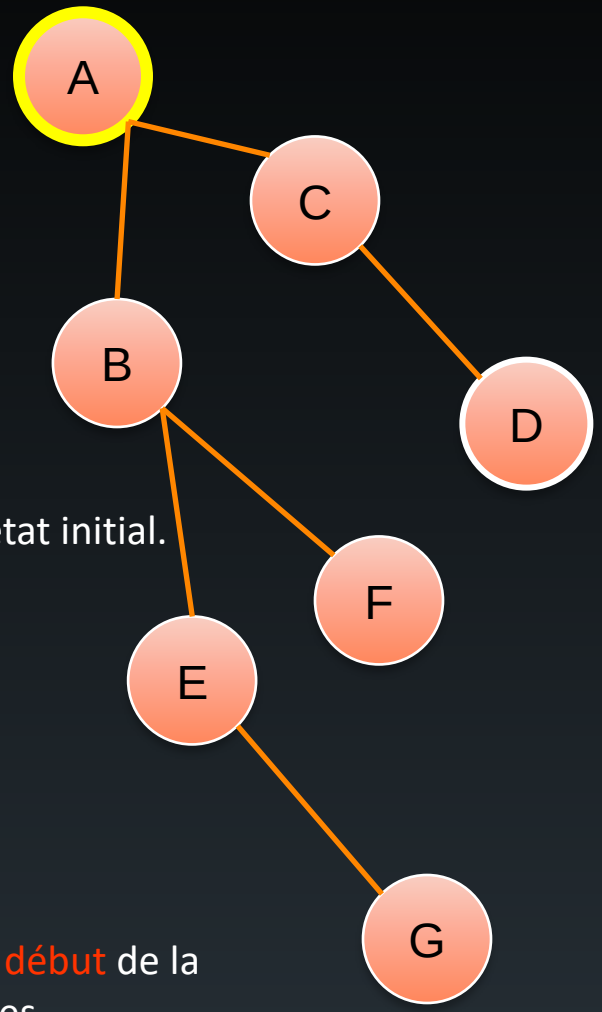
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)

EXPLOREE

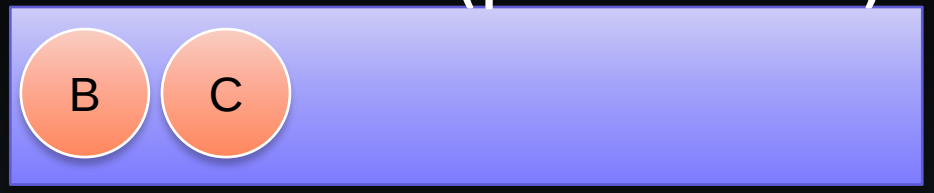
A



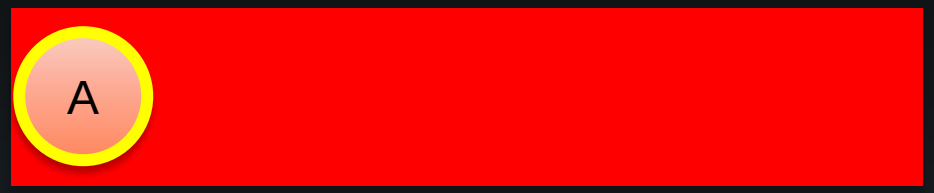
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

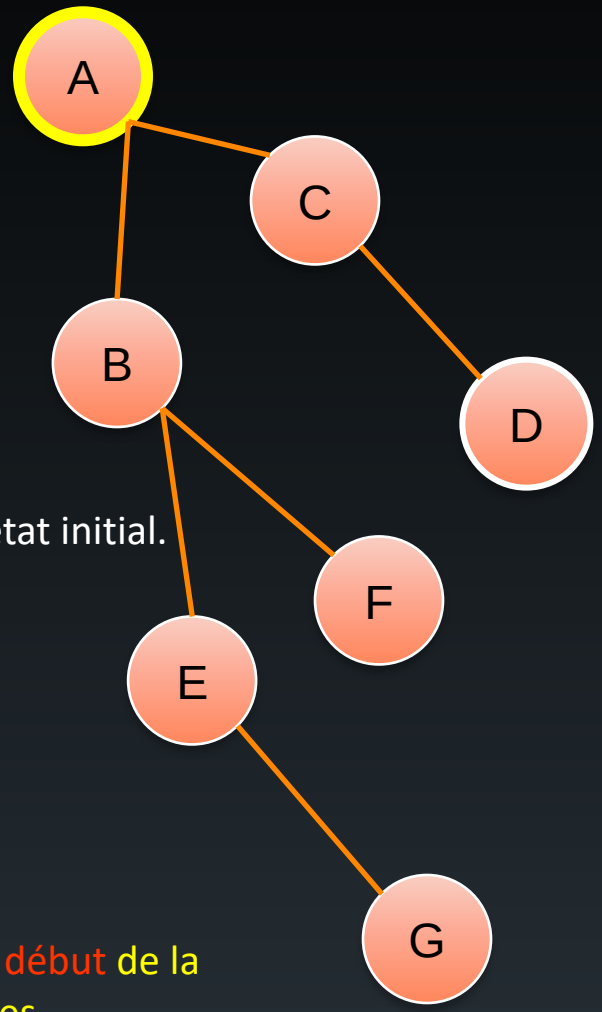
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE

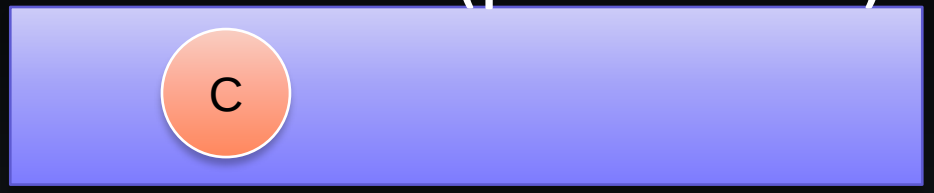




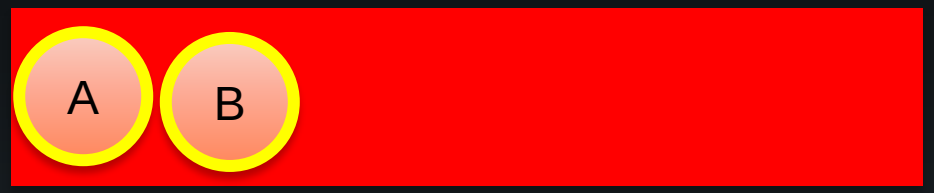
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

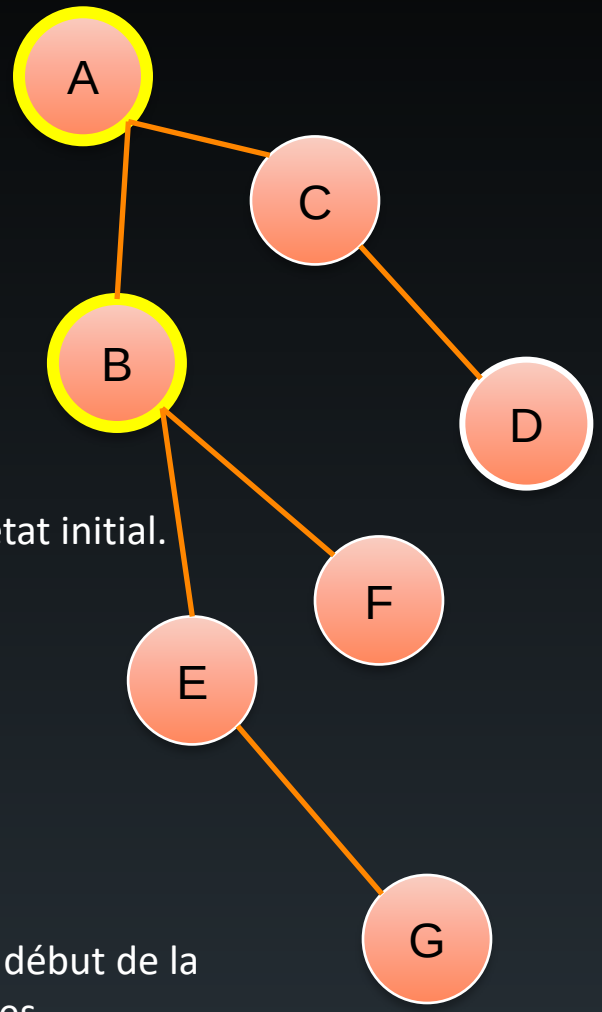
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE

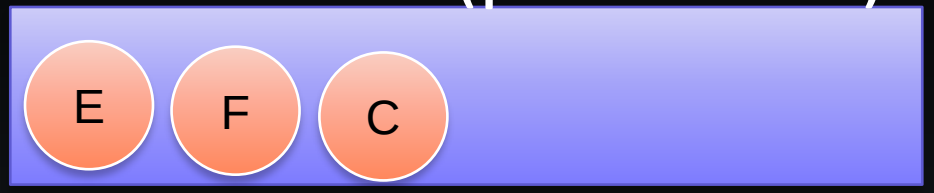




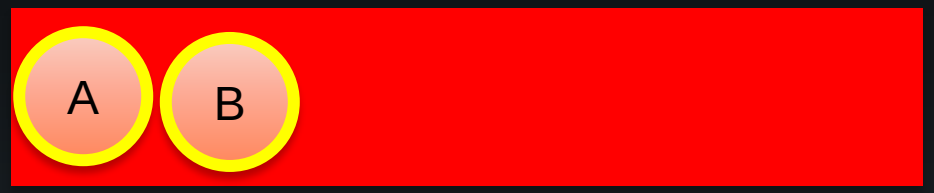
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

