UNIVERSITÉ DE LORRAINE

Loria

Techniques de résolution de problèmes

laurent.bougrain@loria.fr
Equipe Neurorhythms, LORIA

Mécanisme de résolution de problèmes

Les algorithmes de recherche sont un mécanisme de résolution de problèmes général et puissant très utilisé en IA.

- Il se déroule dans un espace appelé espace d'états.
- Il recherche une séquence d'actions (appelée chemin) menant à une solution.
- Dans un état donné, il explore toutes les actions possibles (parfois, seule une partie des actions est envisagée).

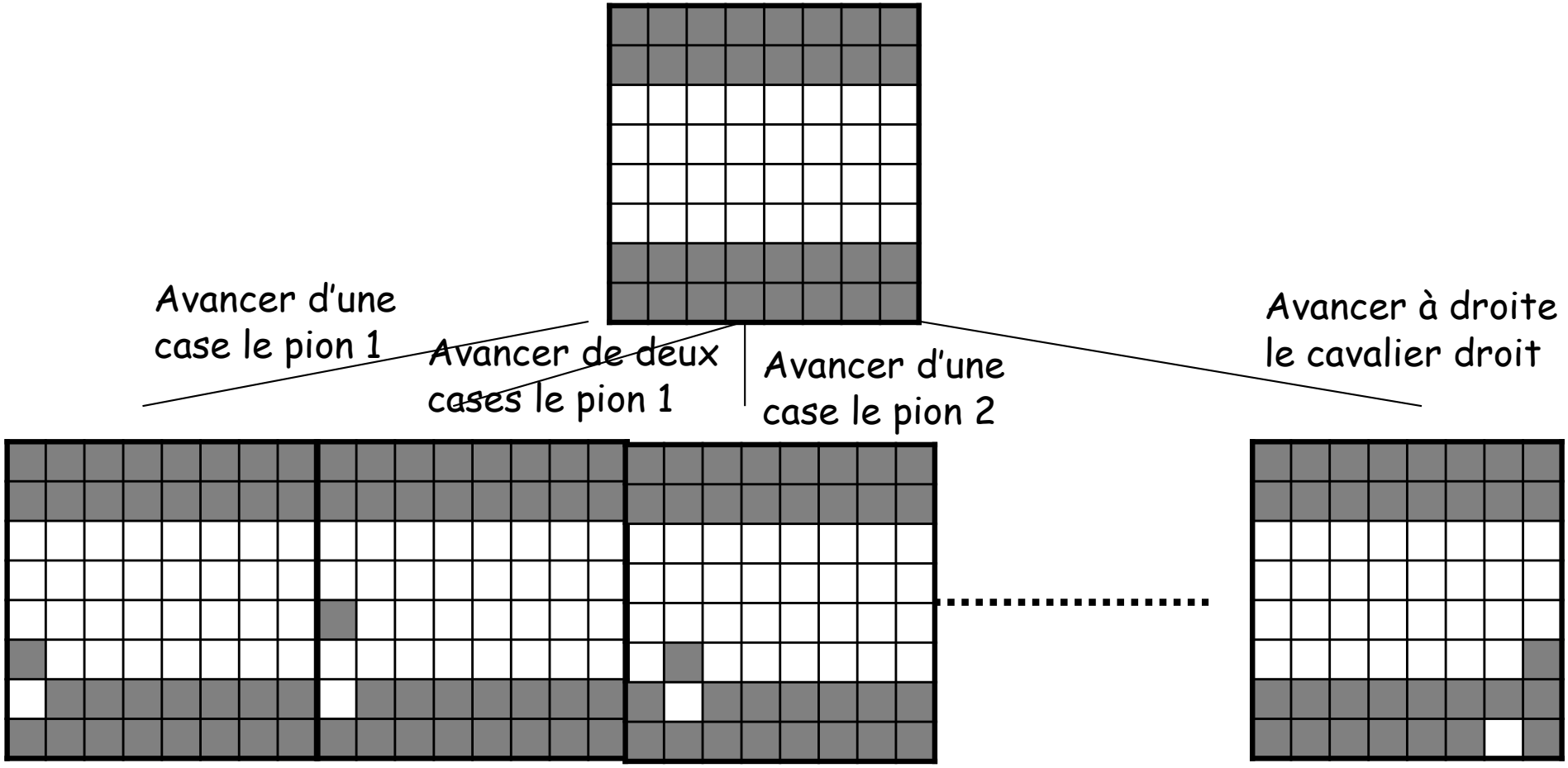
Espace d'états

- Il est décrit par un **état initial** et par un **ensemble d'actions** possibles (opérateurs) ayant un coût donné.
- Un état est une représentation qui permet de connaître la situation actuelle sans ambiguïté et ne tient compte que des informations pertinentes pour résoudre le problème.

Test-but : fonction applicable à chaque état qui détermine si cet état est solution.

Coût-chemin : détermine le coût d'une succession d'actions.
La meilleure solution est la résolution la moins coûteuse.

Exemple : le jeux d'échecs



Le problème des cruches

Enoncé :

Soient deux cruches de 3 et 4 litres, et de l'eau en quantité illimitée.

Comment faire pour obtenir un volume de deux litres ?

Travail à réaliser :

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
4. Établir la liste des actions permettant d'atteindre une solution

Variante :

Soient deux cruches non graduées de 2 et 5 litres, et de l'eau en quantité illimitée.

Comment faire pour obtenir un volume d'un litre dans la cruche de 2 litres, si au départ la cruche de 2 litres est vide et celle de 5 litres est pleine ?

Le problème des cruches (solution)

1. Choisir un formalisme permettant de représenter les états possibles

   Définition par intention :

   Les états possibles du système peuvent être représentés par un couple de valeurs (g,p) où g et p sont respectivement le nombre de litres contenu dans la cruche de 4 litres et dans la cruche de 3 litres ($g \in \{0,1,2,3,4\}$ et $p \in \{0,1,2,3\}$).

