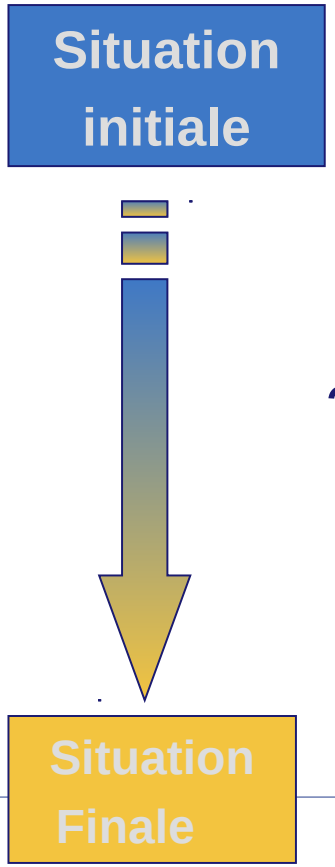
Chapitre 2

Résolution de problème en IA
Par recherche

INSTITUT SUPERIEUR INFORMATIQUE ISI
المعهد العالي للاعلامية

Introduction

- Résoudre un pb c'est chercher un chemin qui permet d'aller d'une situation initiale à une situation finale (but)



? Trouver ce chemin

Introduction

Pour résoudre un problème il arrive qu'on puisse le décomposer en sous-problèmes puis décomposer ceux-ci, etc.,

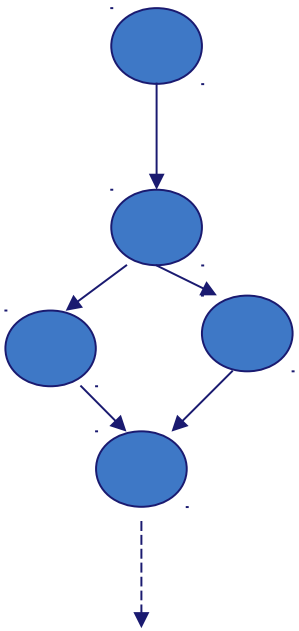
→jusqu'à n'avoir plus que des problèmes dont la solution est considérée comme immédiatement accessible sans qu'il soit nécessaire de les décomposer à leur tour.

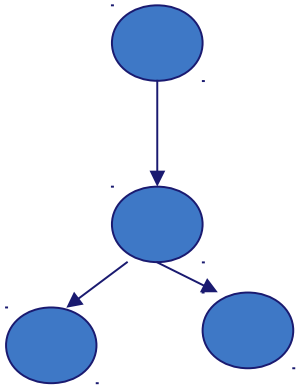
- L'ensemble des décompositions possibles peut être représenté par un " graphe des sous-problèmes ".

→La résolution d'un problème est alors ramenée à la recherche d'un certain sous-graphe du graphe des sous-problèmes.

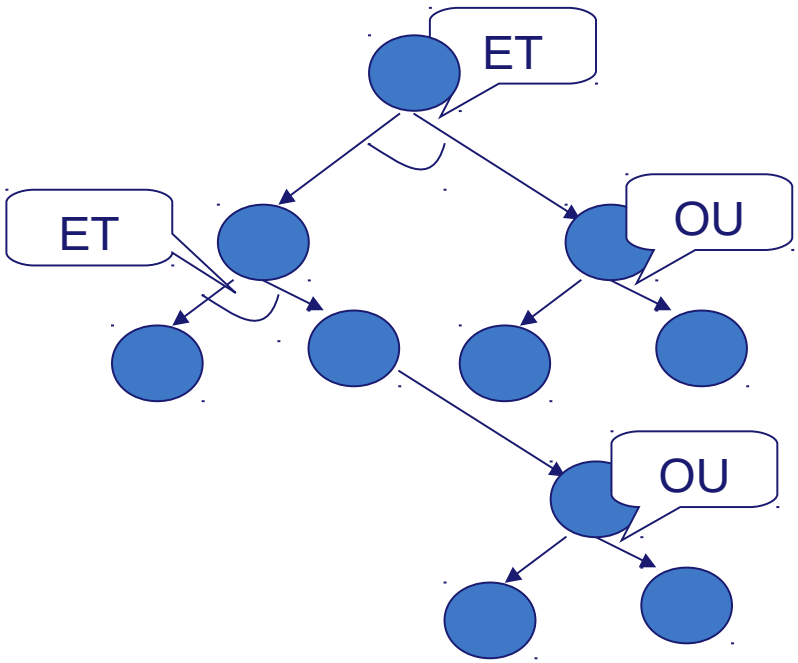
Introduction

Un graphe sans cycle → arbre





Arbre ET/OU



Certains sommets marquent une conjonction de sous problèmes dont la résolution implique celle du problème décomposé.

D'autres sommets marquent une disjonction de décompositions possibles.

Exemple

- un état est une configuration du tableau 4x4

Etat initial

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

On distingue:

- **L'état initial**
- Un ou plusieurs **états finaux**

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

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

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

Représenter le problème

Il faut définir:

- Les états du problème (abstrait) = ensemble d'états réels
- L'objectif à atteindre: solution (abstraite) = ensemble de chemins-solutions dans le monde réel
- Les opérateurs de transformations (abstrait) = combinaison d'actions réelles (représentation par graphe)

Le monde réel est excessivement complexe

l'espace d'états doit être une abstraction de la réalité

Types de problèmes

- déterministe, accessible → problème à état unique
  - état exact connu
  - effets des actions connus
- déterministe, inaccessible → problème à états multiples
  - un ensemble parmi plusieurs ensembles d'états
  - effets des actions connus
- non déterministe, inaccessible → problème contingent
  (perception limitée, algorithmes plus complexes)
  - besoin de percevoir durant l'exécution
  - solution a une structure d'arbre
  - souvent mélange entre recherche et exécution
  - effet conditionnel des actions
- espace d'états inconnu → problème d'exploration ("online")
  - exécution révèle les états
  - besoin d'expérimenter pour trouver la solution
  - le plus difficile

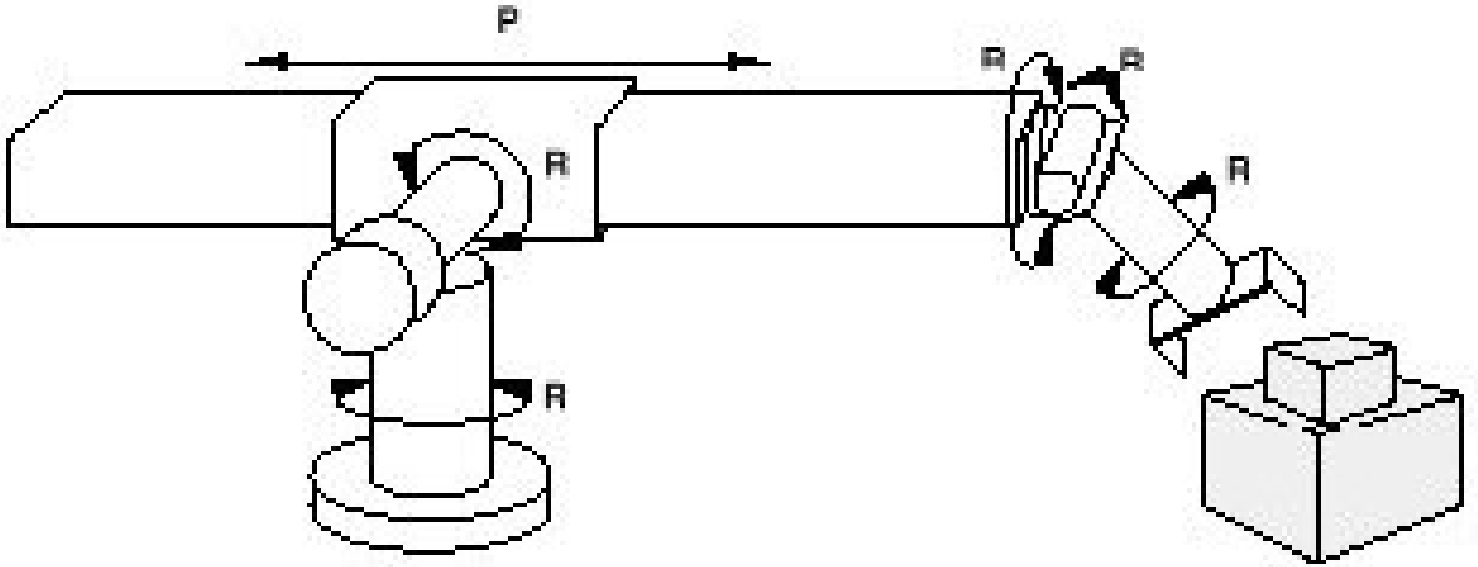
Formulation d'un problème à état unique

Un problème est défini par 4 éléments:

- **état initial**
- **opérateurs** (ou fonction successeur S(x))
- **Test-but**: fonction applicable à un état qui détermine si c'est l'état solution.
- **Coût-chemin**: permet de déterminer quel est le meilleur chemin menant à la solution si plusieurs chemins existent.

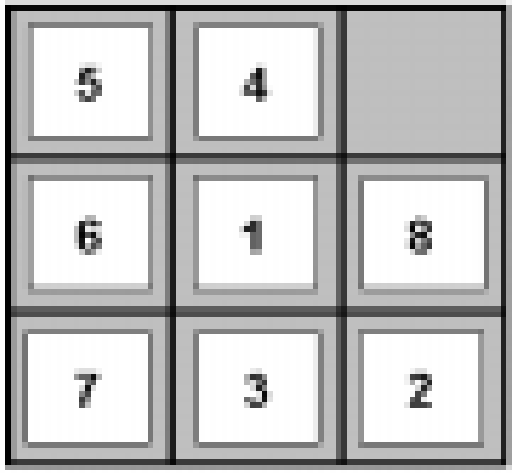
une **solution** est une séquence d'opérateurs menant de l'état initial à l'état final (solution)

Exemple: assemblage automatique



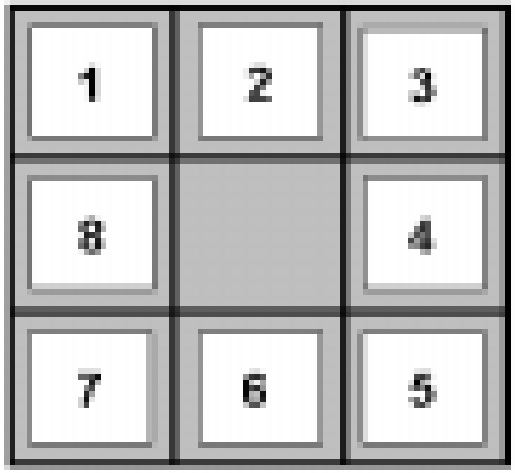
- état initial: coordonnées des articulations du robot et pièces à assembler
- opérateurs: mouvements continus du bras robotique
- test-but: assemblage terminé, robot en position de repos
- coût-chemin: temps d'exécution

Exemple : jeu de taquin (puzzle)



Configuration initiale

Etat initial



Configuration finale

Etat final

- état initial: positions des 8 plaquettes dans une des 9 cases
- opérateurs: déplacer la case vide
- test-but: état courant = état final
- coût-chemin: chaque déplacement coûte 1,
  coût total = nombre de déplacements