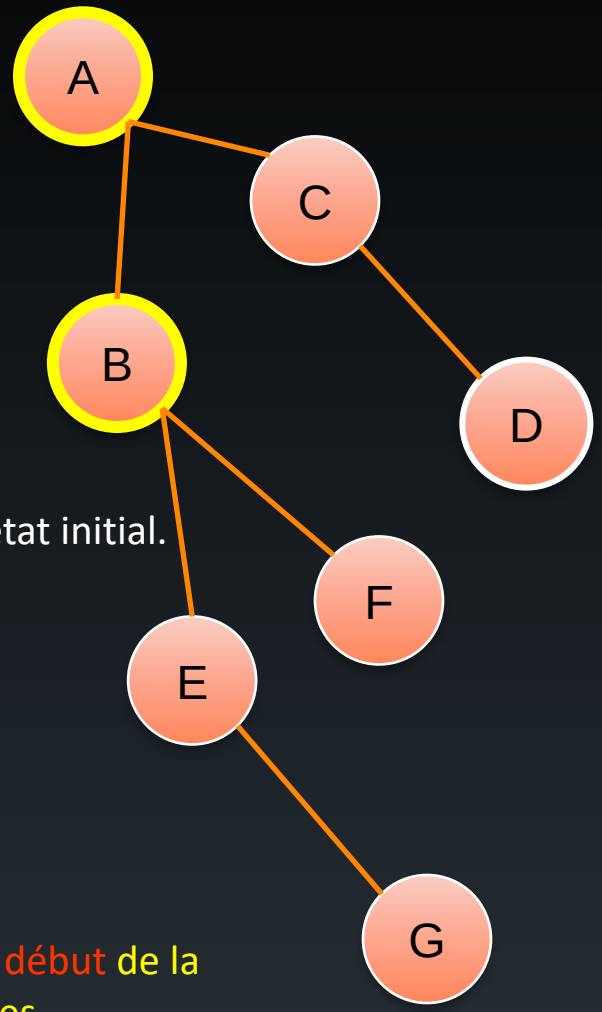
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE

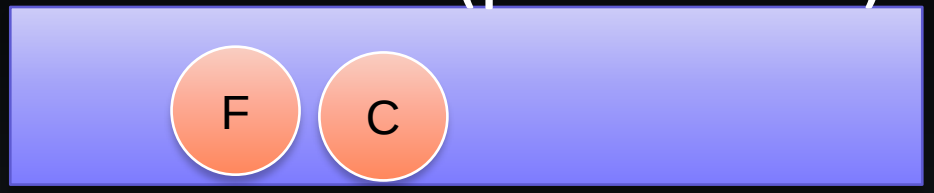




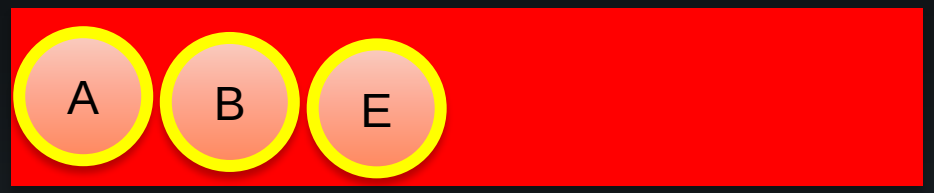
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

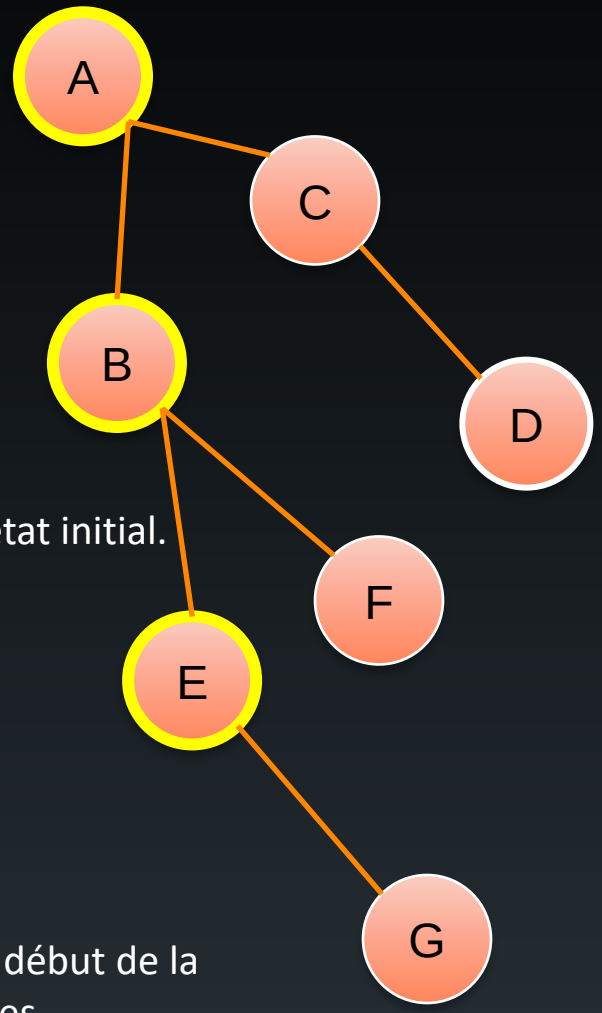
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE

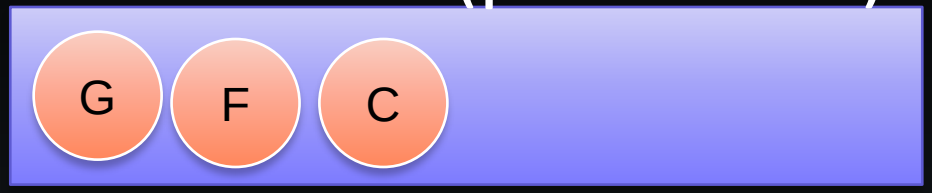




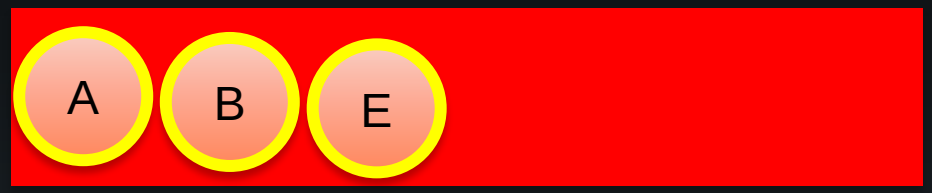
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

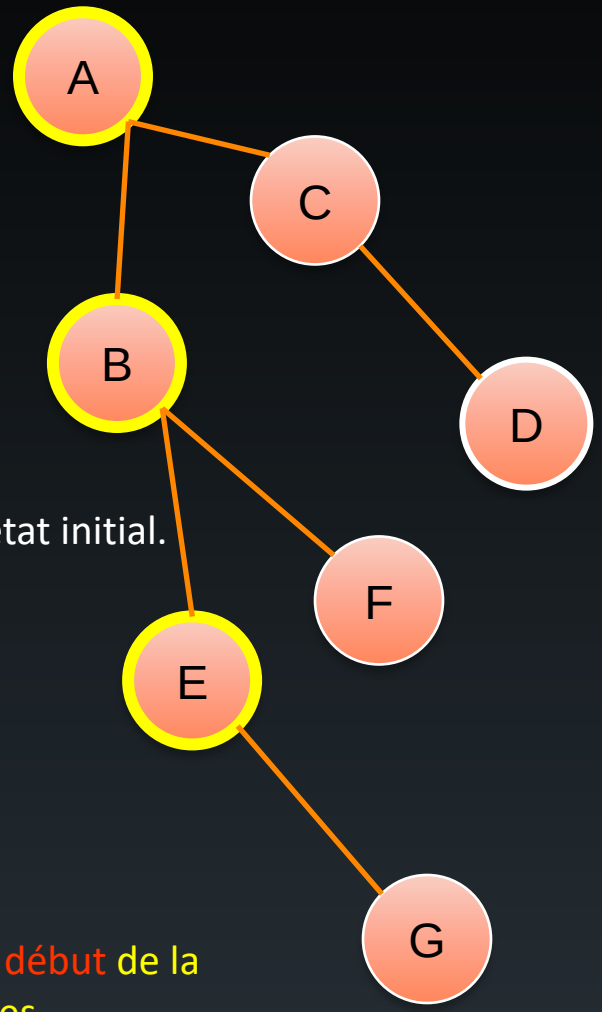
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



EXPLOREE

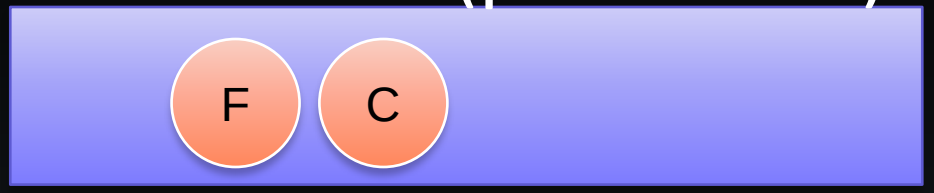




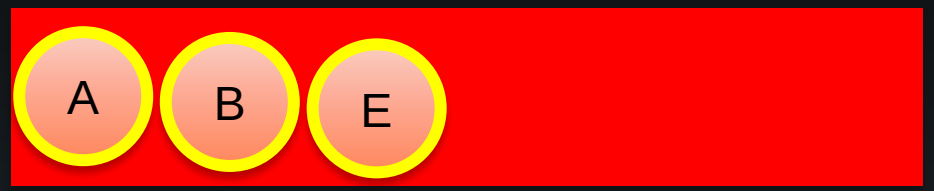
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

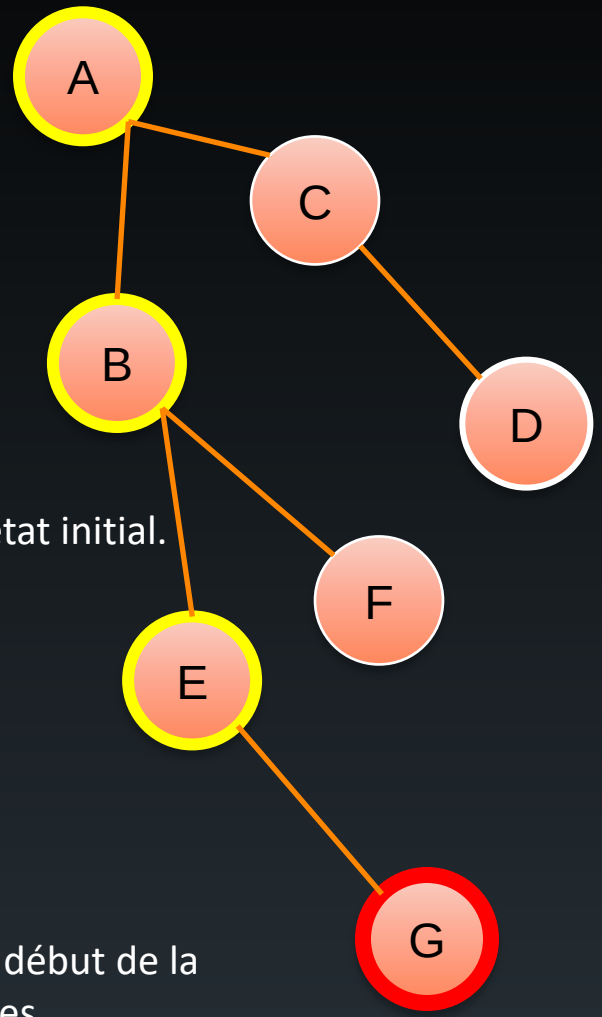
Recherche en profondeur d'abord

FRONTIERE (pile = LIFO)



