



Cours 2 : Algorithmes de recherche aveugle



A. Belaïd
Université de Nancy 2





Références

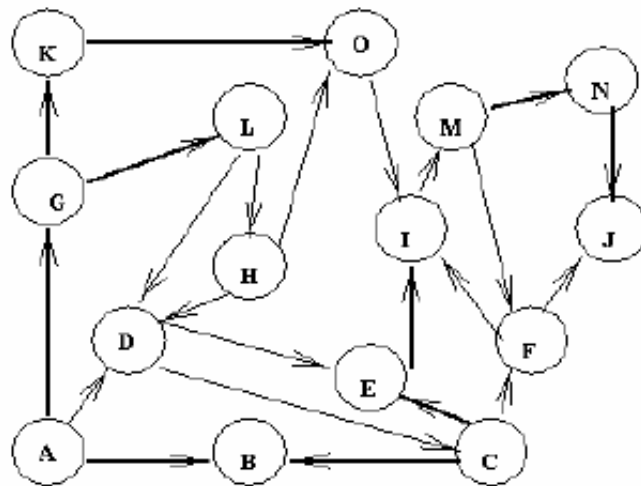


- Pellegrini C.A. (2000), *Introduction à l'Intelligence Artificielle*, Transparents de cours 2000-2001 à CUI, <http://cui.unige.ch/ScDep/Cours/IA/ia00-01>
- http://turing.cs.pub.ro/auf2/html/chapters/chapter3/chapter_3_2_1.html
- <http://www.cours.polymtl.ca/inf4215/documentation/CoursRechercheSol.pdf>

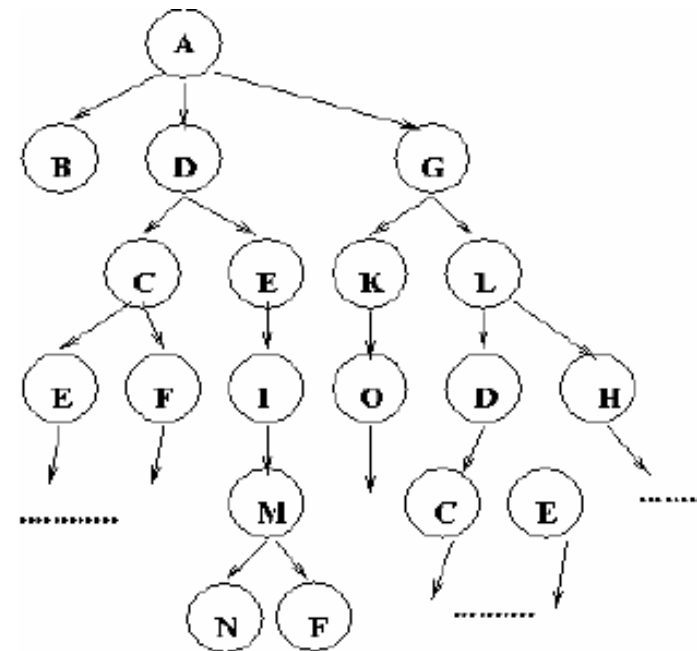
Problème général de recherche



Problème :
exprimé par un graphe d'états



Transformé en un arbre de recherche
exprimant les cheminements de
raisonnement




Raisonner sur un problème, c'est construire cet arbre en identifiant **le chemin de raisonnement** le plus intéressant qui correspond à la solution

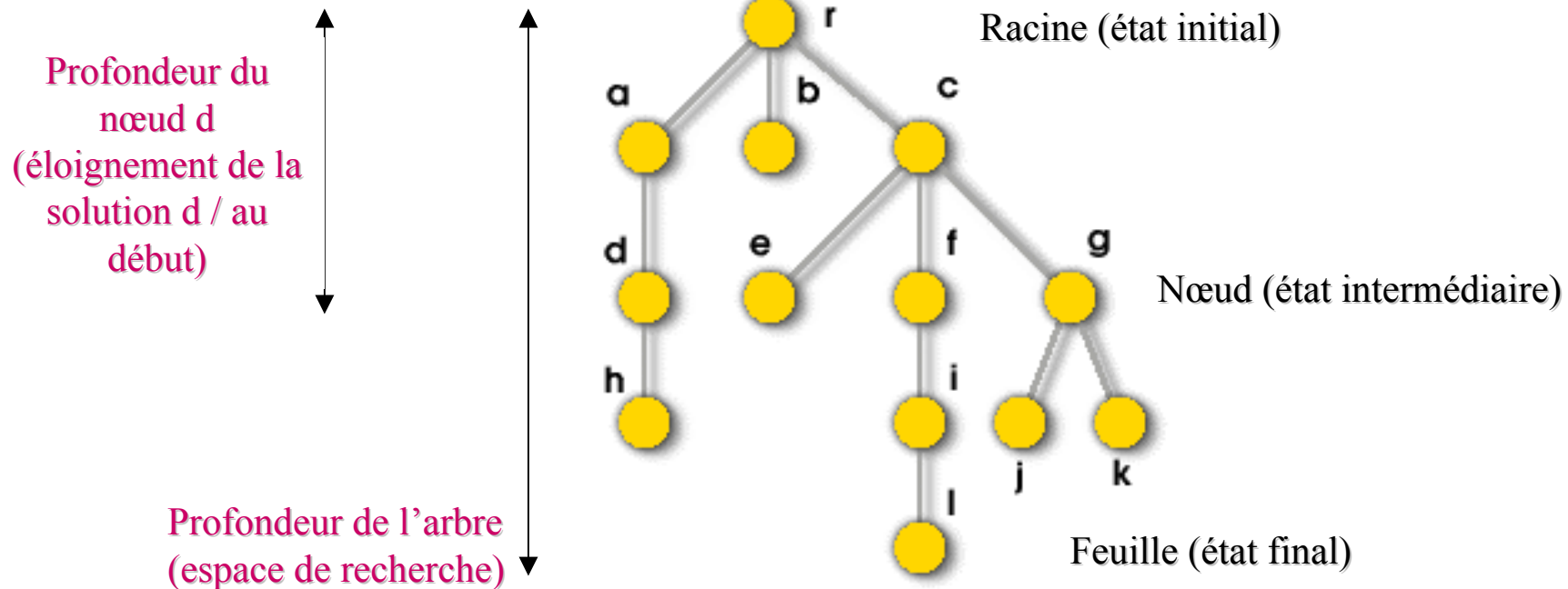


Algorithme général de recherche



- 
- Construction de l'arbre de recherche :
 - Exploration **simulée** de l'espace d'états (graphe d'états) en générant les états successeurs des états déjà explorés
 - Les nœuds sont étudiés puis développés ou étendus par un processus appelé "**expansion**" d'états
 - Chaque chemin correspond à un chemin de raisonnement qui conduit soit à une solution soit à un échec

Problème général de recherche





Critères d'évaluation



- Les méthodes de recherche sont évaluées selon :
 - **Optimalité**
 - c'est la caractéristique d'un algorithme de recherche qui trouve la meilleure solution (pour un problème qui en admet plusieurs)
 - **Complétude**
 - si une solution existe l'algorithme garantit qu'elle sera trouvée
 - **Complexité en temps**
 - estimation du temps nécessaire pour résoudre un problème
 - **Complexité en espace**
 - estimation de l'espace mémoire nécessaire à l'algorithme pour résoudre le problème

Complexité des algorithmes




- Les complexité en temps et en espace sont mesurées en fonction de :
 - b = facteur de branchement de l'arbre de recherche (= nombre maximum de successeurs pour un état)
 - d = profondeur à laquelle se trouve le (meilleur) nœud-solution
 - m = profondeur maximum de l'espace de recherche (peut être ∞)

Algorithme général de recherche



Fonction rechercheArbre (*problème, stratégie*) **retourne** une solution, ou échec
initialiser l'arbre de recherche en utilisant l'état initial du problème
Boucle faire
 si pas de candidats pour l'expansion **alors retourne** échec
 choisir un nœud feuille pour l'expansion en fonction de la *stratégie*
 si nœud contient un *état but* **alors**
 retourne la solution correspondante
 sinon étendre le nœud et **ajouter** les nœuds résultats à l'arbre de recherche
fin



Algorithme général de recherche



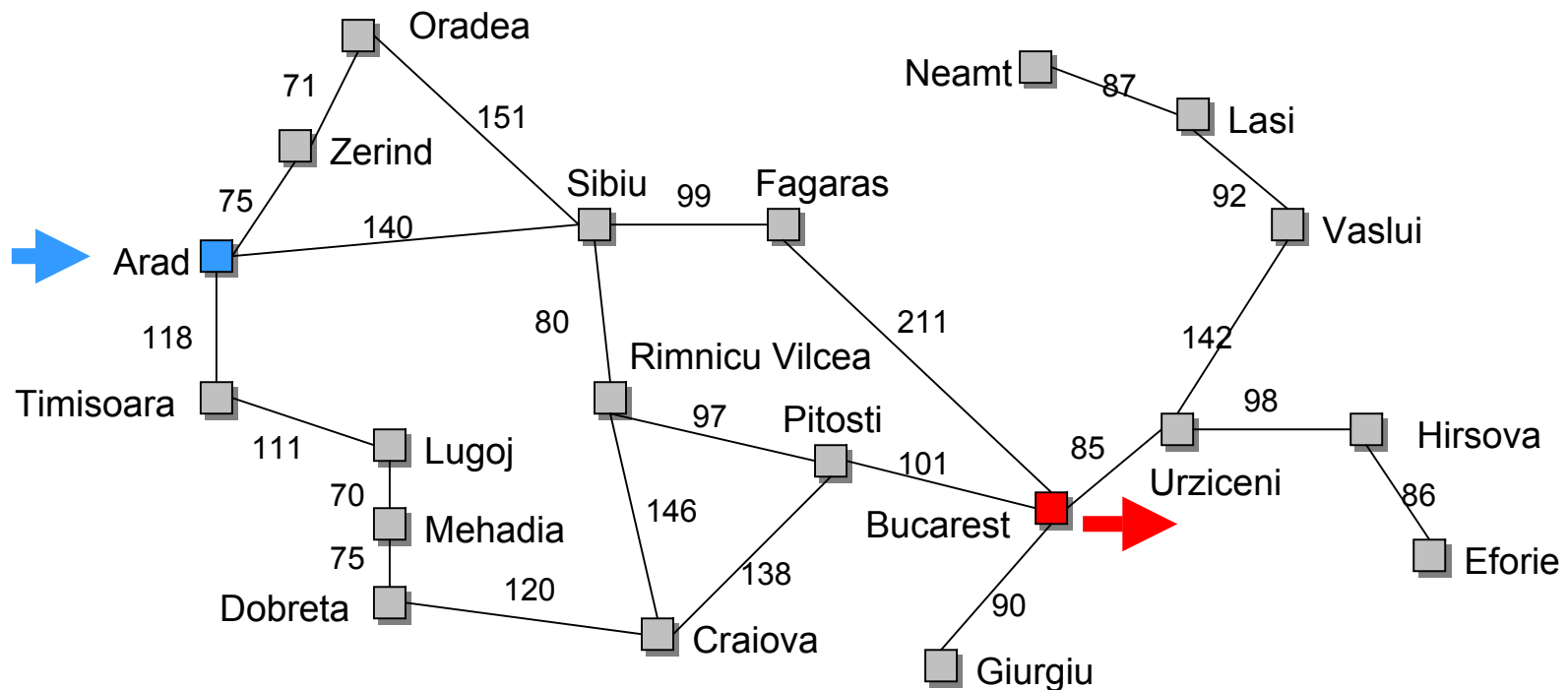
■ Exemple :

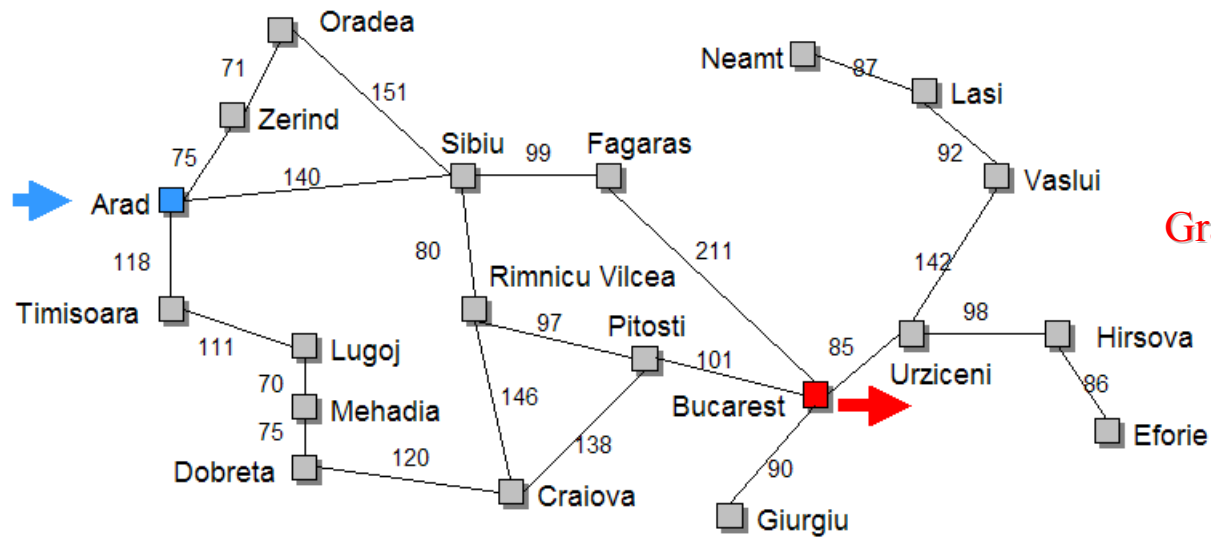
- Recherche d'un chemin (itinéraire) conduisant à une ville d'arrivée (but) à partir d'une ville de départ (état initial)
- Le graphe d'états :
 - Nœuds : noms des villes
 - Arêtes : distances entre ces villes

Algorithme général de recherche



- Exemple : ici le graphe d'états est le même que le graphe de connexions des villes



