INTELLIGENCE ARTIFICIELLE

PR. IDRISSI NAJLAE
UNIVERSITE SULTAN MOULAY SLIMANE
FACULTE DES SCIENCES ET TECHNIQUES
DÉPARTEMENT INFORMATIQUE
BÉNI MELLAL

©2019-2024

RÉSOLUTION DE PROBLÈMES

CHAP. 3
VERSION 1

PLAN

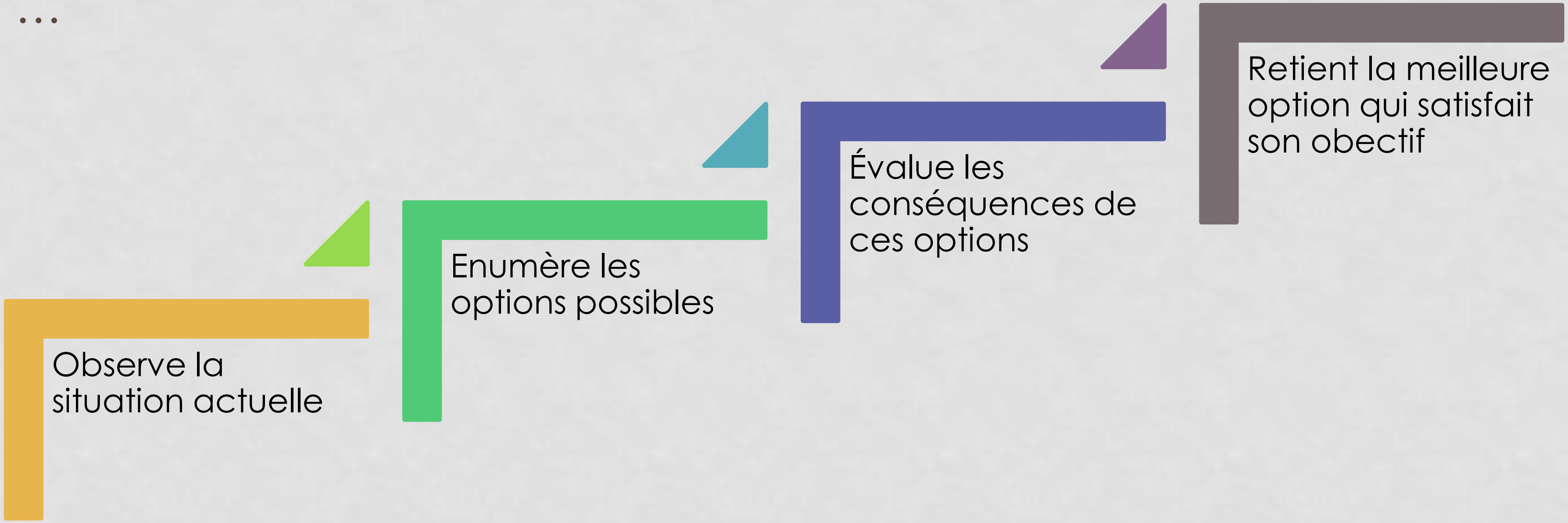
- Introduction
- Problèmes
- Résolution des problèmes
- Méthodes de recherche aveugles
- Méthodes de recherche heuristiques

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PROBLÈME – HUMAIN?

- Pour l'humain, un problème est une situation/ un fait pour lequel il doit trouver une solution ou prendre une décision.

- Objectif qui doit atteindre
- …



© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

AGENTS AVEC BUTS

Algorithme

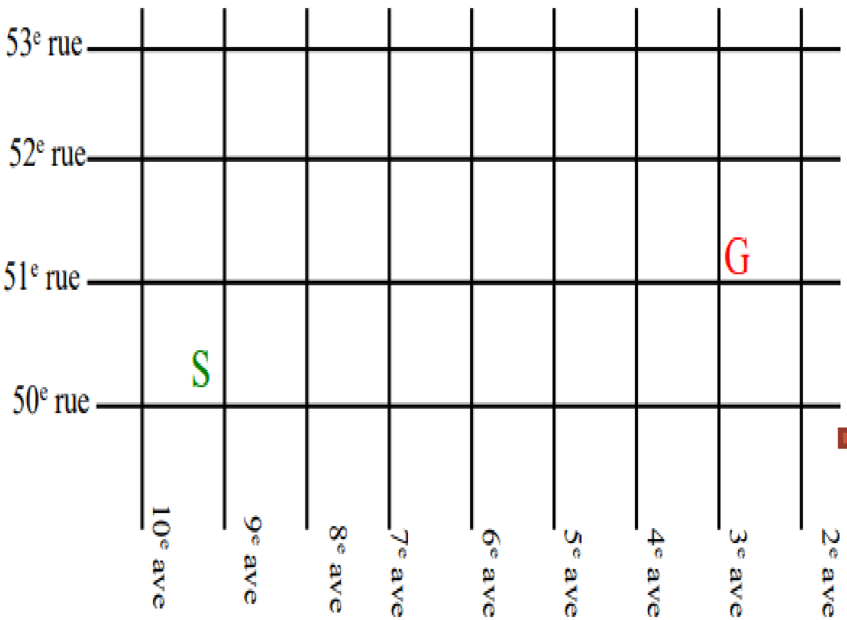
```
funtion SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
    input  : p, a percept
    static : s, an action sequence, initially empty
             state, some description of the current world state
             g, a goal initially null
             problem, a problem formulation
    state ← UPDATE-STATE(state, p)
    if s is empty then
        g ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, g)
        s ← SEARCH(problem)
    action ← FIRST(s), s ← REST(s)
    return action
```

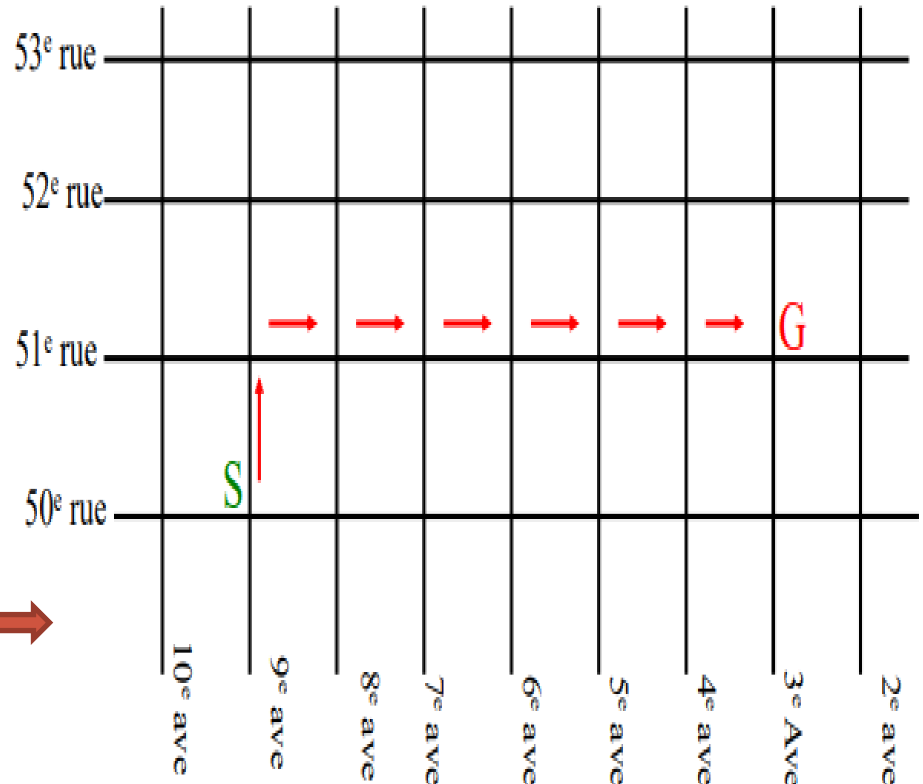

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

EXEMPLES
RECHERCHE UN CHEMIN

Trouver un chemin de la 9e ave & 50e rue à la 3e ave et 51e rue

(Exemple de Henry Kautz, U. of Washington)