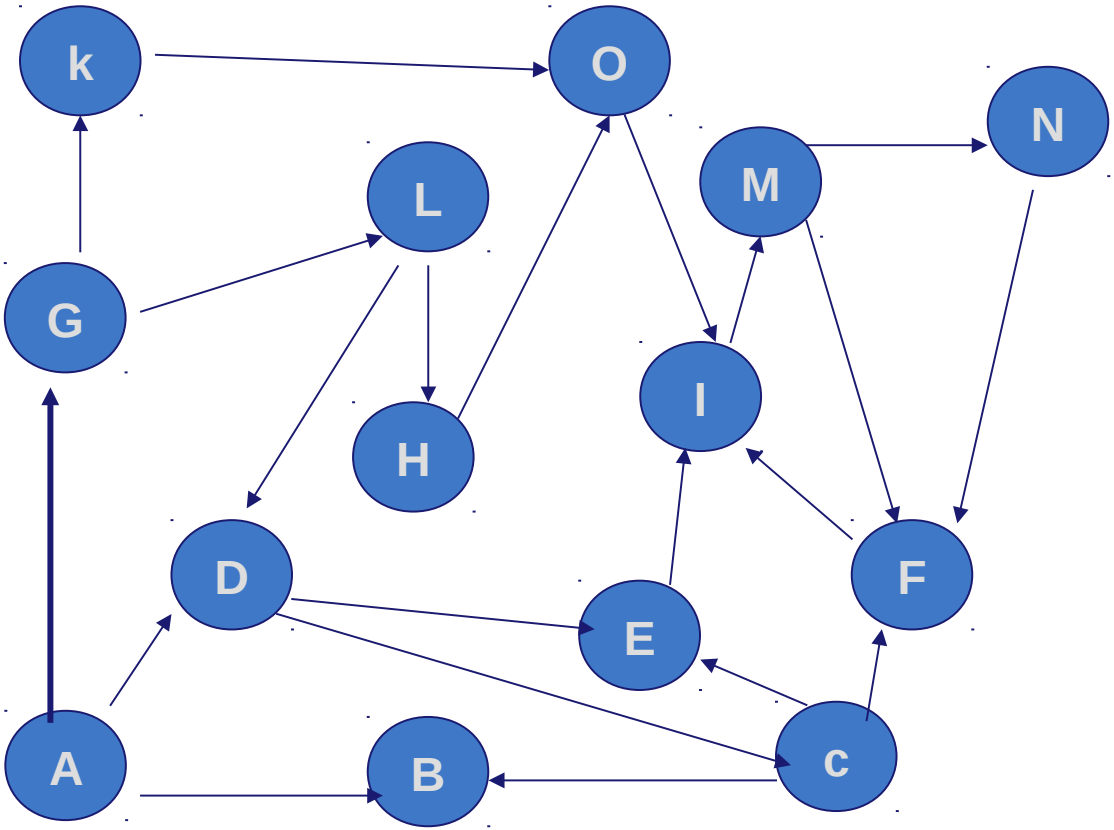
Opérateurs du jeu taquin

- Un opérateur transforme un état en un autre état.
- Il existe quatre opérateurs pour le taquin:
  - déplacer la case vide en haut
  - déplacer la case vide en bas
  - déplacer la case vide à gauche
  - déplacer la case vide à droite

Représentation par graphes d'états

L'application des opérateurs sur les états en démarrant de l'état initial conduit à la construction d'une **arborescence**

Problème général de recherche



Graphe d'état général

?

Représentation par un arbre

États vs. noeuds

• Un état est une représentation d'une configuration physique

• un noeud est un élément d'une structure de donnée constitutive d'un arbre de recherche; il possède les informations de:
– parent, enfants, profondeur, coût de chemin g(x)

Méthodes de recherche

Stratégies d'exploration

- Méthodes de recherche « aveugles » = Explorations non informées
  - recherche en largeur
  - recherche en profondeur
  - recherche en profondeur limitée
  - recherche par approfondissement itératif
- Méthodes de recherche heuristiques = Explorations informées

Critères d'évaluation

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

- **Complétude**: est-ce que la méthode garantit de trouver une solution si elle existe?
- **Complexité en temps**: combien de temps faut-il pour trouver la solution?
- **Complexité en espace**: quel espace mémoire faut-il pour effectuer la recherche?
- **Optimalité**: est-ce que la méthode trouve la meilleure solution s'il en existe plusieurs?

  - Les complexités en temps et en espace sont mesurée en fonction de:
  - **b** = facteur de branchement maximum de l'arbre de recherche
  - **d** = profondeur à laquelle se trouve le (meilleur) noeud-solution
  - **m** = profondeur maximum de l'espace de recherche (espace d'états ou arbre de recherche) - peut être infini

Exercice 1: problème du fermier

- quartes acteurs le fermier(f), le loup(l), la chèvre (c) et le chou (C) se trouvent sur la rive gauche d'une rivière.
- On considère:
  - Un bateau qui peut transporter le fermier seul ou avec un des trois acteurs restants de gauche à droite
  - Un bateau qui peut transporter le fermier seul ou avec un des trois acteurs restants de droite à gauche
  - Le loup peut manger la chèvre sans présence du fermier
  - La chèvre peut manger le chou sans présence du fermier

Pb?

Comment faire pour passer les 4 acteurs à l'autre rive

Algorithme de recherche

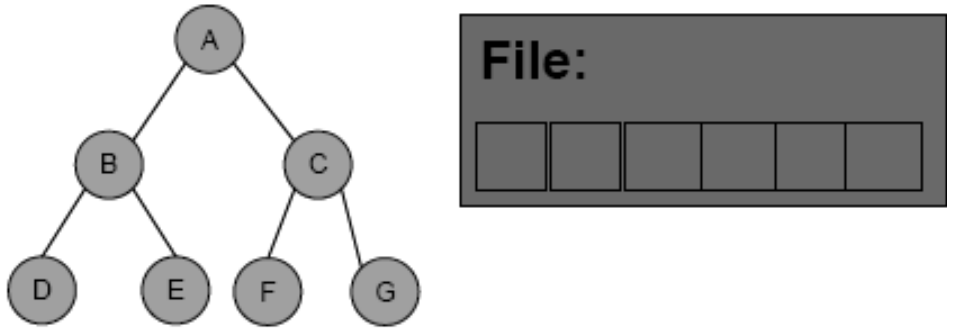
Largeur d’abord
(breadth-first-search)

Principe de la recherche en largeur

- L’expansion des noeuds les moins récemment engendrés s’effectue en premier
- L’arborescence est construite niveau après niveau et donc de manière horizontale

- **Stratégie**: étend le noeud le moins profond
- **Implémentation**:
- insertion des successeurs à la fin de la file d'attente

Exemple largeur d’abord



Ordre de visite: A – B – C – D – E – F - G

Algorithme

1-Insérer le noeud initial s dans une liste appelée OPEN

2-Si OPEN est vide alors échec sinon continuer

3-Retirer le premier noeud de OPEN et l'insérer dans une liste appelée CLOSED. Soit n ce noeud.

4-S'il n'existe pas de successeur alors aller à 2. Engendrer les successeurs de n et les insérer à la queue de OPEN. Créer un chaînage de ces noeuds vers n

5-Si parmi les successeurs, il existe un état final alors succès: la solution est obtenue en suivant le chaînage arrière de ce noeud vers la racine, sinon aller à 2

Propriétés de la recherche en largeur

- **Complétude** Oui (si b est fini)
- **Complexité en temps**

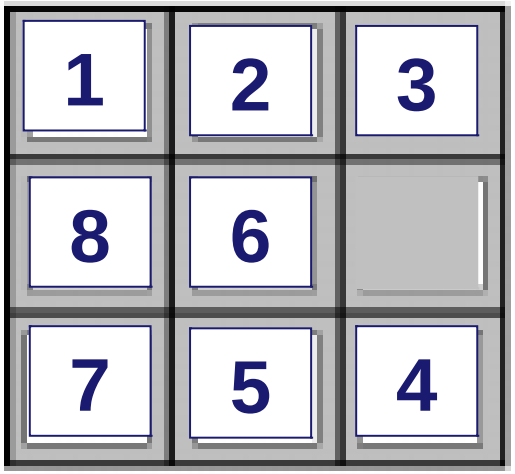
$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (exponentiel en d)

- **Complexité en espace** $O(b^d)$ (il faut garder chaque noeud en mémoire)
- **Optimalité** Oui (si coût = 1 par étape)

en général pas optimal

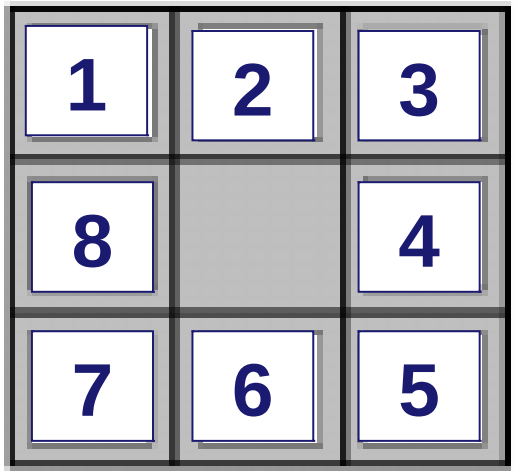
Exercice 2 : taquin 3x3

- Appliquer la recherche en largeur d'abord sur la donnée suivante:



Configuration initiale

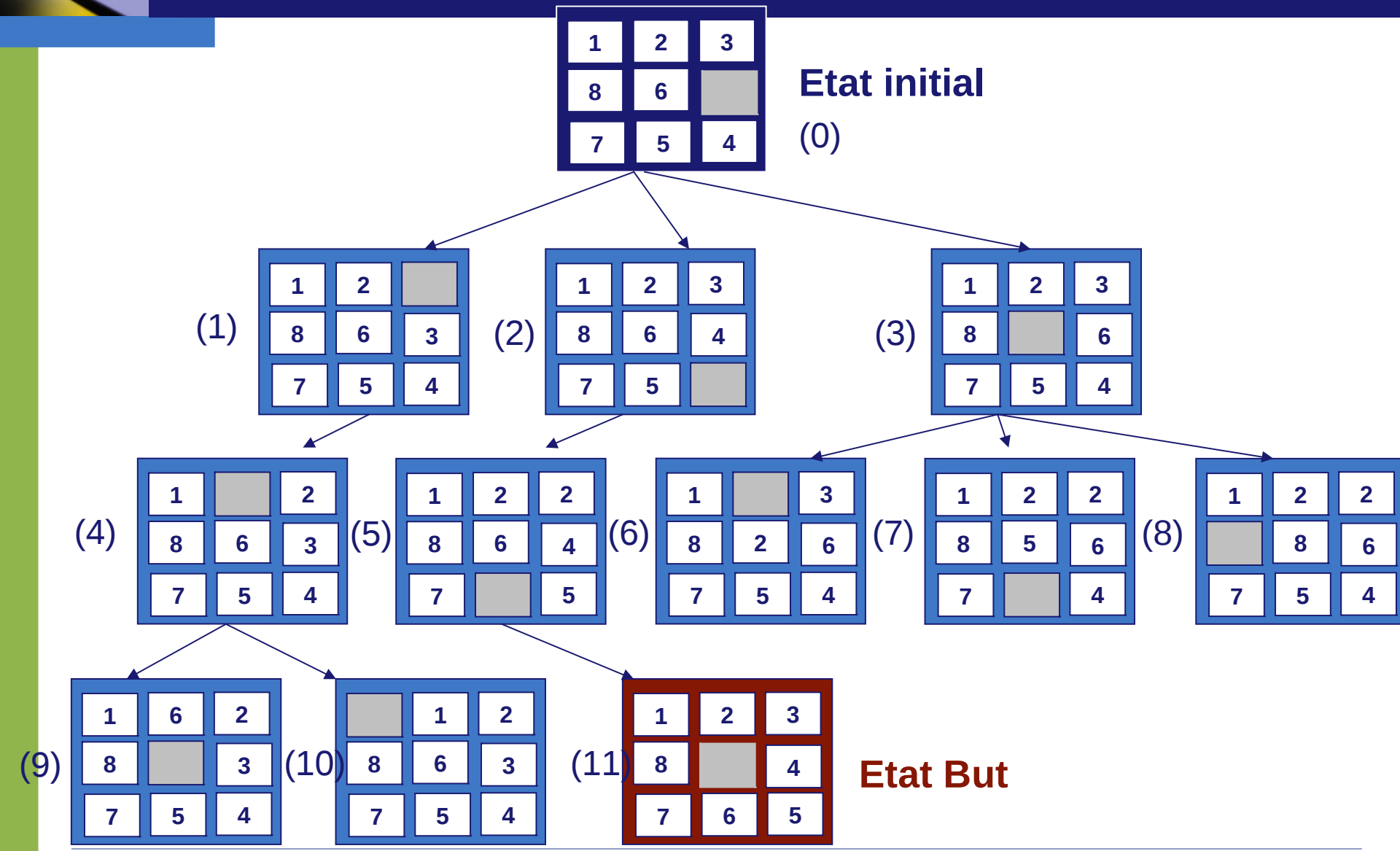
Etat initial



Configuration finale

Etat final

Solution Exercice 2



( ): numéro donnant l'ordre de développement

Algorithme de recherche

profondeur d’abord
(depth-first-search)