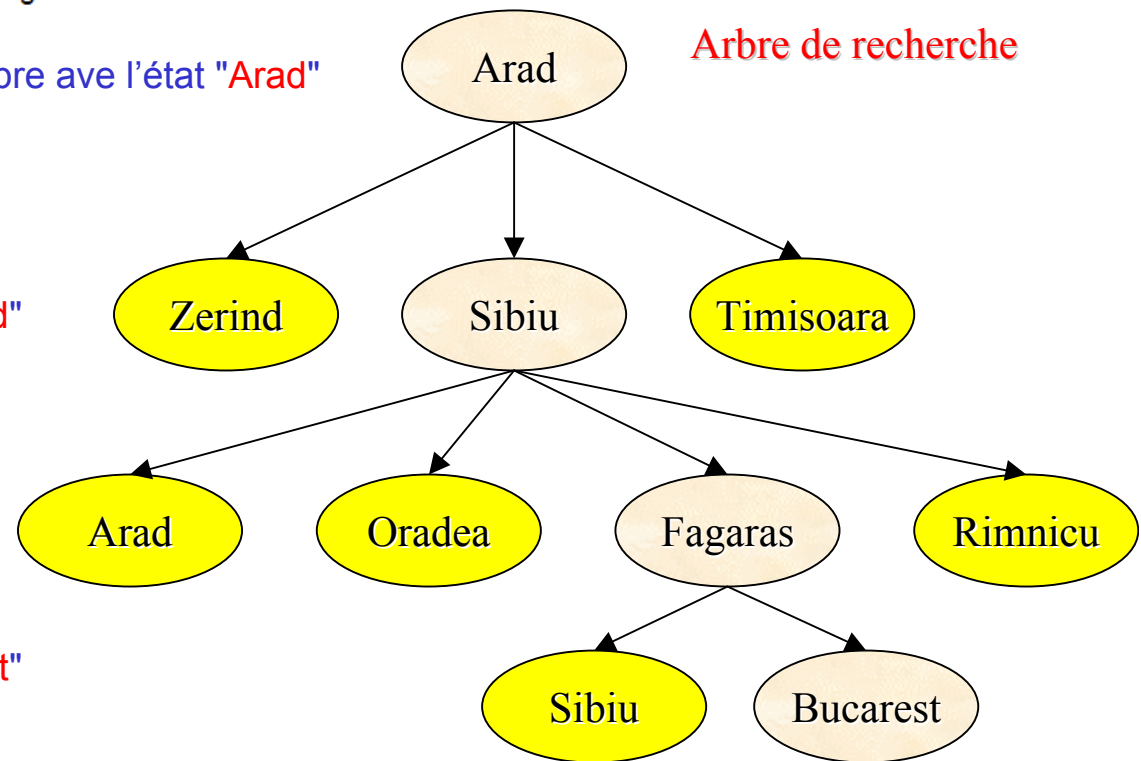


Graphe d'états



Initialisation de l'arbre avec l'état "Arad"

Arbre de recherche



Expansion du nœud "Arad"

Expansion du nœud "Fagaras"

Atteinte de l'état but "Bucarest"

Concepts de base pour la recherche



■ Exercice

- Reprendre le problème du loup, de la chèvre et du chou
 - Choisir un formalisme permettant de représenter les états possibles
 - Identifier, dans ce formalisme, l'état initial et le(s) état(s) final(ux)
 - Définir l'ensemble des opérateurs en précisant les prémisses, les contraintes et les conséquences
 - Établir la liste des actions permettant d'atteindre une solution

Concepts de base pour la recherche



■ En résumé

- Un algorithme de recherche repose sur les éléments suivants :
 - Arbre de recherche
 - Recherche d'un nœud
 - Expansion d'un nœud
 - Stratégie de recherche
 - A chaque étape du processus de recherche, déterminer quel est le nœud à "étendre"

Concepts de base pour la recherche



■ Définition structurelle (algorithmique) d'un arbre

- On définit 2 types : $\text{arbre} = \text{Arbre}(V)$ et nœud

initArbre : $V \rightarrow \text{arbre}$

noeudvide : $\text{arbre} \rightarrow \text{booléen}$

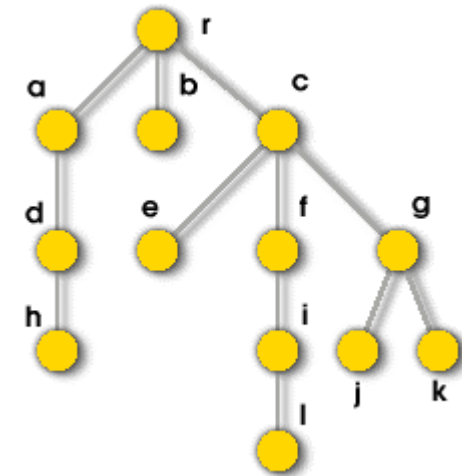
racine : $\text{arbre} \rightarrow \text{nœud}$

val : $\text{arbre} \times \text{nœud} \rightarrow V$

fils : $\text{arbre} \times \text{nœud} \rightarrow \text{liste}$

père : $\text{arbre} \times \text{nœud} \rightarrow \text{nœud}$

étendre : $\text{arbre} \times \text{nœud} \rightarrow \text{arbre}$





Concepts de base pour la recherche



■ Exercice

- Soit A un arbre binaire donné
- Q1 : Écrire l'algorithme de recherche d'un nœud particulier
- Q2 : Écrire l'algorithme d'extraction de tous les chemins dans l'arbre



Concepts de base pour la recherche



- Remarque

- Cette manière réursive masque la vraie gestion des choix

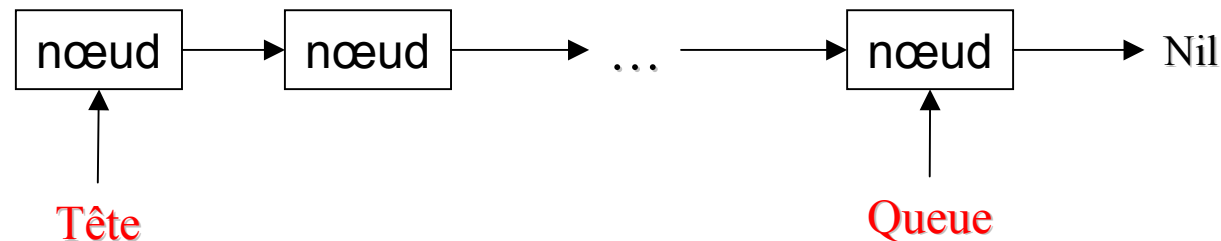
- D'où :

- Comment gérer les choix ?
- Comment gérer les retours en arrière pour explorer d'autres chemins ?
- Bref : comment expliciter cette gestion ?

Solution : on associe une structure de données à l'espace



- La gestion des nœuds de l'arbre de recherche repose sur :
 - l'utilisation d'une structure de données particulière :
 - Une liste (file d'attente)
 - contient les nœuds de recherche pas encore traités, mais triés par ordre de prise en compte :
 - Ordre de haut en bas : père puis fils
 - Ordre de gauche à droite sur les frères





File d'attente ou liste



■ Algorithme général de recherche :

fonction Recherche (*problème*, stratégie) **res** : nœud **ou** échec

liste-nœuds ← **initialiserListe** (**créerNœud** (État-initial
[*problème*]))

Répéter

si **listeVide**(*liste-nœuds*) **alors** **res** ← échec

sinon *nœud* ← **extraireListe**(*liste-nœuds*)

fsi

si **test-but** (*problème*, *nœud*) **alors** **res** ← *nœud*

sinon *liste-nœuds* ← **insérerListe** (*liste-nœuds*,
étendreListe(*liste-nœuds*, *nœud*, Opérateurs [*problème*]))

fsi

FinRépéter



File d'attente ou liste



■ Opérations sur la liste :

<i>initListe :</i>	<i>vide</i>	\rightarrow	<i>liste</i>
<i>listeVide :</i>	<i>liste</i>	\rightarrow	<i>booléen</i>
<i>insertListe :</i>	<i>liste \times nœud</i>	\rightarrow	<i>liste</i>
<i>inseriListe :</i>	<i>liste \times nœud \times nœud</i>	\rightarrow	<i>liste</i>
<i>inserqListe :</i>	<i>liste \times nœud</i>	\rightarrow	<i>liste</i>
<i>extratListe :</i>	<i>liste</i>	\rightarrow	<i>nœud</i>
<i>extraqListe :</i>	<i>liste</i>	\rightarrow	<i>nœud</i>
<i>extraiListe :</i>	<i>liste \times nœud</i>	\rightarrow	<i>nœud</i>
<i>sucListe :</i>	<i>liste \times nœud</i>	\rightarrow	<i>nœud</i>
<i>valListe :</i>	<i>liste \times nœud</i>	\rightarrow	<i>V</i>



File d'attente ou liste



■ Réécriture en utilisant les méthodes définies sur la liste :

```
fonction Recherche (problème, stratégie) res : nœud ou échec
    liste-nœuds ← initListe
    liste-nœuds ← insertListe (créerNœud (État-initial [problème]))
    Répéter
        si listeVide(liste-nœuds) alors res ← échec
        sinon nœud ← extratListe(liste-nœuds )
    fsi
    si test-but (problème, nœud) alors res ← nœud
        sinon liste-nœuds ←
            inserqListe(liste-nœuds, nœud, Operateurs [problème]))
    fsi
    frépéter
```



■ Algorithme général de recherche : autre écriture

Fonction RechercheGénérale(étatInitial, ensemble_opérateurs)

S = ConstruireVide()

Insérer (S, Nœud(étatInitial))

tant que non Vide(S) **faire**

NœudCourant = Extraire (S)

Si Test_But(NœudCourant)=**vrai**

alors

Détruire (S)

retourne NœudCourant

sinon

pour chaque op **dans** ensemble_opérateurs **faire**

x = Successeur(NœudCourant, op)

si Valide(x) **alors** Insérer(S, x) **fin si**

fin pour

fin si

fin tant que

Détruire (S)

retourne vide

Fin





File d'attente ou liste



■ Structure de donnée

- S : construite vide au début et détruite à la fin

■ Fonctions

- Insérer (S, x) : introduit l'élément x dans la structure S
- Extraire (S) : fournit l'élément qui se trouve "en face" de la structure S et l'élimine aussi
- Vide(S) : fonction booléenne qui retourne vrai si la structure S est vide
- Test_But(NœudCourant) : fonction booléenne qui retourne vrai si le NœudCourant représente l'état solution ;
- Successeur(NœudCourant, op) : fonction qui retourne le nœud successeur du NœudCourant obtenu suite à l'application de l'opérateur op; si l'opérateur op ne peut pas être appliqué au NœudCourant alors la fonction retournera nul
- Valide(x) : fonction booléenne qui retourne vrai si x n'est pas nul et si x est un nœud qui peut être inséré dans la structure S.



File d'attente



- Les stratégies de recherche diffèrent les unes des autres selon
 - l'ordre dans lequel les nœuds sont "étendus"
 - (c.à.d les successeurs sont produits et mis dans la file d'attente)
 - ➔ rôle de la fonction :
 - *étendreListe*

Classes d'algorithmes de recherche



- On distingue deux classes d'algorithmes de recherche :
 1. algorithmes non-informés ou "aveugles" (brute force)
 - qui réalisent une recherche exhaustive, sans utiliser aucune information concernant la structure de l'espace d'états pour optimiser la recherche
 2. algorithmes informés ou heuristiques
 - qui utilisent des sources d'information supplémentaires
 - ces algorithmes parviennent ainsi à des performances meilleures

Stratégies aveugles vs stratégies heuristiques



■ Exemple : taquin-8

8	2	
3	4	7
5	1	6

ETAT
N1

Pour une stratégie aveugle, N1 et N2 sont simplement deux noeuds (à une certaine profondeur de l'arbre de recherche)

1	2	3
4	5	
7	8	6

ETAT
N2

1	2	3
4	5	6
7	8	

But

Stratégies aveugles vs stratégies heuristiques



■ Exemple : taquin-8

8	2	
3	4	7
5	1	6

ETAT
N1

1	2	3
4	5	
7	8	6

ETAT
N2

Pour une stratégie heuristique comptant le nombre de plaquettes mal placées, N2 est plus prometteur que N1

1	2	3
4	5	6
7	8	

But



Méthodes de recherche aveugles



■ Types de méthodes

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

Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)

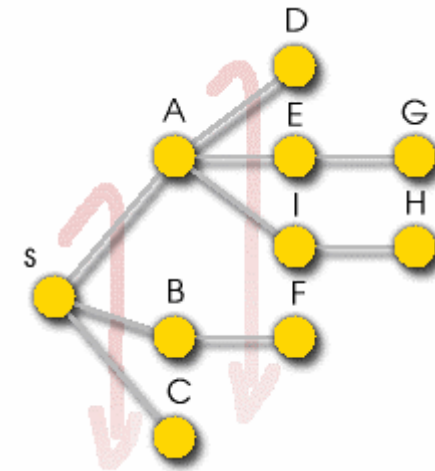


■ Stratégie :

- Explorer tous les fils d'un nœud d'abord
→ étendre le nœud le moins profond

■ L'algorithme

- cherche à épuiser la liste des sommets proches de **s** avant de poursuivre l'exploration de l'arbre
- s'arrête dès la rencontre d'un **état but**
- au cas où la solution n'est pas trouvée, le processus est itéré au niveau de profondeur suivant



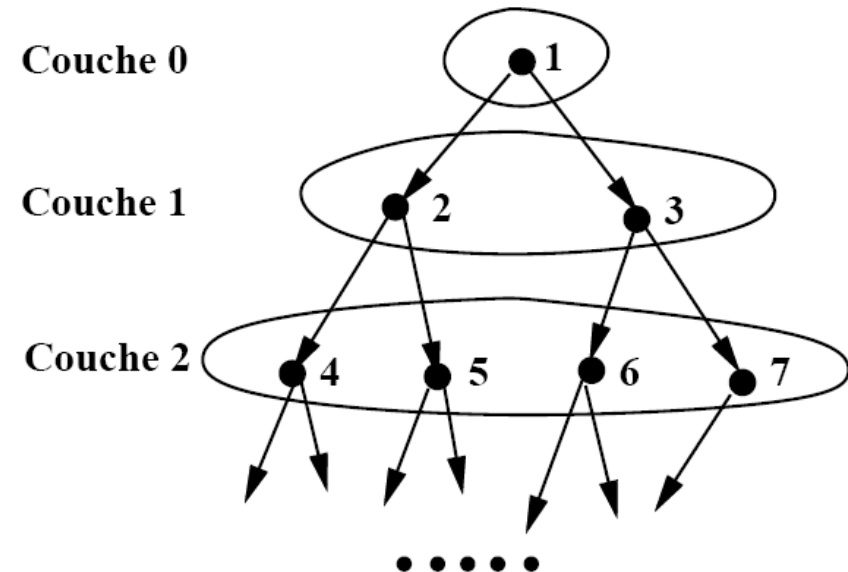
Stratégies aveugles

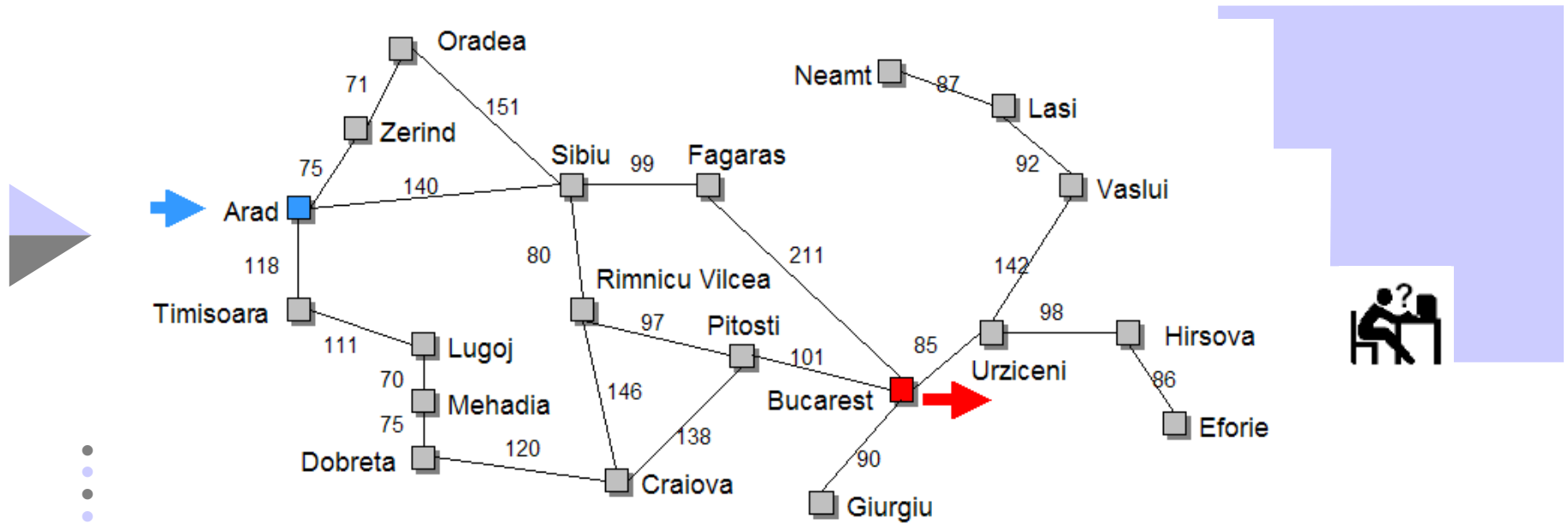
Recherche en largeur d'abord ou BFS (*Breadth First Search*)