   Il y a donc en théorie 5x4=20 états possibles.

2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)

   L'état initial est (0,0).

   La fonction test-but rend vrai si et seulement si g=2 ou p=2

Le problème des cruches (solution)

3.Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences

Action	Condition(s)	Transition	Coût
RemplirG	**g ≠ 4**	**(g,p)→(4,p)**	**4**
RemplirP	**p ≠ 3**	**(g,p)→(g,3)**	**3**
ViderG	**g ≠ 0**	**(g,p)→(0,p)**	**0**
ViderP	**p ≠ 0**	**(g,p)→(g,0)**	**0**
TransvaserGdansP	**g ≠ 0 et p ≠ 3**	**(g,p)→(max(p+g-3 , 0), min(g+p , 3))**	**0**
TransvaserPdansG	**g ≠ 4 et p ≠ 0**	**(g,p)→(min(p+g , 4), max(g+p-4 , 0))**	**0**

Le problème des cruches (solution)

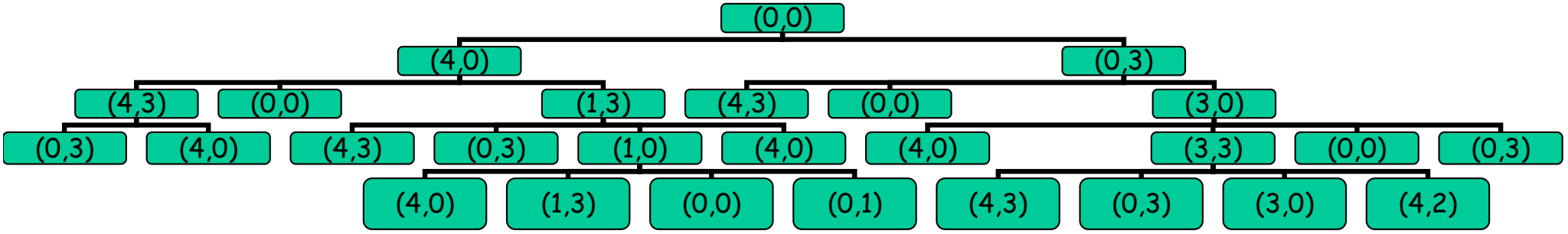
4. Établir la liste des actions permettant d'atteindre une solution

Étape	Action	g	p
		0	0
1	RemplirG	4	0
2	TransvaserGdansP	1	3
3	VideP	1	0
4	TransvaserGdansP	0	1
5	RemplirG	4	1
6	TransvaserGdansP	2	3

ou

Étape	Action	g	p
		0	0
1	RemplirP	0	3
2	TransvaserPdansG	3	0
3	RemplirP	3	3
4	TransvaserPdansG	4	2

Le problème des cruches (Arbre de recherche)



Problèmes classiques

Les cruches
Le taquin
Les tours de Hanoï
Le loup, la chèvre et le chou
Les missionnaires et les cannibales
Le voyageur de commerce
La crypt-arithmétique
Les mots croisés
Les huit reines
Les dominos
La fausse pièce
Le nim
Les jeux d'échecs, de dames, ...
Les blocs

Problèmes réels

Recherche de parcours

- itinéraires automatiques, guidage routier, planification de routes aériennes, routage sur réseaux informatiques, …

Robotique

- assemblage automatique, navigation autonome, …

Planification et ordonnancement

- horaires, organisation de tâches, allocation de ressources, …

Configuration de circuits VLSI

Le problème du loup, de la chèvre et du choux

Enoncé :

Un loup, une chèvre et un choux se trouvent sur la rive gauche d'une rivière.

Comment les faire traverser de l'autre coté à l'aide d'une barque ne contenant que deux places (dont une pour le fermier) sachant que si le loup et la chèvre restent sur la même rive la chèvre sera mangée et que si la chèvre et le choux restent ensemble c'est le choux qui sera mangé ?

Travail à réaliser :

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
4. Établir la liste des actions permettant d'atteindre une solution

Le problème des missionnaires et les cannibales

Enoncé :

Trois missionnaires et trois cannibales sont sur un même coté d'une rivière qu'ils doivent traverser à l'aide d'une barque à deux places nécessitant au moins une personne pour la manipuler. Si sur un coté (et la barque est toujours considérée comme étant sur un des cotés) il y a strictement plus de cannibales que de missionnaires, ces derniers sont mangés.

Comment faire traverser tout le monde sans que les missionnaires de soient mangés ?

Travail à réaliser :

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
4. Établir la liste des actions permettant d'atteindre une solution

Le problème de la traversée du pont

Enoncé :

Quatre personnes doivent traverser un pont en 17 minutes. Chacune d'entre elles marche à une vitesse maximale donnée. Une personne peut traverser le pont en 1 minute, une autre en 2 minutes, une autre en 5 minutes et la dernière en 10 minutes.
Il est impossible de traverser le pont sans torche, et ces quatre personnes n'ont en tout qu'une torche. Le pont ne peut supporter que le poids de 2 personnes.
On cherche à déterminer l'ordre dans lequel ces quatre personnes doivent traverser.

Travail à réaliser :

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
4. Établir la liste des actions permettant d'atteindre une solution

Le problème des tours de Hanoï

Enoncé :

Dans ce solitaire, on dispose d'anneaux de tailles toutes différentes et à tout moment, au dessus de tout anneau, il n'y a que des anneaux plus petits.
Comment amener tous les disques à l'unique position légale où tous les disques se trouvent sur l'axe de gauche à celle où tous se trouvent sur l'axe de droite ?

