Intelligence artificielle

ENSIIE

Semestre 4 – 2019/20

Organisation du cours

Évaluation :

- contrôle continu (rendu de TP) $\left(\frac{1}{2}\right)$
- 1 projet, 1 à 2 demie(s) journée(s) $\left(\frac{1}{2}\right)$

Introduction

Définition

Ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine.

- "The artificial intelligence problem is taken to be that of making a machine behave in ways that would be called intelligent if a human were so behaving."
- Dartmouth 1956 McCarthy, Minsky, Rochester, Shannon

Historique

- Traitement de symboles
- “Artificial Intelligence” : 1956 (Mc Carthy)
- Etapes de développement :
- 1950-1970 : premiers systèmes
  Échecs, interrogation de BD en langue naturelle, GPS
  Logique, LISP
- 1970-1980/90 : connaissances
  Compréhension de textes et dialogues HM
  Systèmes experts, Prolog
  Approches numériques : Réseaux de neurones

Thèmes

- Résolution de problèmes - Planification
- Représentation des connaissances, Raisonnement, Apprentissage
- Architecture multiagent
- Connexionisme

Domaines d'application

- Traitement de la langue : écrit, oral
- Systèmes experts (diagnostique, . . .) , EIAO
- Robotique, reconnaissance des formes

Techniques d'IA

- Raisonnement adaptable à chaque occurrence de problème
- Connaissances :
  - volumineuses
  - difficiles à caractériser finement $\rightarrow$ structuration, généralisation
  - évolutives $\rightarrow$ modifiables aisément, compréhensibles
  - utilisables dans de nombreuses situations

Exemple 1 : Jeu (résolution de problème) : morpion

1^{re} solution

- Représentation du pb : vecteur de 9 éléments à trois valeurs + n° du tour à jouer
- Algorithme principal : stratégie selon le n° du tour (sans considérer différents cas)

solution relativement facile à modifier si on change de stratégie, mais stratégie programmée et connaissance de la solution a priori

2e solution

- Représentation du pb : vecteur de 9 éléments pouvant prendre 3 valeurs différentes
- Algorithme : à chaque coup,
  - regarder les coups possibles
  - les évaluer / réponses de l'adversaire
  - choisir le meilleur

technique d'IA, extensible à d'autres jeux du même type, stratégie générale

Plan du cours

- Algorithmes de recherche
- Algorithmes de jeux
- Programmation logique, par contraintes
- Systèmes experts

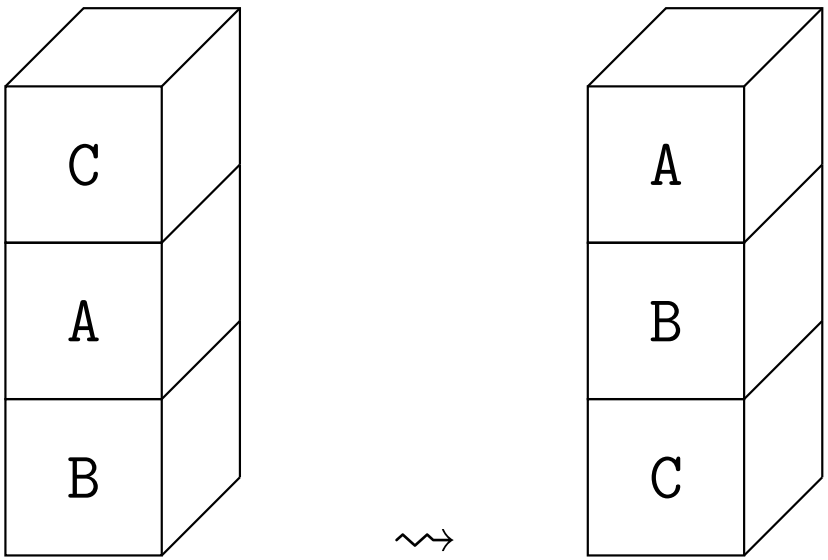
Algorithmes de recherche

Résolution de problèmes par recherche

- On représente un problème par un espace d'états (arbre/graphe).
- Chaque état est une configuration possible du problème.
- Résoudre le problème consiste à trouve un chemin dans le graphe.
- Parcours aveugles non informés : profondeur, largeur.
- Parcours informés.