Principe de recherche en profondeur

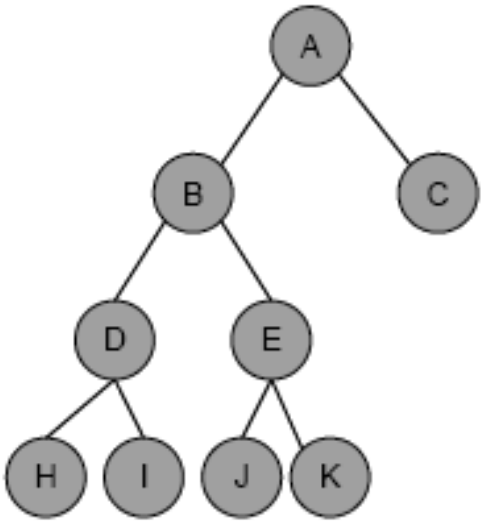
- **Stratégie**: étend le noeud le plus profond
- **Implémentation**: insertion des successeurs en tête de la file d'attente

Attention aux cycles infinis !

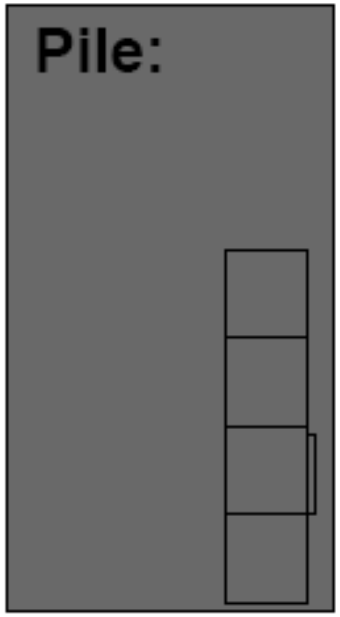
Il faut un espace de recherche fini et sans cycle,

nécessité d'éliminer les nœuds déjà rencontrés.

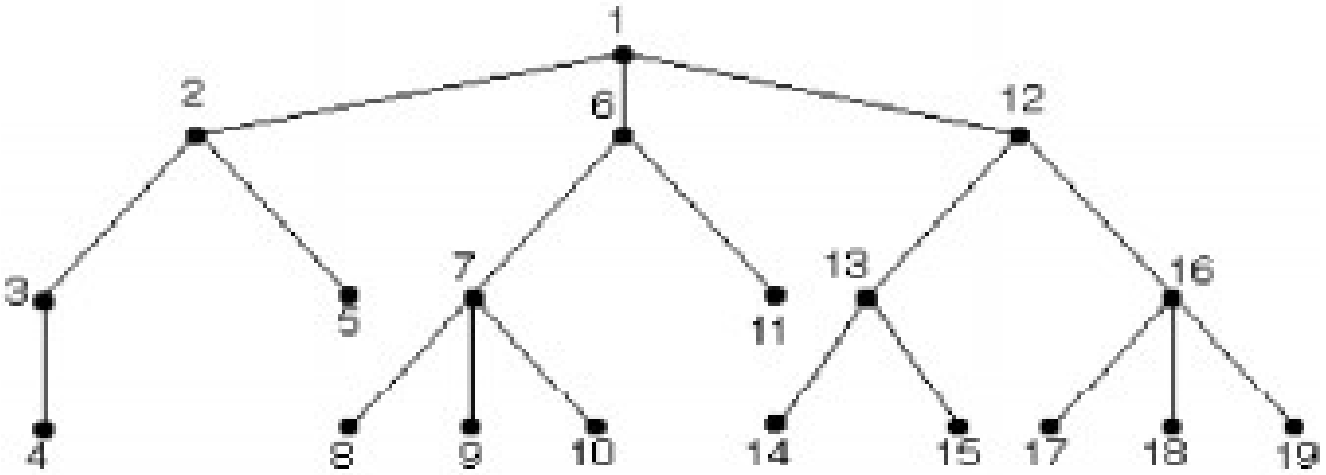
Exemple profondeur d'abord



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



Algorithme de recherche en profondeur



Les nœuds sont numérotés dans l'ordre de leur exploration

Propriétés de la recherche en profondeur

- **Complétude** : Non

  échoue dans les espaces infinis ou avec cycle

  → complet dans les espaces finis acycliques
- **Complexité en temps** :

  $O(b^m)$ = terrible si m est beaucoup plus grand que d
- **Complexité en espace:** O(b * m) linéaire!
- **Optimalité** : Non
- **discussion**: besoins modestes en espace

– pour b = 10, d = 12 et 100 octets/noeud:

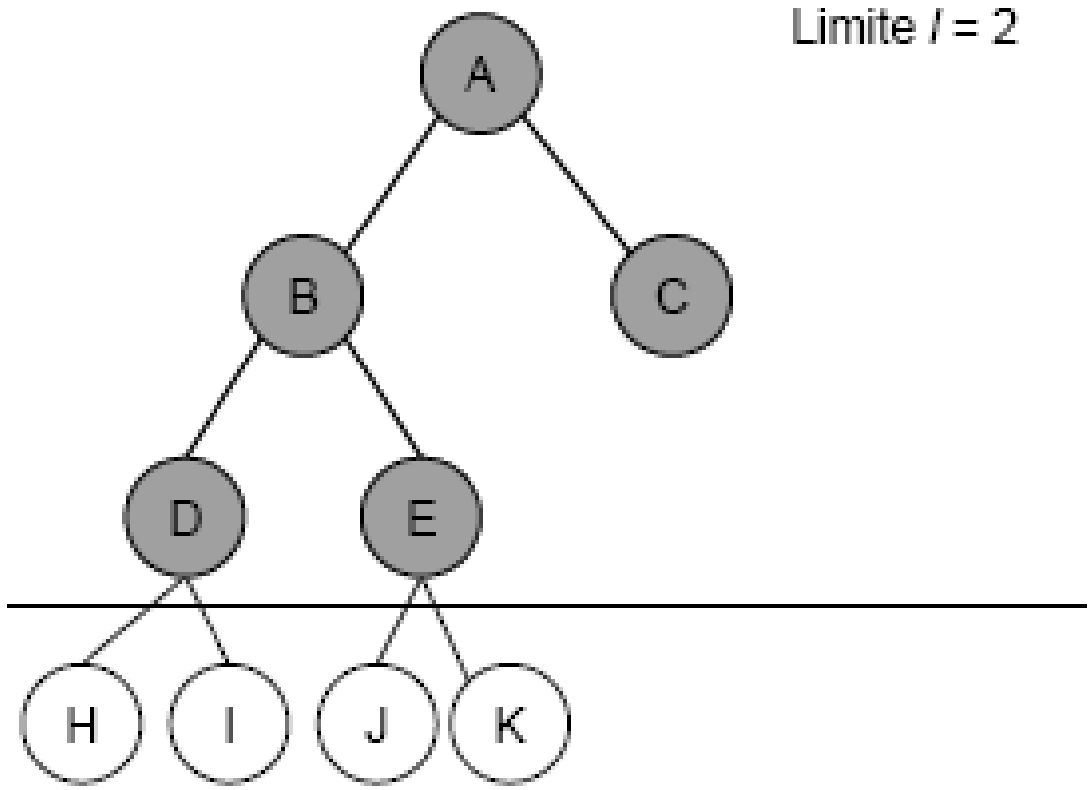
- recherche en profondeur a besoin de 12 Koctets
- recherche en largeur a besoin de 111 Tera-octets

* 10^{10} !!!

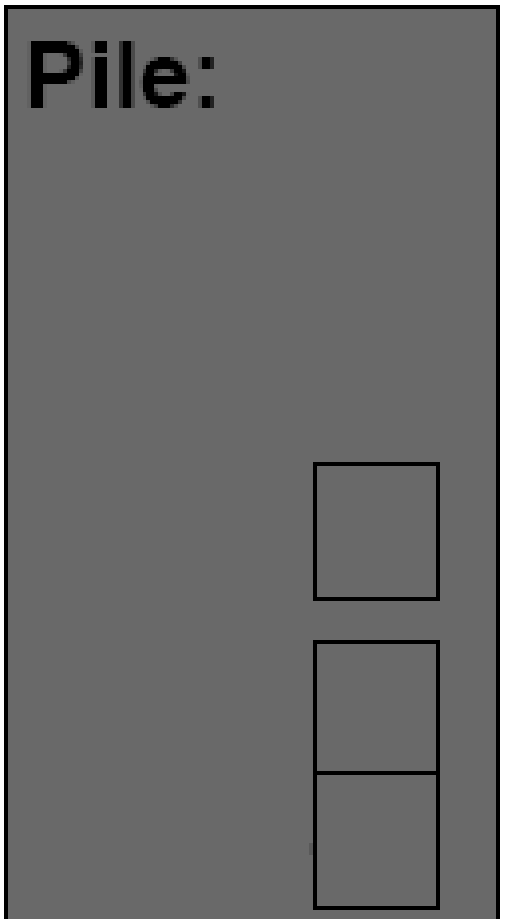
Algorithme de recherche

profondeur limitée

Principe de la recherche en profondeur limitée



Ordre de visite: A – B – D – E - C



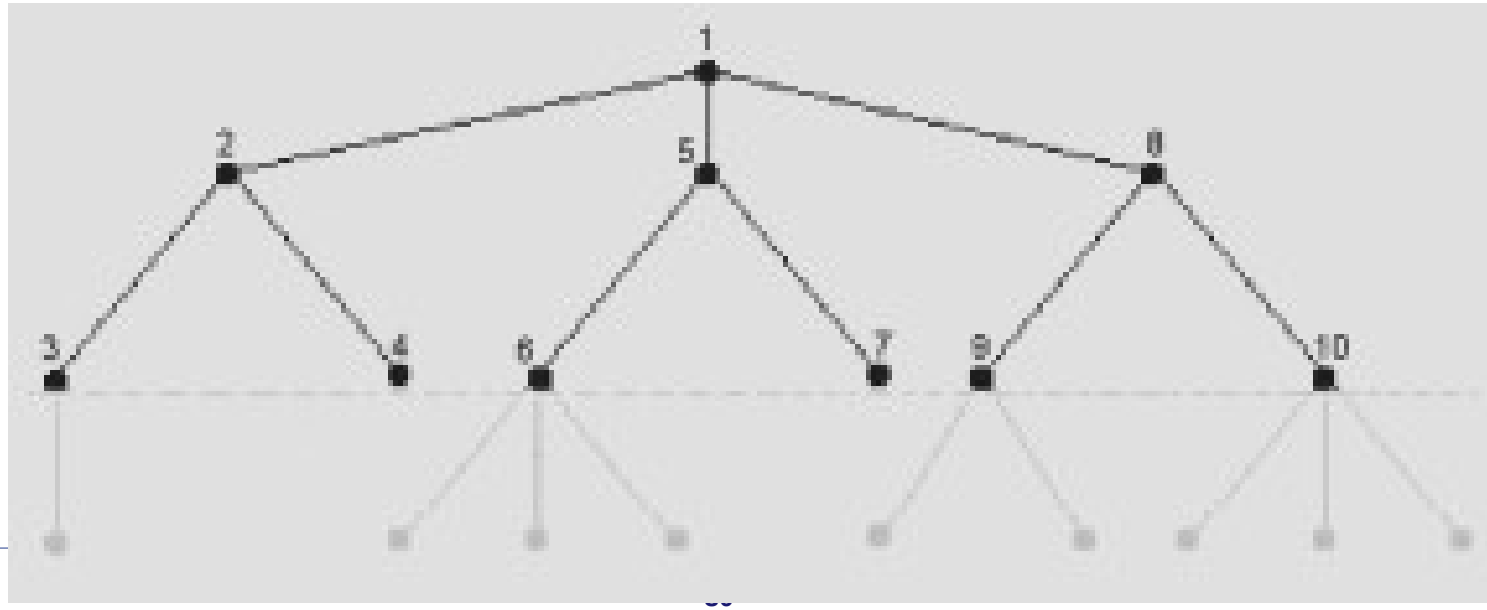
Principe de la recherche en profondeur limitée