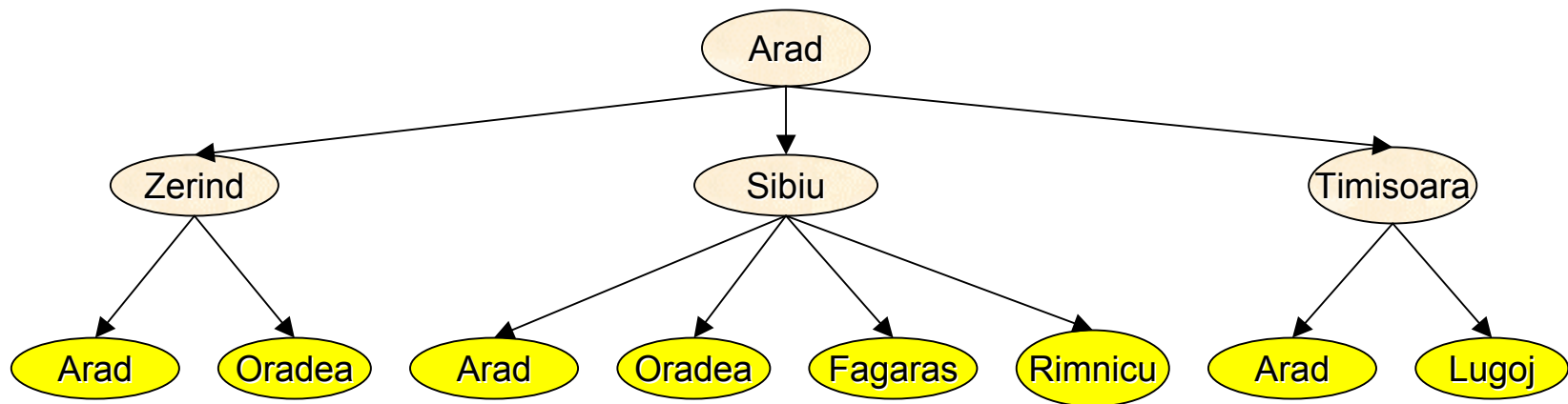
■ Stratégie :

- Parcourt l'arbre de recherche couche par couche
- Trouve toujours le chemin le plus court
- Exige beaucoup de mémoire pour stocker toutes les alternatives à toutes les couches

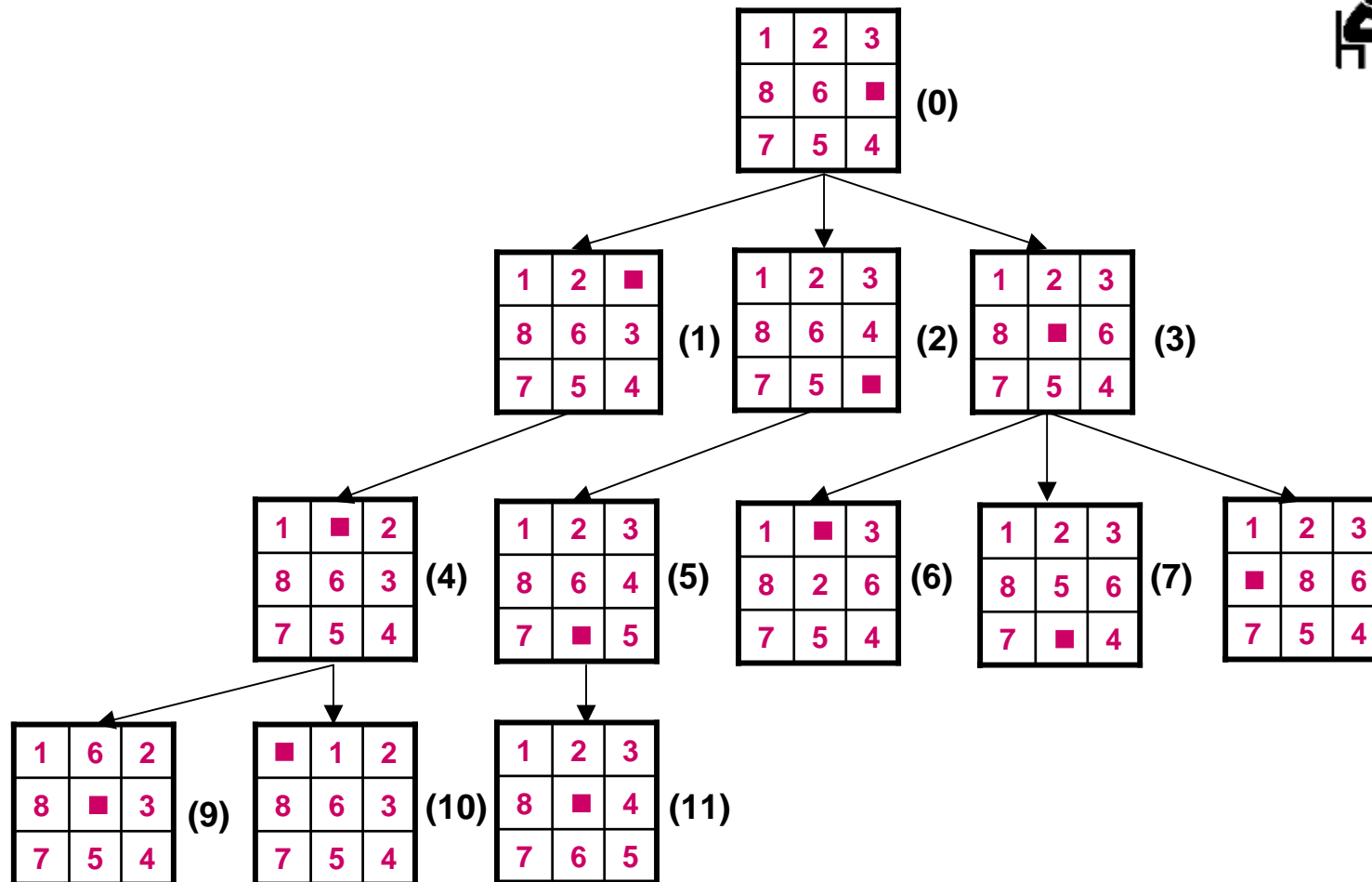




■ Exemple



BSF : exemple du taquin



Stratégies aveugles

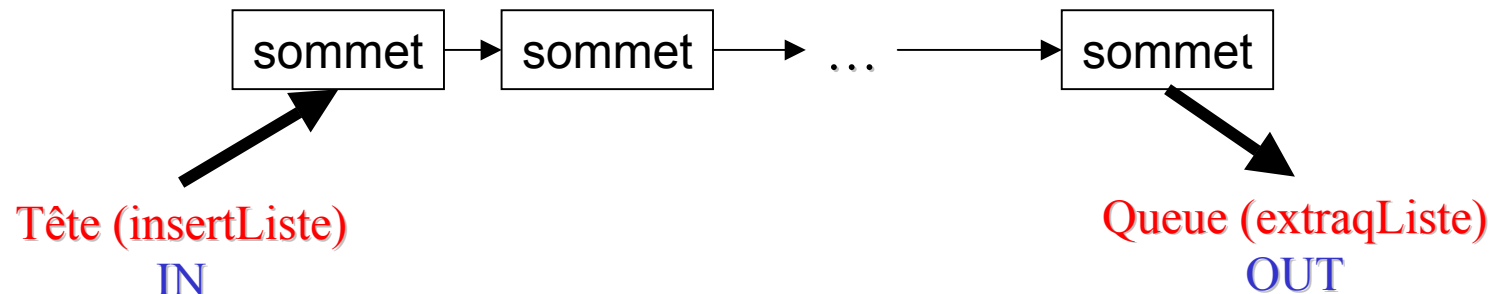
Recherche en largeur d'abord ou BFS (*Breadth First Search*)



■ Structure de données :

– File : liste FIFO FirstIn/FirstOut

- (premier entré/premier sorti)
- structure de données telle que c'est le plus ancien des nœuds mémorisés au niveau de profondeur p qui est développé en priorité pour engendrer les nœuds de niveau $p+1$
- L'état initial est traditionnellement situé à la profondeur 0



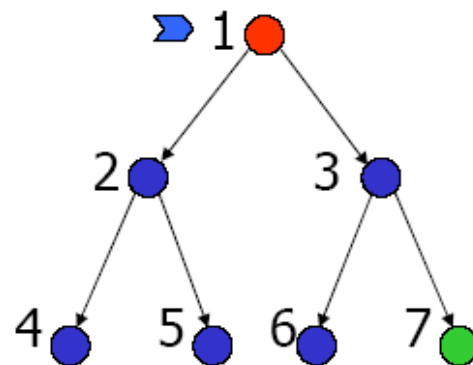


Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



Les nouveaux nœuds sont insérés à la fin de la file d'attente



Entrée de : 1

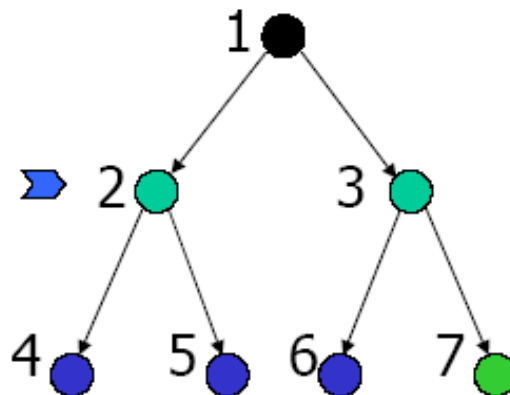
File : 1

Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



Les nouveaux nœuds sont insérés à la fin de la file d'attente



Traitement du : 1, entrée de 2, 3

File :

3	2
---	---

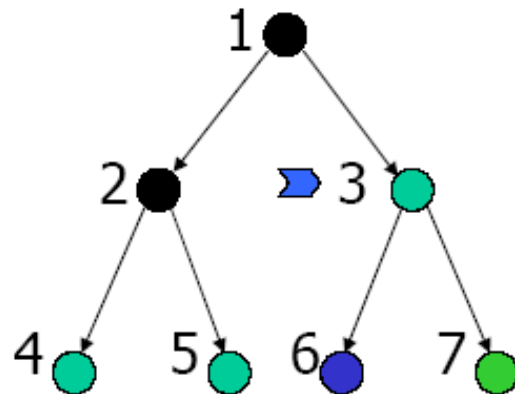


Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



Les nouveaux nœuds sont insérés à la fin de la file d'attente



Traitement du : 2, entrée de 4, 5

File :

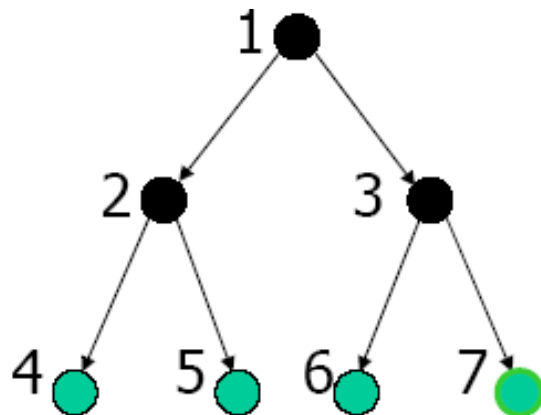
5	4	3
---	---	---

Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



Les nouveaux nœuds sont insérés à la fin de la file d'attente



Traitement du : 3, entrée de 6, 7

File :

7	6	5	4
---	---	---	---



Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



■ Opérations sur la file :

<i>initFile :</i>	<i>nœud</i>	\rightarrow	<i>file</i>
<i>insérerFile :</i>	<i>file</i> \times <i>nœud</i>	\rightarrow	<i>file</i>
<i>fileVide :</i>	<i>file</i>	\rightarrow	<i>booléen</i>
<i>extraFile :</i>	<i>file</i>	\rightarrow	<i>nœud</i>
<i>étendreFile</i>	<i>file</i> \times <i>nœud</i>	\rightarrow	<i>file</i>

Où :

insérerFile = *insertListe*

extraFile = *extraqListe*



Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



```
fonction BFS (problème, file) res : nœud ou échec
  file-nœuds ← initFile (créer-nœud (État-initial [problème]))
  Répéter
    si fileVide(file-nœuds) alors res ← échec
    sinon nœud ← extraFile(file-nœuds )
  fsi
  si test-but (problème, nœud) alors res ← nœud
  sinon file-nœuds ← insérerFile (file-nœuds,
    étendreFile(file-nœuds, nœud, Operateurs [problème]))
  fsi
fRépéter
```



Stratégies aveugles

Recherche en largeur d'abord ou BFS (*Breadth First Search*)



Autre manière de l'exprimer

On crée un arbre T recouvrant en profondeur, au fur et à mesure

Fonction BFS (G : graphe ; x : point de départ)

marquer (x) ;

mettre x dans une file F ;

tant que (file pas vide) **faire**

 enlever le premier sommet y de la file ;

 traiter (y) ;

pour toutes les arêtes (y , z) avec z non marqué **faire**

 marquer (z) ;

 ajouter arête (y , z) dans T ;