Exemple : cubes

Exemple : la tour de cube

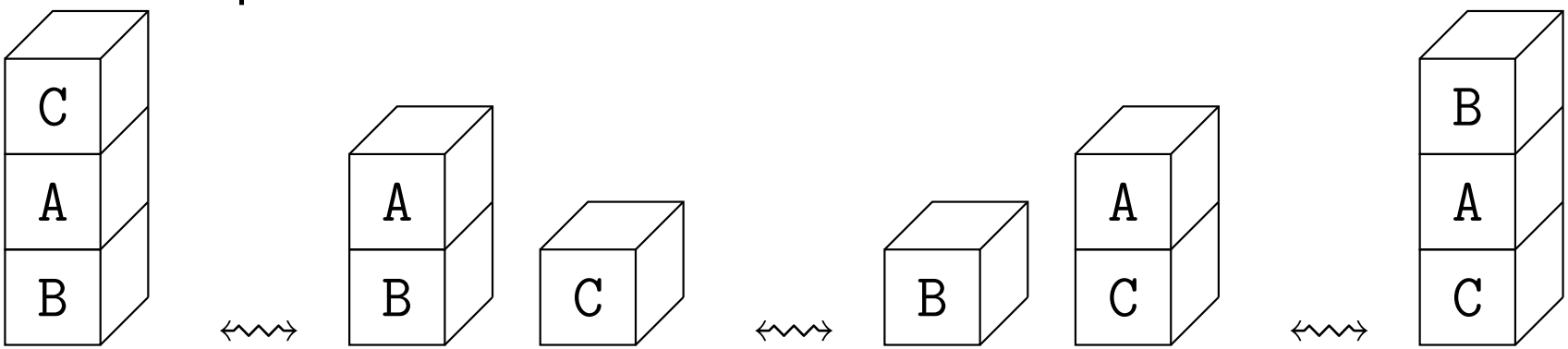


Contraintes

- On veut réarranger la pile de cubes, en ne déplaçant qu'un cube à la fois
- On peut déplacer un cube uniquement s'il n'a pas un autre au dessus de lui
- On peut poser un cube sur la table ou sur un autre cube

Configurations et actions possibles

Par exemple :



Graphe de recherche

Sommets :

- configurations possibles du jeu (états)

Arêtes :

- transitions entre les configurations (opérateurs)

Il y a (au moins) un état initial (par exemple C A B)
Il y a (au moins) un état final

- Explicite (p.e. A B C).
- Implicite (décrit par une propriété).