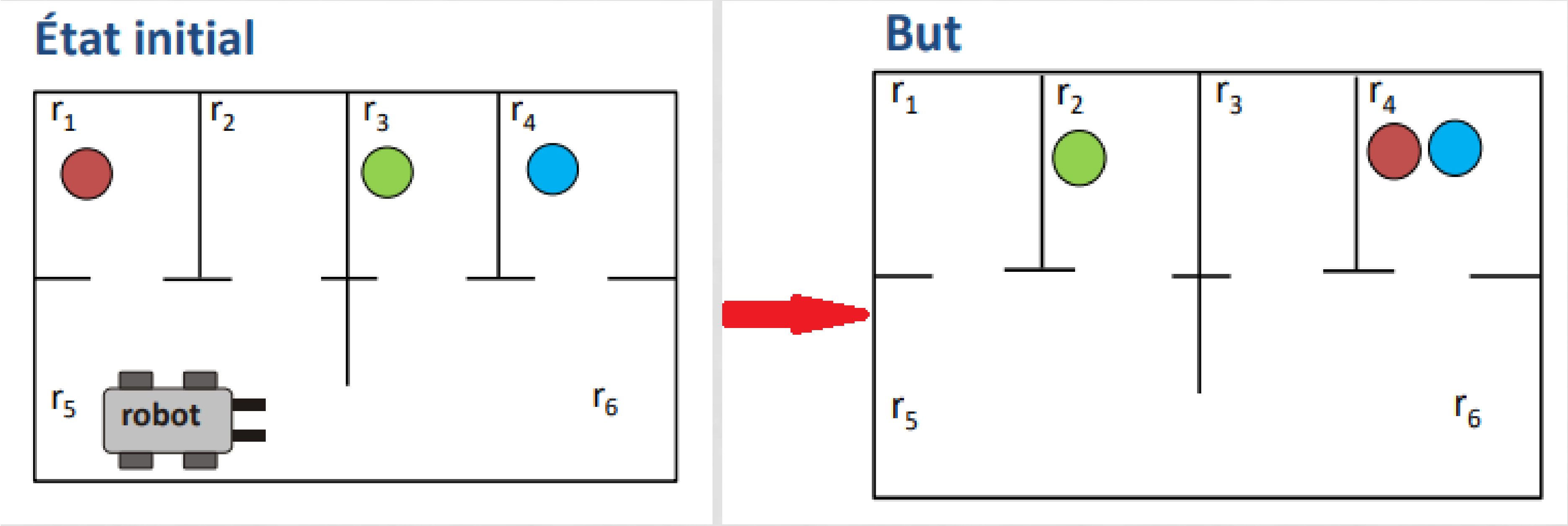
(Illustration par Henry Kautz, U. of Washington)

État initial

But

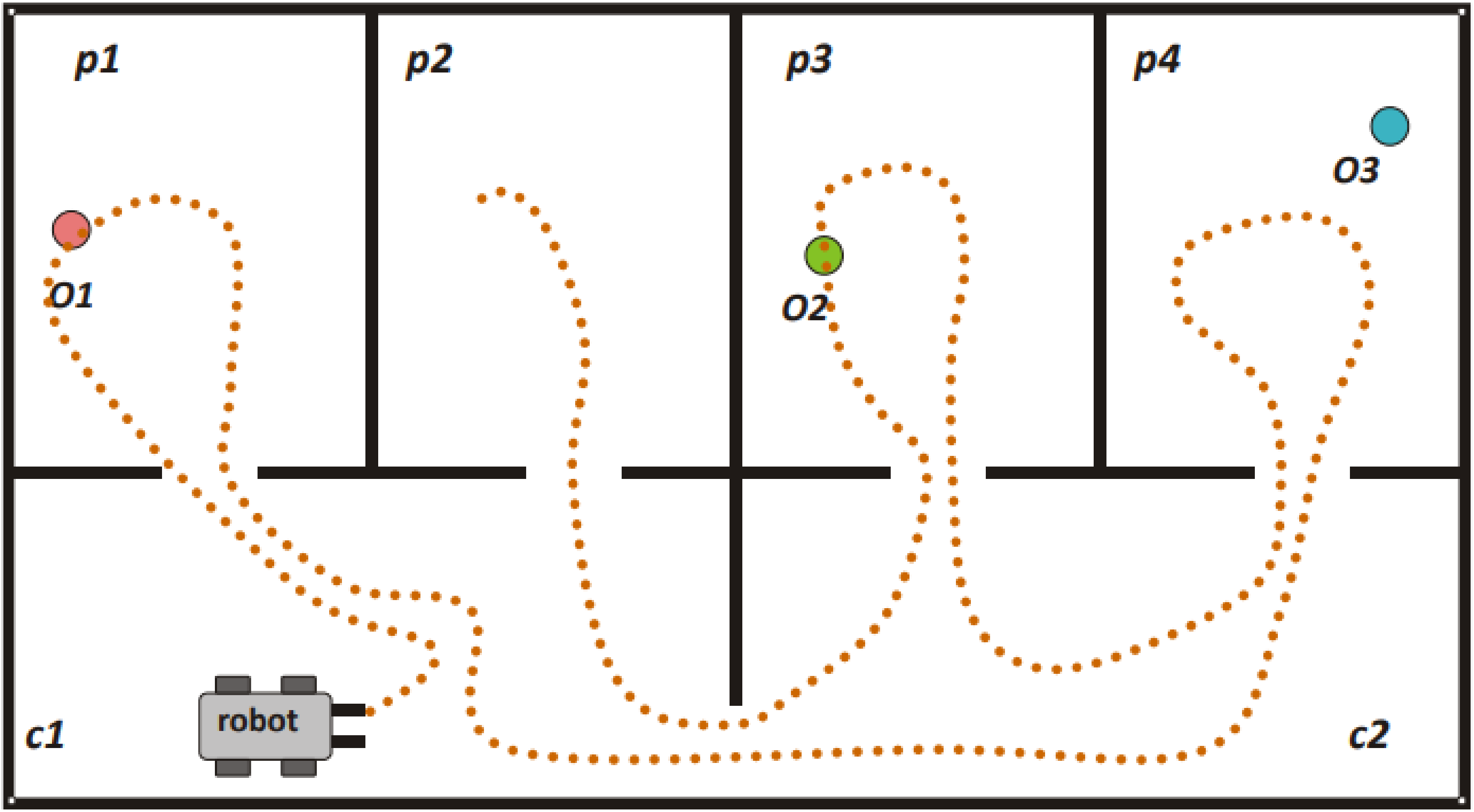
© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ROBOT LIVREUR

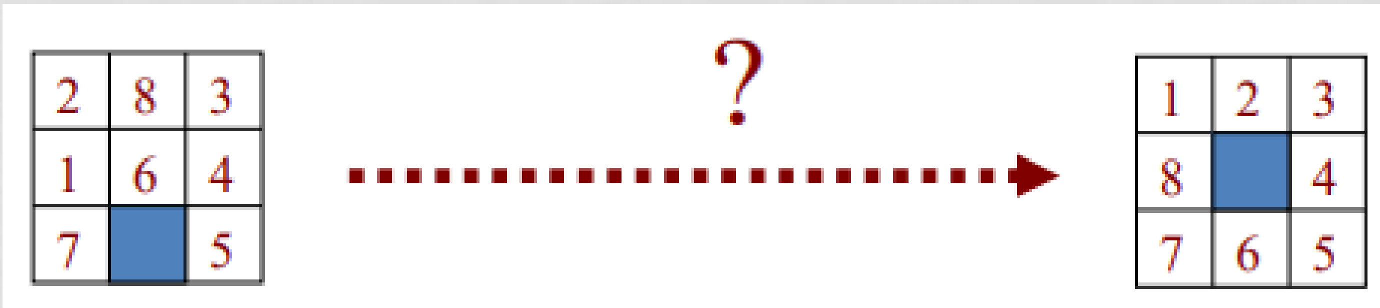


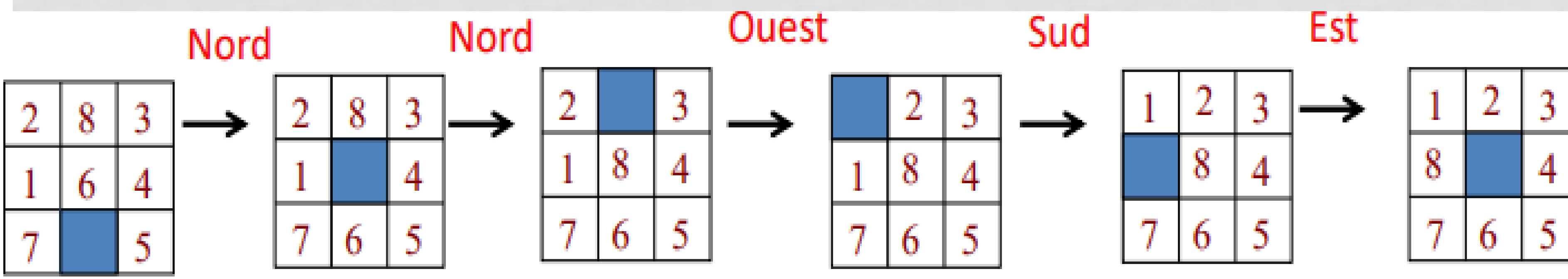
© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ROBOT LIVREUR



JEUX DE TAQUIN





© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PLAN

- Introduction
- Problèmes
  - Définition d'un problème
  - Types de problèmes
- Résolution des problèmes
- Méthodes de recherche aveugles
- Méthodes de recherche heuristiques