Travail à réaliser :

1. Choisir un formalisme permettant de représenter les états possibles
2. Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
3. Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
4. Établir la liste des actions permettant d'atteindre une solution

Le problème de l'atelier

Enoncé :

On considère une chaîne de production de voitures :

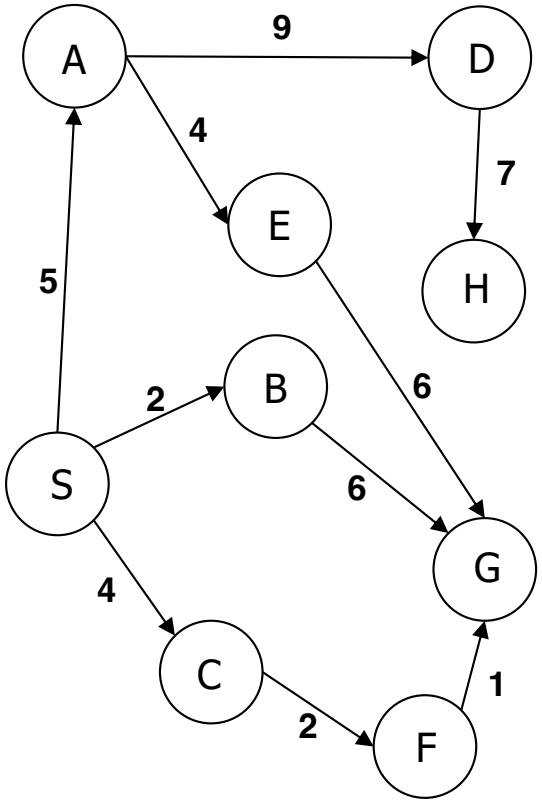
- la chaîne de production est composée de 4 ateliers A, B, C et D.
- chaque voiture doit passer successivement par l'atelier A, puis le B, puis le C et enfin le D.
- chaque atelier a une capacité de production à l'heure, qui dépend du nombre d'opérateurs qui lui sont affectés : l'atelier A peut traiter 5 voitures par opérateur et par heure, l'atelier B, 3, l'atelier C, 2 et l'atelier D, 3.
- il y a 4 opérateurs : l'opérateur 1 travaille de 8 h à 13 h, l'opérateur 2 de 8h à 13h, l'opérateur 3 de 9h à 14h et l'opérateur 4 de 10h à 15h.
- au départ, la chaîne de production est dans un état initial donné par le nombre de produits en attente pour chaque atelier, par exemple: à 8 heures, il y a 30 voitures en attente de l'atelier A, et 0 voiture en attente des autres ateliers.

Travail à réaliser :

Il s'agit alors d'établir l'emploi du temps indiquant heure par heure quels opérateurs sont affectés à chaque station, de telle sorte que le maximum de voitures soient passées par les 4 ateliers à la fin de la journée.

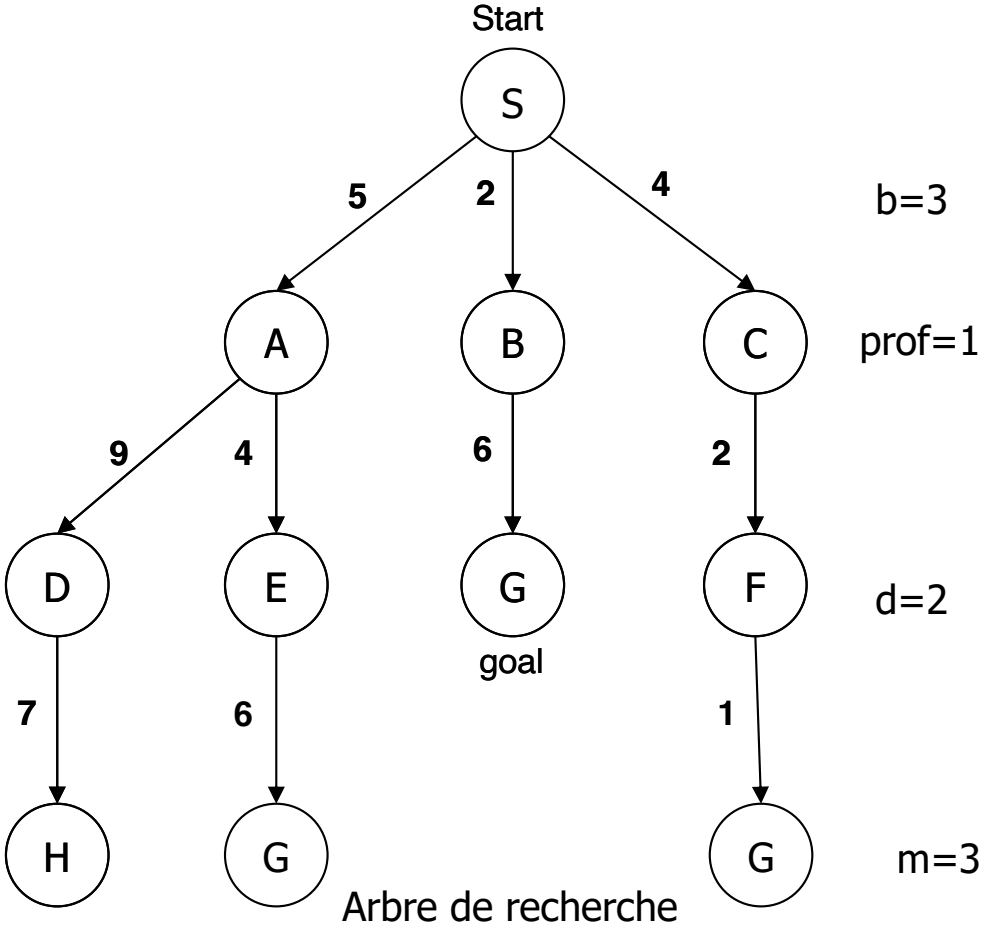
Représentations

Du monde



Espace d'états

De la recherche



Arbre de recherche

b = facteur de branchement maximum de l'arbre de recherche
P = profondeur du nœud
d = profondeur à laquelle se trouve le (meilleur) nœud-solution
m = profondeur maximum de l'espace de recherche (parfois ∞)
L = limite de la profondeur de recherche

Méthodes de recherche

Critères d'évaluation

Méthodes de recherche simple (aussi appelée non informée)