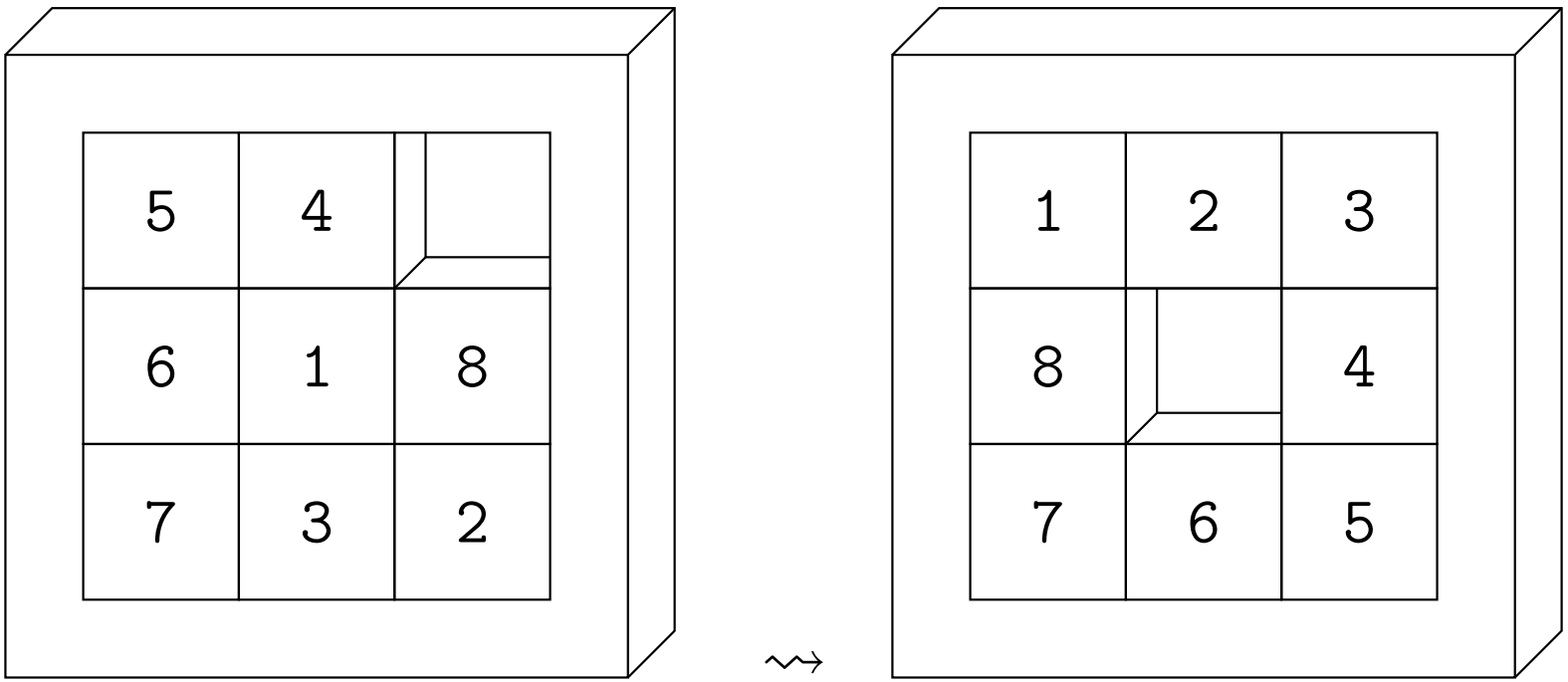
Solution

Le graphe est implicite ! Il n'est pas calculé en entier en général.
Solution :

- séquence finie de transitions permettant d'atteindre un état final à partir d'un état initial

Trouver une solution consiste donc à trouver un chemin dans le graphe (p.e. un chemin de l'état C A B vers l'état A B C)
Coût du chemin : somme de distances, nb d'opérateurs, etc.

Autre exemple : taquin



Graphe

États :

- Toutes les configurations possibles des pièces 1 à 8 dans le puzzle

Arêtes

- échanger le blanc avec la pièce à gauche, à droite, au dessus ou en dessous

État initial :

- Une configuration quelconque

État final :

- La configuration de droite

Coût :

- nombre de mouvements

Algorithmes de recherche

Idée générale

1. Démarrer la recherche avec la liste contenant l'état initial du problème
2. Tant la liste n'est pas vide alors :
   - 2.1 Choisir (à l'aide d'une stratégie) un état e à traiter
   - 2.2 Si e est un état final alors retourner recherche positive
   - 2.3 Sinon, rajouter les successeurs de e à la liste (ou une partie d'entre eux)
3. Si on sort de la boucle, retourner recherche négative

Stratégies

C'est un critère qui permet de choisir un ordre pour traiter les états du problème.
On tiendra compte de :

- La complétude
  - Si une solution existe, va-t-on la trouver (en temps fini) ?
- L'optimalité (selon le coût)
  - La solution trouvée est-elle la meilleure ?
- La complexité (en temps et en espace) mesurée par :
  - b : branchement maximal de l'arbre de recherche
  - d : profondeur de la meilleur solution
  - m : profondeur maximale de l'espace d'états

Deux familles de stratégie

Recherche aveugle, ou non-informée

- stratégie générique

Recherche heuristique, ou informée

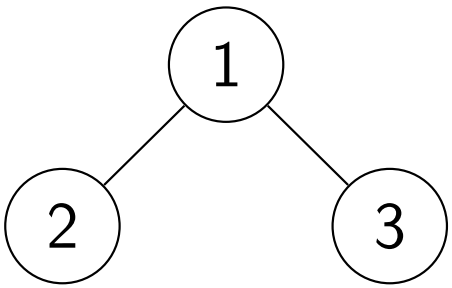
- stratégie utilisant des spécificités du problème