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

FORMALISATION DU PROBLÈME DE RECHERCHE

- **Démarche générale :**
  1. Trouver une bonne représentation du problème
  2. Trouver des opérateurs pour manipuler cette représentation
  3. Effectuer un contrôle de stratégie

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

AGENT DE RÉSOLUTION DE PROBLÈMES

- Formulation du but : un ensemble d'états à atteindre.
- Formulation du problème : les états et les actions à considérer.
- Recherche de solution : examiner les différentes séquences d'actions menant à un état but et choisir la meilleure '*en mesurant un coût de solution pour chacune*'.
- Exécution : accomplir la séquence d'actions sélectionnée.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

DÉFINITION D'UN PROBLÈME

- Espace d'états : description (abstraction) du monde réel définissant le problème
  - Exemple : réseau des bus, trams, rue 5, . . .
- Buts : sous ensemble de l'espace d'états
  - Exemple : état final du jeux de taquin, robot livreur, …
- Action (opérateurs) : (ou *fonction de transition*) déplacement dans l'espace d'états {actions, successeurs}
  - une fonction qui donne l'état qui résulte d'avoir effectué une action depuis un état.
  - successeur : désigne tout état accessible à partir d'un état donné avec une seule action
  - Exemple: {droite(7,51), (6,51)}, ….
- *On peut représenter l'état initial, les actions et le modèle de transition par **un graphe***

DÉFINITION D'UN PROBLÈME

- Solution du problème : la séquence d'actions menant de l'état initial à l'état objectif
  - Exemple : {(p1,o1),p4},{(p4,o3,),p3},{(p3,o2),p2}

- Algorithme de Recherche : procédure qui calcule une (ou plusieurs) solution(s) à partir d'un problème (état initial, actions, états objectifs).

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

JEU DE TAQUIN

- *Espace d'états : ?* positions des huit pièces dans les cases + case vide
- *État initial:* n'importe quel état.

- *But: ?* atteindre la configuration désirée

- *Actions : ?* déplacement d'une case: droite, gauche, haut, bas.

- *Modèle de transition: ?* étant donnée un état et une action, cela renvoie l'état résultant

- *Test de but : ?* un état correspond-il à l'état final.

- *Solution?:* séquence d'états et actions de l'état initial à l'état final

- *Coût du chemin : ?* Chaque déplacement coûte 1, donc le coût total est le nombre d'étapes pour atteindre le but.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ASPIRATEUR

- *Espace d'états : ?* position de l'aspirateur, état du sol: propre ou sale
- *État initial:* n'importe quel état.
- *Test de but : ?* les deux pièces doivent être propres
- *Actions : ?* droite, gauche, aspirer
- *Modèle de transition: ?* étant donnée un état et une action, cela renvoie l'état résultant
  - ex: {aspirer(1), droite}, …
- *Coût du chemin : ?* une action vaut 1

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ROBOT ASSEMBLEUR

- *Espace d'états?* coordonnées du robot, angles, position de l'objet à assembler...
- *État initial:* position quelconque
- *Actions?* déplacements continus
- *Test du but?* objet complétement assemblé
- *Coût du chemin?* le temps d'assemblage

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

AUTRE MODÉLISATION

- On peut aussi modéliser un problème par un quadruplet (S, E_0, F, T) où :

  - S est l'ensemble de tous les états.
  - E_0 est l'état initial, $E_0 \in S$.
  - F est l'ensemble des états finaux $F \subset S$.
  - T est la fonction de transition : associe à chaque état E_i un ensemble de couples (A_{ij}, E_{ij}) tq A_{ij} soit une action élémentaire permettant de passer de l'état E_i à l'état E_{ij} .

➔➔➔ **Résolution** : *Trouver une séquence $E_0, E_1, .., E_j, ..E_n$ tel que $\exists A_1, ..A_j, ..A_n$ tels que $(A_j, E_j) \in T(E_{j-1})$ et $E_n \in F$.*

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

TYPES DE PROBLÈMES

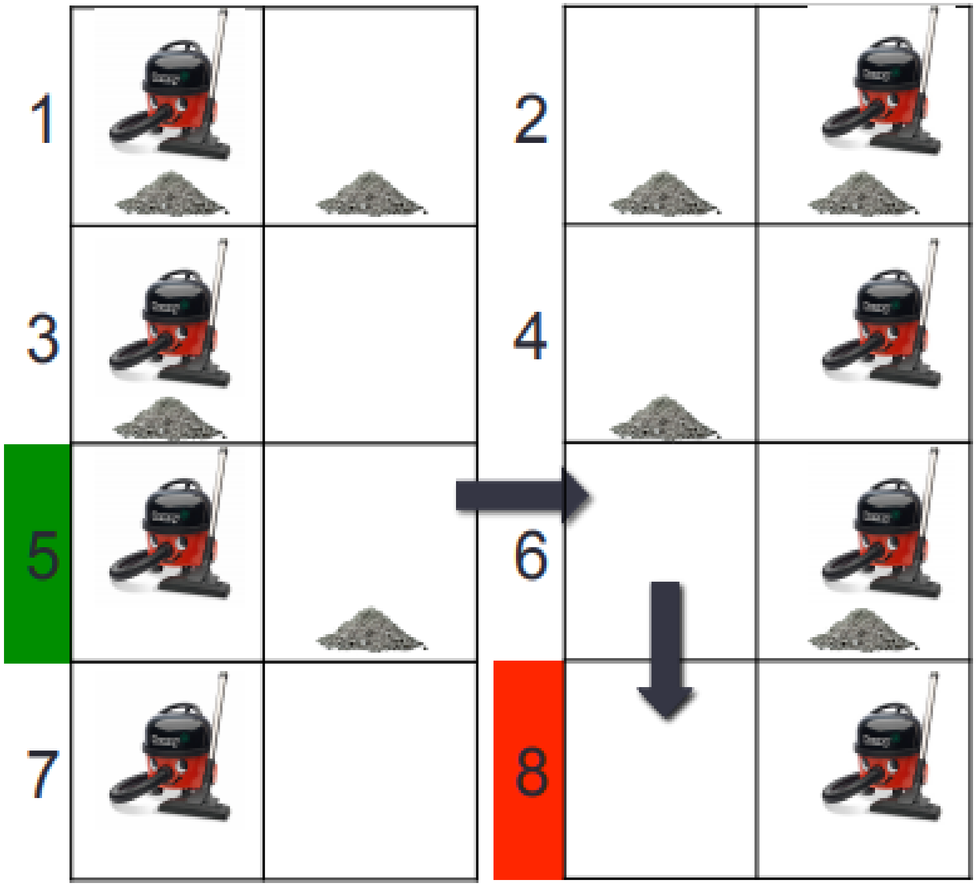
- ***Problèmes à état unique : déterministe***
  - L'agent connaît exactement l'état dans lequel il est
  - L'agent connaît précisément l'effet de ses actions
  - on peut à tout moment calculer l'état dans lequel se trouvera un agent après une action

- ***Problèmes à états multiples*** : déterministe, mais le monde n'est pas complètement accessible (données manquantes initialement)
  - L'agent ne sait pas exactement dans quel état réellement il se trouve (seulement un ensemble d'états possibles)
  - L'état actuel est caractérisé par un ensemble d'états
  - L'effet de chaque action est connu