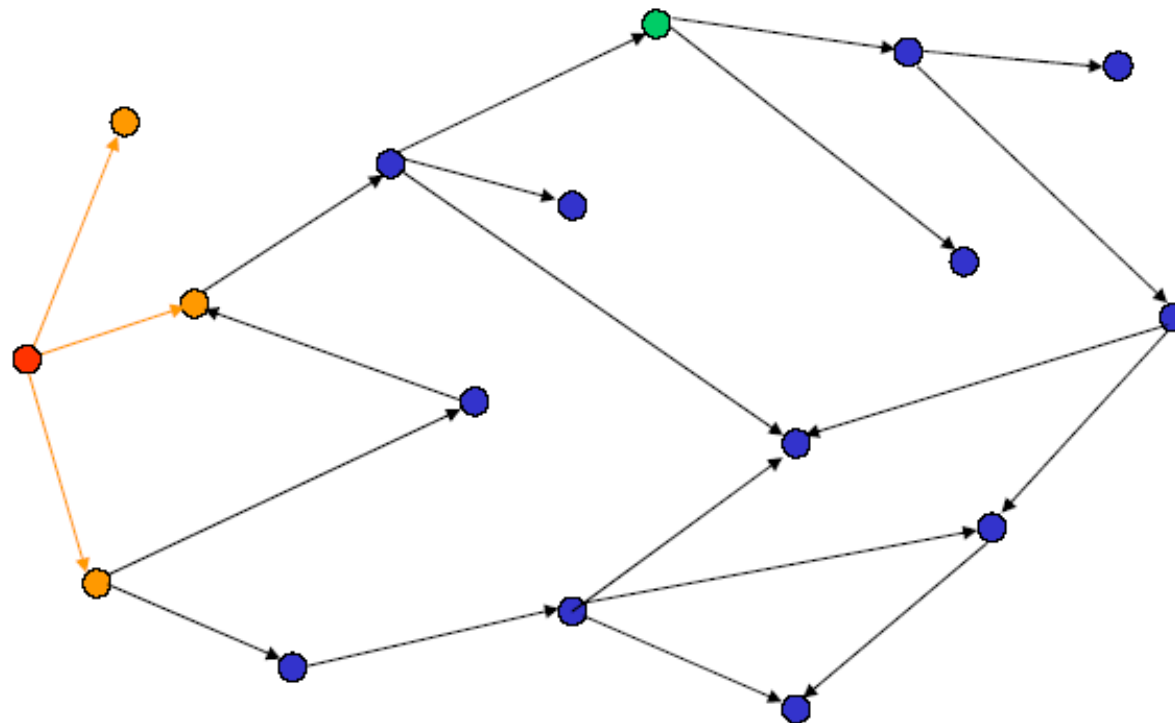
 mettre z dans F ;

fpour

ftque

Stratégies aveugles

Recherche en largeur d'abord ou BFS
(*Breadth First Search*)





Stratégies aveugles

Recherche en largeur d'abord ou BFS



- Complétude :

- Oui (si facteur de branchement b est fini)

- Complexité en temps :

$$1 + b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b)$$
$$= O(b^{d+1}) \text{ (exponentielle en } d\text{)}$$

- Complexité en espace :

$O(b^d)$ (il faut garder tous les nœuds en mémoire)

- Optimalité :

Oui (si coût unitaire par étape) en général pas optimal

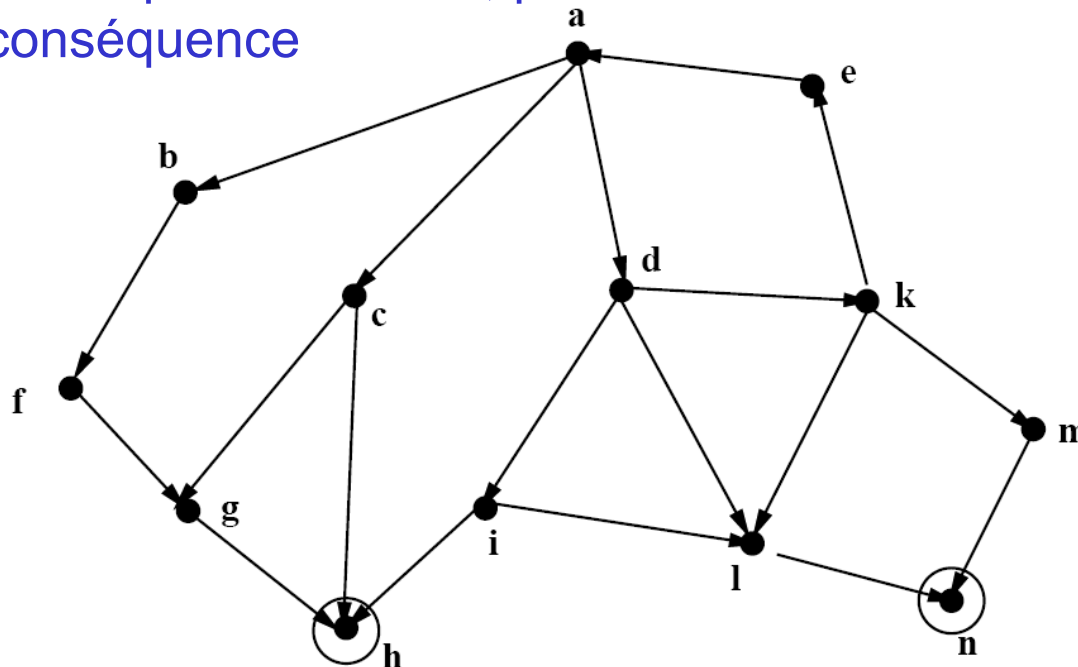
Stratégies aveugles

Recherche en largeur d'abord ou BFS



■ Exercice

1. Quel est le résultat de la BFS appliquée à cet exemple ?
2. Traduire la file par un tableau, puis réécrire la fonction BFS en conséquence





Stratégies aveugles

Recherche en largeur d'abord ou BFS



- Exercice : programmation du jeu du taquin
 - Proposez une structure de donnée pour représenter un état du taquin
 - Proposez une fonction permettant de tester l'état but
 - Proposer une structure de donnée pour représenter la file des états



Stratégies aveugles

Recherche en coût uniforme



■ Variante de la BFS

- Utilise une fonction coût g (=distance, importance, etc.) associée aux nœuds
- Similitude avec la BFS
 - La BFS trouve le nœud-solution le moins profond
 - La recherche en coût uniforme, quant à elle :
 - étend systématiquement le nœud de coût **le plus faible**, trouvé par une fonction de coût $g(n)$: n nœud
 - le coût d'un chemin ne doit jamais décroître, c.à.d qu'il ne doit pas y avoir de coûts partiels négatifs
 - $\forall n \ g(\text{Successeur}(n)) \geq g(n)$

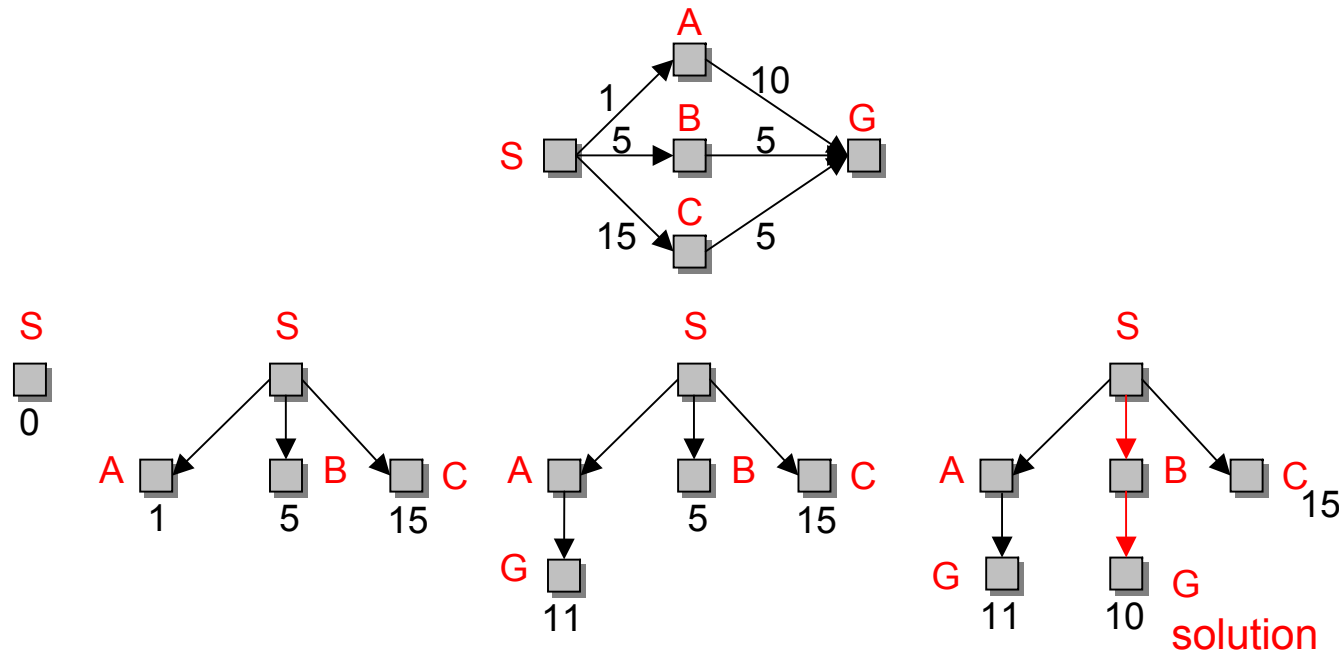
Stratégies aveugles

Recherche en coût uniforme



■ Exemple

- La solution sélectionnée est celle avec un coût $g = 10$

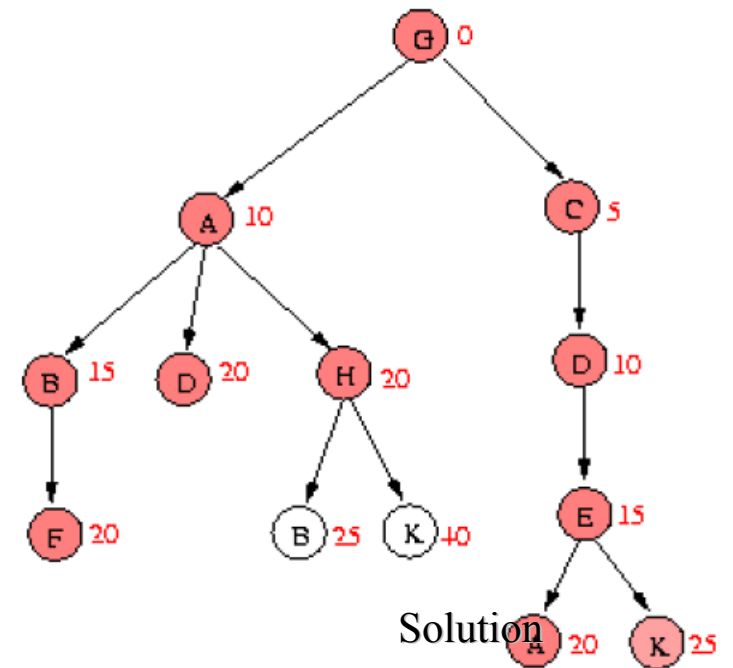
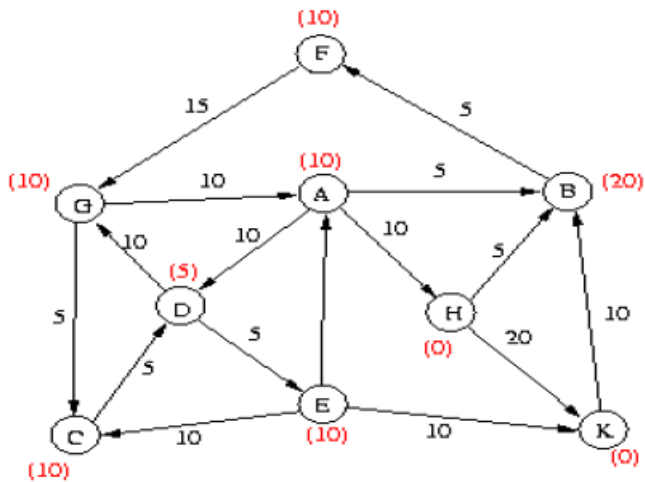


Stratégies aveugles

Recherche en coût uniforme



- Autre exemple : départ de G, arriv





Stratégies aveugles

Recherche en coût uniforme



■ Exercice

- Appliquer la recherche en coût uniforme au graphe routier roumain, en considérant les distances entre les villes comme des coûts :
 - dessiner le graphe d'état,
 - trouver le chemin qui mène à Bucarest ainsi que son coût
 - comment fonctionne insérerFile dans ce cas



Stratégies aveugles

Recherche en coût uniforme



- Complétude :
 - Oui
- Complexité en temps
 - $O(b^d)$
- Complexité en espace :
 - $O(b^d)$
- Optimalité :
 - Oui

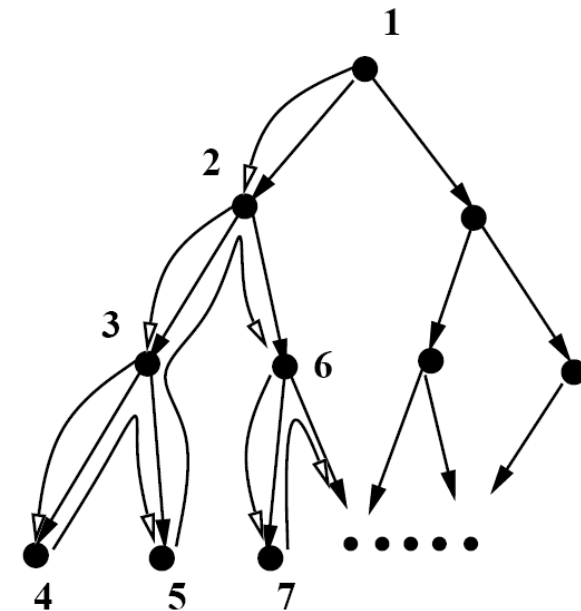
Stratégies aveugles

Recherche en profondeur d'abord Depth First



■ Stratégie :

- étendre le nœud le plus profond
- Dans l'exploration, l'algorithme cherche à aller très vite "profondément" dans le graphe, en s'éloignant du sommet s de départ
- La recherche sélectionne à chaque étape un sommet voisin du sommet marqué à l'étape précédente
- En cas d'échec (branche conduisant à un cas d'échec), on revient en arrière au niveau du père, et si possible, on recherche un autre de ses successeurs





Stratégies aveugles

Recherche en profondeur d'abord



- **Stratégie**

- expansion du premier nœud trouvé jusqu'à ce qu'il n'y ait plus de successeurs
- retour en arrière (backtrack) : retour à un niveau supérieur et essai de la prochaine possibilité

- **Avantage :**

- peu de mémoire requise : liste des nœuds "ouverts" et de leurs successeurs encore non explorés

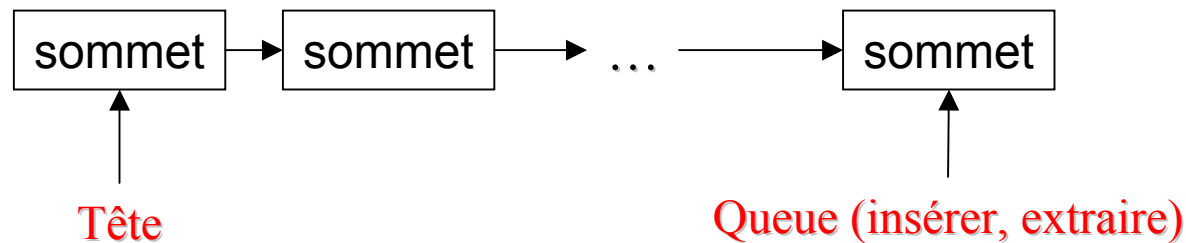


Stratégies aveugles

Recherche en profondeur d'abord



- Liste = pile : liste LIFO *LastIn/FirstOut* (dernier entré/premier sorti)
 - Cette structure permet de mémoriser les nœuds en suspens de sorte que c'est le nœud le plus récemment mémorisé qui est traité en priorité





Stratégies aveugles

Recherche en profondeur d'abord



■ Opérations sur la file :