Recherche en largeur d'abord

breadth-first search

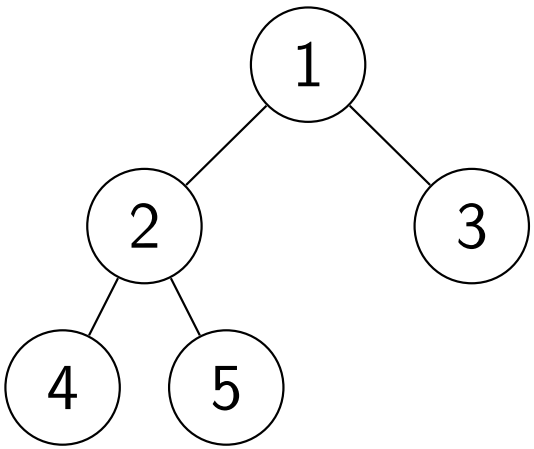
Recherche en largeur d'abord

breadth-first search



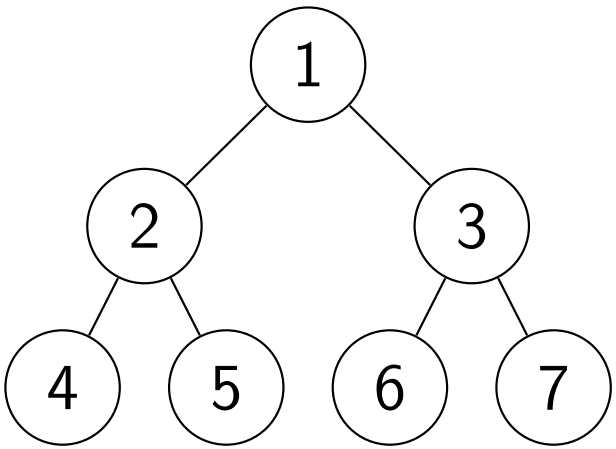
Recherche en largeur d'abord

breadth-first search



Recherche en largeur d'abord

breadth-first search



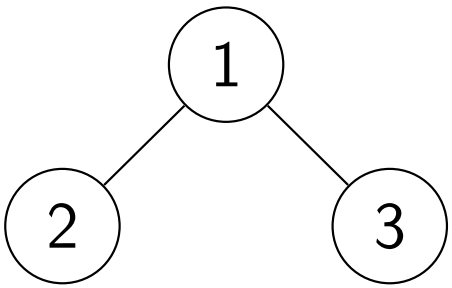
Recherche en largeur d'abord

breadth-first search

Utilisation d'une file (FIFO, *queue*)
1

Recherche en largeur d'abord

breadth-first search

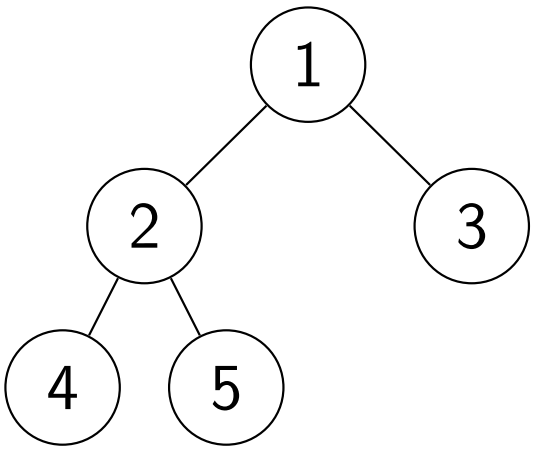


Utilisation d'une file (FIFO, *queue*)

2 3

Recherche en largeur d'abord

breadth-first search

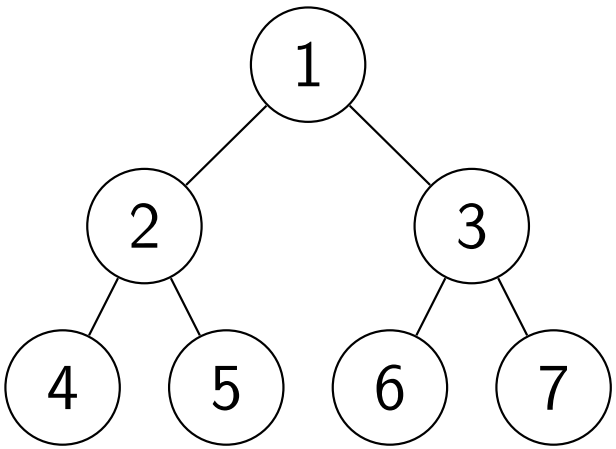


Utilisation d'une file (FIFO, *queue*)

3 4 5

Recherche en largeur d'abord

breadth-first search



Utilisation d'une file (FIFO, *queue*)

4 5 6 7

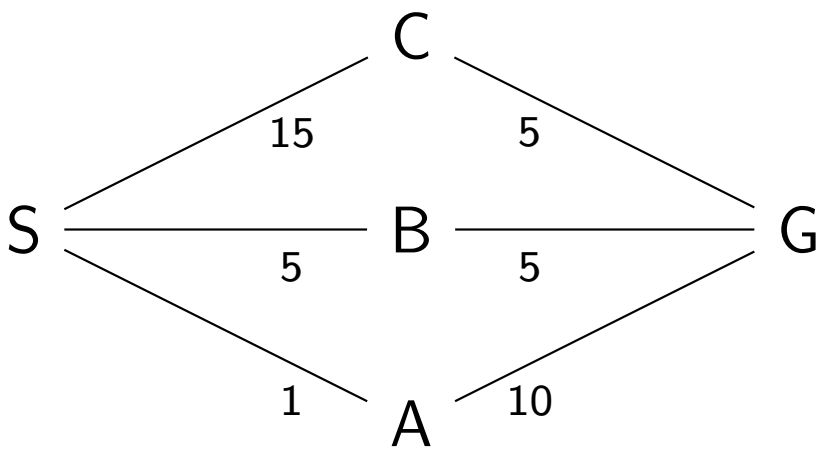
Caractéristiques de la recherche en largeur d'abord