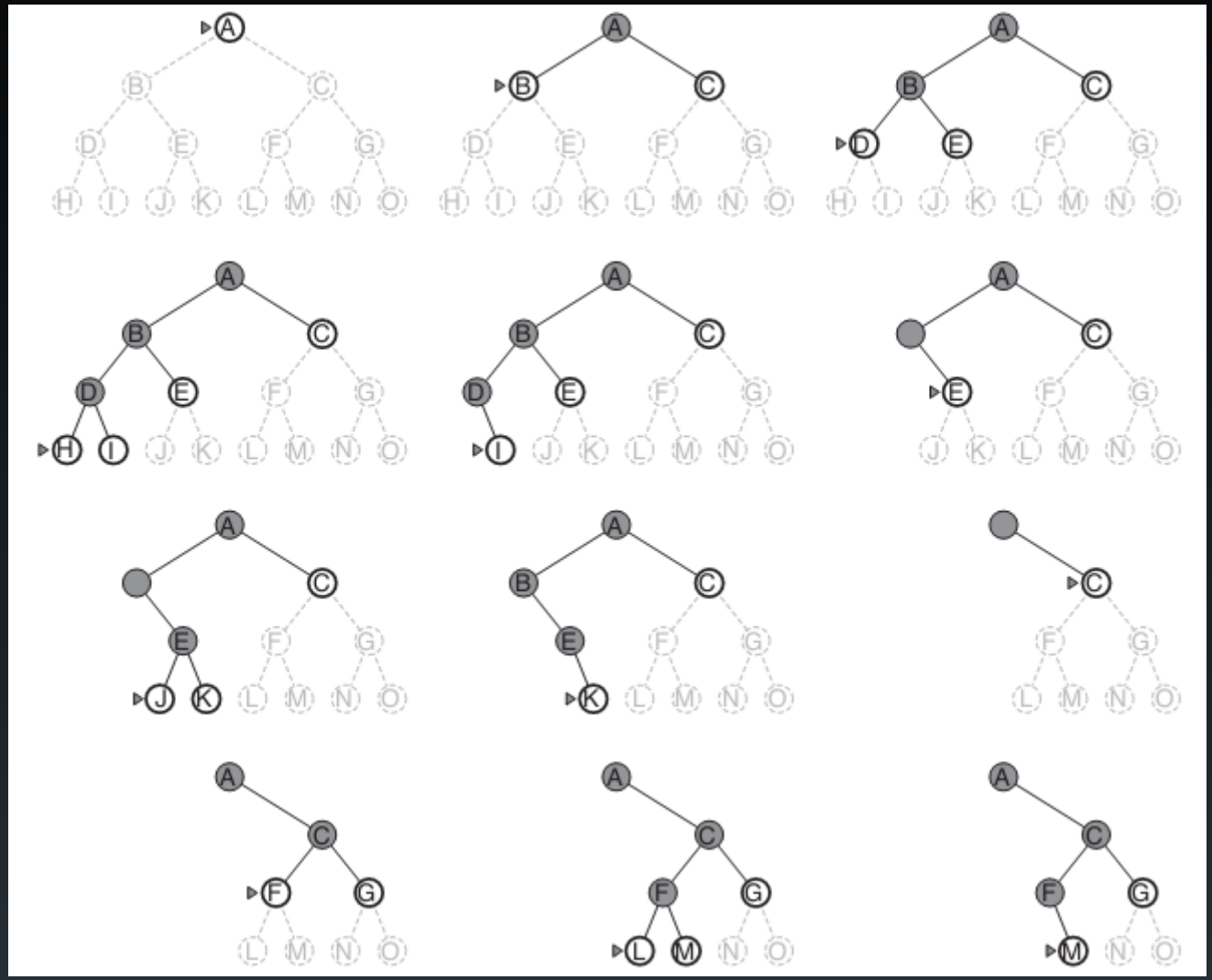
EXPLOREE



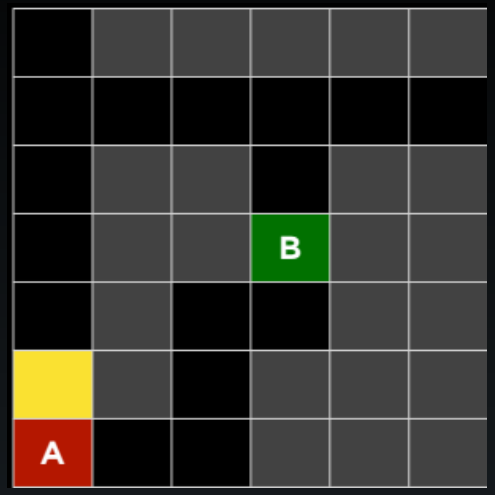


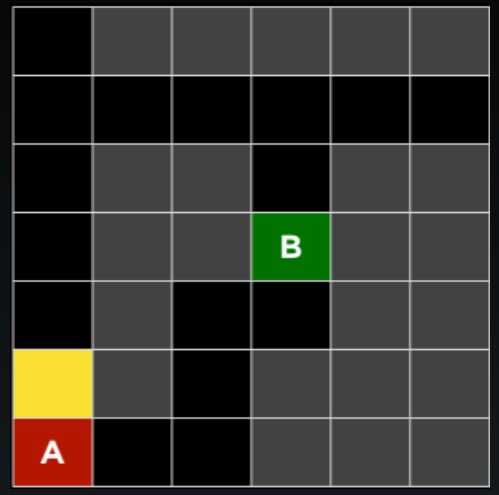
- Commencer par une liste appelée FRONTIERE contenant l'état initial.
- Et une liste EXPLOREE vide
- Répéter :
  - Si FRONTIERE est vide alors aucune solution.
  - Tirer le premier nœud de la liste FRONTIERE.
  - Si le nœud contient l'état but, retourner la solution.
  - Ajouter le nœud à la liste EXPLOREE
  - Développer le nœud et ajouter les nœuds résultants au début de la liste FRONTIERE si ces nœuds n'existent pas dans les listes FRONTIERE et EXPLOREE.