<i>initPile :</i>	<i>nœud</i>	\rightarrow	<i>pile</i>
<i>insérerPile :</i>	<i>pile</i> \times <i>nœud</i>	\rightarrow	<i>pile</i>
<i>pileVide :</i>	<i>pile</i>	\rightarrow	<i>booléen</i>
<i>extrairePile:</i>	<i>pile</i>	\rightarrow	<i>nœud</i>
<i>étendrePile</i>	<i>pile</i> \times <i>nœud</i>	\rightarrow	<i>pile</i>

Où :

insérerPile = *inserqListe*

extraPile = *extraqListe*



Stratégies aveugles

Recherche en profondeur d'abord



```
fonction DFS (problème, pile) res : nœud ou échec
  pile-nœuds ← initPile (créer-nœud (État-initial [problème]))
  Faire pour
    si pileVide(file-nœuds) alors res ← échec
    sinon nœud ← extraPile(pile-nœuds )
  fsi
  si test-but (problem, nœud) alors res ← nœud
  sinon pile-nœuds ← inserPile (pile-nœuds ,
    étendrePile(pile-nœuds, nœud , Operateurs [problème]))
  fsi
fpour
```



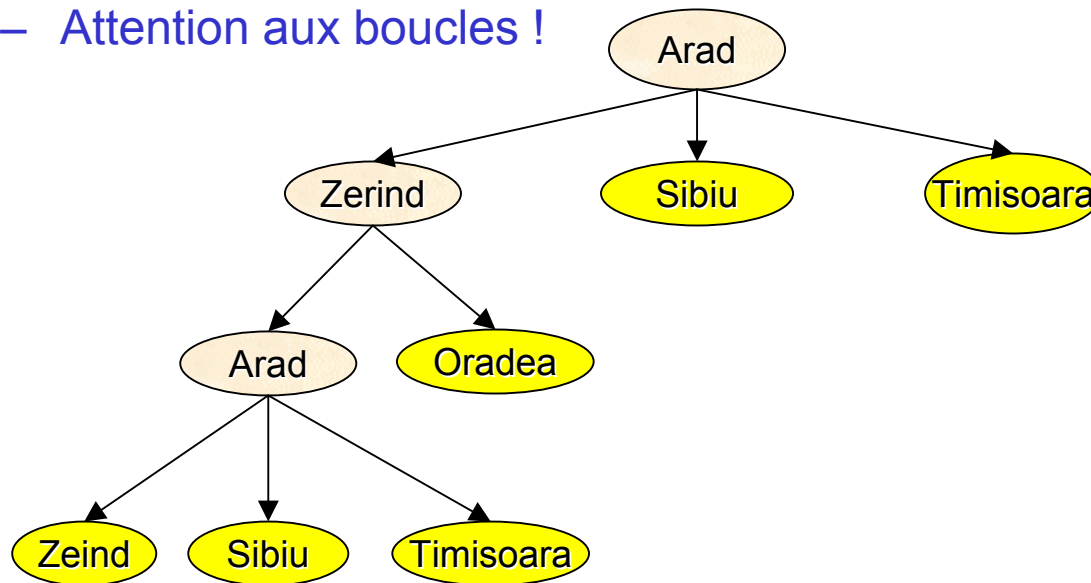
Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation :

- L'algorithme de recherche progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S
- Attention aux boucles !