- en largeur
- en coût uniforme = Dijkstra
- en profondeur
- en profondeur limitée
- par approfondissement itératif
- bi-directionnelle

Méthodes de recherche heuristique

- du meilleur d'abord
  - gourmande
  - Algorithmes A, A*, IDA*, SMA*

Critères d'évaluation

Les différentes méthodes de recherche sont évaluées selon les critères suivants:

Complétude (trouve une solution si elle existe)
Optimalité (trouve la meilleure solution)
Complexité en temps
Complexité en espace mémoire

Ces critères vont dépendre de :
b = facteur de branchement maximum de l'arbre de recherche
d = profondeur à laquelle se trouve le meilleur nœud-solution
m = profondeur maximum de l'espace de recherche (parfois ∞)

Idée de base

Exploration simulée de l'espace d'états en générant les états successeurs des états déjà explorés

fonction **recherche-generale (*probleme*, *strategie*)** rend ***solution* ou *echec***
    initialiser l'arbre de recherche par l'*etat initial* du *probleme*
    boucle
        si **il n'y a pas de candidats pour l'expansion** alors rendre ***echec***
        sinon **choisir une *feuille* pour l'expansion en fonction de la *strategie***
        si **le *nœud* contient un *état final*** alors rendre **la *solution***
        sinon **étendre le *nœud* et ajouter les *nœuds fils* dans l'arbre**
    fin

Les méthodes de recherche diffèrent les unes des autres selon l'ordre dans lequel les nœuds sont explorés.

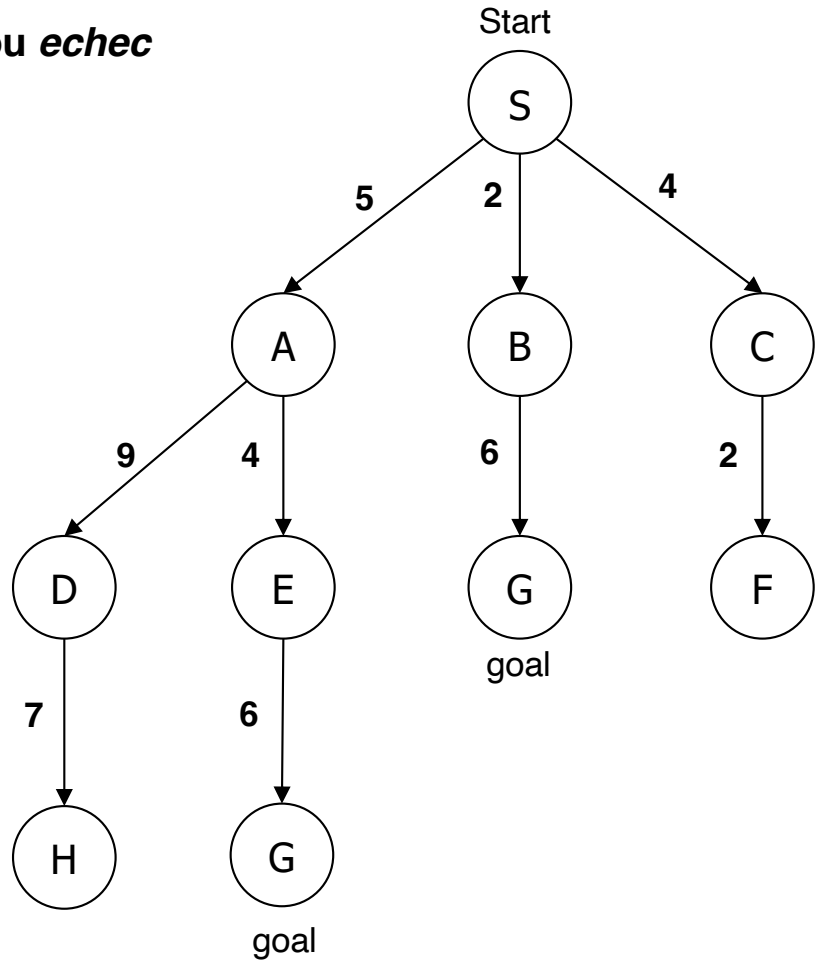
Recherche en largeur d'abord

Stratégie : étendre le nœud le moins profond
Implantation : insertion des successeurs à la fin de la file d'attente

fonction **recherche-largeur (*probleme*)** rend ***solution* ou *echec***
recherche-generale (*probleme*, *insere-a-la-fin*)

Nœuds étudiés (FERMES)	Nœuds à étudier (OUVERTS)
Ø	{S}
{S}	{A,B,C}
{S,A}	{B,C,D,E}
{S,A,B}	{C,D,E,G}
{S,A,B,C}	{D,E,G,F}
{S,A,B,C,D}	{E,G,F,H}
{S,A,B,C,D,E}	{G,F,H,G}

Chemin : S,B,G
Coût : 8
Nœuds testés : 7
Nœuds étendus : 6



Exemple de la complexité en largeur d'abord

Profondeur	Nœud	Temps	Mémoire
0	1	1 ms	100 octets
1	11	11 ms	1.1 Ko
2	111	111 ms	11.1 Ko
4	10^4	11 s	1 Mo
6	10^6	18 minutes	111 Mo
8	10^8	31 heures	11 Go
10	10^{10}	128 jours	1 To
12	10^{12}	35 ans	111 To
14	10^{14}	3500 ans	11 111 To