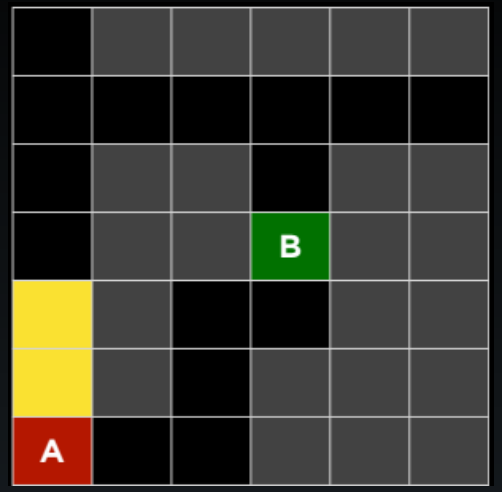
Recherche en profondeur d'abord

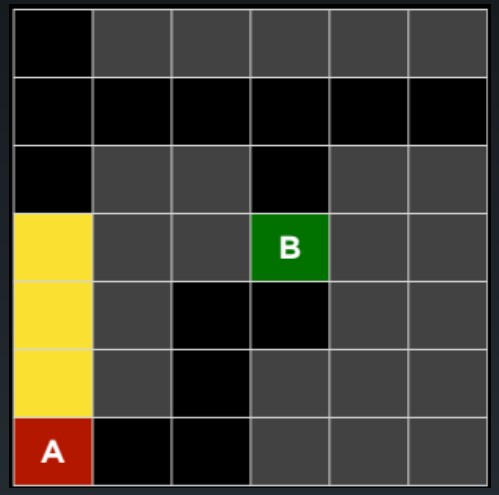


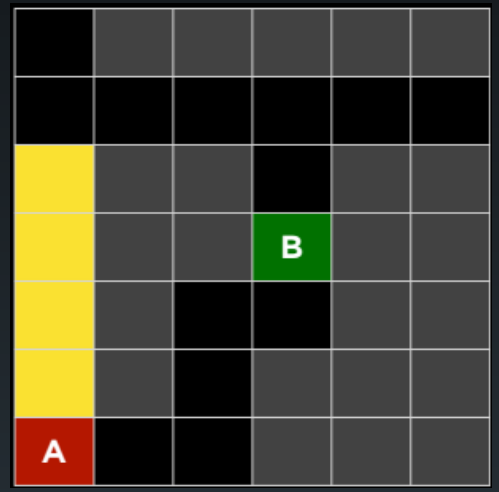
Recherche en profondeur d’abord

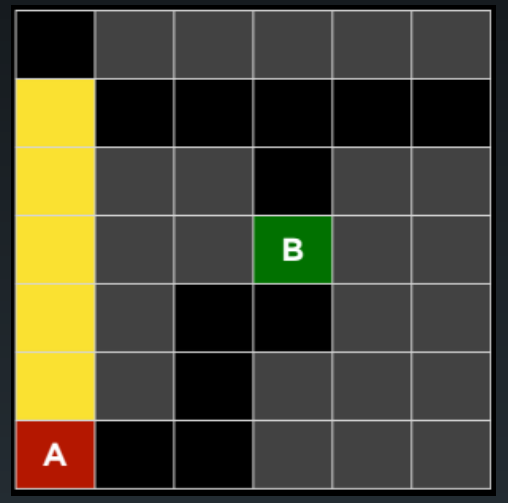




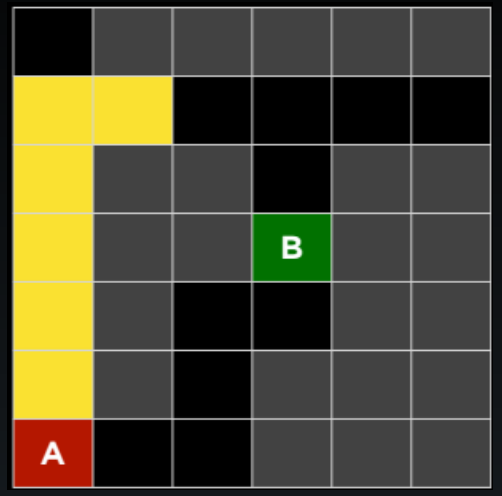


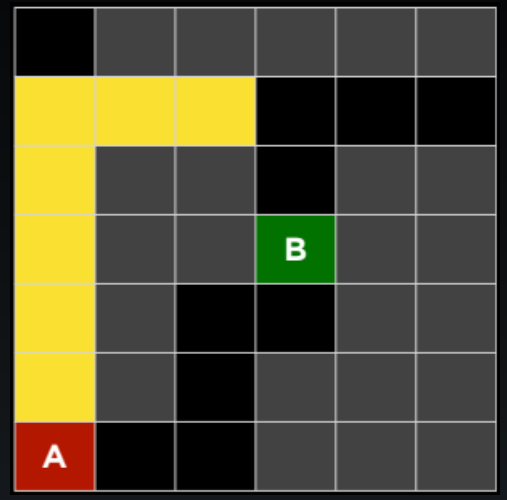


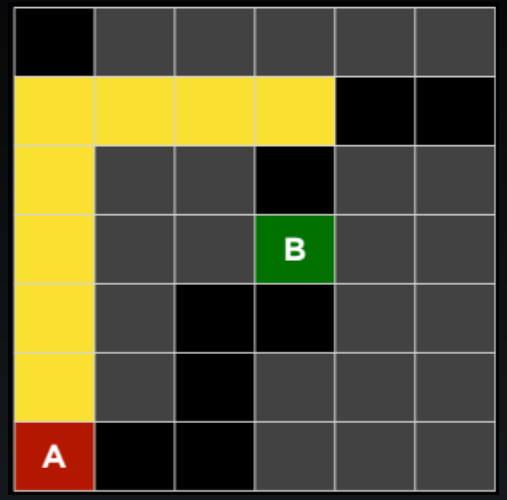


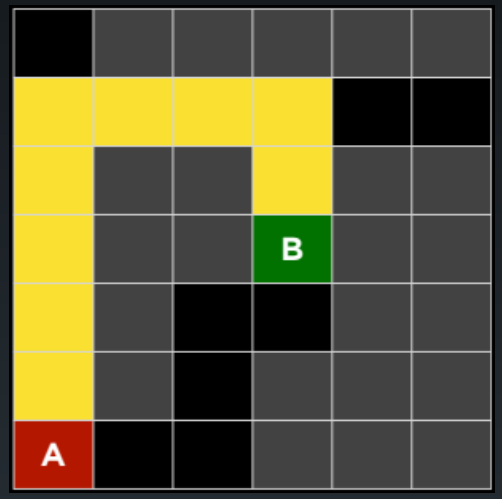


Recherche en profondeur d’abord





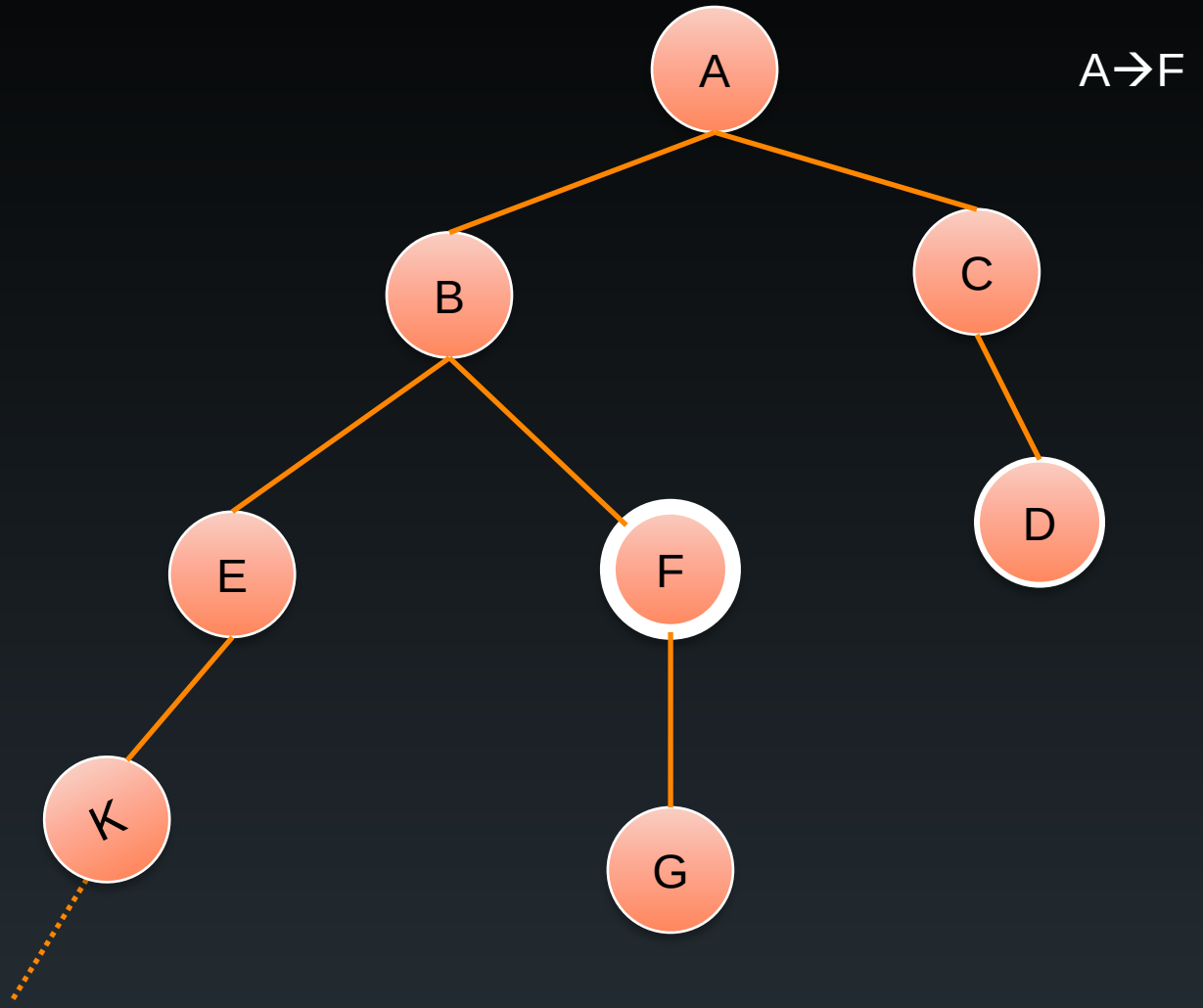




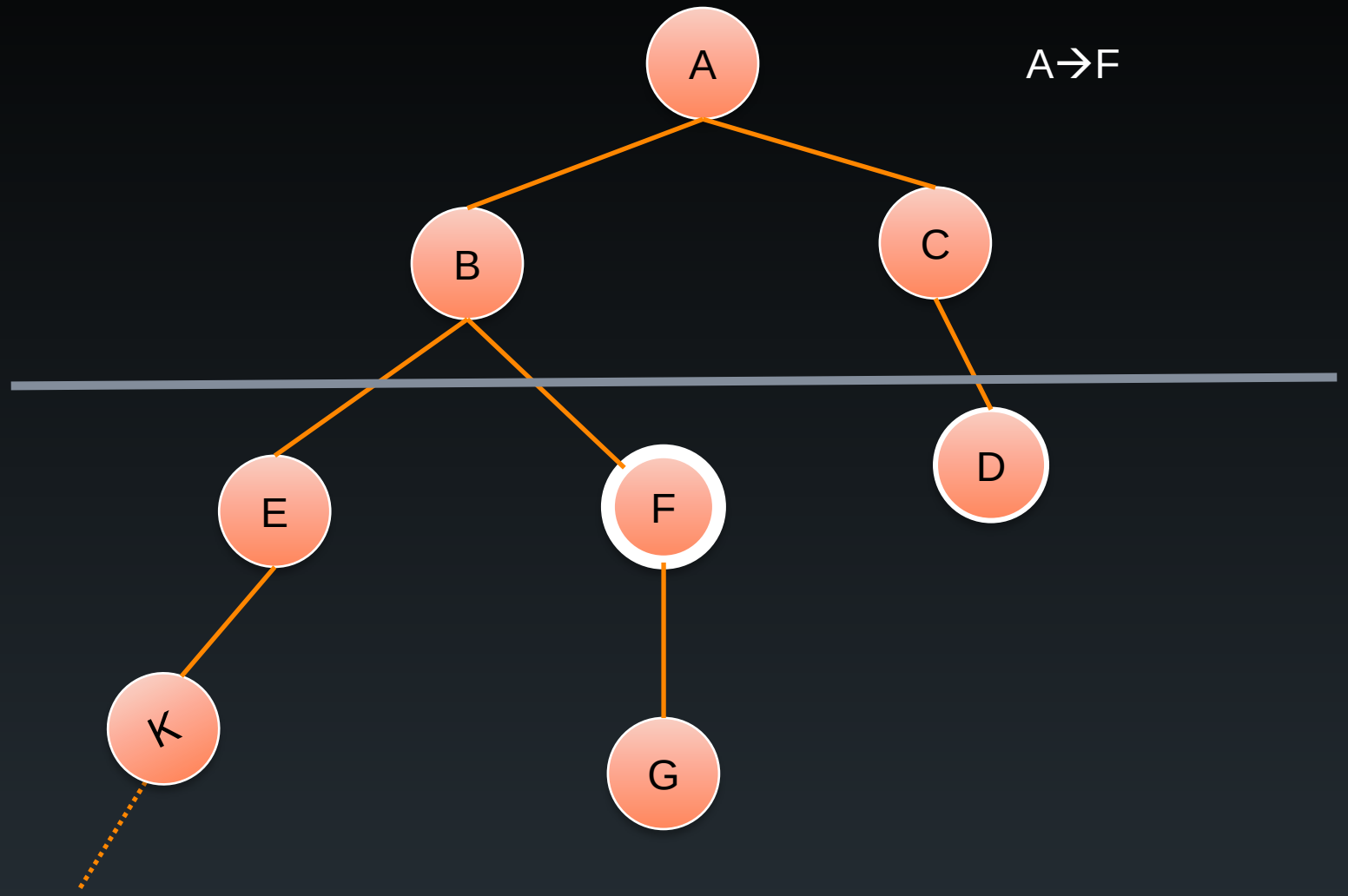
Recherche en profondeur d’abord

- Non complète: pour les espaces d’états infinis.
- Complexité en temps :
  $O(b^m)$ (m est la profondeur maximale)
- Complexité en espace :
  $O(bm)$(pour la version de recherche dans un arbre)
- Non Optimale