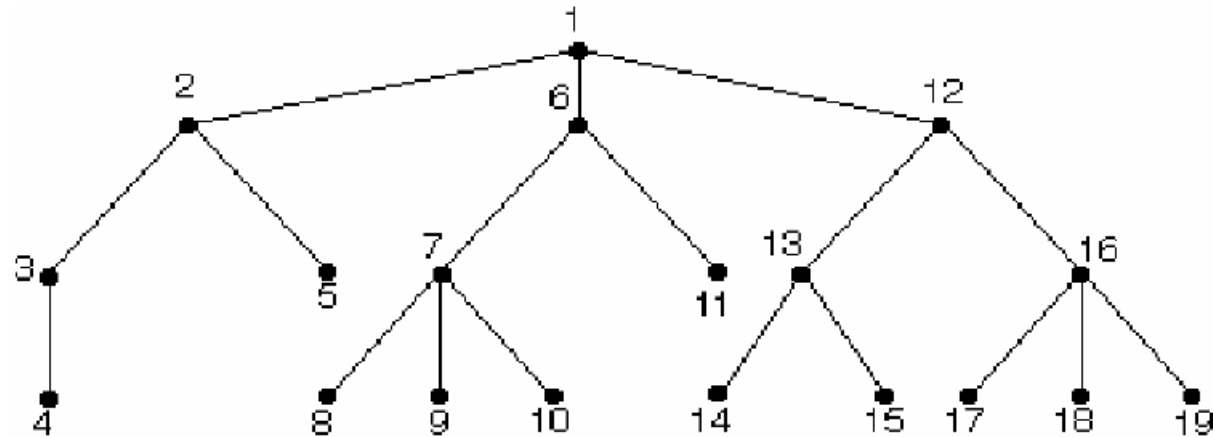


Stratégies aveugles

Recherche en profondeur d'abord



- Implémentation :
 - Les nœuds sont numérotés dans l'ordre de leur exploration



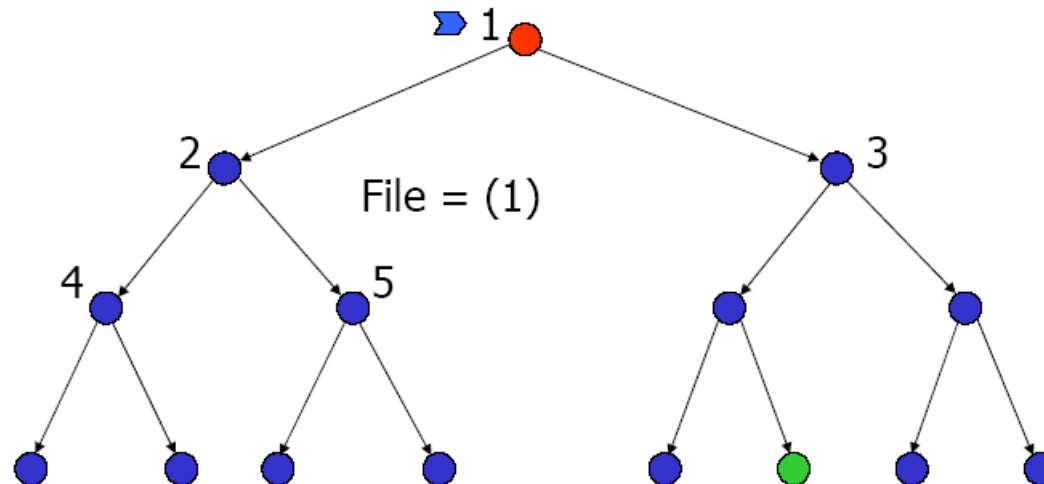


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



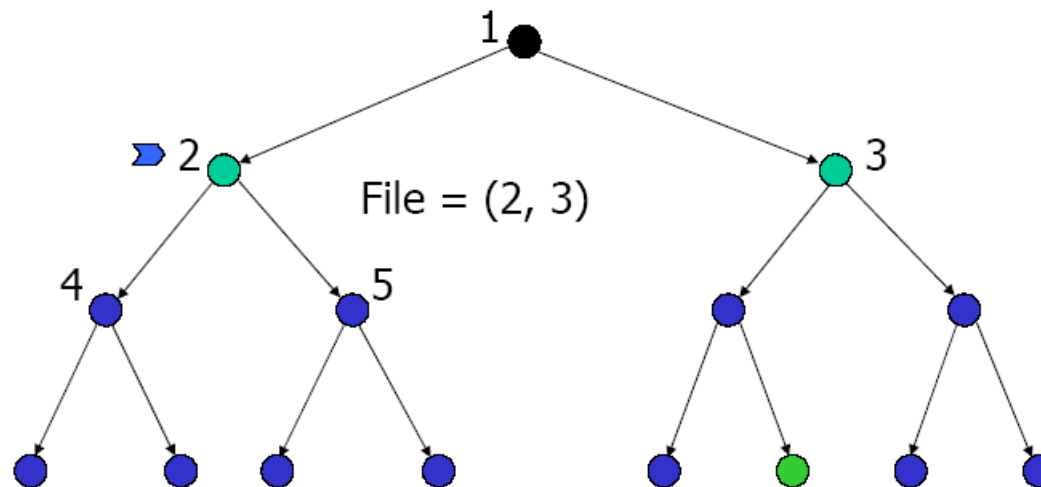


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



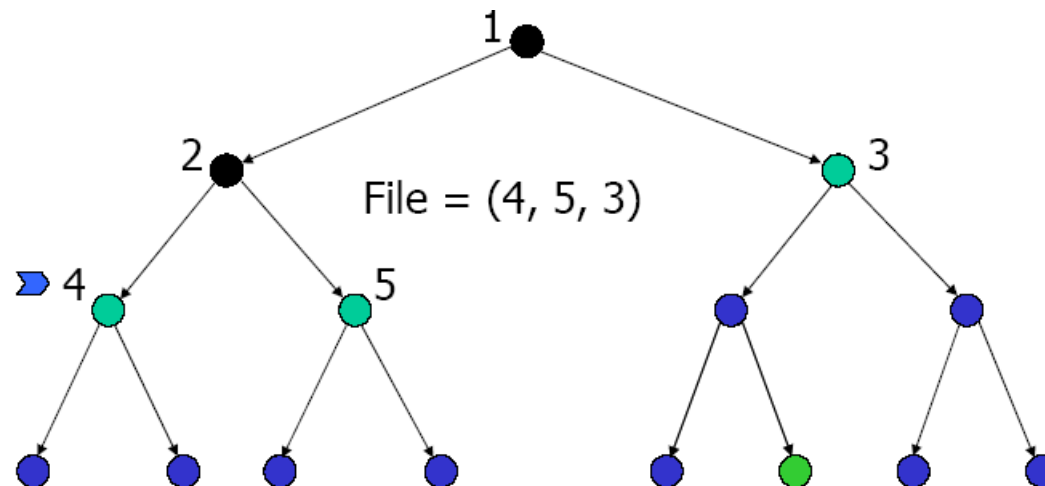


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



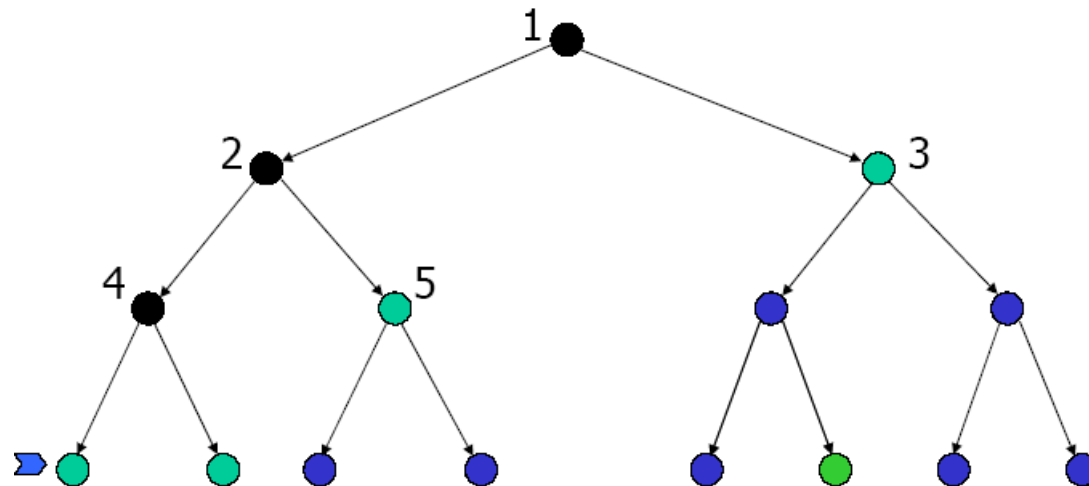


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



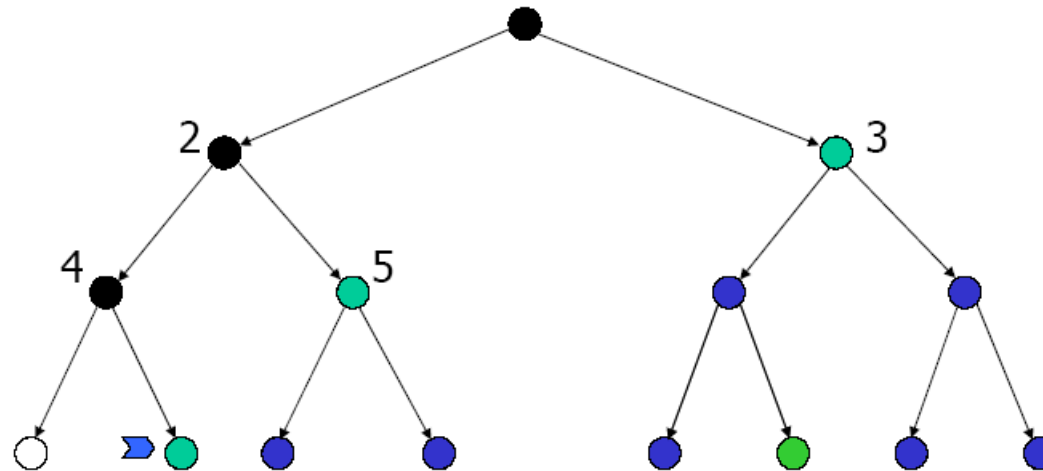


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



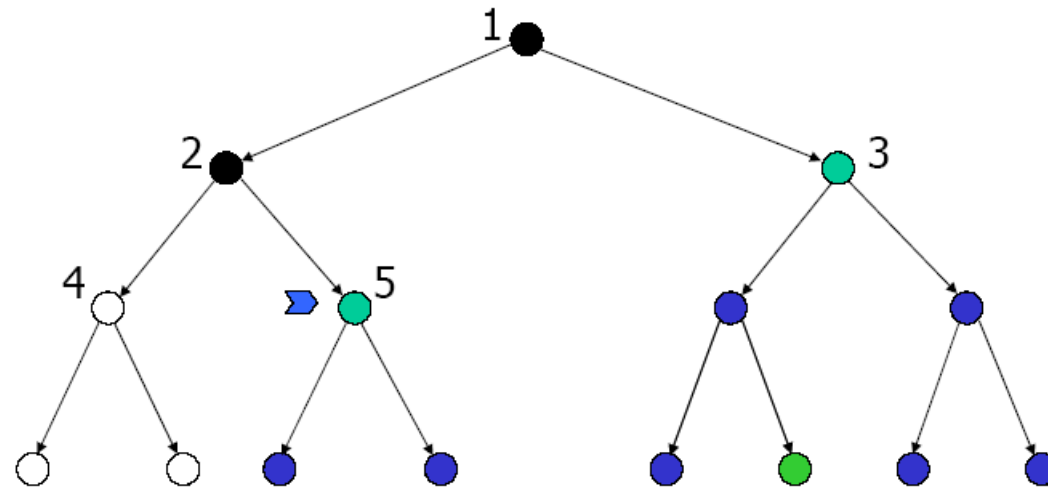


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



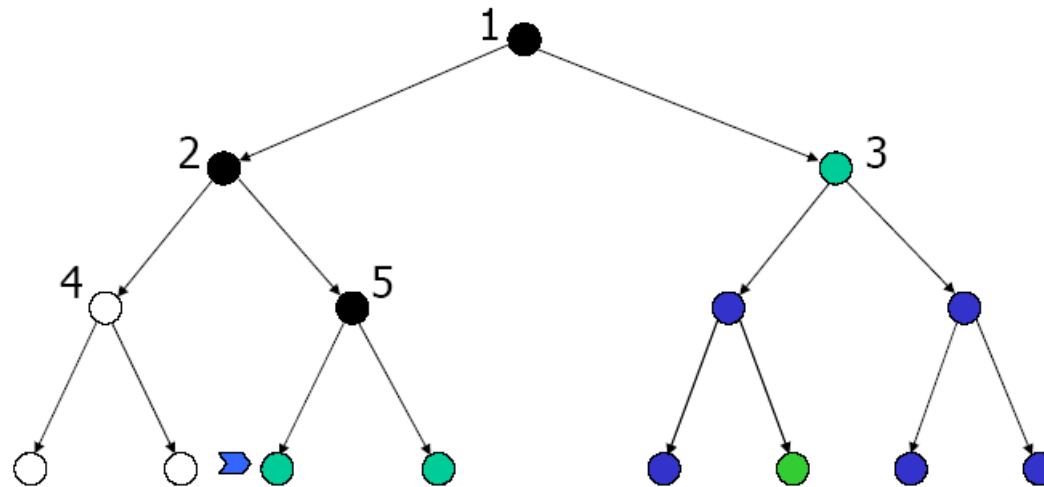


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



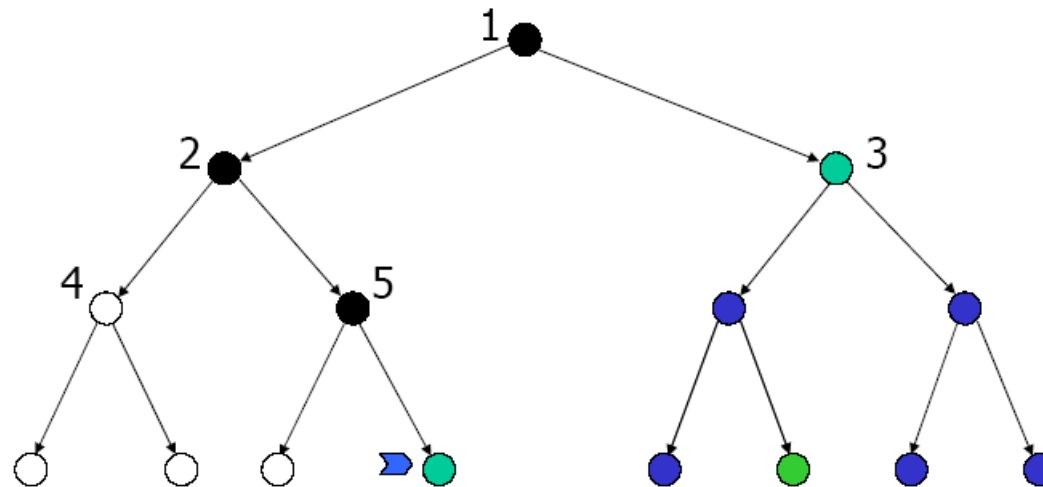


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



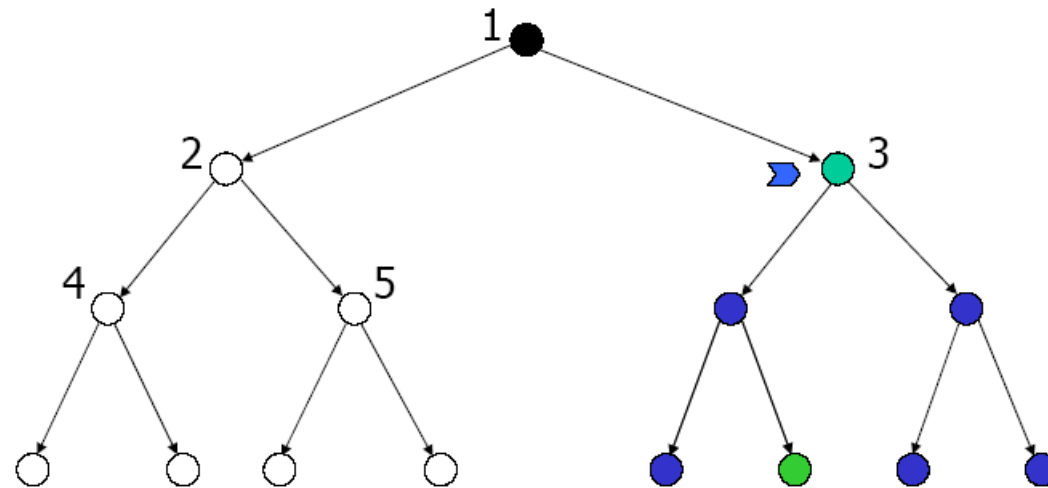


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



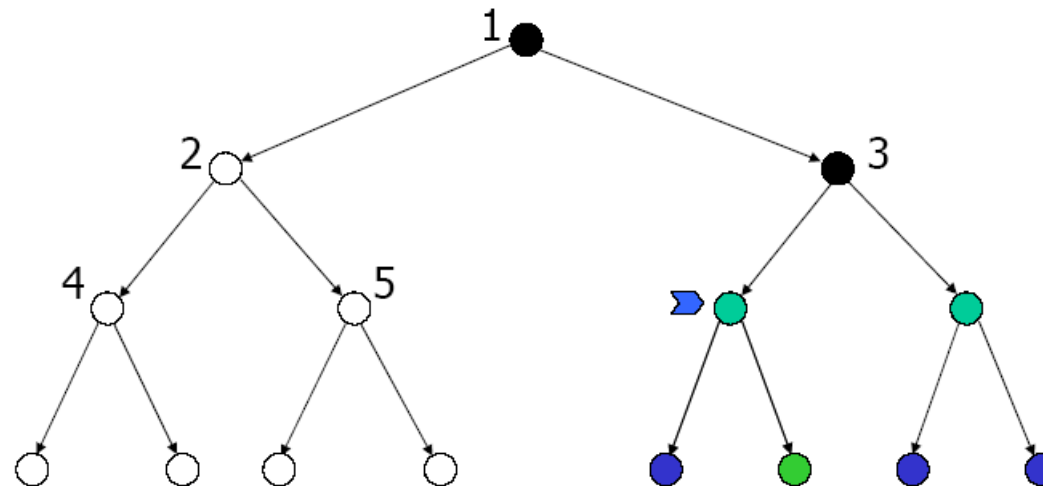


Stratégies aveugles

Recherche en profondeur d'abord



■ Implémentation : Exemple



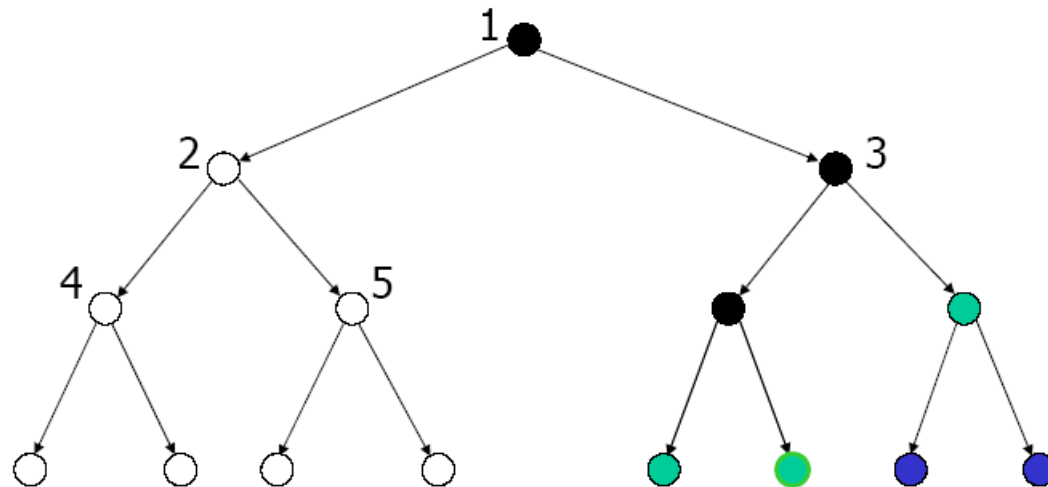


Stratégies aveugles

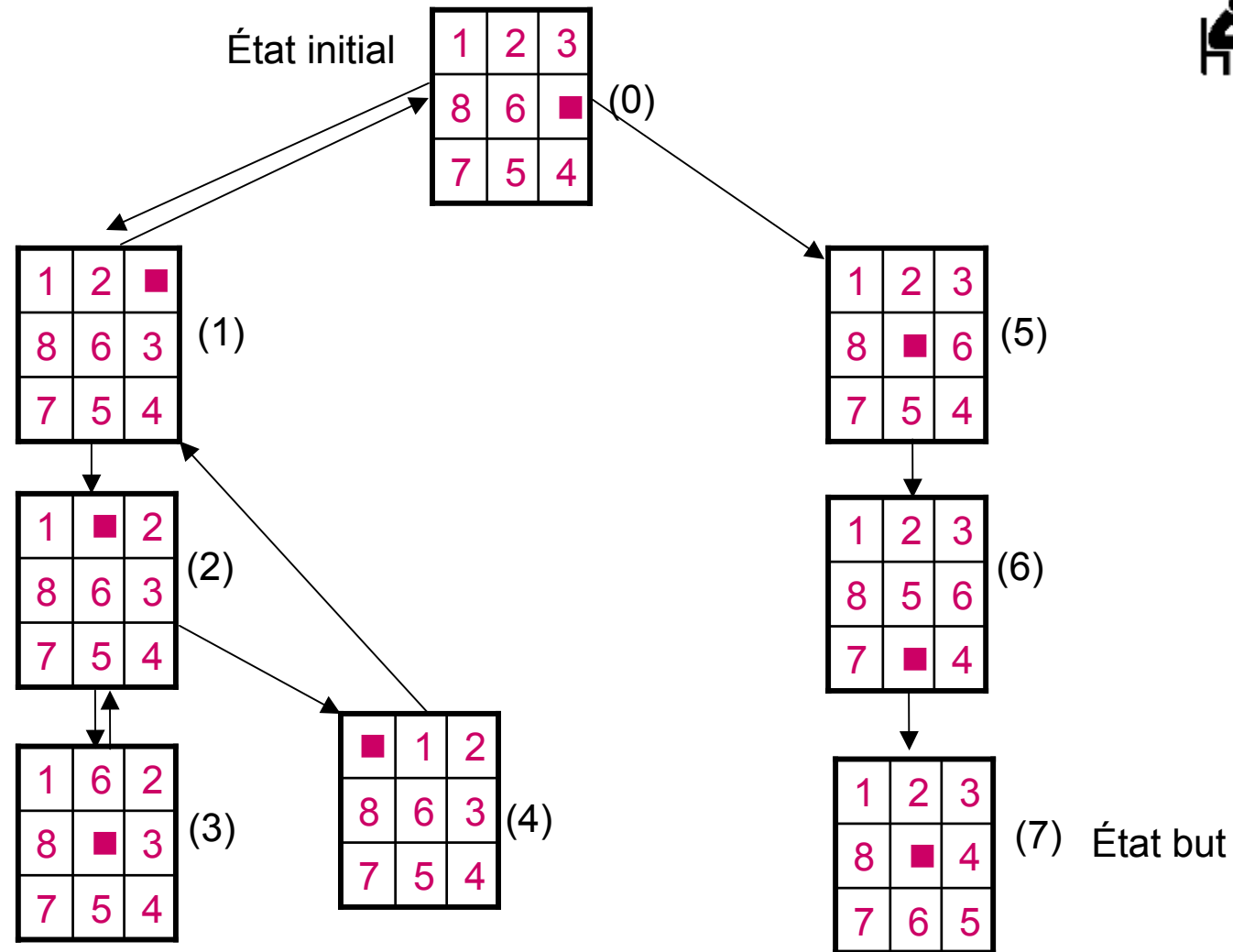
Recherche en profondeur d'abord



■ Implémentation : Exemple



Exemple : taquin





Stratégies aveugles

Recherche en profondeur d'abord



- Problème des 8-reines par la DSF
 - Regardons comment il faut s'y prendre :
 - On a notre échiquier vide (état initial)
 - On place la 1^{ère} dame (c'est pas trop dur, il vous faut choisir parmi 64 cases !)
 - au-delà, vous placez la 2^{ème} dame en tenant compte des règles du problème (une dame se déplace en ligne, en colonne ou en diagonale)
 - puis vous placez la 3^{ème} dame avec le même raisonnement sur les règles
 - et ainsi de suite jusqu'à ce que vous ayez vos 8 dames sur l'échiquier



Stratégies aveugles

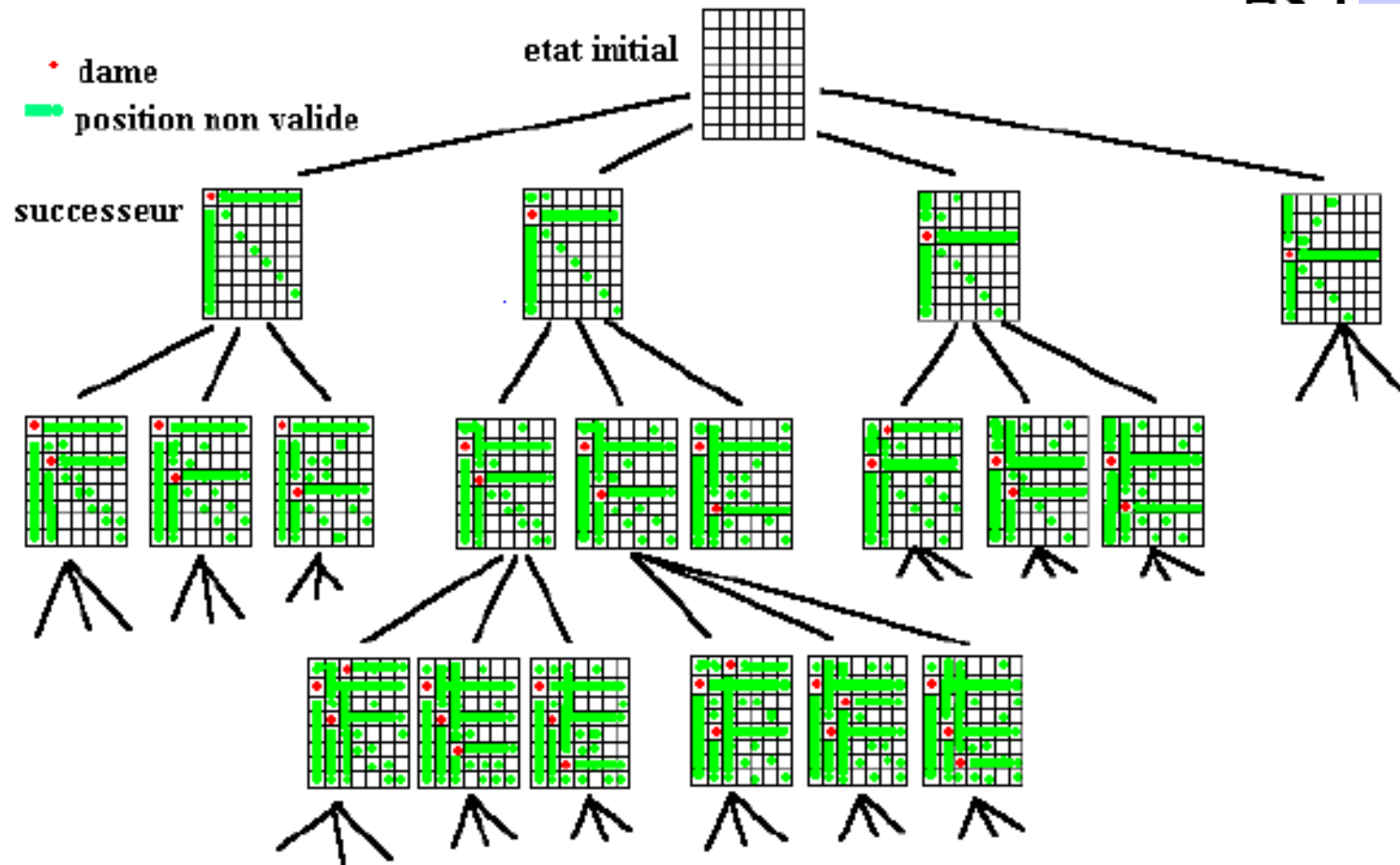
Recherche en profondeur d'abord



■ Problème des 8-reines par la DSF (suite 1)