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

Exemple : Le monde de l'aspirateur
Problème déterministe complètement observable

- État initial #5
- État final #8
- Solution? <droite, aspirer>

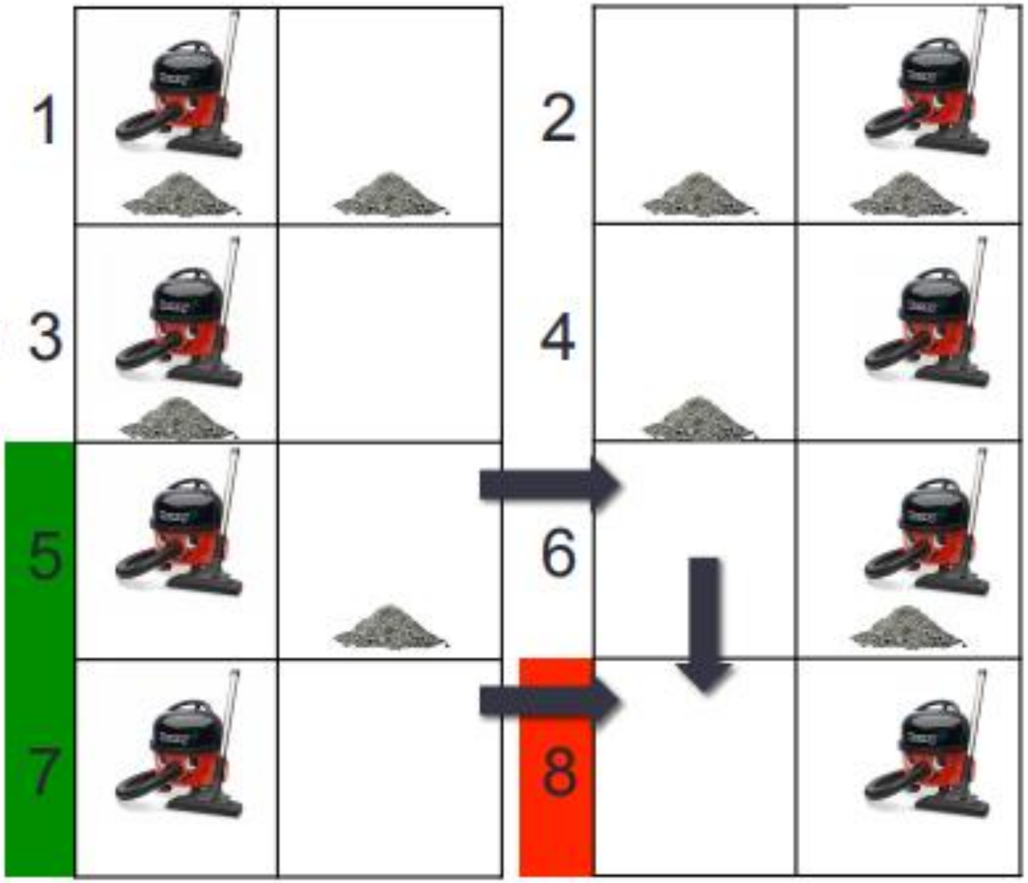


- ***Problèmes non déterministe/ partiellement observable*** (besoin de "capter" les états)
  - Les perceptions fournissent de nouvelles informations sur l'état courant
  - Le choix d'une action nécessite de tester l'environnement durant l'exécution
    - Souvent les phases de recherche et d'exécution sont alternées (ex.: jeux à deux joueurs)
  - L'effet d'une action est inconnu
    - Se déplacer à droite, mène à une case qu'on ne sait pas son état

- ***Problèmes d'exploration :*** l'espace d'états est inconnu, à découvrir par exploration

Exemple : Le monde de l'aspirateur
Problème non-déterministe et partiellement observable

- Non-déterminisme
  - aspirer ne garantit pas que le sol soit propre
- Partiellement observable
  - on ne sait pas si le sol dans l'autre pièce est propre
- États initiaux {#5, #7}
  -



© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

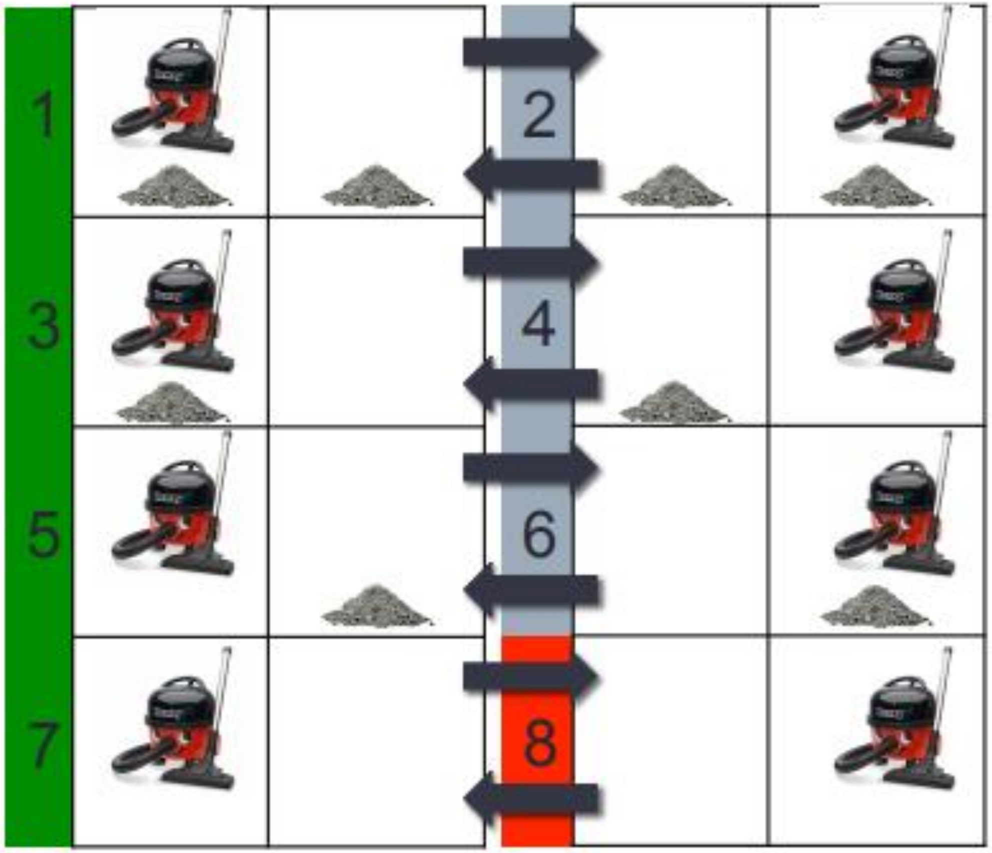
Exemple : Le monde de l'aspirateur
Problème non-observable

Solution pour { #1,#3,#5,#7}?

-

Solution pour { #2,#4,#6,#8}?

-



© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

NATURE DE PROBLÈME

Problème	
Jouets	**Monde réel**
Sert à illustrer ou expérimenter différentes méthodes de résolution de problèmes	Problèmes dont les solutions intéressent les gens
Jeu de taquin, jeu de reines, échecs, aspirateur, robot livreur, …	Problèmes de recherche de trajet, de planification, d’assemblage, …

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PLAN

- Introduction
- Problèmes
- Résolution des problèmes
  - La résolution ?
  - Méthodes de résolution constructive
  - Graphe d'états
  - Evaluation de la recherche
  - Type de recherches
- Méthodes de recherche aveugles
- Méthodes de recherche heuristiques

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

UNE MÉTHODE DE RÉSOLUTION?

- Démarche à trouver une solution pour un problème donnée

- Nécessité
  - de décrire formellement le problème
  - d'évaluer la solution

➔ Résolution constructive = Proposer + Evaluer une solution

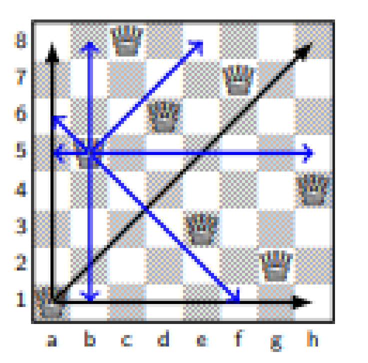
➔ Résolution par logique classique

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

RÉSOLUTION CONSTRUCTIVE

- Méthodes de résolution constructive
  - Avancer par étapes, par solutions partielles
    - Le taquin, les reines, …
  - Décomposer en sous-problèmes
    - Tours de hanoï, énigme du chien, loup et la chèvre, …

PROBLÈME DES 8 REINES



- But : placer 8 reines sur un échiquier de 8 × 8, sans menace possible

- Etat initial : échiquier vide

- Actions/opérateurs : ajouter une reine sur l'échiquier

- Résoudre des solutions partielles :
  - Placer la première reine,
  - puis placer les suivantes dans les cases restantes de l'échiquier

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PROBLÈME DES TOURS DE HANOÏ

- **But :** 3 mâts/bâtons (A, B, C) ; Déplacer la pyramide de N disques de taille croissante (taille(disque N) > taille(disqueN−1)) du mât A au mât C. Avec un coût minimum
- **Etat initial** : pyramide sur le mât A
- **Actions/opérateurs :** déplacer un disque à la fois d'un mât à un autre
  - Contrainte: un disque de taille m ne peut se poser sur un disque de taille m' < m
- **Coût :** 1 point par déplacement
- **Résoudre des sous problèmes**
  - Placer N-1 disques de A vers B
  - Placer le Nième disque de A vers C
  - Placer les N-1 disques de B vers C

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

GRAPHE D'ÉTATS

- *Représentation par graphe d'états*

  - Les ***nœuds*** représentent les états

  - Un ***arc (i,j)*** représentent l'opération/l'action permettant de passer de l'état i à l'état j

  - Une ***Solution*** = chemin entre l'état initial et l'état final

  - ***Recherche de Solution*** = Recherche du/d'un chemin entre l'état initial et l'état final

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ARBRE DE RECHERCHE

- La racine de l'arbre correspond à l'état initial du problème.

- Les feuilles de l'arbre correspondent à des états sans successeur dans l'arbre ou des nœuds qui n'ont pas encore été développés.