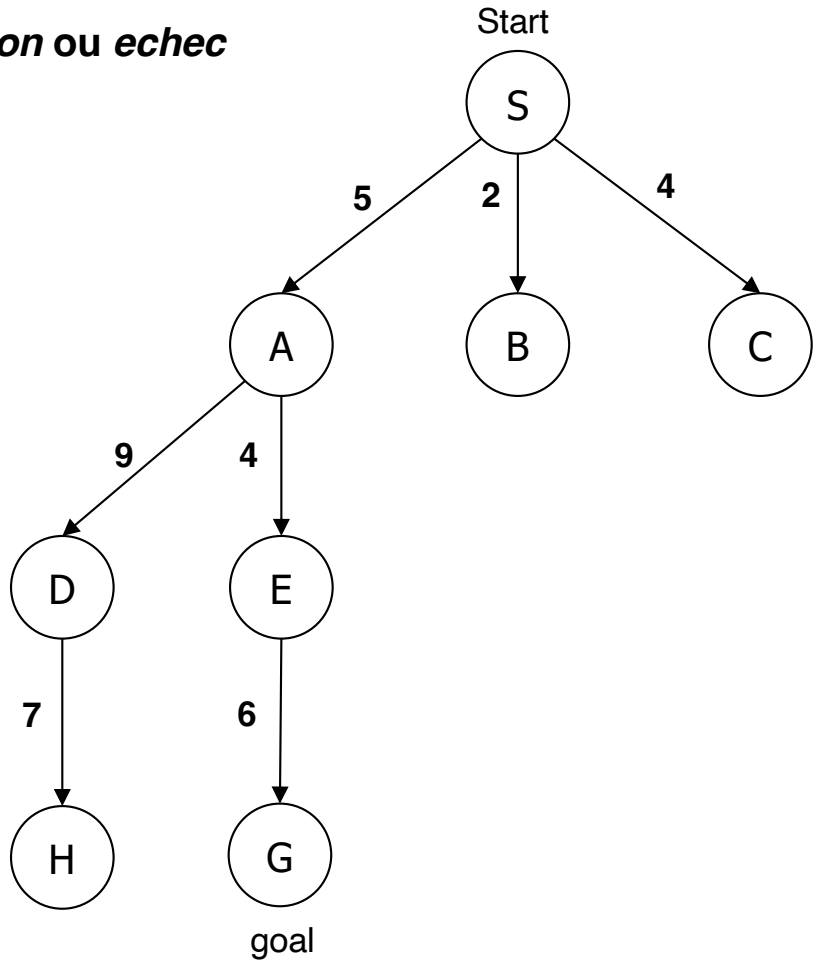
Recherche en profondeur

Stratégie : étendre le nœud le plus profond
Implantation : insertion des successeurs en tête de la file d'attente

fonction **recherche-profondeur (*probleme*)** rend ***solution* ou *echec***
recherche-generale (*probleme*, *insere-en-tete*)

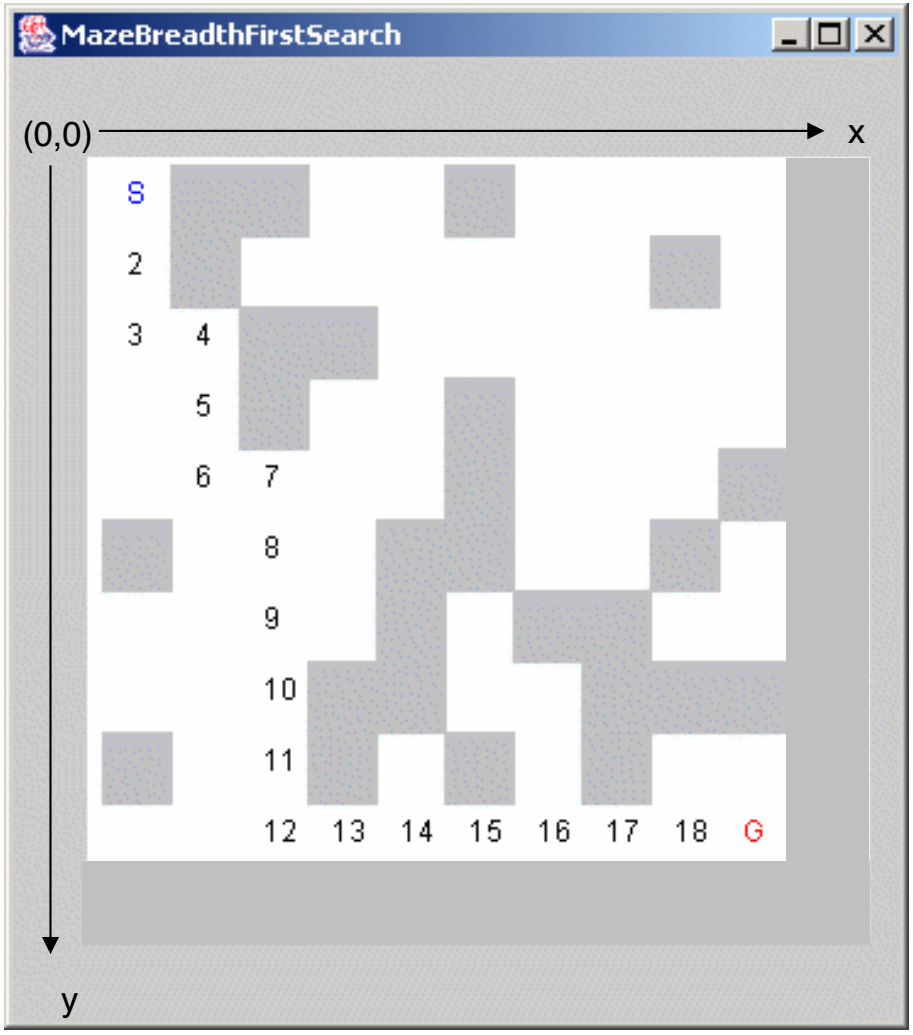
Nœuds étudiés (FERMES)	Nœuds à étudier (OUVERTS)
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G	{B,C}