Recherche en profondeur limitée



Recherche en profondeur limitée

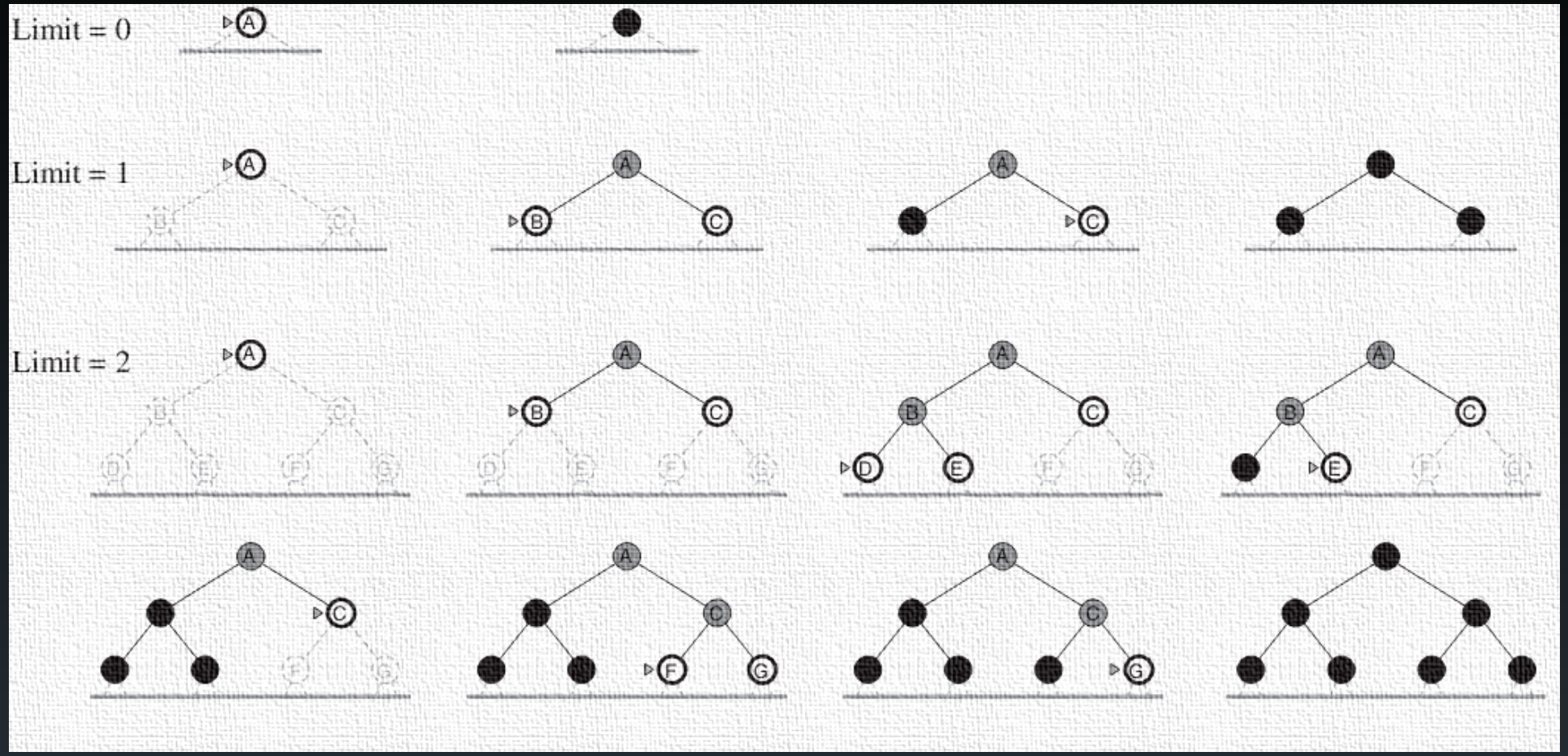


Recherche en profondeur itérative

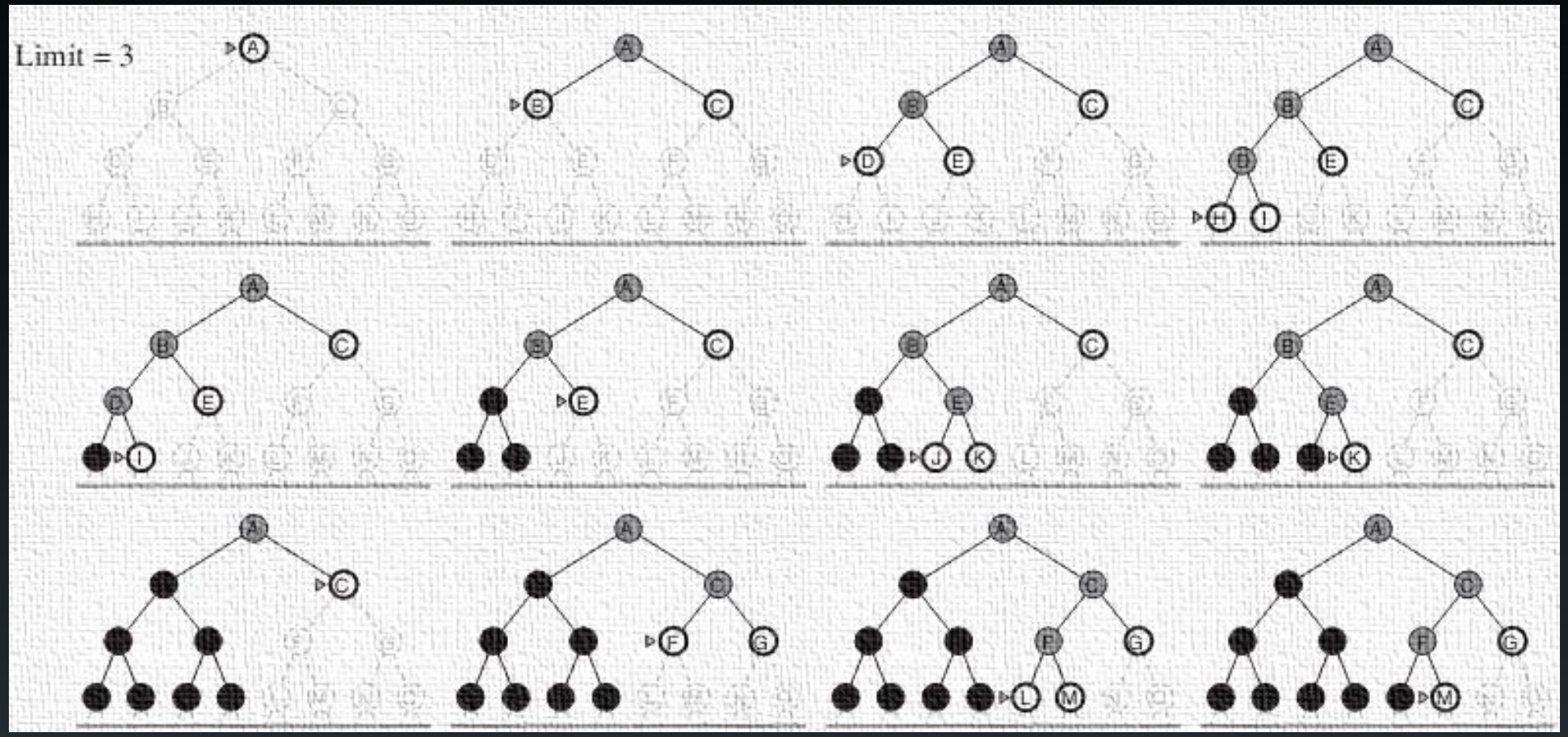
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

- Profondeur limitée, mais en essayant toutes les profondeurs: 0, 1, 2, 3, : : :
- Evite le problème de trouver une limite pour la recherche profondeur limitée
- Combine les avantages de l'exploration en largeur d'abord (complète et optimale), et la complexité en espace de l'exploration en profondeur d'abord

Recherche en profondeur itérative



Recherche en profondeur itérative



Recherche en profondeur itérative

- Peut paraître du gaspillage car beaucoup de nœuds sont étendus de multiples fois,
  - mais la plupart des nouveaux nœuds étant au niveau le plus bas
  - ce n'est pas important de développer plusieurs fois les nœuds des niveaux supérieurs.

Si b=10 et d =5 on a :

$$N(IDS) = 50 + 400 + 3000 + 20000 + 10000 = 123450$$
$$N(BFS) = 10 + 100 + 1000 + 10000 + 10000 = 111110$$

- Complete
- Complexité en temps :
- $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots\ldots\ldots + b^d = O(b^d)$
- Complexité en espace : $O(bd)$
- Optimale : oui, si le coût de chaque action est de 1.