- Un chemin est une séquence de sommets partant du sommet à une feuille.
  - Le coût d'un chemin est défini par la somme des coûts de chaque opérateur intervenant dans la construction du chemin.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

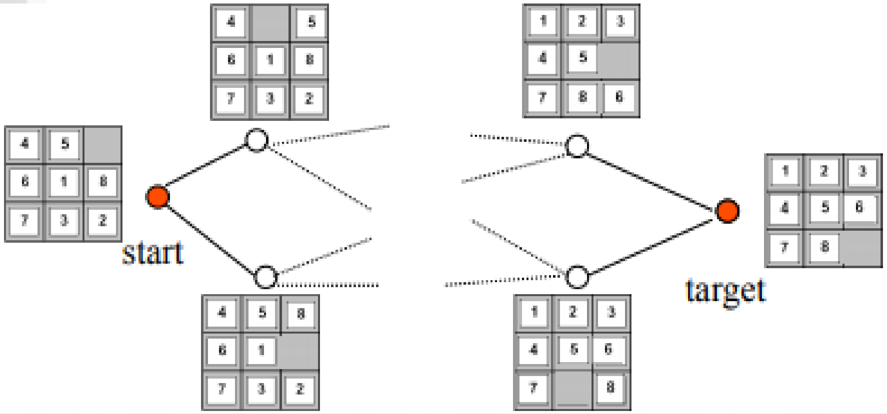
ARBRE DE RECHERCHE

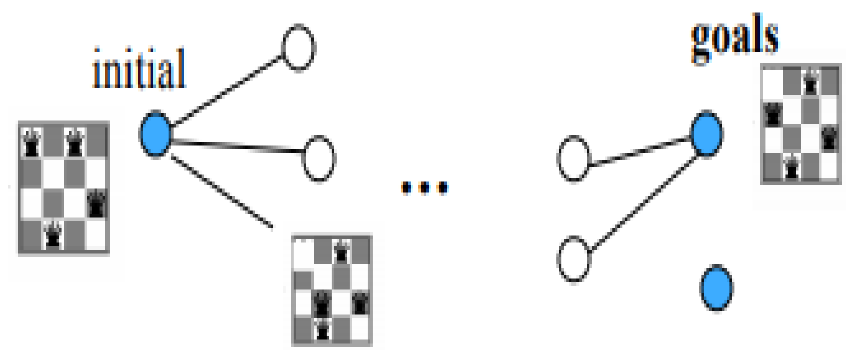
- La racine de l'arbre : l'état initial E_0.

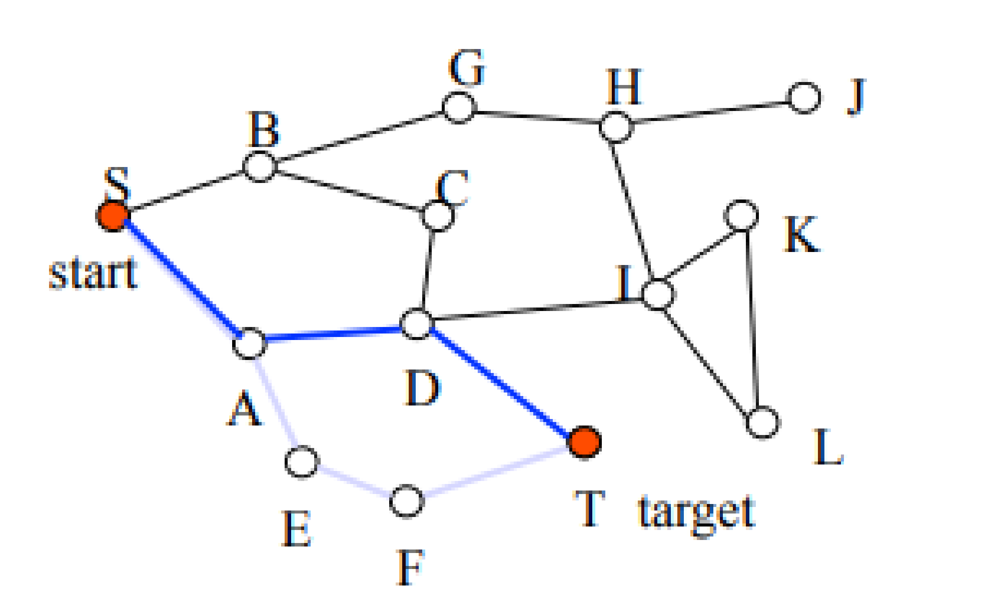
- Un état E_{ij} est fils d'un autre état E_i s'il existe une action qui permet d'obtenir E_{ij} à partir de E_i.

- Si une des feuilles correspond à un état final E_n, la solution est donc trouvée = $\{E_0, \ldots, E_i, \ldots, E_n\}$.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024


start
target

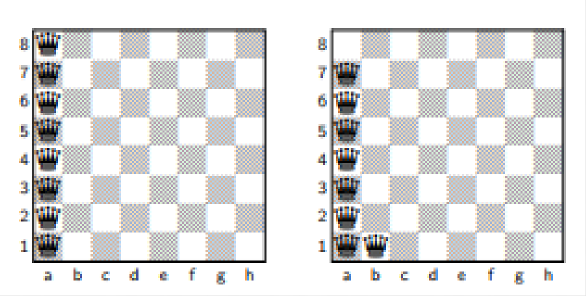

initial
...
goals


G
H
J
B
S
C
start
K
I
D
A
L
E
F
T target

IMPORTANCE DU CHOIX DE GRAPHE D'ÉTATS

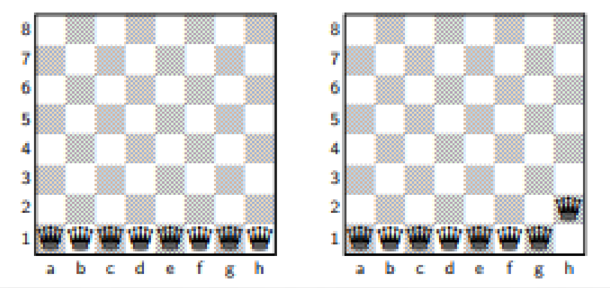
Reprenons le problèmes des 8 reines sur un échiquier 8 × 8

- **Si 1 état = 1 case**
  - on a donc : 64 × 63 × 62 × 61 × 60 × 59 × 58 × 57 = 64! − 56! = 1, 78.1018 états possibles : case(i, j) ∈ {reine, ∅}



  - Action: placerReine(i,j)
  - Test: PriseImpossible, ToutesReinesPlacees

- **Si 1 état = 1 position dans une ligne**
  - On aura donc: 8 × 7 × 6 × · · · × 1 = 8! = 40320 états possibles : ligne(i) ∈ {reine, ∅}



  - Action/Opérateur : placerReineDansLigne(i)
  - Test : PriseImpossible, ToutesReinesPlacees

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

IMPORTANCE DU CHOIX DE L'ACTION

- Sur le problèmes des 8 reines sur un échiquier 8 × 8
  - **Avec 1 état = 1 position dans une ligne**
  - Action/Opérateur : **vérifierDispoLigne(i)**, **placerReineDansDispoLigne(i)**
  - Test : PriseImpossible, ToutesReinesPlacees

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PROCESSUS DE RÉSOLUTION

- Le processus de résolution de problème consiste donc à construire un arbre/graphe de recherche qui se superpose à l'espace des états du problème.
- Chaque nœud du graphe correspond soit à l'état initial du problème, soit à un développement du sommet parent par un des opérateurs du problème.
- On choisit un nœud à développer et on teste si les fils sont des états finaux.

➔➔ ***Quelle politique pour choisir le nœud à développer? (choix de la stratégie ou méthode)***

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

FONCTION GÉNÉRALE DE RECHERCHE

```
Node or nil <- GeneralSearch(Problem p, QueuingFn strategy) // retourne une solution ou un échec
Queue Nodes <- MakeQueue(MakeNode(p.InitialState));
Loop do
    if Empty?(nodes) then return nil;
    Node <-b RemoveFront(nodes);
    if GoalTest(n.state) then return n;
    Nodes <- strategy(Expand(n,p.operators),nodes)
End
```


© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

PERFORMANCE D'UNE STRATÉGIE DE RÉSOLUTION

- *Une stratégie de recherche* est définie comme un moyen/critère qui permet de choisir un ordre pour développer les nœuds.

- Critères de mesure de performance d'une stratégie:
  - *Complétude :* est-ce que la stratégie garantit de trouver une solution s'il y en a une indique qu'il n'y en a pas lorsque aucune n'existe?
  - *Optimalité :* est ce que la stratégie trouve toujours la solution optimale (selon le coût) ?
  - *Complexité de temps:* la technique est-elle coûteuse « en temps ? » combien de temps ça prend pour trouver une solution ?
  - *Complexité d'espace :* combien d'espace mémoire nécessaire pour effectuer la recherche?