Stratégie

- algorithme de recherche en profondeur avec une limite de profondeur d'exploration L

Implémentation

- les noeuds de profondeur L n'ont pas de successeurs
- exemple avec L = 2

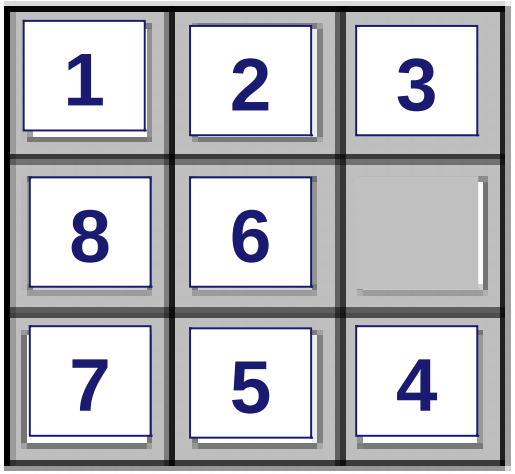


Propriétés de la recherche en profondeur limitée

- **Complétude:** Oui si $L \geq d$
- **Complexité en temps:** $O(b^L)$
- **Complexité en espace:** $O(b * L)$
- **Optimalité:** Non

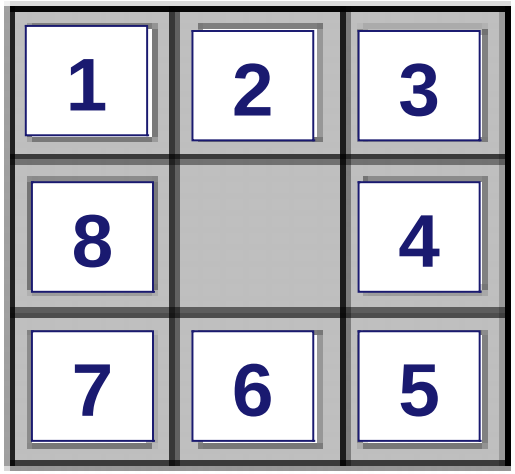
Exercice 3 : taquin 3x3

- Reprendre l'exercice 2 en appliquant la recherche en profondeur limitée à 3 sur la donnée suivante:



Configuration initiale

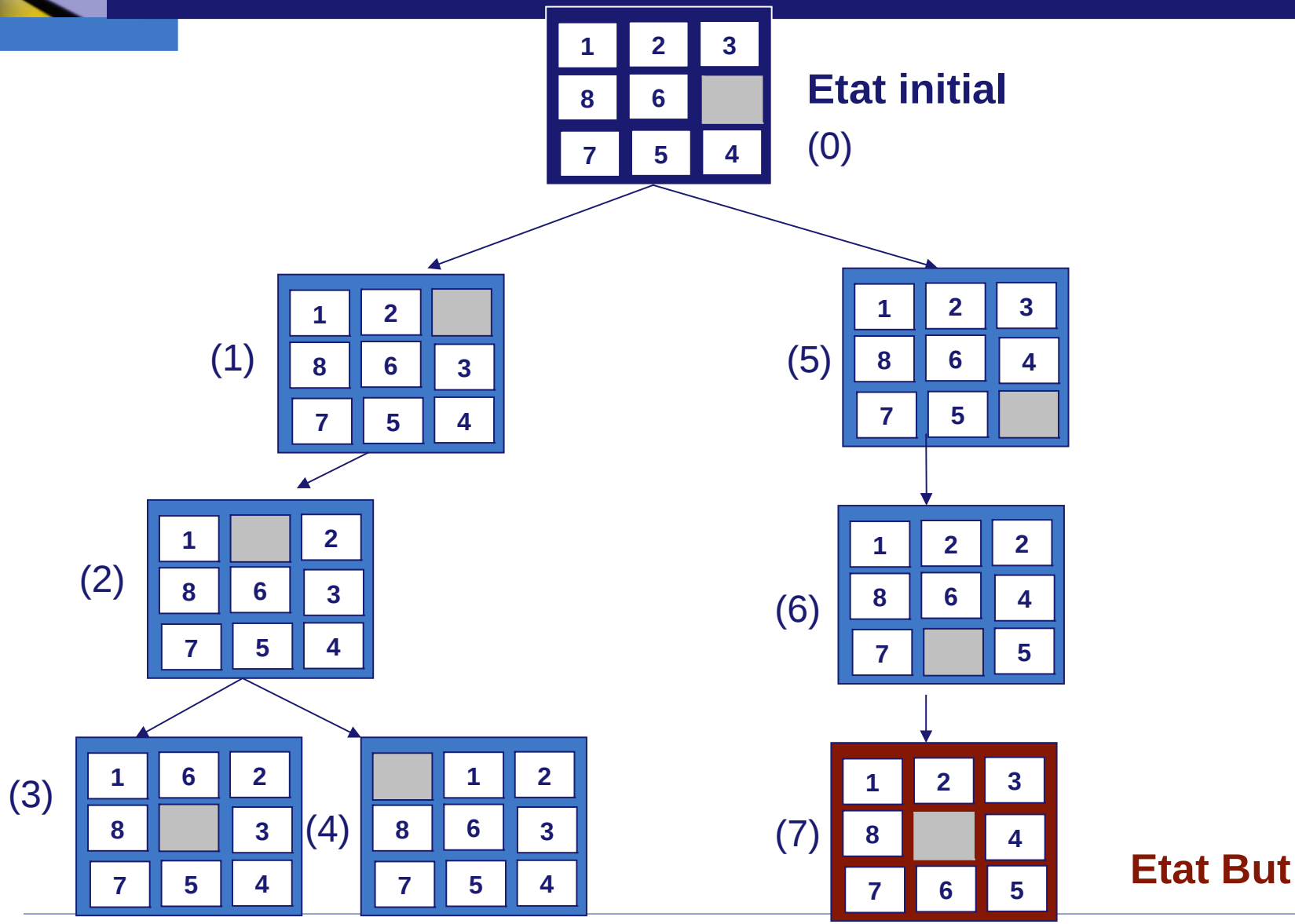
Etat initial



Configuration finale

Etat final

Solution Exercice 3



( ): numéro donnant l'ordre de développement

Algorithme de recherche

approfondissement itératif

=

Itérative en profondeur

Principe de la recherche itérative en profondeur

→ **Le problème avec la recherche en profondeur limitée est de fixer la bonne valeur de L**

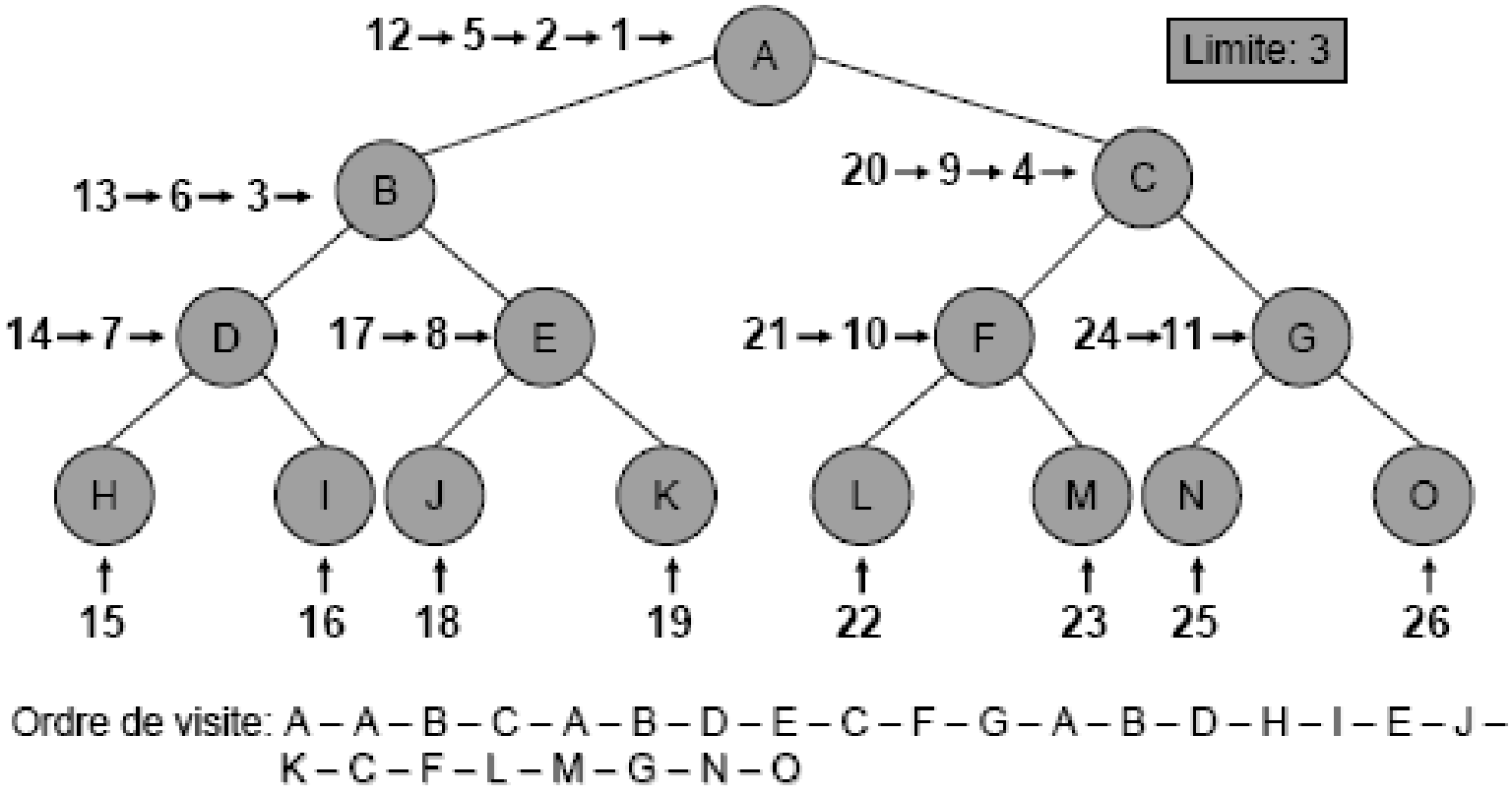
• approfondissement itératif = essayer toutes les valeurs possibles de L à partir de L = 0 (en incrémentant la limite)

• combine les avantages de la recherche en largeur et en profondeur

– optimal et complet comme la recherche en largeur
– économe en espace comme la recherche en profondeur

• c'est l'algorithme de choix si l'espace de recherche est grand et si la profondeur de la solution est inconnue

Exemple de la recherche itérative en profondeur



Propriétés de la recherche itérative en profondeur

- **Complétude** Oui

- **Complexité en temps**

$(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

- **Complexité en espace** $O(b * d)$

- **Optimalité** Oui si coût = 1 par étape

Les méthodes heuristiques (exploration informée)

Introduction aux heuristiques

- Les méthodes aveugles (non informées) sont des méthodes systématiques peu efficaces
- Toute technique visant à accélérer la recherche est basée sur une information appelée **heuristique**
- Une heuristique signifie '**aider à découvrir**'
- Ils utilisent des sources d'information supplémentaires. Ces algorithmes parviennent ainsi à des performances meilleures en terme de :
  - Complexité spatiale et/ou
  - Complexité temporelle
- Les méthodes utilisant des heuristiques sont dites méthodes de recherche heuristiques

Implémentation des méthodes heuristiques

- Utiliser un critère pour réordonner tous les nœuds qui sont explorés (au lieu d'être mis dans une pile ou file)

- Une certaine mesure doit être établie pour évaluer « la promesse » d'un nœud.