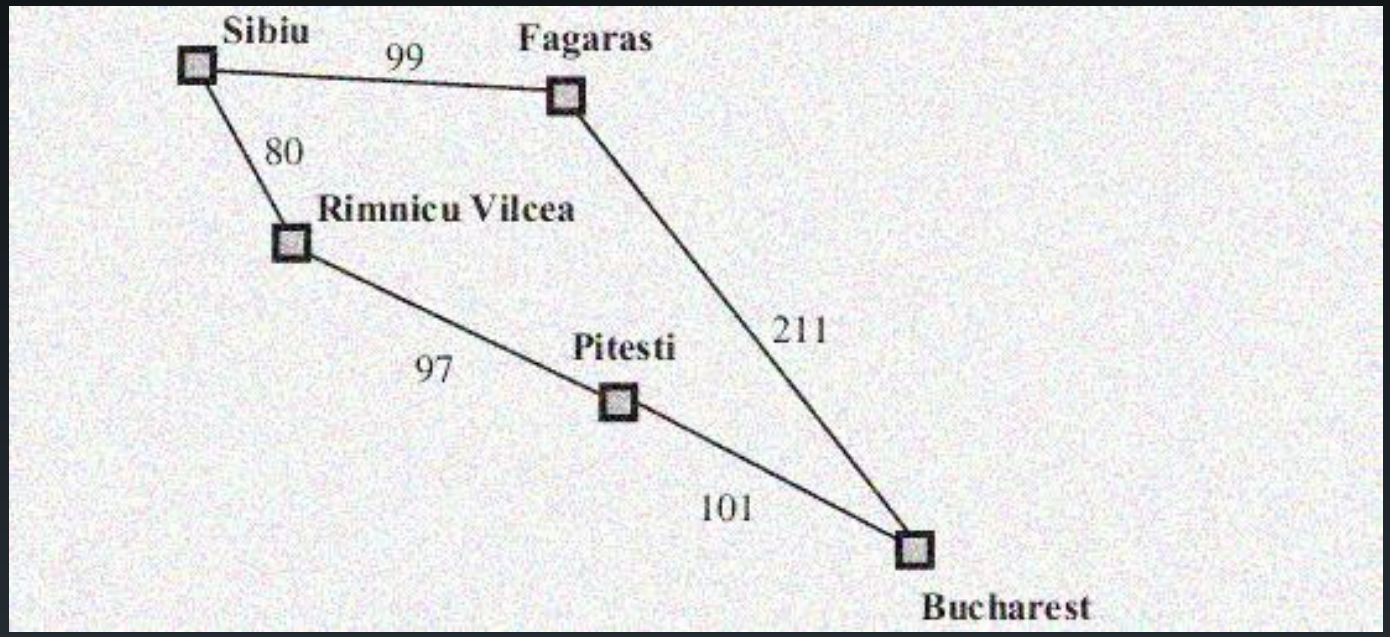
Recherche à coût uniforme

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Recherche à coût uniforme

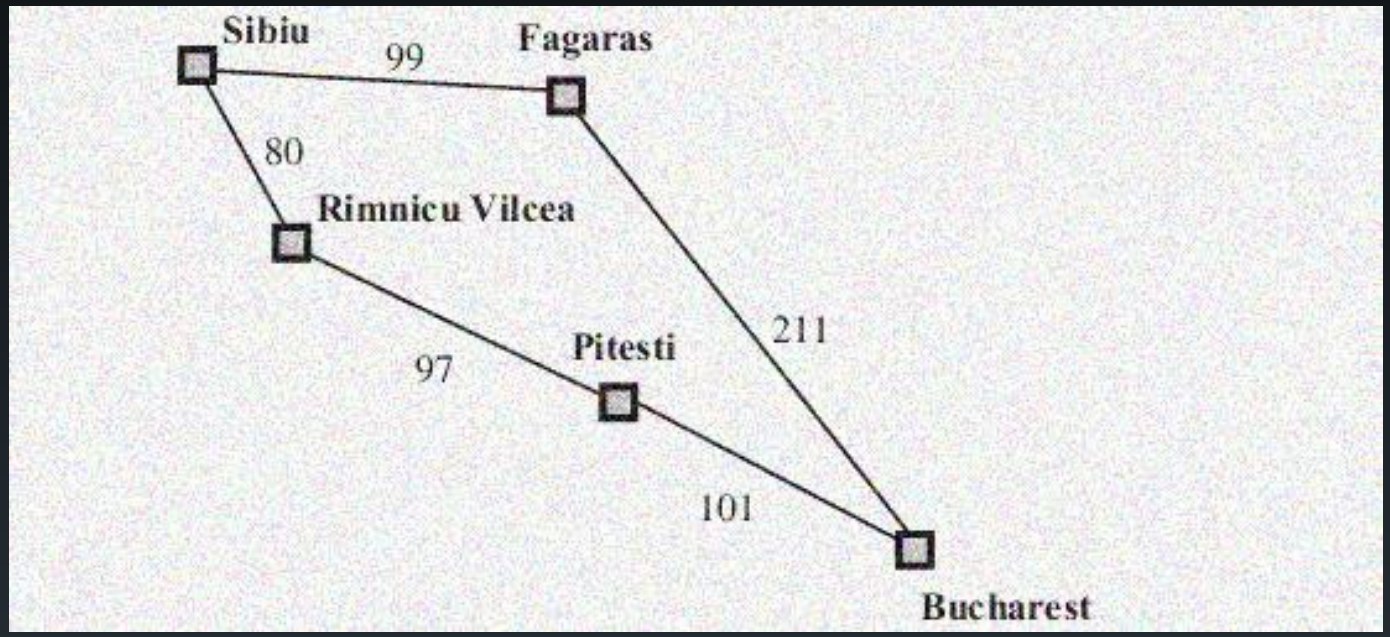
Frontière

Sibiu, 0



Recherche à coût uniforme

Frontière

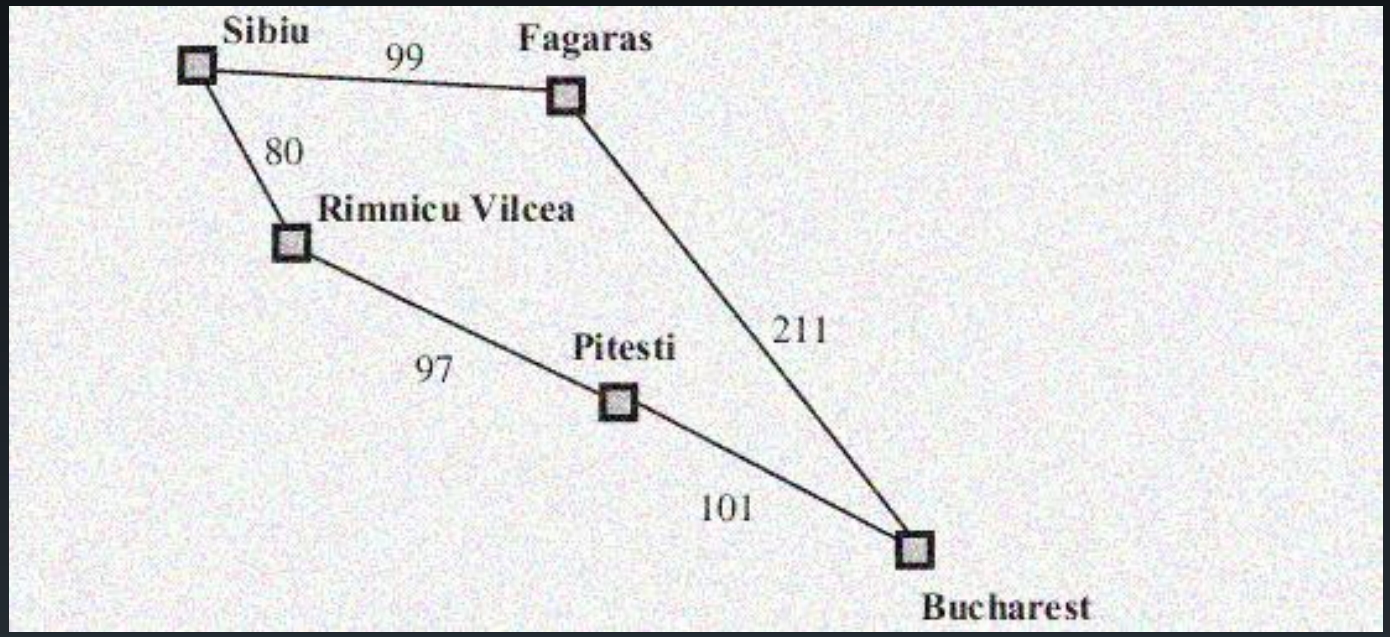


Recherche à coût uniforme

Frontière

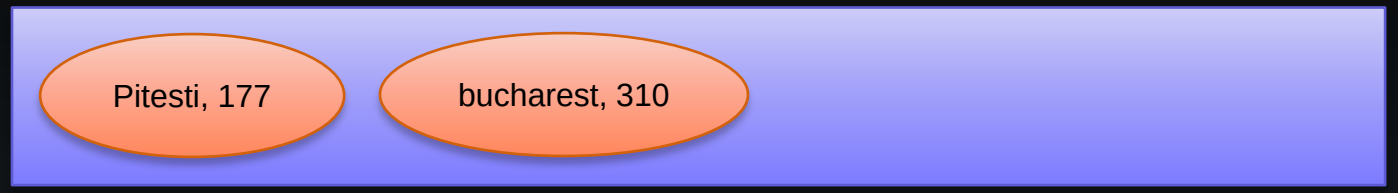
Fagaras, 99

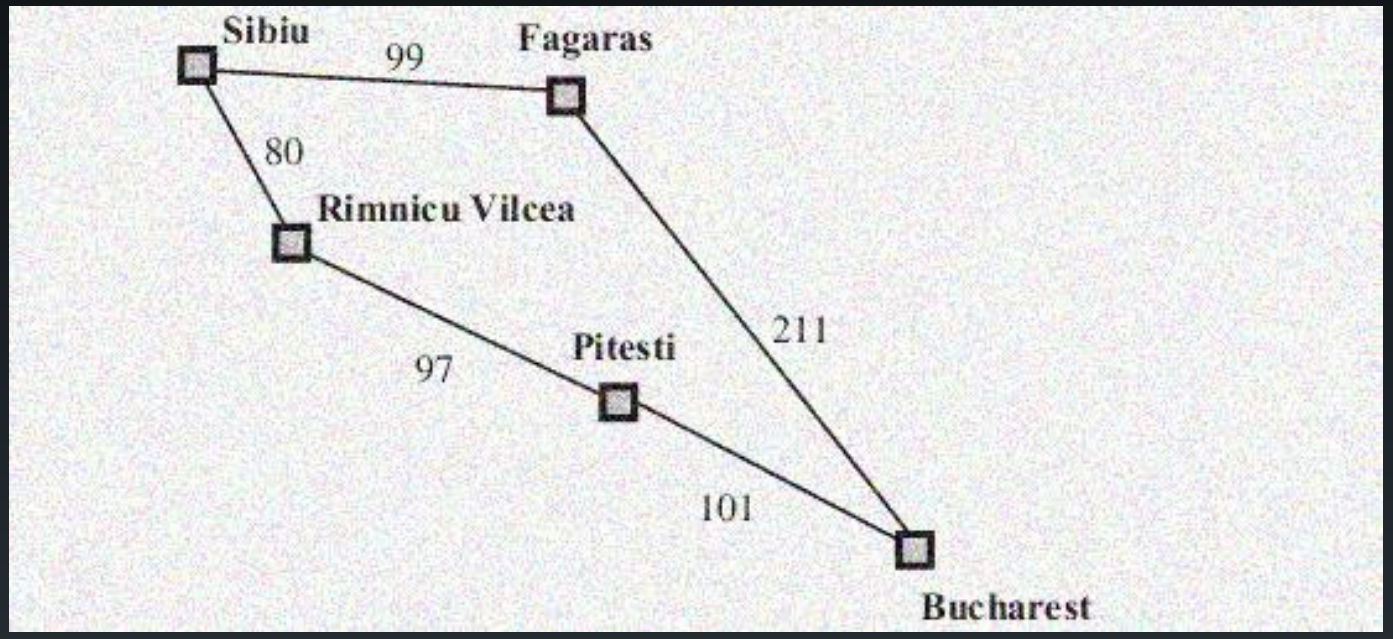
Pitesti, 80+7=177



Recherche à coût uniforme

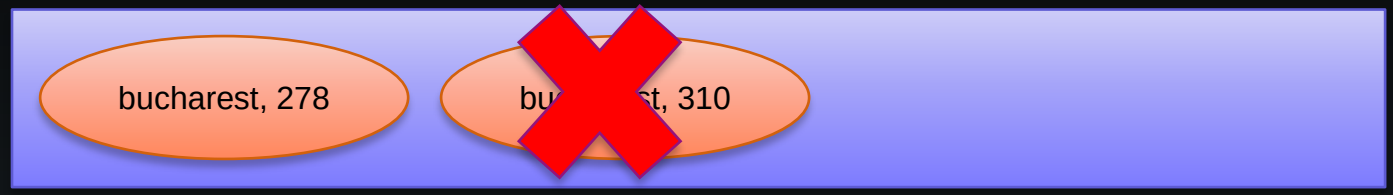
Frontière

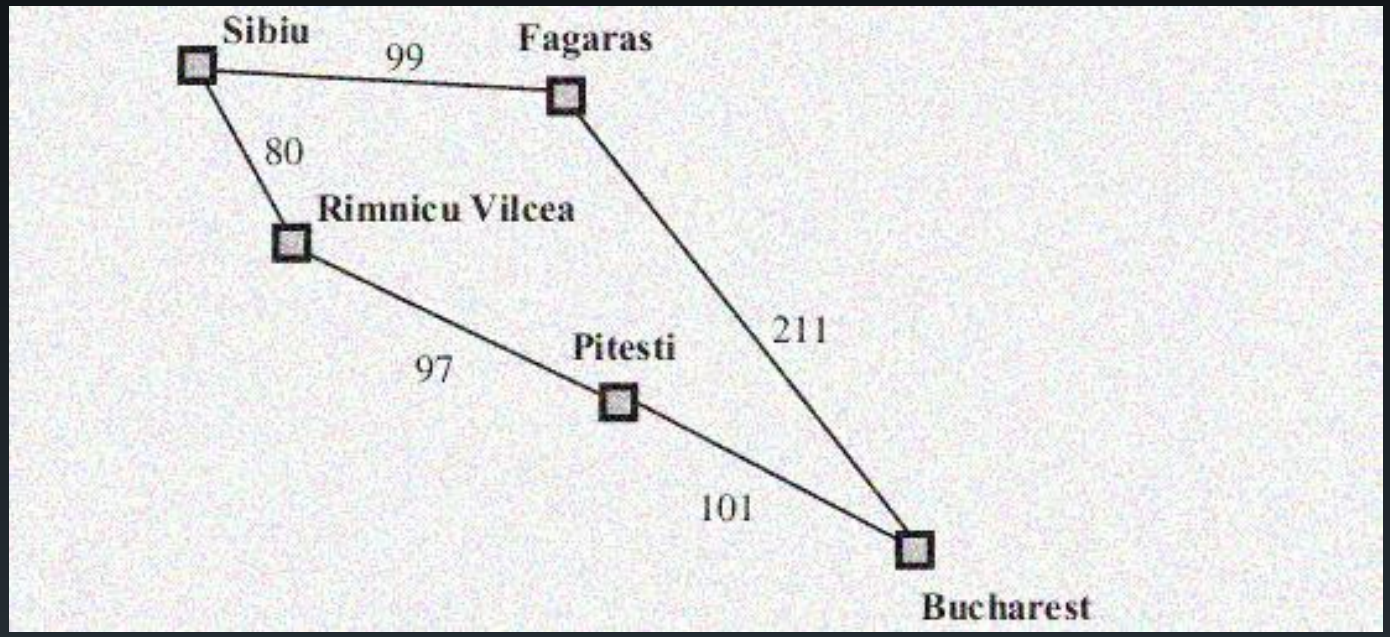




Recherche à coût uniforme

Frontière

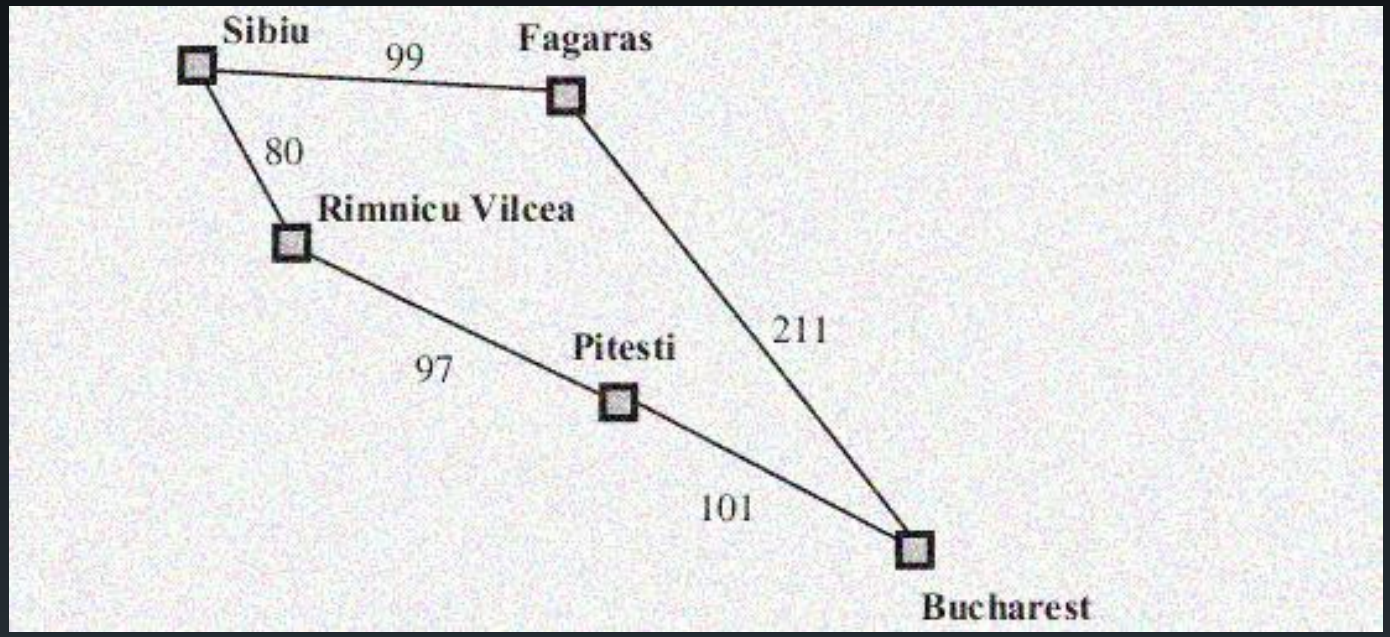