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

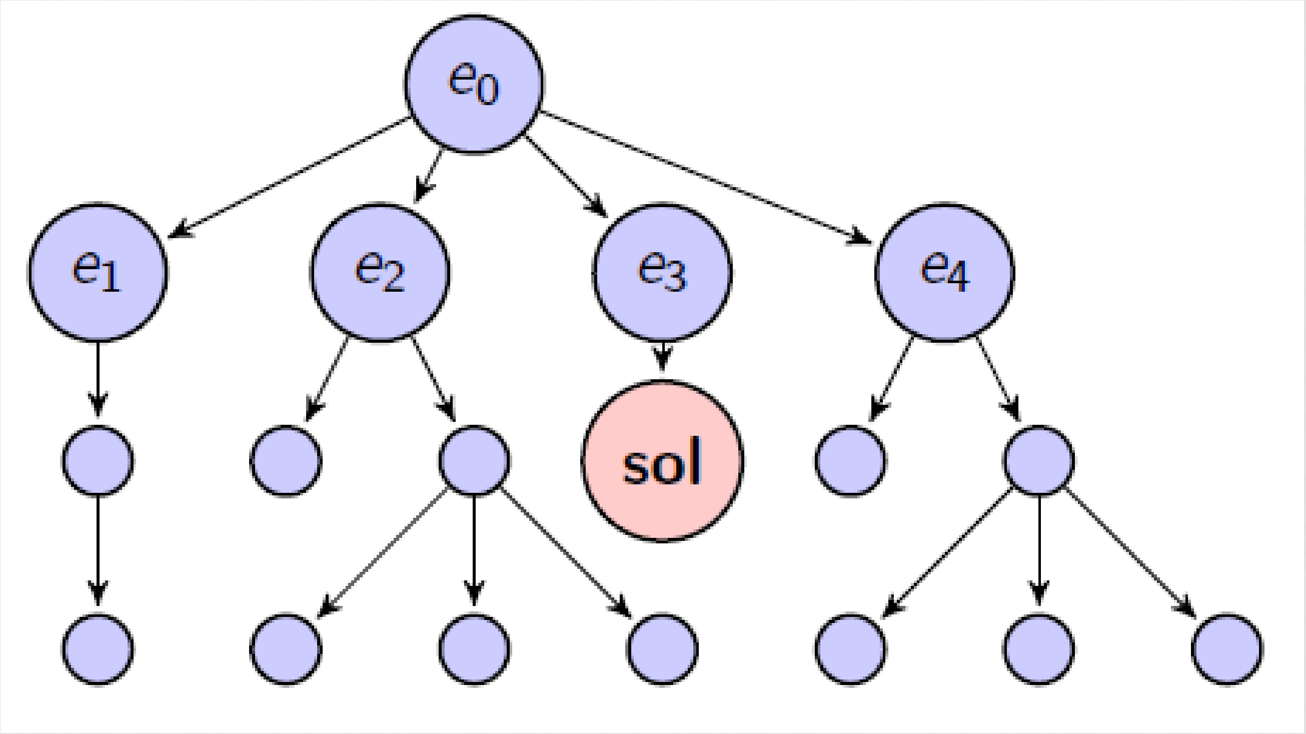
COMPLEXTITÉ

Elle est exprimée en utilisant les quantités suivantes :

- ***b***, le facteur maximum de branchement: le nombre maximum de successeurs à un nœud de l'arbre de recherche
- ***d***, la profondeur du **nœud but** le moins profond.
- ***m***, la longueur maximale d'un chemin dans l'espace d'états.

- Complexité en temps : le nombre de nœuds générés pendant la recherche.
- Complexité en espace: le nombre maximum de nœuds en mémoire.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024



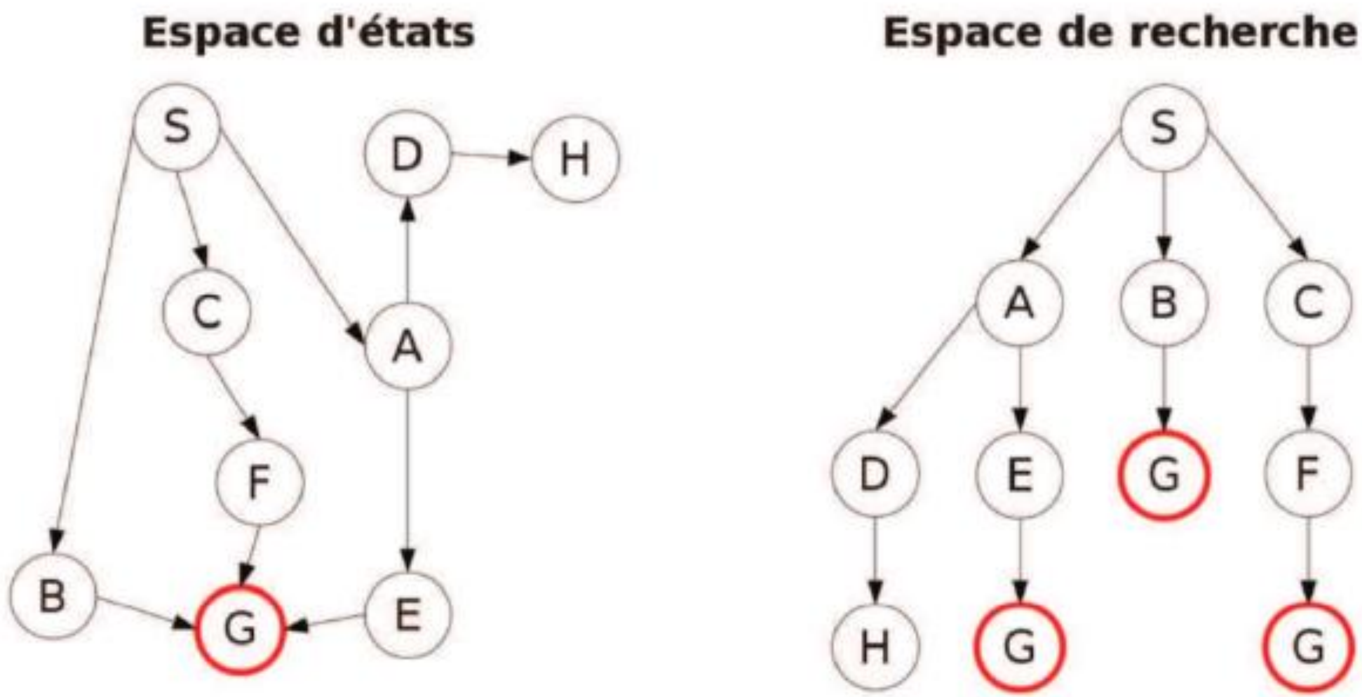
Pour cet arbre de recherche:
La racine : e_0, le but = sol, la solution : {e_0, e_3, sol}
Les paramètres de complexité sont:
b = 4,
d = 2
m = 3

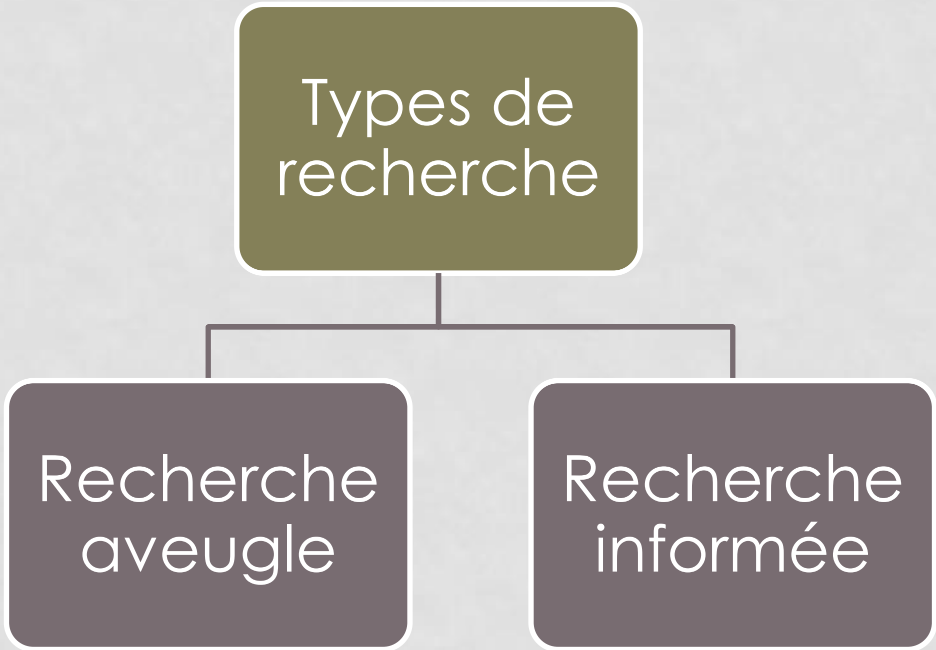
Algorithm Recherche générale

```
Argument : problème
Sortie : échec ou une solution
Initialiser l'arbre de recherche du problème par l'état initial
Boucler
    Si il n'y a pas de candidat (noeud) possible alors
        Retourner échec
    Sinon
        Choisir un candidat
        Si le candidat satisfait le but alors
            Retourner solution
        Sinon
            étendre le noeud
            ajouter les noeuds fils dans l'arbre
        Fin si
    Fin si
Fin Boucler
```

Activer Windows
Accédez aux paramètres p

Exemple





utilise seulement les informations disponibles dans le problème

- recherche en largeur
- recherche en profondeur
- recherche en profondeur limitée
- recherche par approfondissement itératif

Utilise une fonction d’estimation (heuristique) pour choisir les nœuds à visiter.