→Cette mesure est appelée **fonction d'évaluation** ou **d'adéquation** ou **objective**

Fonction d'évaluation

- La recherche ordonnée revient à choisir à développer le meilleur nœud au sens d'un certain critère

  ➔ centrée sur le nœud ayant les meilleurs chances de mener au but

- L'utilisation d'une heuristique h dans la fonction d'évaluation f

- **h(u)** : fonction heuristique qui estime le coût du passage de l'état u à l'état final.

- h(n) = 0 si n est un état but.

- La fonction permet d'ordonner la recherche

Algorithme du meilleur d'abord

Méthode heuristique A : Best-first

- Examiner les nœuds qui semblent les plus proches d'un état but, dans l'espoir d'aboutir plus vite à une solution.

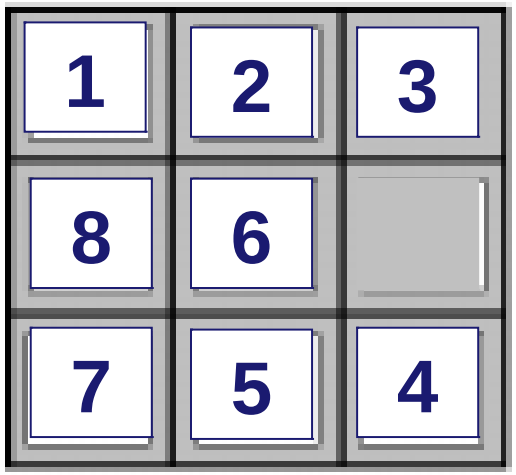
- Dans ce cas: $f(n) = h(n)$

Exercice : Best-first

- f(n)=h(n)

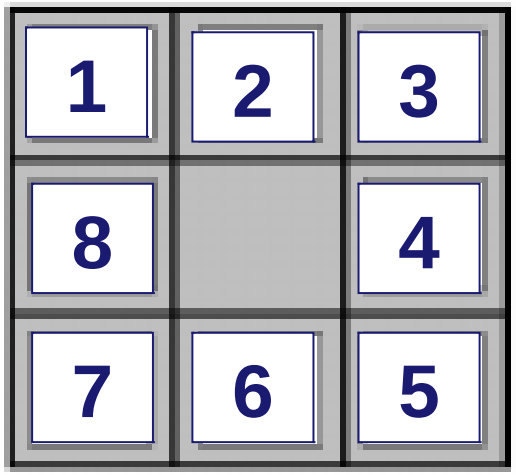
Avec :

- h: nombre de jetons mal placés



Configuration initiale

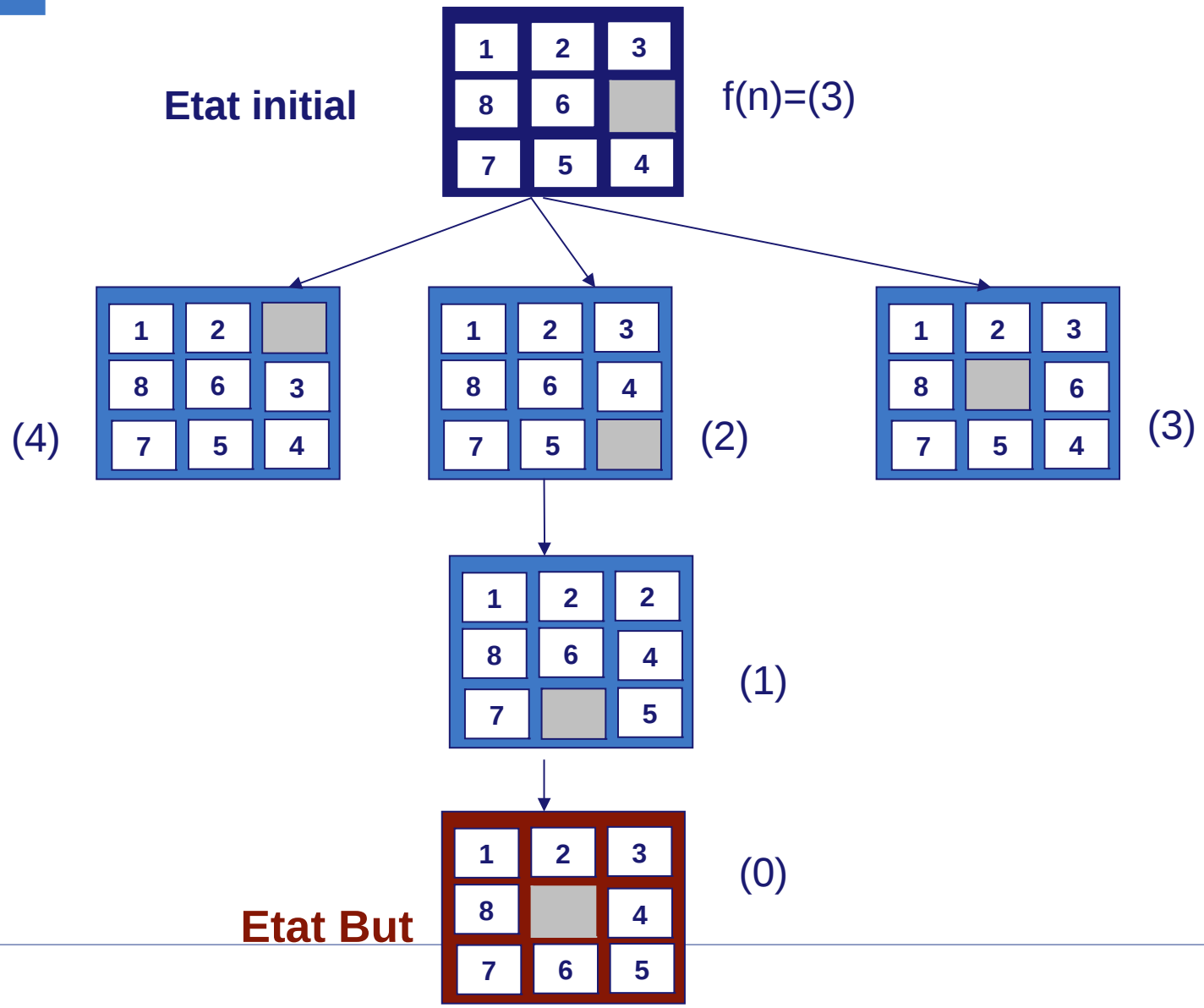
Etat initial



Configuration finale

Etat final

Solution : Best first



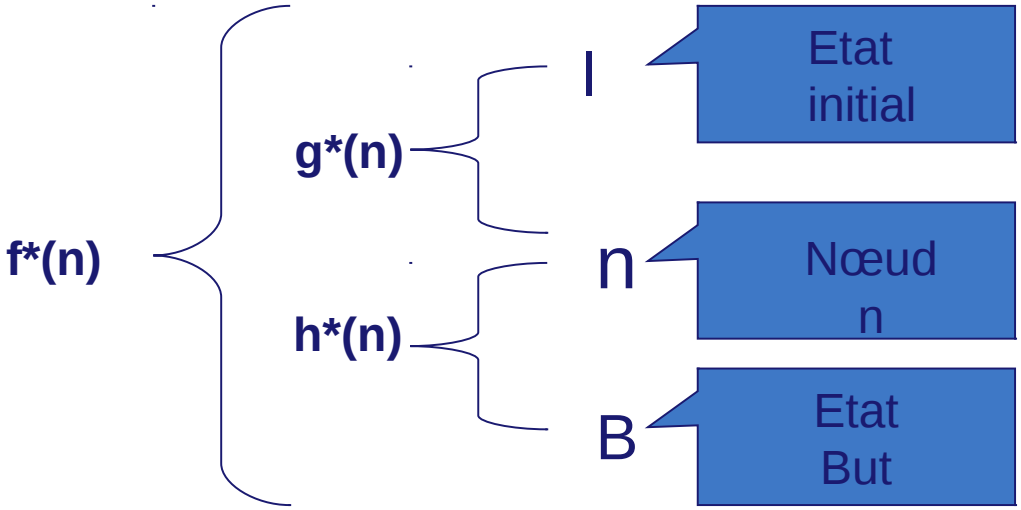
Algorithme A*

- **Algorithme A***: stratégie du meilleur en premier dans le cadre des graphes OU et des problèmes de minimisation des coûts.

- Algorithme A* permet de calculer le plus court chemin menant de l'état initial à l'état final.

Méthode d'évaluation

- **f*(n) représente le coût idéal du chemin** passant par un nœud n pour arriver au but



f*(n) = g*(n) + h*(n)

- **g*:** le coût du meilleur chemin déjà rencontré de l'état initial I à n

- Le choix de g est très dépendant du domaine

- Exemple : pour le jeu de taquin

  g(n) : le nombre de jeton déplacé

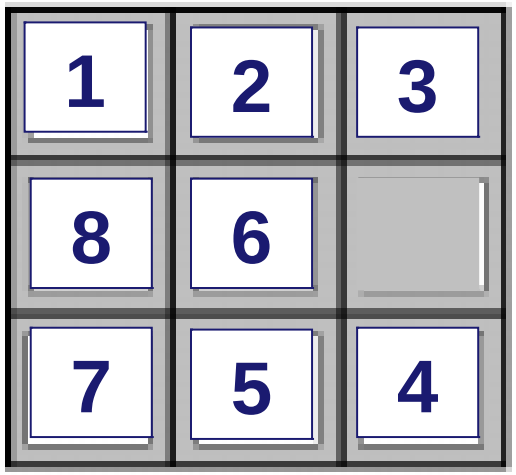
  → la longueur de la chaîne entre la racine et n

Exercice : Recherche heuristique avec A*

- f(n)=g(n) +h(n)

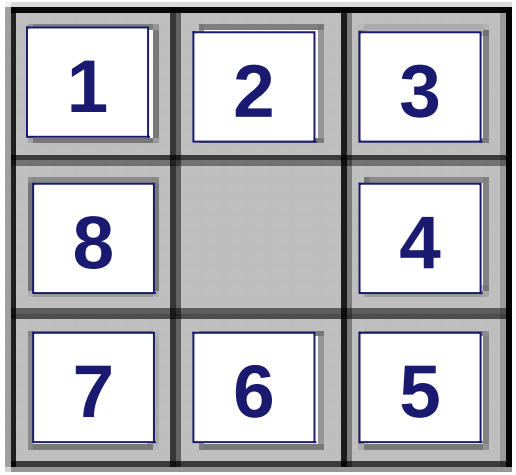
Avec :