- Seulement, il se peut que vous soyez bloqué au bout de la 5^{ème} ou 6^{ème} dame
- Dans ce cas, que faites vous ?
 - Vous revenez en arrière. Pourquoi ?
 - Car à un moment donné, vous avez placé une dame qui vous a donné un échiquier (état intermédiaire) qui ne conduira jamais à une solution (état final)
 - Pour représenter les états intermédiaires séparant l'état initial de l'état final (ou des états finaux), on se sert d'un arbre de recherche
 - L'arbre de recherche est utilisé pour pouvoir garder en mémoire les différentes étapes de la recherche

Exemple : les 8-reines





Stratégies aveugles

Recherche en profondeur d'abord



■ Problème des 8-reines par la DSF (suite 2)

- Vous voyez que la recherche est volumineuse !!!
- On voit donc que l'on place petit à petit nos dames. Il est évident que chaque nœud de l'arbre n'a pas obligatoirement 3 ou 4 fils
 - Par exemple, l'état initial a 64 fils !!!
- Il se peut aussi qu'une branche de l'arbre ne conduise jamais à une solution
- On voit donc qu'on place une dame, puis aussitôt après on essaie d'en placer une deuxième,
- puis une troisième...
- Ainsi dès que l'on prend une direction, on essaie de la développer, de la creuser, de l'explorer en profondeur
- D'où le nom de cette méthode de recherche «la recherche en profondeur d'abord»



Stratégies aveugles

Recherche en profondeur d'abord



■ Exercice

- Proposez une structure de donnée pour représenter l'état de l'échiquier à un moment de la résolution du problème
 - A-t-on besoin d'une seule pile ou de 8 piles (une par reine) pour gérer les retours-arrières ?
- Proposez un algorithme informel basé sur cette structure permettant de le résoudre par la DFS



Stratégies aveugles

Recherche en profondeur d'abord



- Complétude : Non
 - échoue dans les espaces infinis
 - complet dans les espaces finis
- Complexité en temps
 - $1 + b + b^2 + b^3 + \dots + b^m = O(b^m)$ terrible si m est beaucoup plus grand que d
- Complexité en espace :
 - $O(b \cdot m)$ ou $O(m)$ linéaire!
- Optimalité : non
- Discussion : besoins modestes en espace
 - pour $b = 10$, $d = 12$ et 100 octets/nœud :
 - recherche en profondeur a besoin de 12 Koctets
 - recherche en largeur a besoin de 111 Terra-octets

□ * 10^{10} !!!



Stratégies aveugles

Recherche en profondeur limitée



■ Problème :

- Le programme (depth first) peut être perdu dans une recherche en profondeur infinie ! espace d'états infini
 - rater le nœud but en explorant indéfiniment un espace d'états infini tout en s'éloignant du but (pb des 8 reines)

■ Solution :

- Pour éviter des branches infinies et non cycliques, on limite la profondeur de la recherche :
 - `depthfirst2(Node, Solution, Maxdepth)`
 - La recherche ne peut pas aller au delà de la limite de la profondeur
 - Décrémenter Maxdepth pendant chaque appel récursif
 - $\text{Maxdepth} \geq 0$

■ Algorithme

Fonction RechercheEnProfondeurLimitée (*problème*,Maxdepth)
est
 pile-nœuds ← **initPile** (créer-nœud (État-initial [*problème*]))
tant que non pilevide(*pile-nœuds*) **faire**
 (NœudCourant, ProfondCourante) = ExtrairePile (*pile-nœuds*)
 si Test_But(NœudCourant)=**vrai**
 alors
 retourne NœudCourant
 sinon
 si ProfondCourante < Maxdepth
 alors
 pour chaque op **dans** ensemble_opérateurs **faire**
 x = Successeur(NœudCourant,op)
 si Valide(x)
 alors insérerPile(*pile-nœuds* ,(x. ProfondCourante+1))
 fsi
 fin pour
 fsi
 fin tantque
 retourne vide
Fin



Stratégies aveugles

Propriété 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
 - 3 possibilités :
 - solution
 - échec
 - absence de solution dans les limites de la recherche

Stratégies aveugles

Recherche par approfondissement itératif (IDS)



- Le problème avec la recherche en profondeur limitée est de fixer la bonne valeur de L
- approfondissement itératif est de répéter pour toutes les valeurs possibles de $L = 0, 1, 2, \dots$
 - 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

Stratégies aveugles

Recherche par approfondissement itératif (IDS)



- Approfondissement itératif $L=0$

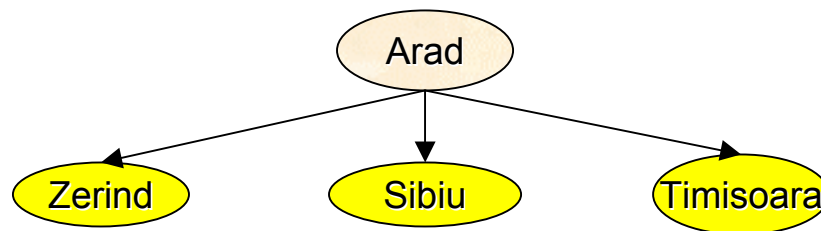
Arad

Stratégies aveugles

Recherche par approfondissement itératif (IDS)



■ Approfondissement itératif L=1

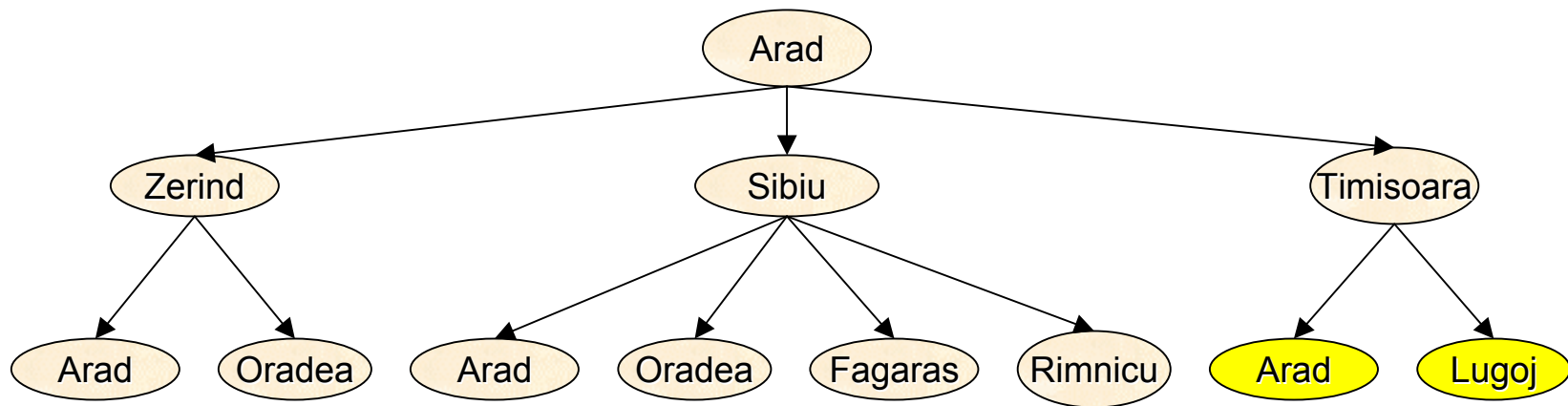


Stratégies aveugles

Recherche par approfondissement itératif (IDS)



■ Approfondissement itératif L=2

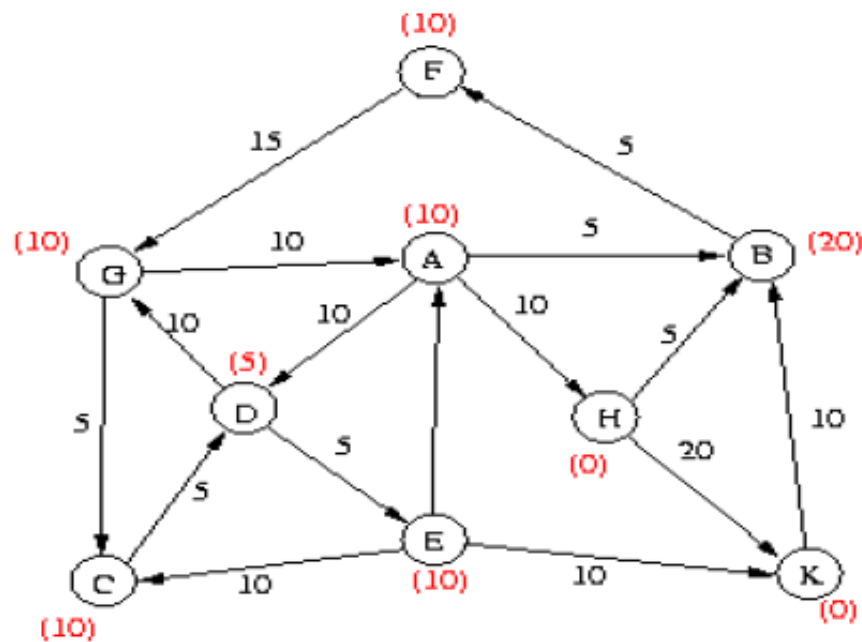


Stratégies aveugles

Recherche par approfondissement itératif (IDS)



■ Autre exemple



Première itération : limite = 0

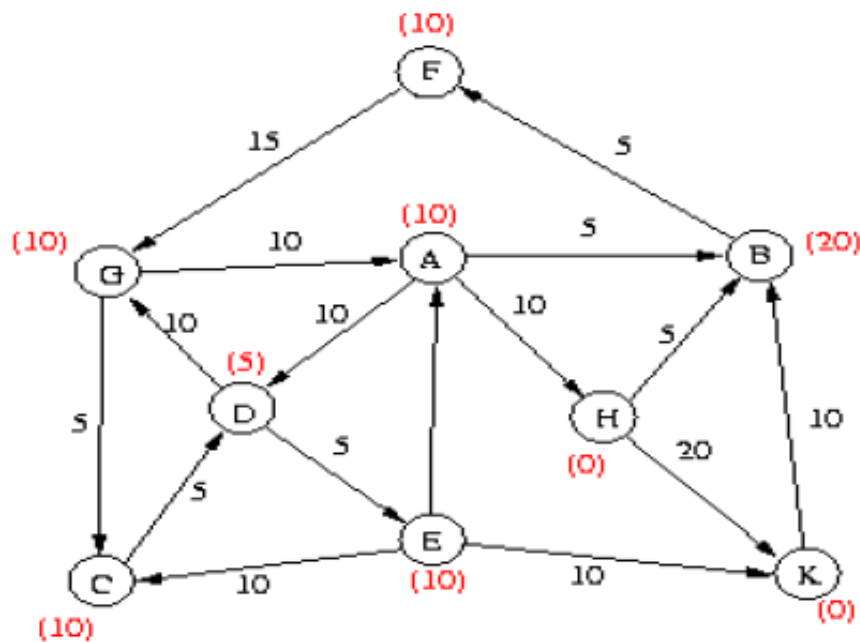


Stratégies aveugles

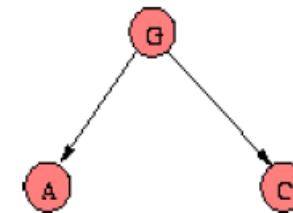
Recherche par approfondissement itératif (IDS)



■ Autre exemple



Deuxième itération : limite = 1

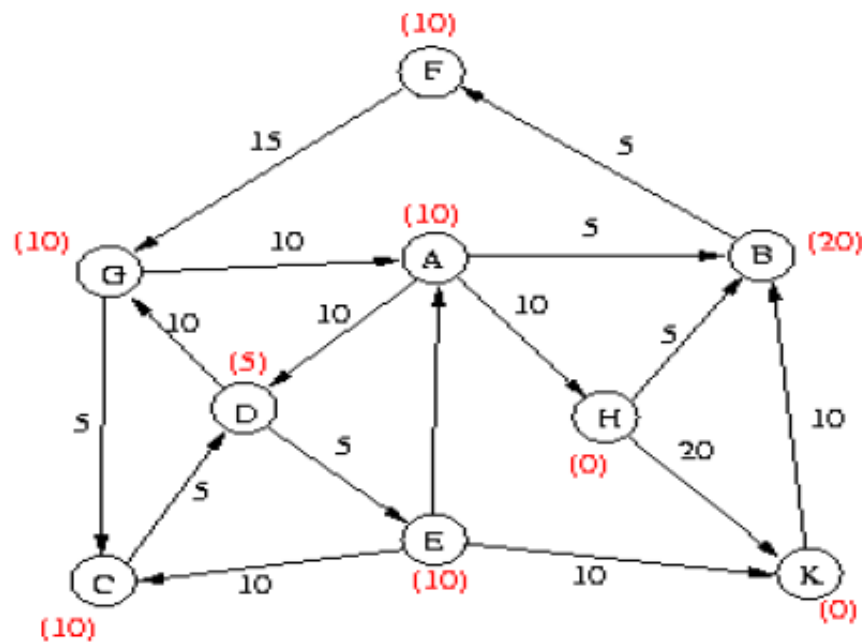


Stratégies aveugles

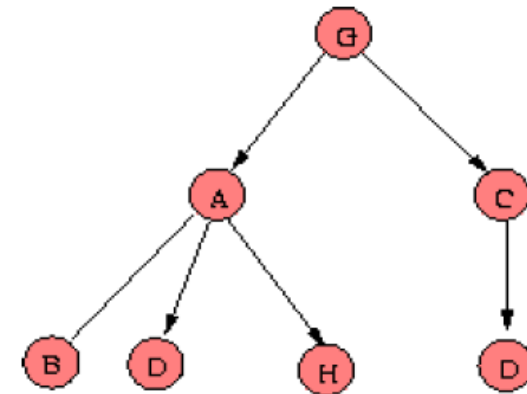
Recherche par approfondissement itératif (IDS)