- Meilleur d'abord
- L'algorithme *A**
- Heuristiques
- Algorithmes génétiques

RECHERCHE EN LARGEUR

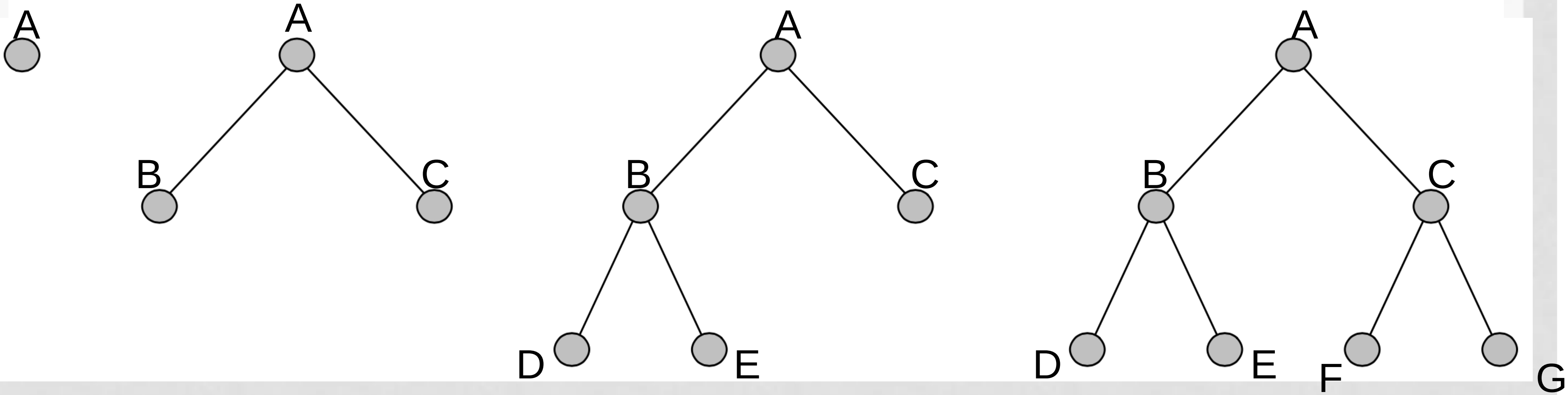
(*Breath -first search*)

- En théorie des graphes: on parcourt les nœuds de l'arbre pointé par niveau croissant.

- Principe :
  - Visiter les états en parcourant l'arbre niveau par niveau.
  - Développement de tous les nœuds au niveau *i* avant de développer les nœuds au niveau i+1
  - la recherche s'arrête quand on atteint l'état final ou bien une profondeur maximale est trouvée.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

EXEMPLE



Ordre du parcours : A – B- C – D – E – F – G

EN LARGEUR

- Complétude : oui, si *b* est fini
- Optimal : non en général, oui si le coût des actions est le même pour toutes les actions
- Complexité en temps : $b^0 + b^1 + b^2 + \ldots + b^d + b^{d+1} = O(b^{d+1})$
- Complexité en espace : $O(b^{d+1})$ (Garde tous les nœuds en mémoire)

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

ALGORITHME

Fonction RechercheLargeur(E_0)

Créer file *F*

Enfiler E_0 dans *F*

Tant que *F* est non vide :

E <- défiler *F*

marquer E comme visité

S <- sucesseurs(E)

∀ s dans S :

Si s n'est pas visité alors :

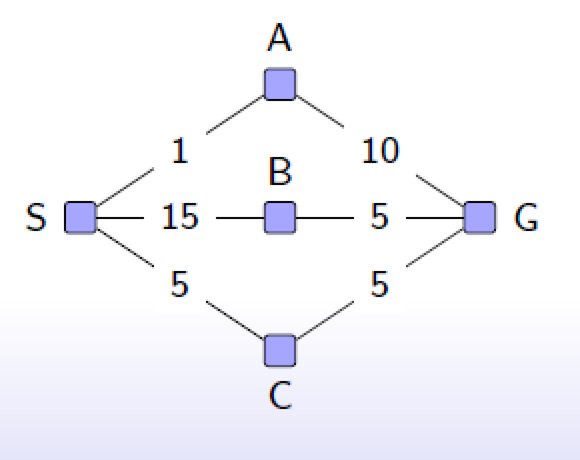
ajouter s dans *F*

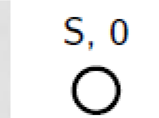
© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

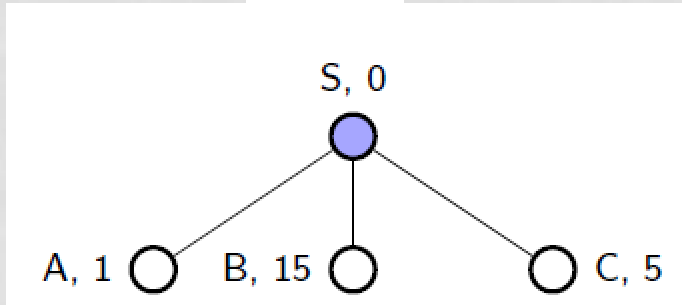
RECHERCHE EN COÛT UNIFORME

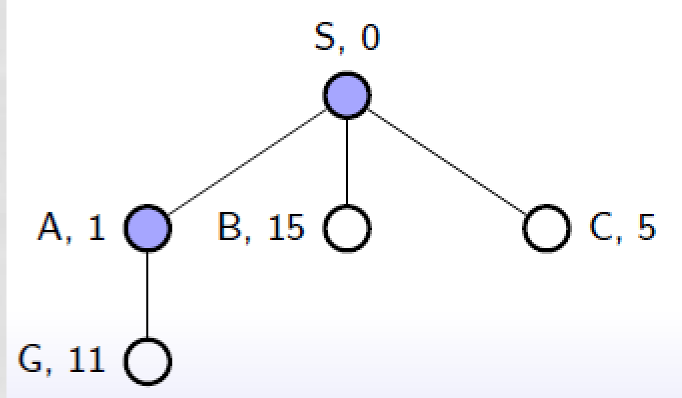
➔ Une solution pour rendre la recherche en largeur optimale

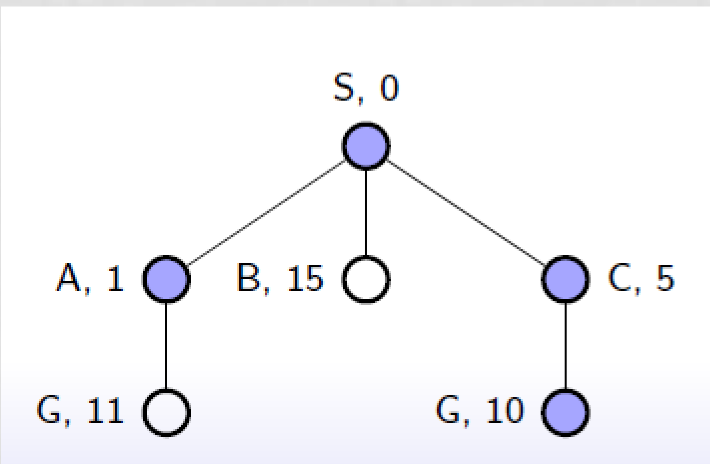
- Développe les nœuds dans l'ordre de leur coût de chemin (noeuds ayant le coût le plus bas)