- g : nombre de jetons déplacés
- h: nombre de jetons mal placés



Configuration initiale

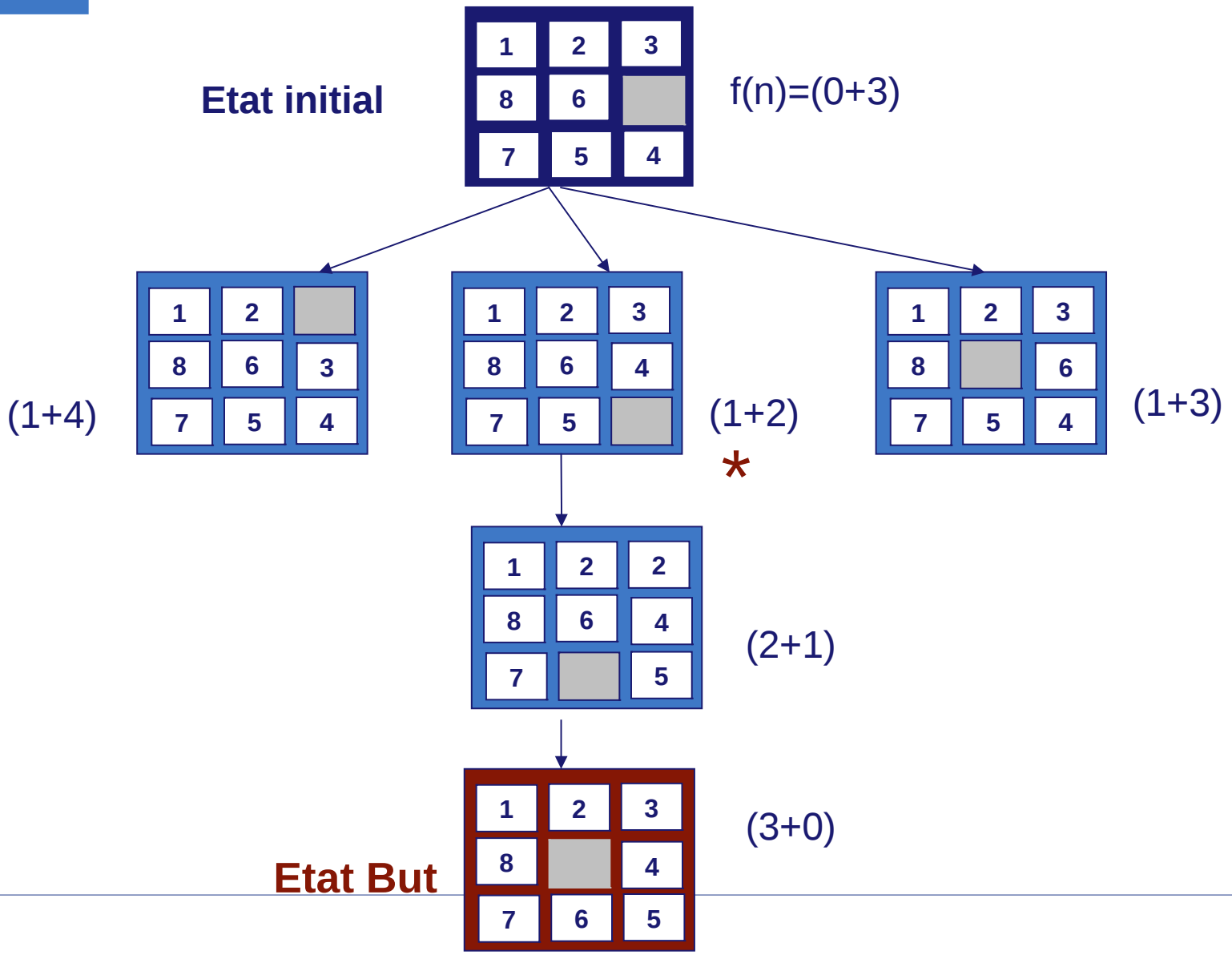
Etat initial



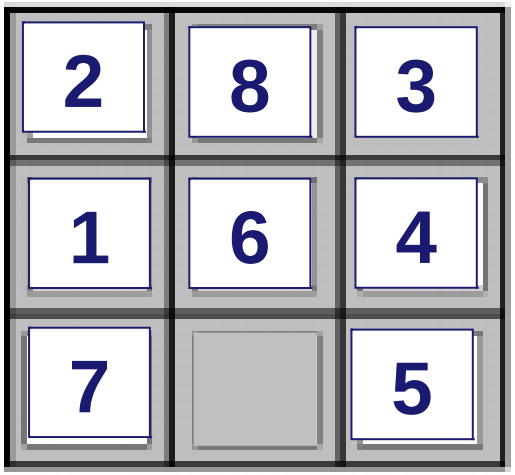
Configuration finale

Etat final

Solution Exercice 4

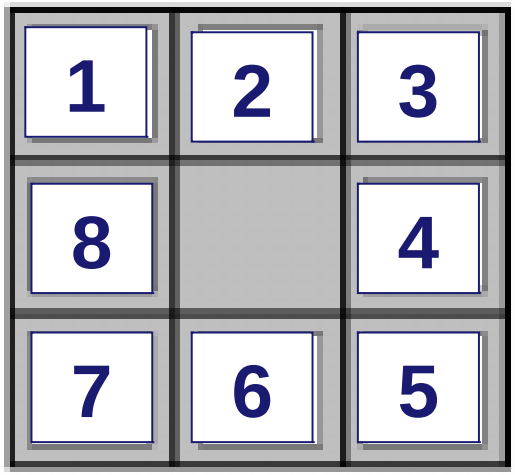


Exercice 5: recherche avec algorithme A*



Configuration initiale

Etat initial



Configuration finale

Etat final

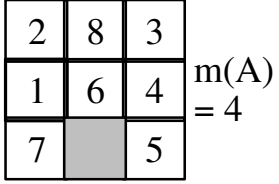
Rappel

Mouvements légaux: Déplace le <blanc> vers:

- le haut
- la droite
- le bas
- la gauche

Contraintes: Les mouvements en diagonal sont interdits

puzzle avec heuristique A*



A, ..., N: mouvements

m = mal placés jetons

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)

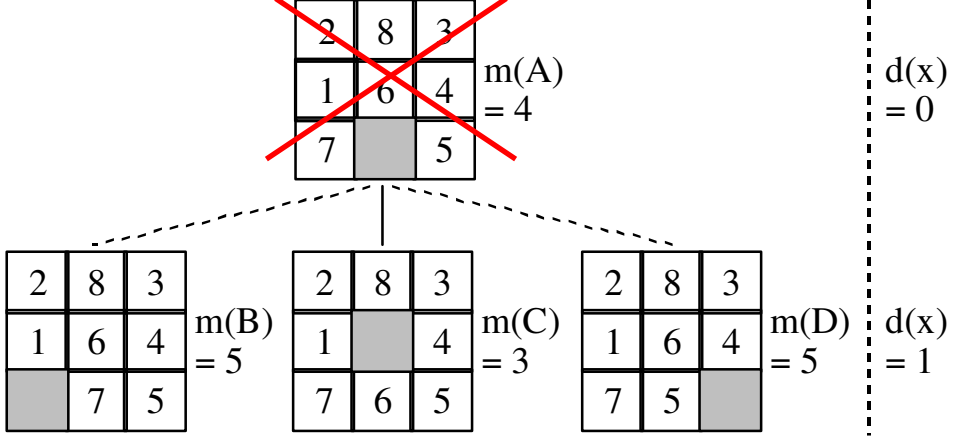
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



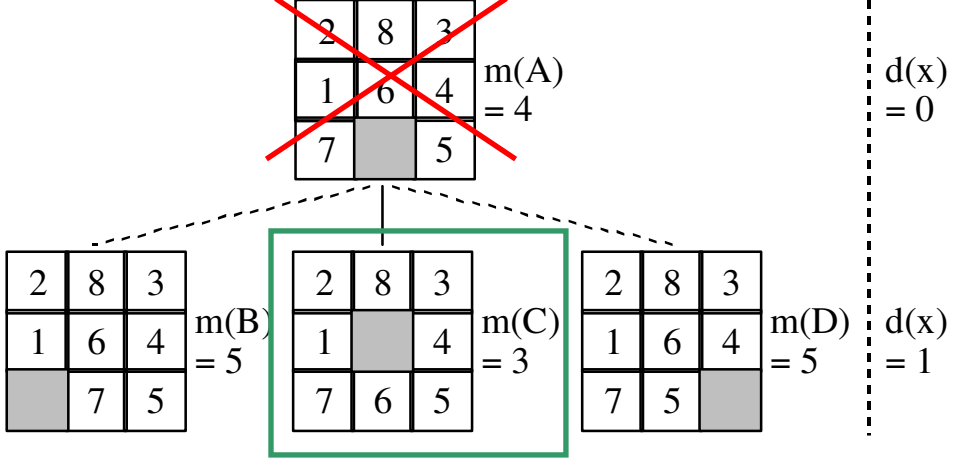
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



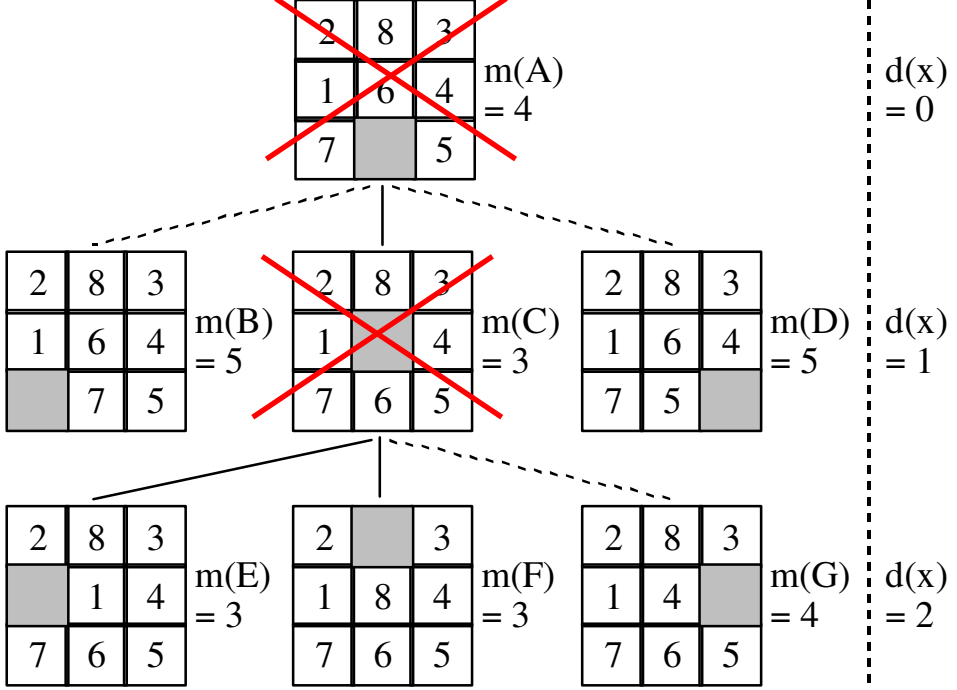
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



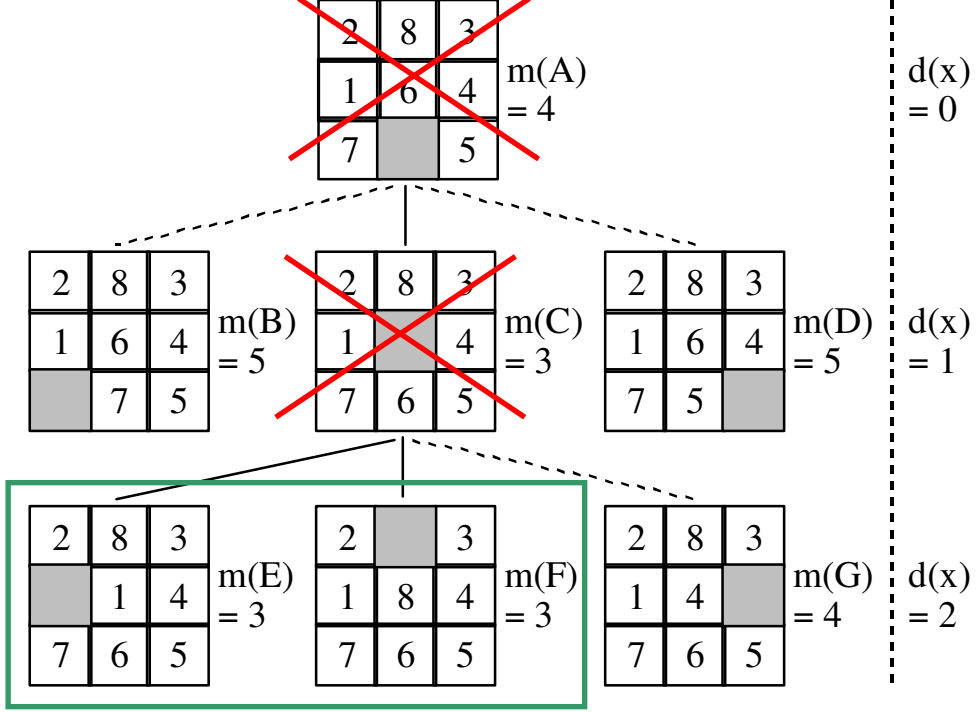
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