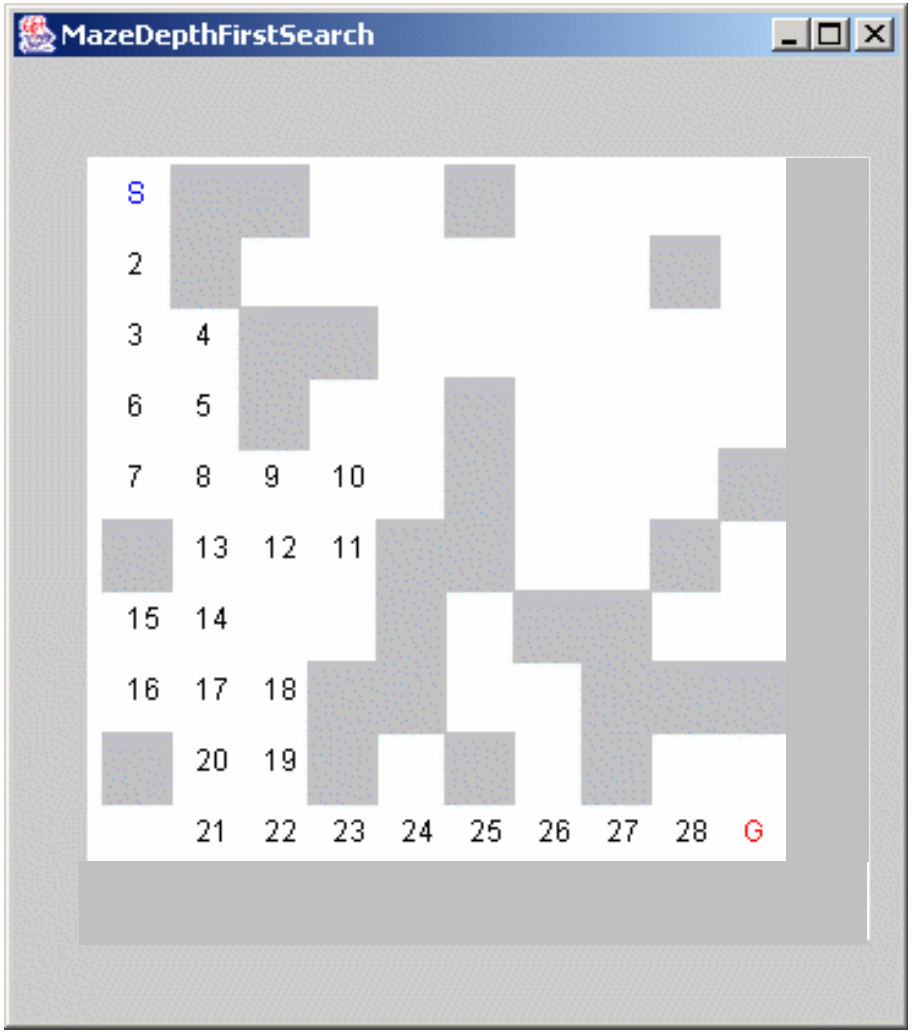
Chemin : S,A,E,G
Coût : 15
Nœuds testés : 6
Nœuds étendus : 5
Il faut un espace de recherche fini et sans cycle



Comparaison recherche en largeur d'abord Vs Profondeur d'abord

Ordre des actions : gauche, droite, haut, bas



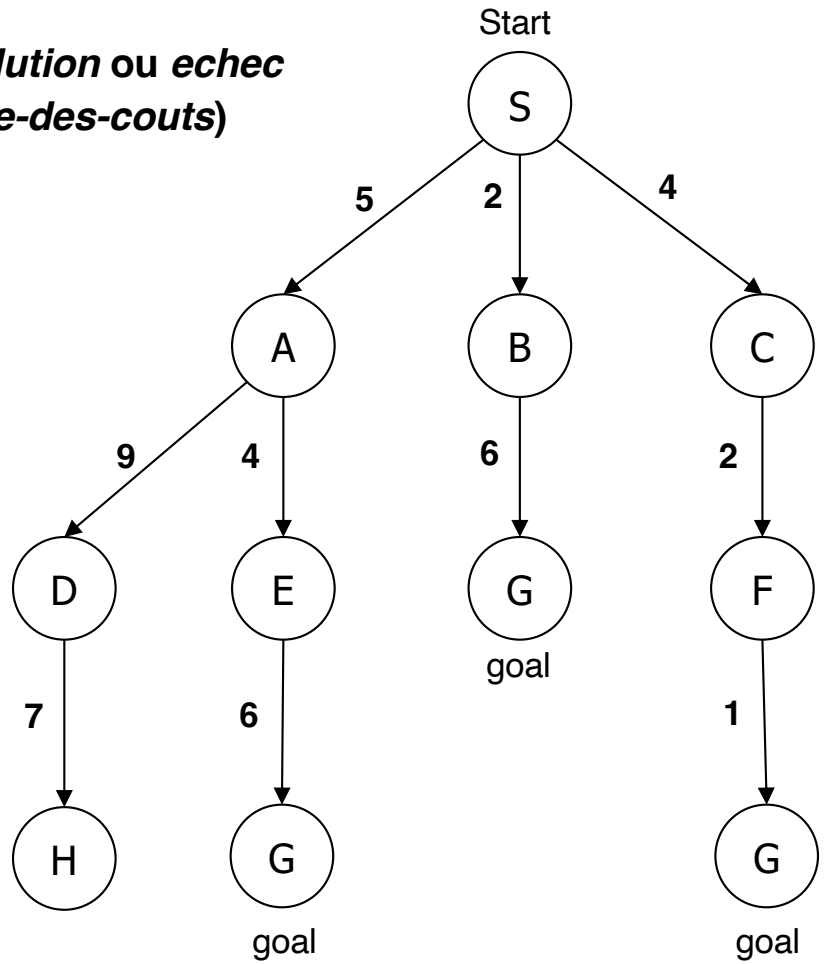


Recherche en coût uniforme (Dijkstra)

Stratégie : étendre le nœud de coût le plus faible
Implantation : insertion des successeurs dans l'ordre croissant des coûts de chemin

fonction **recherche-coût-uniforme (*probleme*)** rend ***solution* ou *echec***
recherche-generale (*probleme*, *insere-dans-ordre-des-couts*)