Recherche à coût uniforme

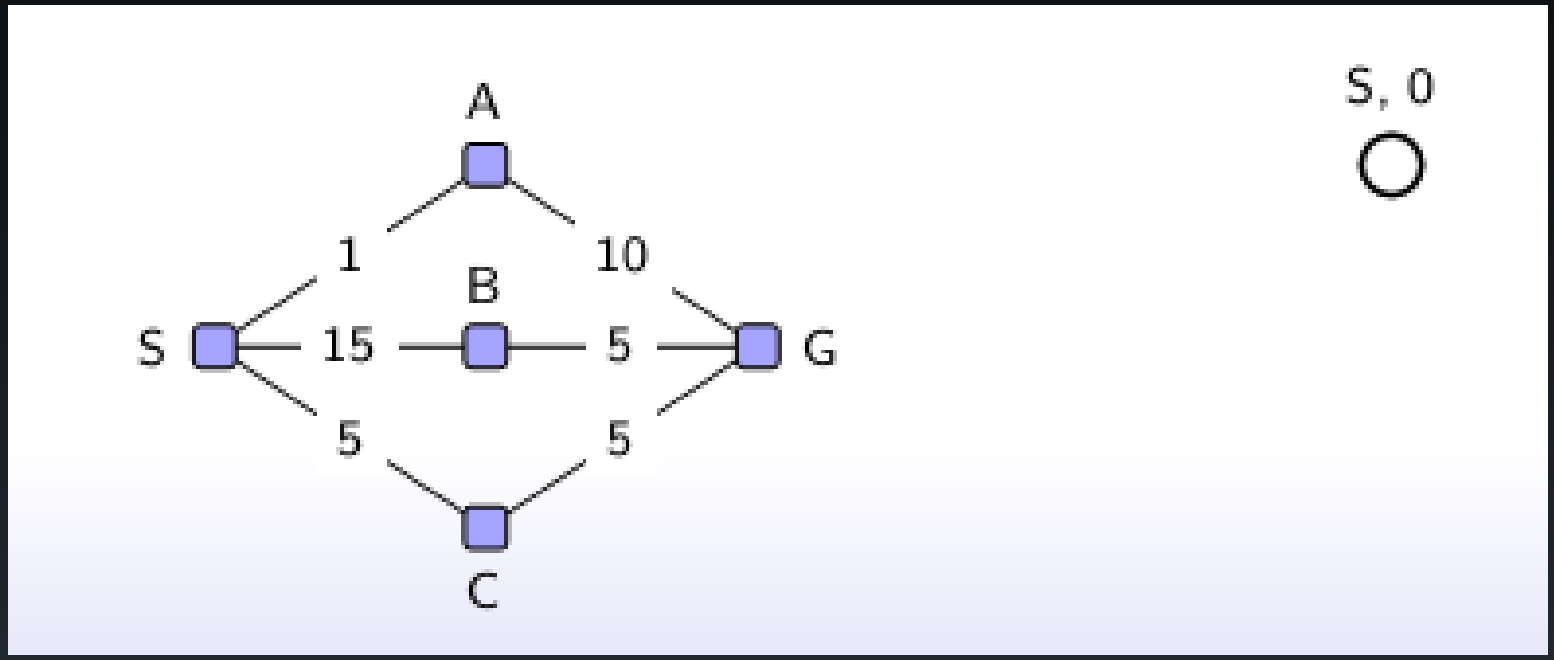
Frontière

bucharest, 278



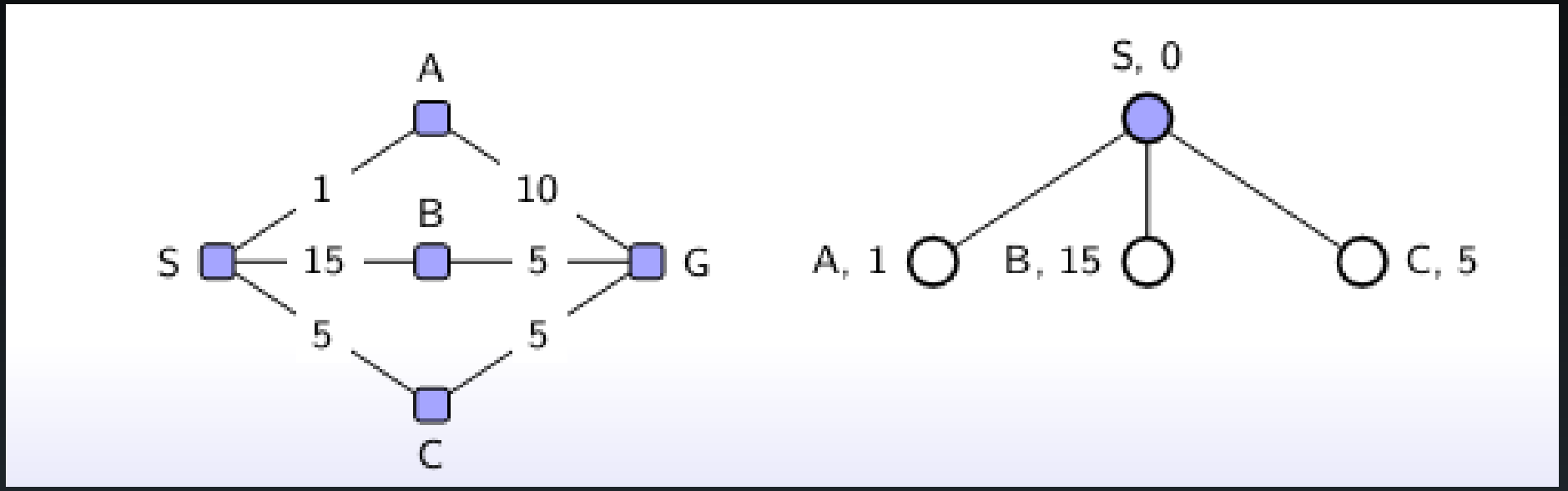
Recherche à coût uniforme

Frontière={(S,0)}



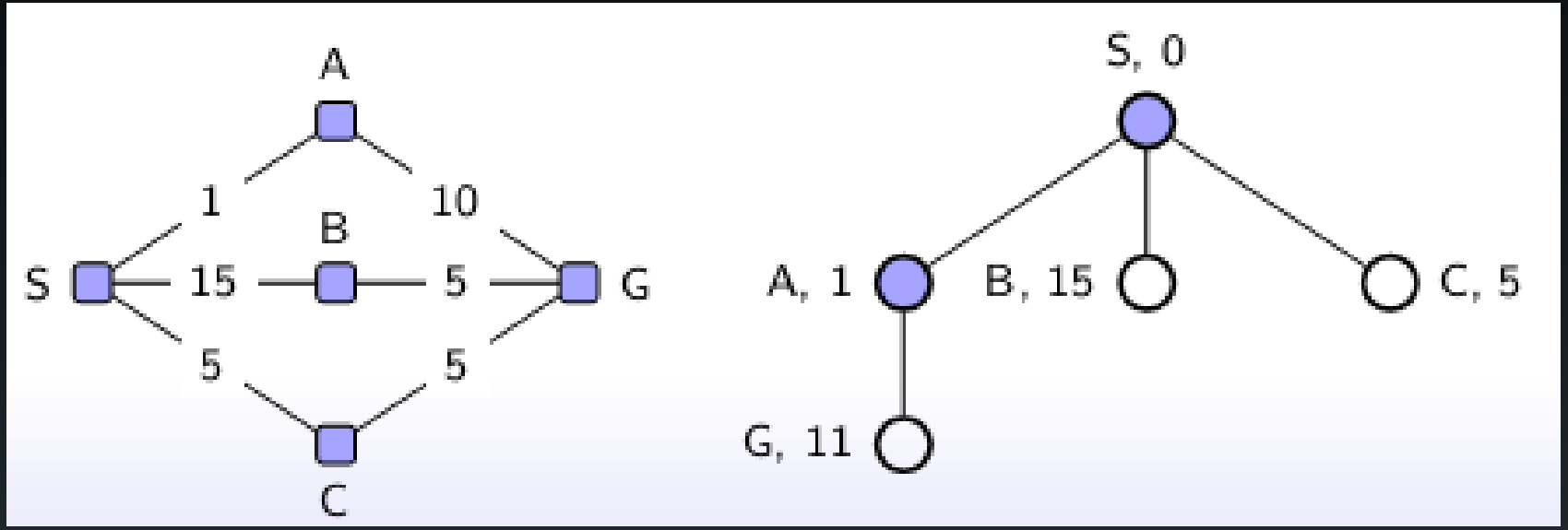
Recherche à coût uniforme

Frontière={(A,1); (C,5); (B,15)}



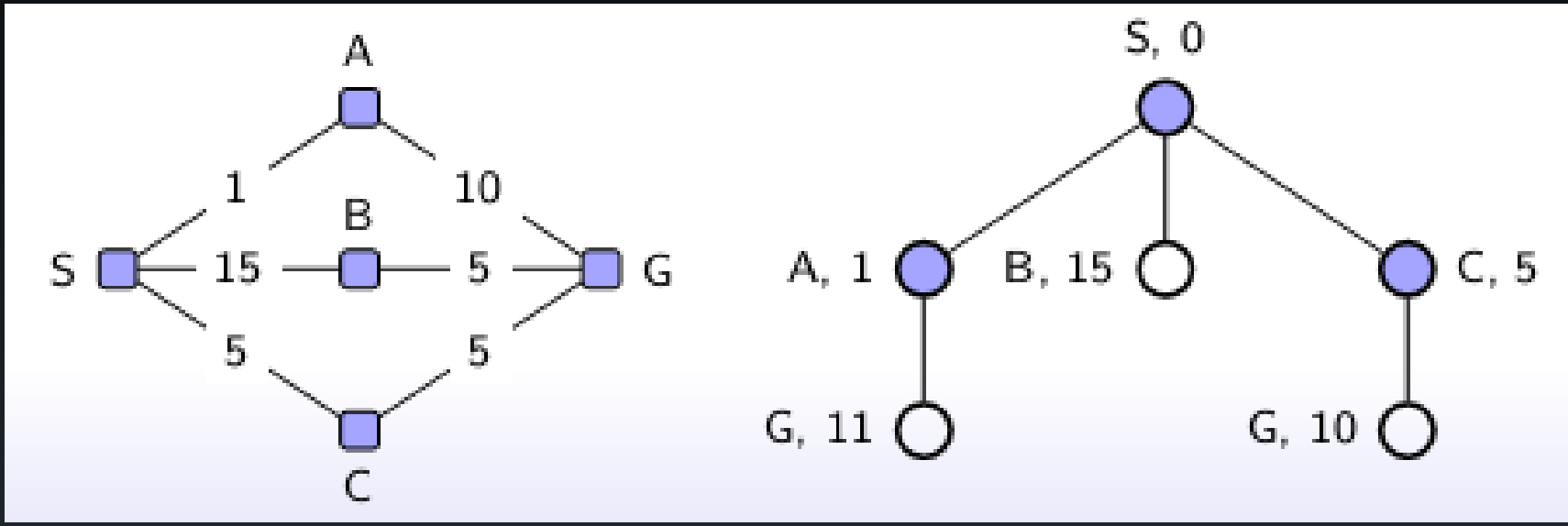
Recherche à coût uniforme

Frontière={ (C,5); (G,11); (B,15)}



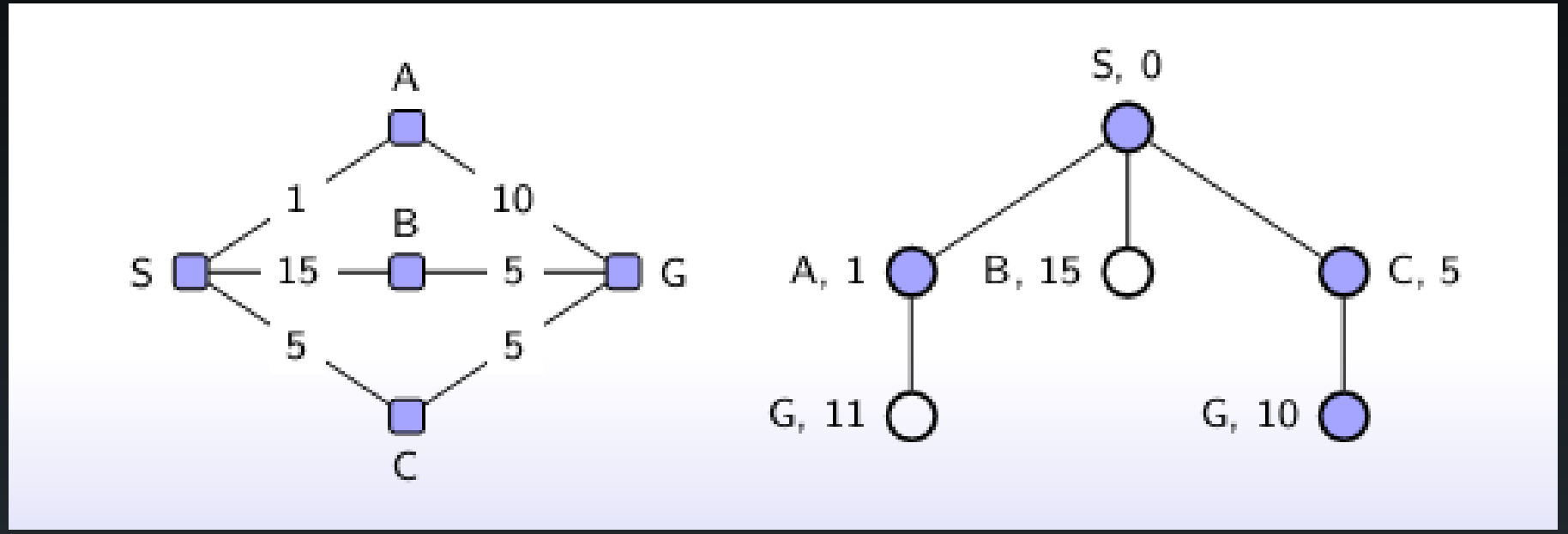
Recherche à coût uniforme

Frontière={ (G,10); (B,15)}



Recherche à coût uniforme

Frontière={ (B,15)}



Recherche à coût uniforme

- Elle est équivalente à la recherche en largeur d'abord si le coût est toujours le même.
- Complete, si le coût est strictement positif.
- Complexité en temps :
  $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- Complexité en espace :
  $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- Optimale

Résumé des algorithmes de recherche non informée

Critères	Largeur d'abord	Coût uniforme	Prof. d'abord	Prof. limitée	Prof. itérative
Complétude	Oui	Oui	Non	Oui si $l \geq d$	Oui
Temps	$O(b^d)$	$O(b^{\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Espace	$O(b^d)$	$O(b^{\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimalité - coût d'une action = 1	Oui	Oui	Non	Non	Oui
Optimalité - cas général	Non	Oui	Non	Non	Non

FIN