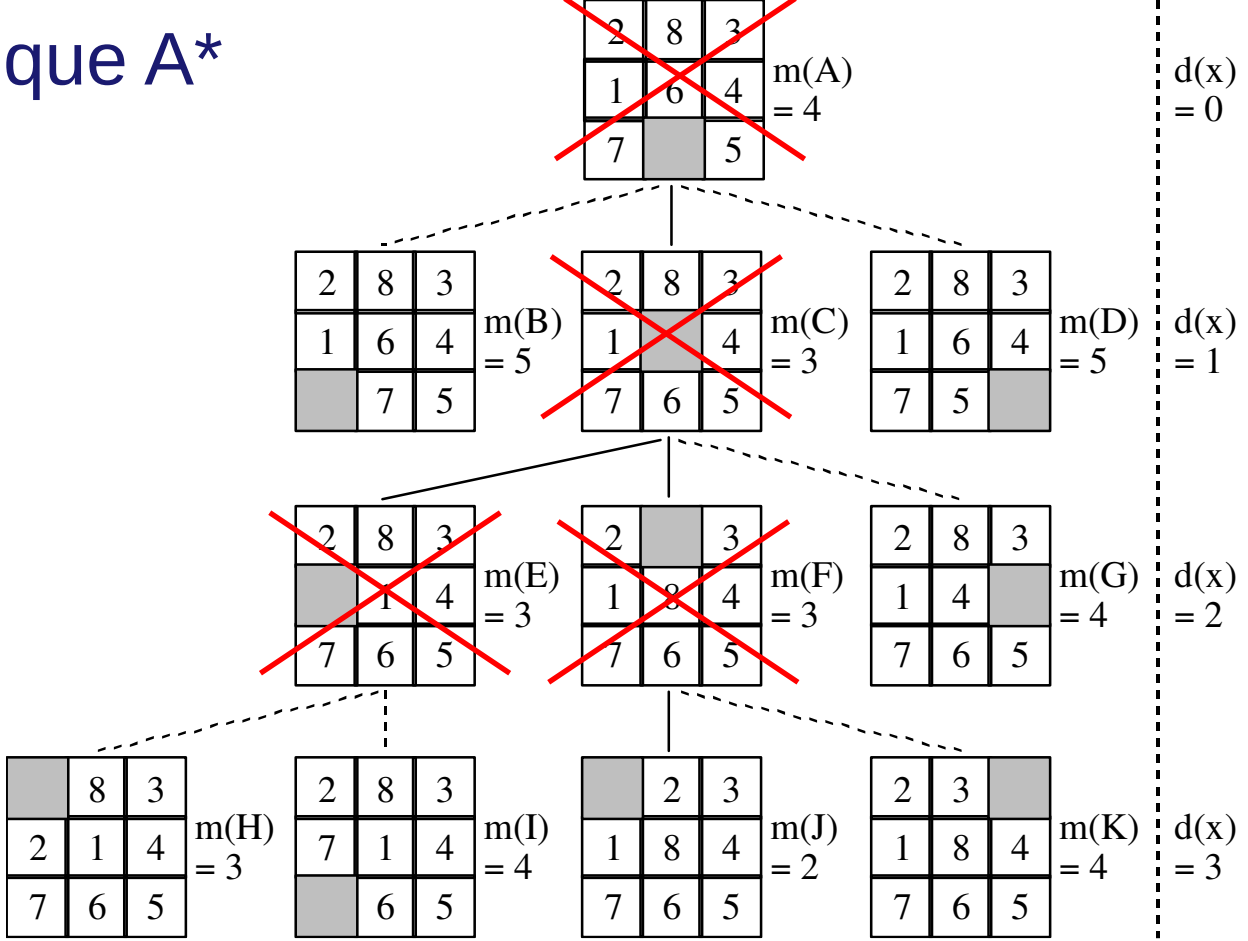
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



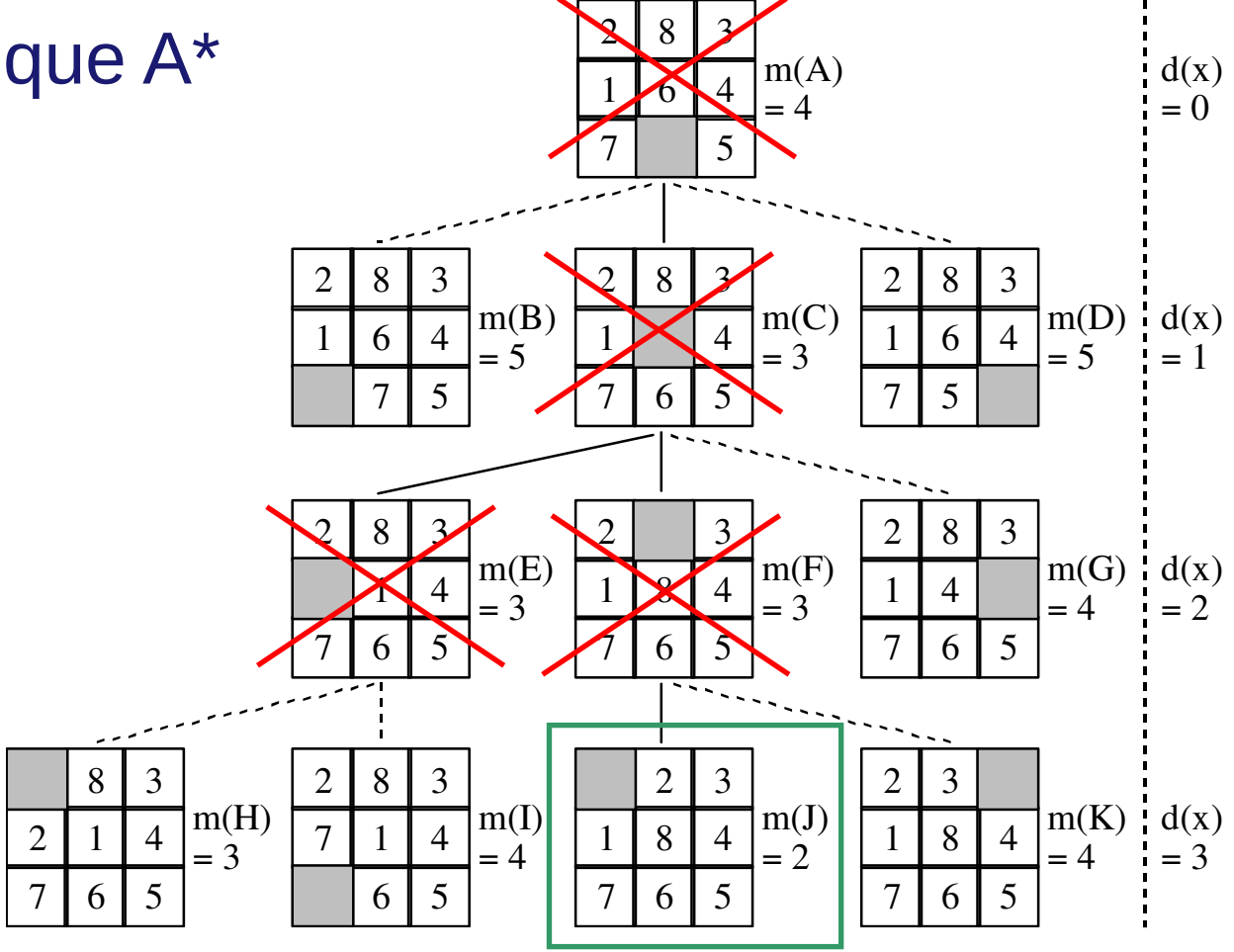
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



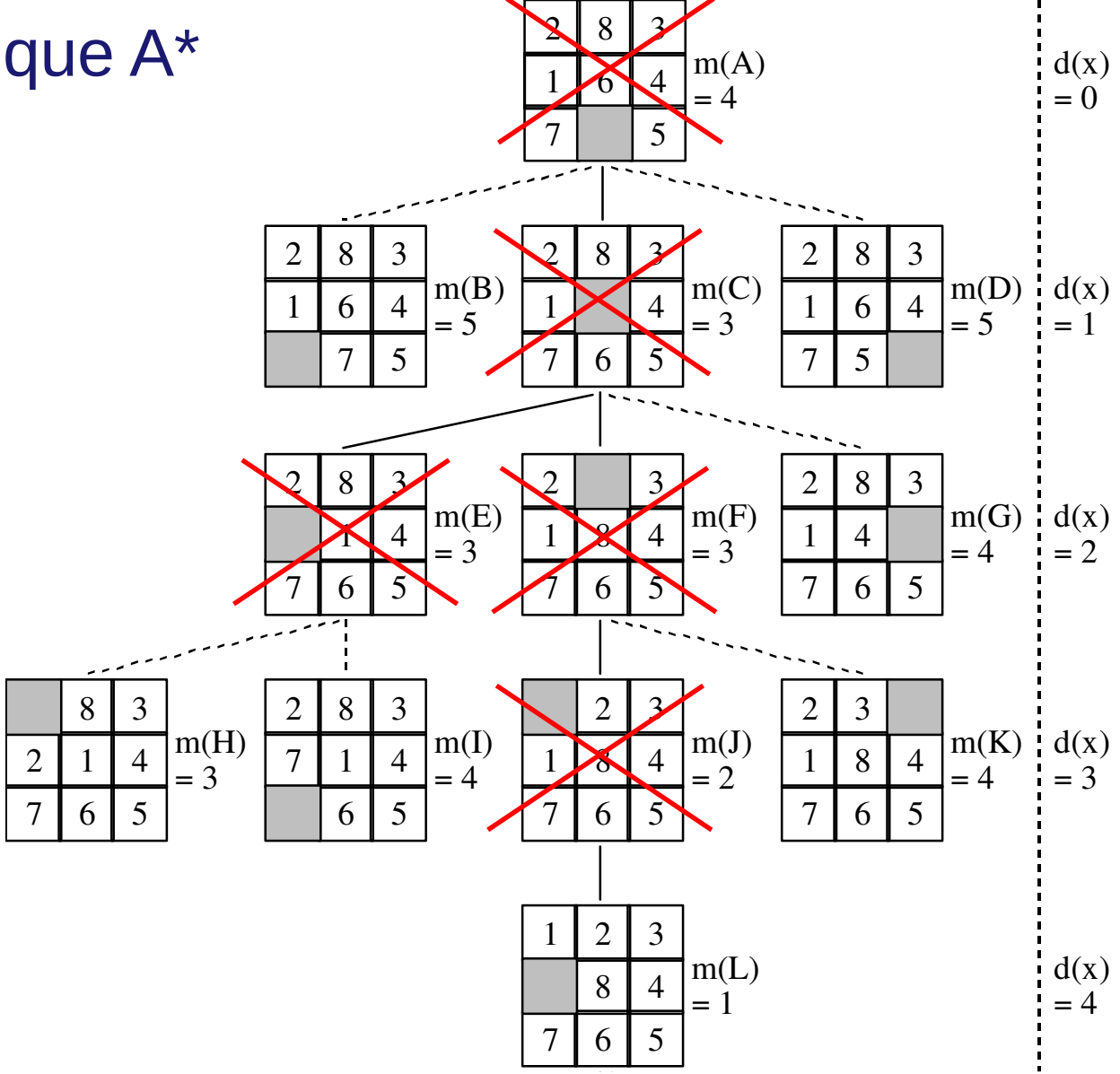
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