Nœuds étudiés (FERMES)	Nœuds à étudier (OUVERTS)
	{S:0}
{S:0}	{B:2,C:4,A:5}
{S:0,B:2}	{C:4,A:5,G:8}
{S:0,B:2,C:4}	{A:5,F:6,G:8}
{S:0,B:2,C:4,A:5}	{F:6,G:8,E:9,D:14}
{S:0,B:2,C:4,A:5,F:6}	{G:7,G:8,E:9,D:14}



Chemin : S,C,F,G
Coût : 7
Nœuds testés : 6
Nœuds étendus : 5
Le coût d'un chemin ne doit jamais décroître

Comparaison

Critères	Largeur	Dijkstra	Profondeur	Profondeur limitée	Profondeur itérative	Bi-directions
Complétude	Oui	Oui	Non sauf esp. fini acyclique	Oui, si $L \geq d$	Oui	Oui
Optimalité	Oui, si coûts identiques	Oui	Non	Non	Oui, si coûts identiques	Oui
Temps	b^d	b^d	b^m	b^L	b^d	$b^{d/2}$
Mémoire	b^d	b^d	bm	bL	bd	$b^{d/2}$

b = facteur de branchement maximum de l'arbre de recherche
d = profondeur à laquelle se trouve le (meilleur) nœud-solution
m = profondeur maximum de l'espace de recherche (parfois ∞)
L = limite de la profondeur de recherche

II- Recherche heuristique

Les algorithmes de recherche simple n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de nœuds-solution pour optimiser la recherche

La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles

Un algorithme de recherche heuristique utilise l'information disponible pour rendre le processus de recherche plus efficace.

Une information heuristique est une règle ou une méthode qui presque toujours améliore un processus de décision

Fonction heuristique

Une fonction heuristique h : E→ℜ (E, espace d'états) fait correspondre à un état s∈E un nombre h(s)∈ℜ qui est (généralement) une estimation du rapport coût/bénéfice qu'il y a à étendre le chemin courant en passant par s.

h(s) ≥ 0, pour tout état s

h(s) = 0, implique que s est un état-but

h(s) = ∞, implique que s est un état ne permettant pas d'atteindre un but

Une heuristique est spécifique à un problème. Elle utilise les connaissances du domaine afin d'orienter la recherche pour la rendre plus efficace.

Qualités des heuristiques

Facteur de branchement effectif

soit N = nombre total d'états produits pour obtenir la solution

soit d = profondeur à laquelle la solution a été trouvée

alors b* est le facteur de branchement effectif d'un arbre fictif parfaitement équilibré tel que :

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Une bonne fonction heuristique aura une valeur de b* proche de 1

Dominance

Soient deux fonctions heuristiques h1(n) et h2(n)

Si $h2(n) \geq h1(n), \forall n$ alors on dit que h2 domine h1 et produira une recherche plus efficace

Recherche du meilleur d'abord

Stratégie : étendre le nœud le plus prometteur au regard d'une fonction d'évaluation du coût total f(n) qui tient compte de l'heuristique.

Implantation : insérer les successeurs en ordre croissant de leur valeur f(n).

Combinaison entre la recherche en profondeur et en largeur

- **P: solution trouvée sans avoir à calculer tous les nœuds**
- **L: ne risque pas de rester dans un cul-de-sac**

Cas particuliers :

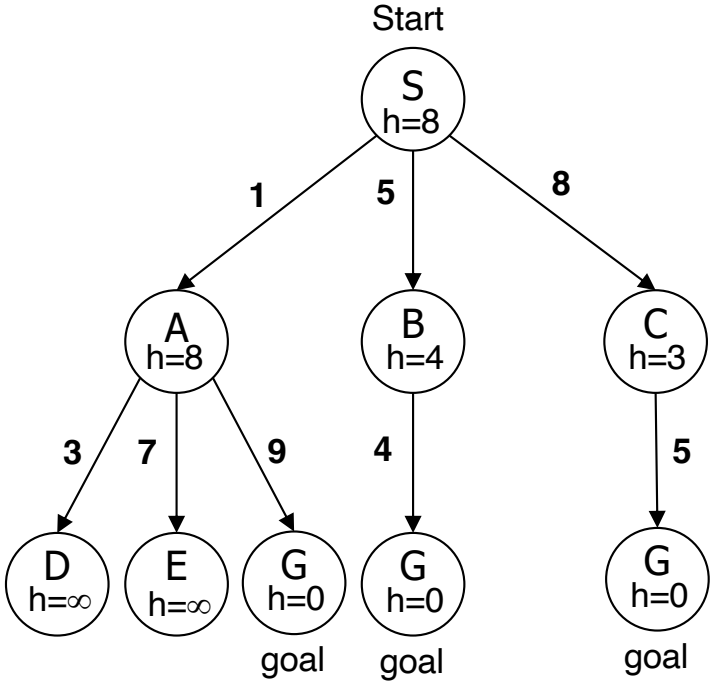
- **recherche gourmande**
- **algorithmes A et A***

Recherche gourmande

Stratégie : minimiser le coût estimé pour atteindre le but. f(s) = h(s)
Implantation : toujours enlever le plus grand morceau du coût restant pour atteindre le but

Nœuds étudiés (FERMES)	Nœuds à étudier (OUVERTS)
	{S:8}
S	{C:3,B:4,A:8}
C	{G:0,B:4,A:8}
G	{B:4,A:8}

Chemin : S,C,G
Coût : 13
Nœuds testés : 3
Nœuds étendus : 2



fonctions heuristiques classiques
- **distance à vol d'oiseau**
- **distance de Manhattan**

Relativement efficace bien que pas toujours optimal

Les algorithmes A et A*

La recherche gourmande minimise le coût estimé h(n) du nœud n au but réduisant ainsi considérablement le coût de la recherche, mais il n'est pas optimal et pas complet

La recherche en coût uniforme minimise le coût g(n) depuis l'état initial au nœud n, il est optimal et complet, mais pas très efficace

Stratégie : Combiner la recherche gourmande et la recherche en coût uniforme, et rechercher le coût total f(n)=g(n)+h(n) du chemin passant par le nœud n

Définition: Une fonction heuristique est admissible si elle ne surestime jamais le coût réel i.e. si $\forall$n, h(n) ≤ h*(n) avec h*(n) = coût réel depuis n au but.