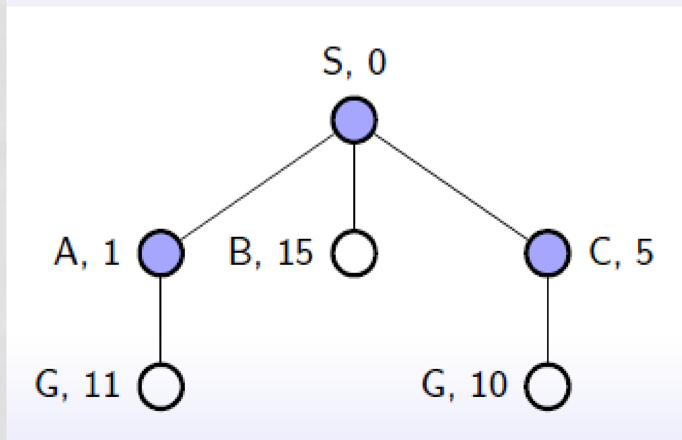

A
1
10
B
S
15
5
G
5
5
C


S, 0


S, 0
A, 1
B, 15
C, 5


S, 0
A, 1
B, 15
C, 5
G, 11


S, 0
A, 1
B, 15
C, 5
G, 11
G, 10


S, 0
A, 1
B, 15
C, 5
G, 11
G, 10

COÛT UNIFORME

- Complète : oui, si le coût de chaque action ≥ ε
- Optimal : oui, parce que les nœuds sont développés en ordre de $g(n)$.
- Complexité en temps : nombre de nœuds avec $g \leq C^*$ où C^* est le coût de la solution optimale.
- Complexité en espace : même que celle en temps

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

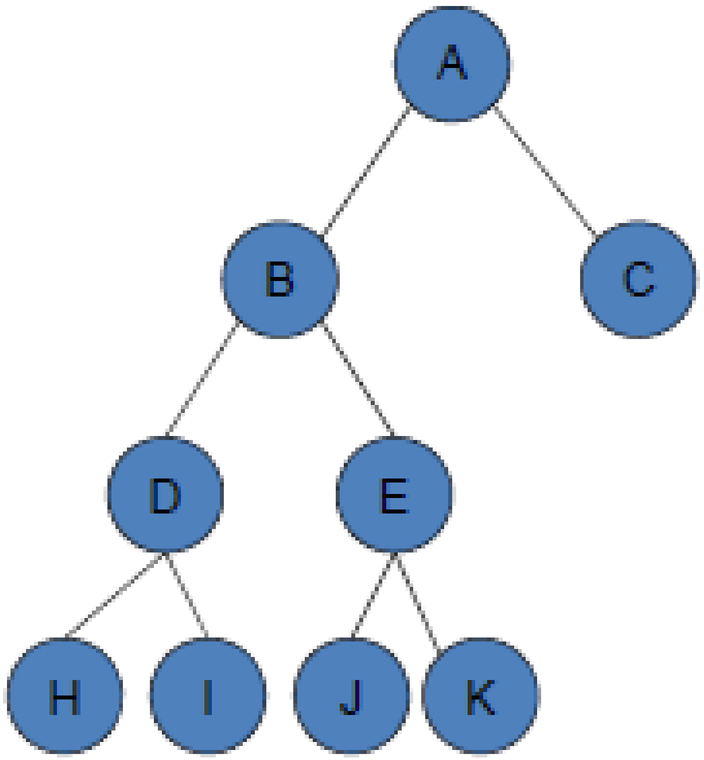
RECHERCHE EN PROFONDEUR

- En théorie des graphes: n parcours en profondeur de G est défini récursivement comme suit:
  - Sélectionner un sommet v_0. A l'étape $k \geq 1$, on choisit un voisin de v_{k-1} qui n'a pas encore été visité. Si un tel voisin n'existe pas, on cherche dans l'ordre, un voisin non sélectionné de v_{k-2}, ..., v_0.

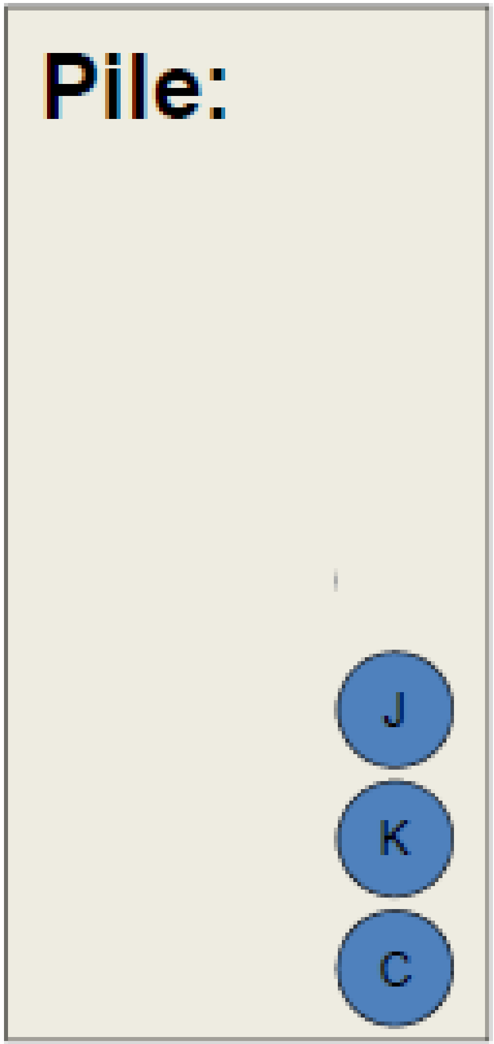
- Principe:
  - Développer les nœuds d'une branche entière avant de parcourir le reste de l'arbre
    - La solution est trouvée : arrêt ou continuer à chercher les autres solutions (backtracking).
    - La solution n'est pas trouvée (état d'échec), poursuivre la recherche (backtracking).
    - Une branche infinie est à explorer : un test d'arrêt à une profondeur maximale sera appliqué.
  - Complexité est liée à l'ordre d'exploration des branches.

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

Exemple profondeur d'abord



Ordre de visite: A – B – D – H – I – E – J – K - C



PROFONDEUR D'ABORD

- Complétude : non, si la profondeur est infinie, s'il y a des cycles. Oui, si on évite les états répétés ou si l'espace de recherche est fini.

- Optimal : Non

- Complexité en temps : $O(b^m)$, très mauvais si m est plus grand que d. m est la profondeur max.

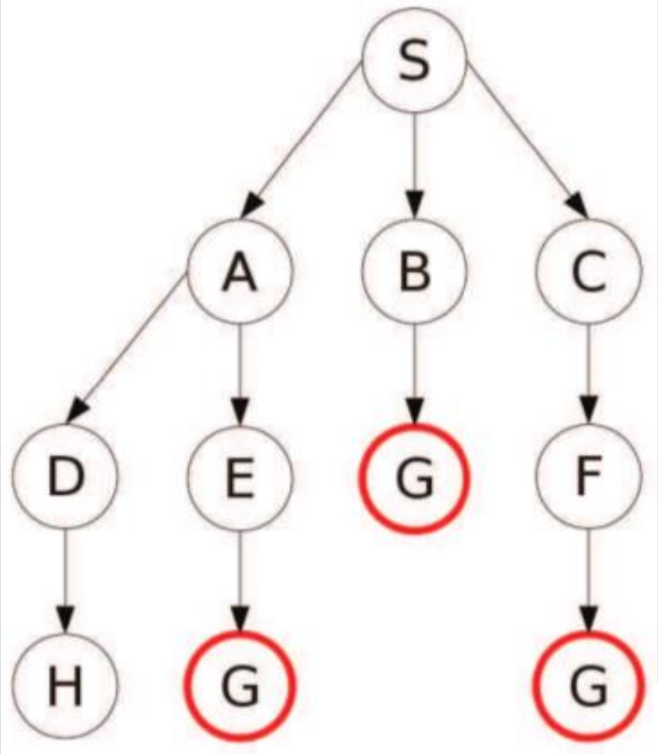
- Complexité en espace : $O(bm)$, linéaire

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024

RECHERCHE PROFONDEUR RÉCURSIVE

```
Fonction RechercheProfondeur(E)
    S <-  sucesseurs(E)
    Pour tout E' dans S :
       Si E' n'est pas sur la pile de visite:
            RechercheProfondeur(E)
```


© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024



Noeuds étudiés (FERMES)	Noeuds à étudier (OUVERTS)
$\emptyset$	$\{S\}$
$\{S\}$	$\{C, B, A\}$
$\{S, C\}$	$\{F, B, A\}$
$\{S, C, F\}$	$\{G, B, A\}$
$\{S, C, F, G\}$	$\{B, A\}$

Noeuds étudiés (FERMES)	Noeuds à étudier (OUVERTS)
$\emptyset$	$\{S\}$
$\{S\}$	$\{C, B, A\}$
$\{S, A\}$	$\{E, D, C, B\}$
$\{S, A, B\}$	$\{G, E, D, C\}$
$\{S, A, B, C\}$	$\{F, G, E, D\}$
$\{S, A, B, C, D\}$	$\{H, F, G, E\}$
$\{S, A, B, C, D, E\}$	$\{H, F, G\}$
$\{S, A, B, C, D, E, G\}$	$\{H, F\}$

RÉFÉRENCES

© IA, Chap.3- Résolution de problèmes _v1, N. Idrissi, 2024