■ Autre exemple



Troisième itération : limite = 2

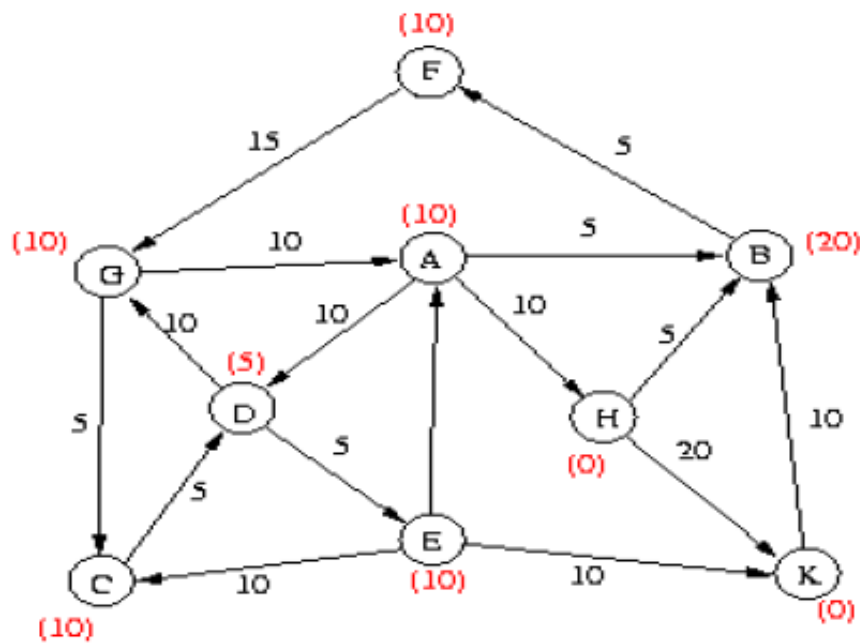


Stratégies aveugles

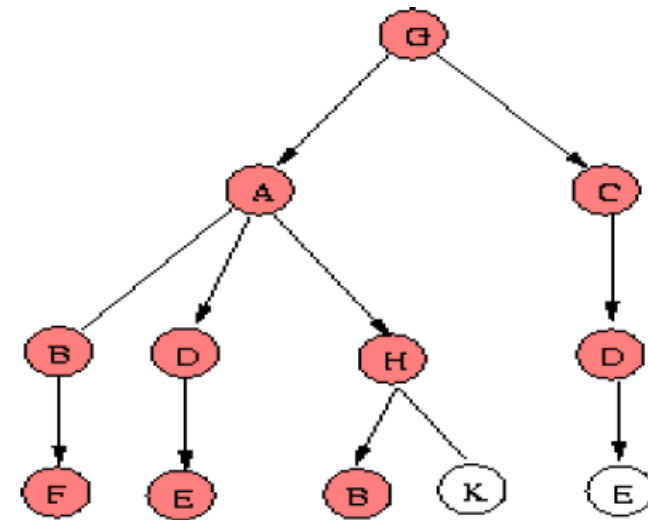
Recherche par approfondissement itératif (IDS)



■ Autre exemple



Quatrième itération : limite = 3



Solution

Stratégies aveugles

Propriété de la recherche par approfondissement itératif (IDS)



- Complétude
 - Oui, il considère systématiquement tous les chemins de longueur 1, 2, 3, ...
- Complexité en temps
 - $(d+1)b^0 + db + (d-1)b^2 + \dots + b^d = O(b^d)$
- Complexité en espace
 - $O(b*d)$
- Optimalité
 - Oui si coût unitaire par étape

Stratégies aveugles

Propriété de la recherche par approfondissement itératif (IDS)



- Peut paraître du gaspillage car beaucoup de nœuds sont étendus de multiples fois !
 - Mais le plupart des nouveaux nœuds étant au niveau le plus bas, ce n'est pas important d'étendre plusieurs fois les nœuds des niveaux supérieurs
 - Comparaison numérique pour $b = 10$ et $d = 5$, solution tout à droite de l'arbre
 - complexité spatiale de la recherche en largeur :
 - $1 + b + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$
 - pour $b = 10$ et $d = 5$ on obtient $\sim 1'111'100$ nœuds
 - complexité spatiale de la recherche par approfondissement successif :
 - $(d+1)1 + (d)b + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$
 - pour $b = 10$ et $d = 5$ on obtient 123'456 nœuds



Stratégies aveugles

Algorithme de recherche bidirectionnelle



■ Stratégie

- Peut être modélisée par l'action de deux agents qui exécutent chacun un algorithme de recherche
 - mais un agent a comme point de départ l'état initial et le deuxième part de l'état solution et se dirige vers l'état initial
- Les agents communiquent en se transmettant leurs frontières de recherche et le processus s'arrête quand les deux agents se rencontrent ou quand un agent arrive à destination
 - = l'état but pour le premier ou l'état initial pour le deuxième



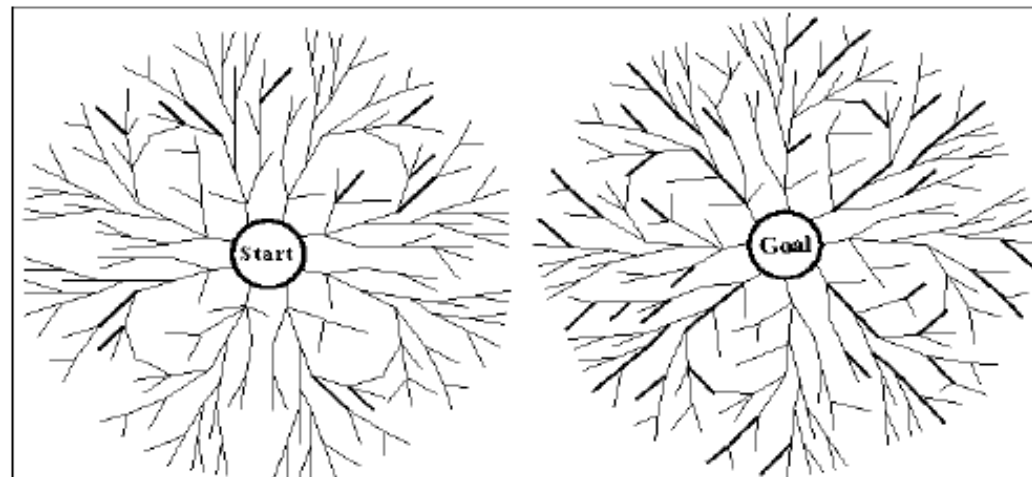
Stratégies aveugles

Algorithme de recherche bidirectionnelle



■ Attention

- Cet algorithme peut être appliqué seulement dans les cas où on dispose des inverses de tous les opérateurs du problème
- ➔ e.g. les déplacements dans un labyrinthe pour trouver la sortie, donc l'espace d'états peut être un graphe non-orienté et on connaît l'état solution



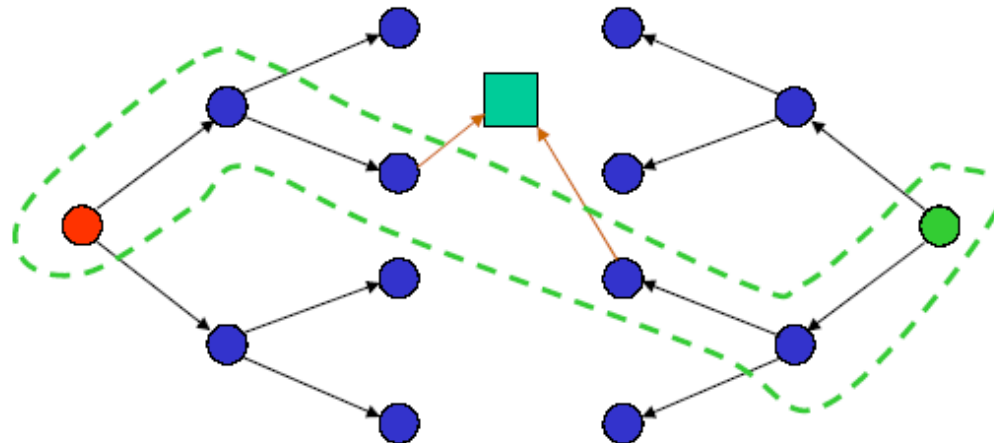


Stratégies aveugles

Algorithme de recherche bidirectionnelle



- Stratégie de recherche
 - 2 files d'attente





Stratégies aveugles

Propriété de la recherche bidirectionnelle



- Complétude
 - Oui
- Complexité en temps
 - $O(b^{d/2}) \ll O(b^d)$
- Complexité en espace
 - $O(b * d/2) \ll O(b * d)$
- Optimalité
 - Oui
- Nécessite :
 - des états-solutions explicitement définis,
 - des opérateurs dont on connaît la fonction inverse (capable de générer l'état prédécesseur d'un état donné)

Stratégies aveugles

Comparaison



- Selon les 4 critères d'évaluation retenus avec :
 - b = facteur de branchement
 - d = profondeur de la solution
 - m = profondeur maximum de l'arbre de recherche
 - l = limite de la profondeur de recherche

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes



Stratégies aveugles

Recherche avec retour en arrière



■ Intérêt

- Une recherche en profondeur peut se révéler dangereuse : l'algorithme risque de développer une branche infinie stérile sans que le mécanisme de retour en arrière puisse se déclencher
- On ajoute dans ce cas une limite de profondeur
- On a maintenant deux possibilités pour le retour arrière :
 - la limite de profondeur est dépassée
 - un sommet est reconnu comme une impasse
- Solution
 - Revenir en arrière, abandonner le nœud en cours et essayer d'étendre un autre nœud



Stratégies aveugles

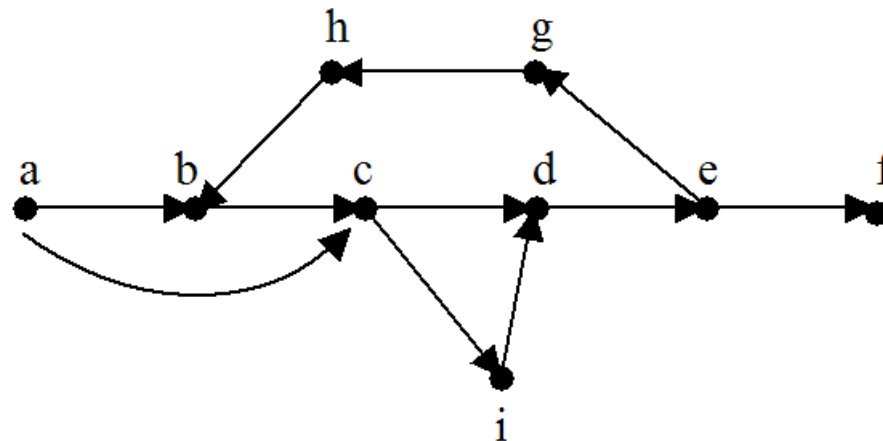
Recherche avec retour en arrière



■ Exercice : recherche de cycles dans les graphes

- Soit le graphe donné ci-dessous, on demande d'écrire l'algorithme de recherche et d'écriture de tous les chemins sans cycle de ce graphe
- Les chemins du graphe donné en exemple sont les suivants :

a b c d e f
a b c i d e f
a c d e f
a c i d e f

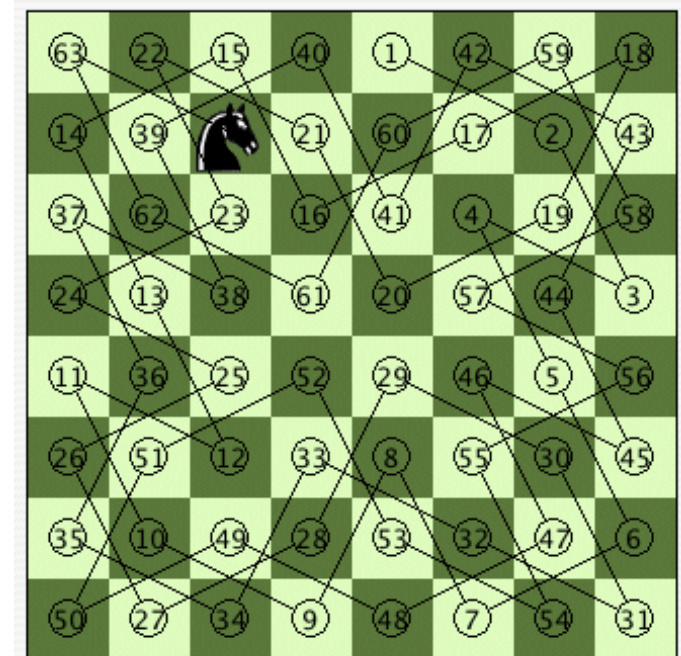
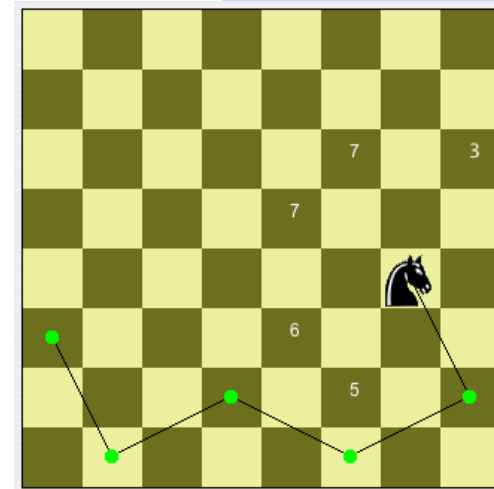


Stratégies aveugles

Projet

■ Exercice : Problème du cavalier

- On demande d'écrire un algorithme donnant les positions successives d'un cavalier qu'on déplace sur un échiquier, à partir d'une position initiale donnée, de sorte que toutes les cases de l'échiquier soit « visitées » une et une seule fois. Une case déjà visitée ne pourra donc pas être utilisée pour un nouveau déplacement.





Stratégies aveugles

Projet



■ Exercice : Problème du cavalier

- La figure rappelle les règles de déplacement d'un cavalier sur un échiquier
- S'il se trouve dans la case notée C, il pourra se rendre dans une des cases numérotées de 1 à 8 :

	1			→	y		8
1							
			3		2		
		4				1	
					C		
		5				8	
x			6		7		
8							

Stratégies aveugles

Projet



■ Exercice : Problème du cavalier

- On remarque qu'à partir des coordonnées (x,y) du cavalier, les coordonnées des 8 nouvelles positions peuvent être calculées en ajoutant dx et dy respectivement à x et y . Les valeurs de dx et dy peuvent être exprimées en fonction du numéro de la nouvelle position. Elles sont résumées dans le tableau, notée *positionRelative*, ci-dessous :

$(-1,2)$	$(-2,1)$	$(-2,-1)$	$(-1,-2)$	$(1,-2)$	$(2,-1)$	$(2,1)$	$(1,2)$
1	2	3	4	5	6	7	8

- Ce tableau est fourni à l'algorithme

Stratégies aveugles

Projet



■ Exercice : Problème du cavalier

- L'échiquier est représenté par un tableau à 2 dimensions de 8 sur 8 cases. Le résultat retourné par l'algorithme sera le tableau représentant l'échiquier et contenant, pour chaque case, un entier indiquant le numéro du déplacement ayant permis d'arriver à cette case. Les déplacements sont numérotés dans l'ordre à partir de 1. Par exemple, voici une solution possible pour un échiquier de 5 sur 5 cases et une position initiale (1,1) :

1	4	9	18	21
10	17	20	3	8
5	2	13	22	19
16	11	24	7	14
25	6	15	12	23