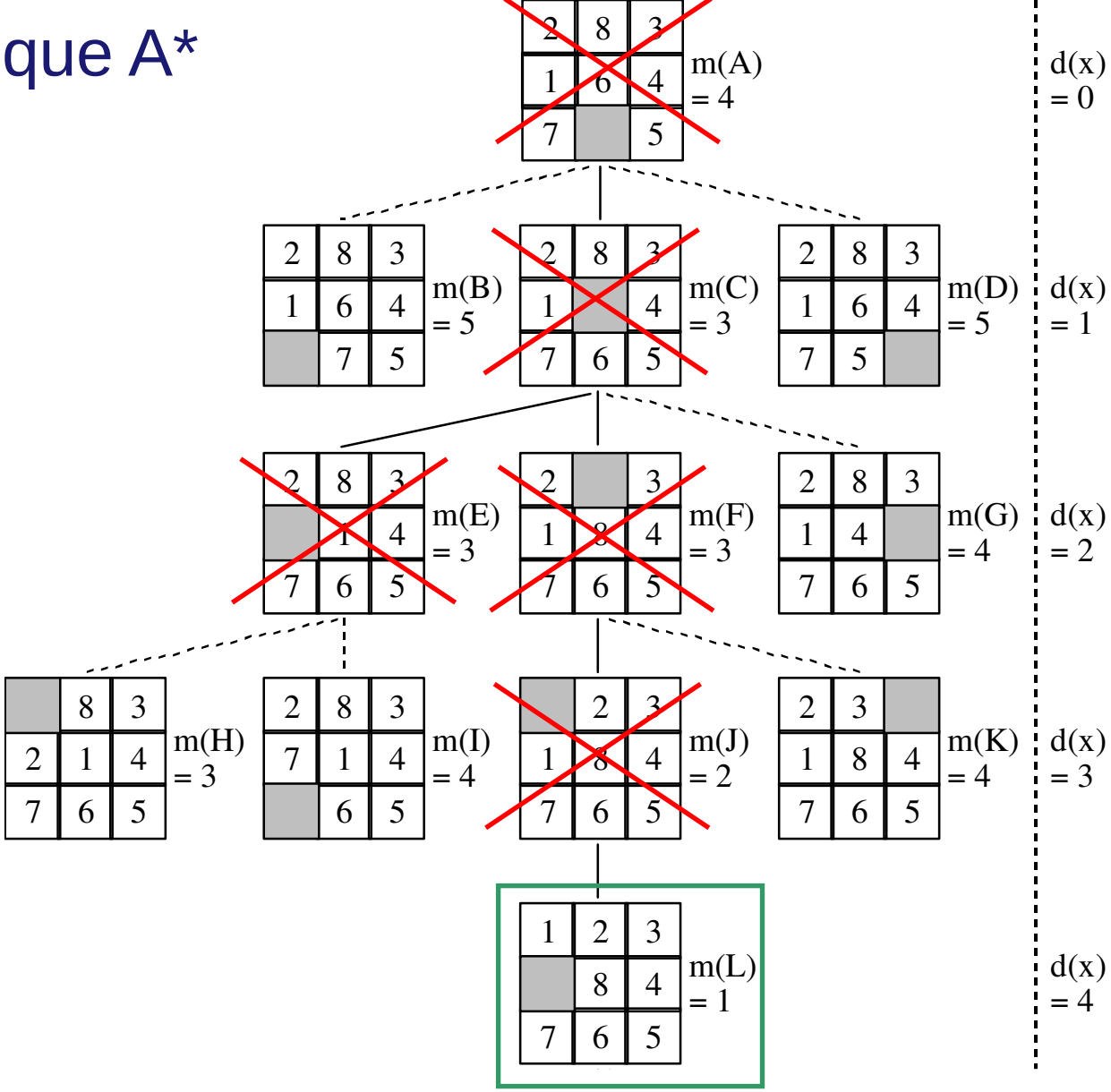
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



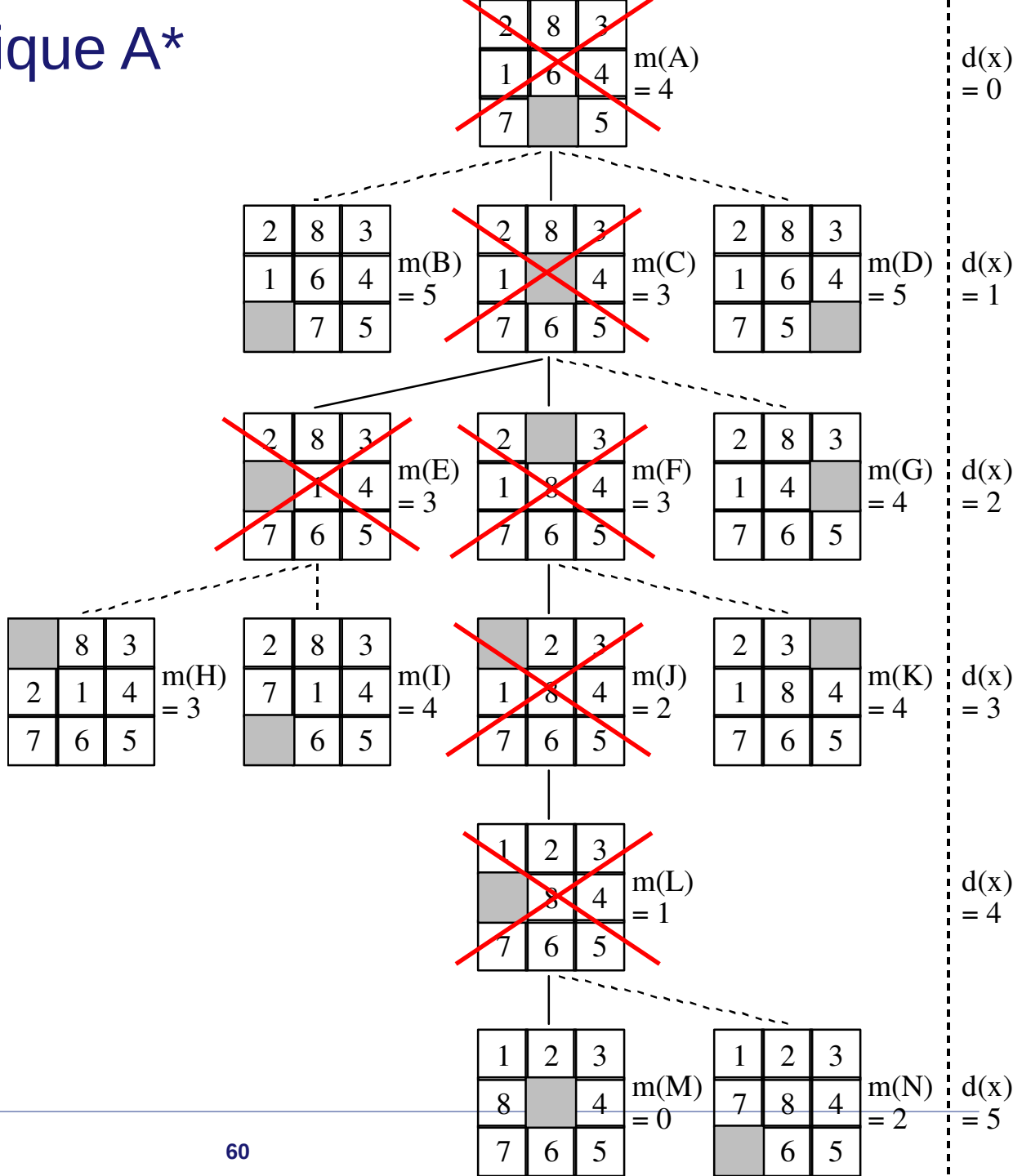
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



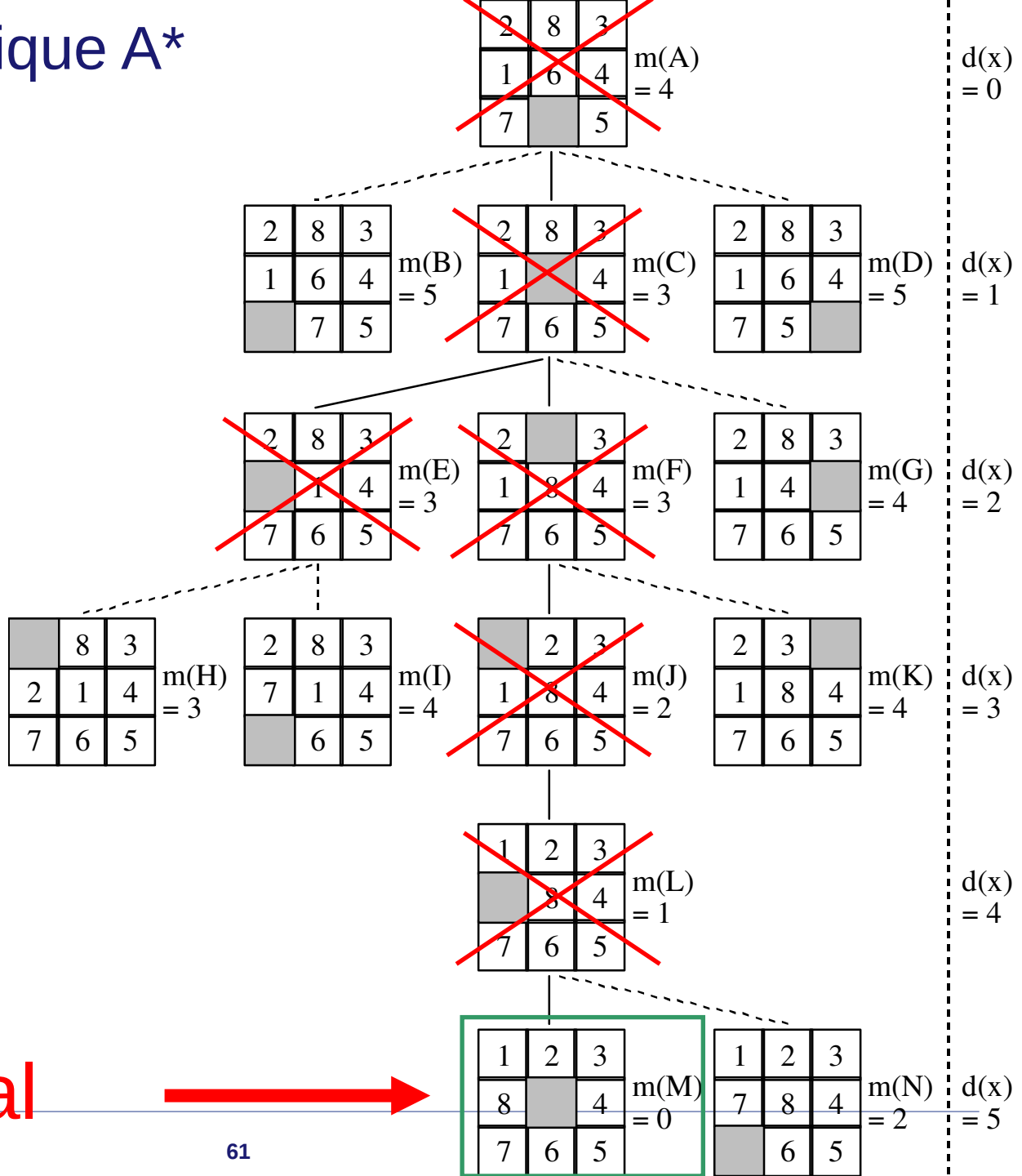
puzzle avec heuristique A*

A, ..., N: mouvements

m = mal placés carreaux

d = niveau dans l'arbre
(profondeur)

f(x) = m(x) + d(x)



État final