Théorème : Si A utilise une fonction heuristique admissible alors A est optimal.

Dans ce cas, l'algorithme est nommé A*.

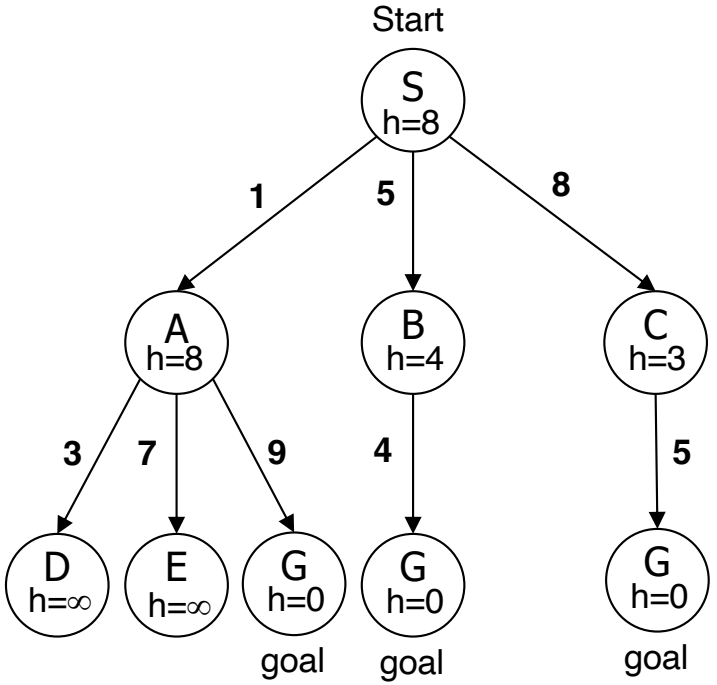
A* : exemple

Stratégie : étendre le nœud dont la valeur f(s) = g(s) + h(s) est la plus faible

Implantation : insertion des successeurs dans la liste dans l'ordre croissant des valeurs de f(n)

fonction **A (*probleme*)** rend ***solution*** **ou *echec***
recherche-meilleur (*probleme*, g+h)

Nœuds étudiés (FERMES)	Nœuds à étudier (N,f,h) (OUVERTS)
	{S:8:8}
S	{A:9:8, B:9:4, C:11:3}
A	{B:9:4, G:10:0 C:11:3, D:∞: ∞, E:∞: ∞}
B	{G:9:0, C:11:3, D:∞: ∞, E:∞: ∞}
G	{C:11:3, D:∞: ∞, E:∞: ∞}



Chemin : S,B,G
Coût : 9
Nœuds testés : 4
Nœuds étendus : 3

efficace et optimal

Algorithme A (et A*)

Soient $\mathbf{s_0}$ l'état initial,
OUVERTS la liste des noeuds à étudier,
FERMES la liste des noeuds étudiés,
tete(OUVERTS) la fonction qui renvoie le premier élément de la liste OUVERTS

OUVERTS $\leftarrow \{s_0\}$, FERMES $\leftarrow \varnothing$, pere(s_0) $\leftarrow$ -, g (s_0) $\leftarrow$ 0, s $\leftarrow s_0$
Tant que OUVERTS $\neq \varnothing$ et tete(OUVERTS) non terminal **Faire**
  supprimer s de OUVERTS et le mettre dans FERMES
    Pour tout noeud s' successeur de s **Faire**
      Si s' $\notin$(OUVERTS $\cup$ FERMES) **ou** g(s') > g(s) + cout(s,s') **Alors**
        g(s') $\leftarrow$ g(s) + cout(s,s')
        f(s') $\leftarrow$ g(s') + h(s')
        pere(s') $\leftarrow$ s
        ranger s' dans OUVERTS dans l'ordre f croissant
        Si s' $\in$ FERMES **Alors**
          FERMES $\leftarrow$ FERMES \ {s'}
    Fin pour
  Si OUVERTS = $\varnothing$ **Alors**
    s $\leftarrow$ tete(OUVERTS)
Fin tant que
Si OUVERTS = $\varnothing$ **Alors** le problème n'admet pas de solution
Sinon le chemin(s) est solution (avec chemin(s) : chemin(pere(s)), s)

Variantes de A*

Les problèmes réels sont très complexes

L'espace de recherche devient très grand

Même les méthodes de recherche heuristique deviennent inefficaces

A* connaît alors des problèmes de place mémoire

2 variantes de A* économes en place mémoire :

- **IDA* = A* avec approfondissement itératif (IDS + A*)**

  On augmente progressivement la valeur limite de f

  Ex : les nœuds n où f(n) > 10 ne sont pas pris en considération

  les nœuds n où f(n) > 20 ne sont pas pris en considération

  …..

- **SMA* = A* avec gestion de la mémoire**

  La taille de file d'attente est limitée pour limiter l'espace mémoire

  Le père du nœud supprimé mémorise la valeur du meilleur descendant oublié

En résumé

La recherche du meilleur d'abord est une méthode de recherche générale où les nœuds de coût minimum sont explorés en premier.

La recherche gourmande utilise une estimation du coût minimum au but pour réduire le temps de recherche mais il n'est ni complet ni optimal.

La recherche A* combine la recherche en coût uniforme et la recherche gourmande : f(n) = g(n) + h(n). L'heuristique ne doit jamais surestimer le coût réel.

- A* est complet, optimal et très rapide mais demande beaucoup de mémoire.
- IDA* et SMA* réduisent l'espace mémoire utilisé
- La qualité de l'heuristique influence la durée de la recherche