- Complète si b est fini
- Complexité en temps : $O(b^d)$
- Complexité en espace : $O(b^d)$
- Optimale si coût $= 1$, non optimale en générale

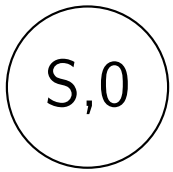
Variante : recherche à coût uniforme

Prendre en compte le coût dans la recherche en largeur
On utilise des files de priorité : chaque état est associé au coût qu'il a fallu dépensé pour l'atteindre

Exemple

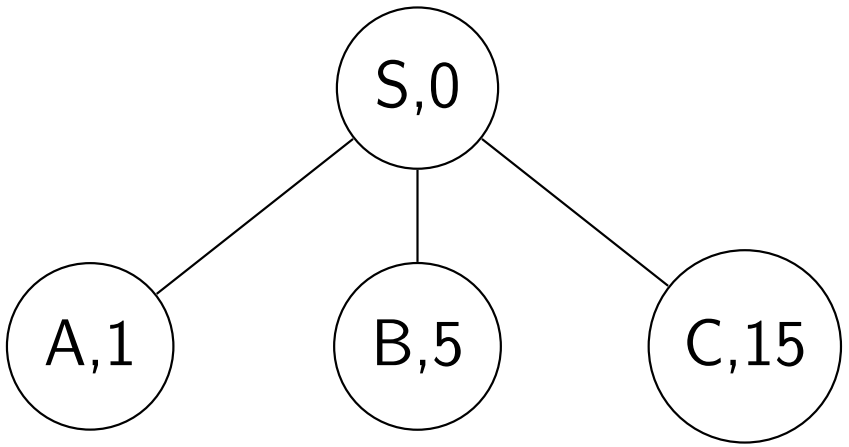


Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)
(S,0)

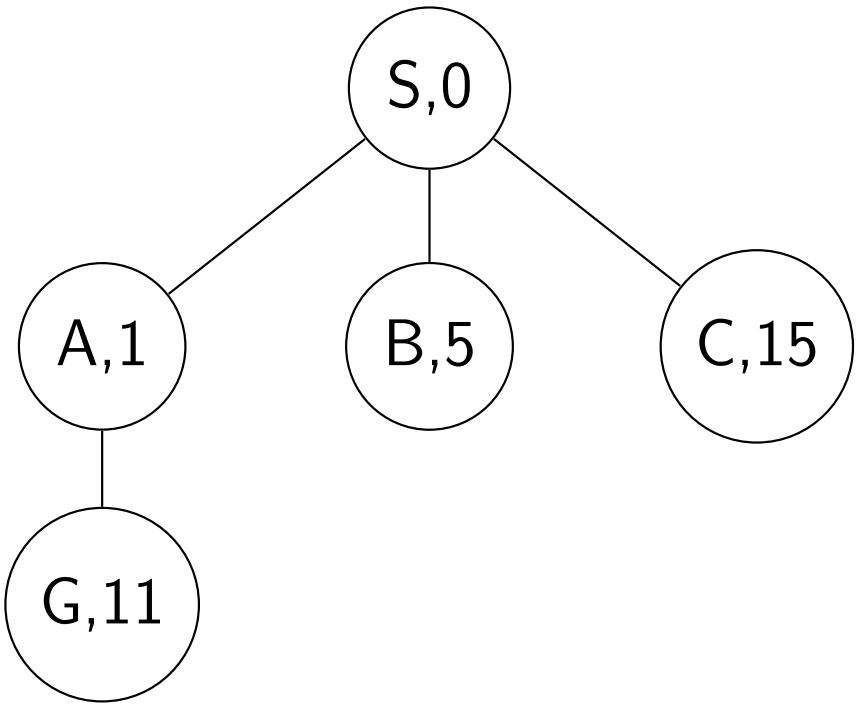
Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)

(A,1) (B,5) (C,15)

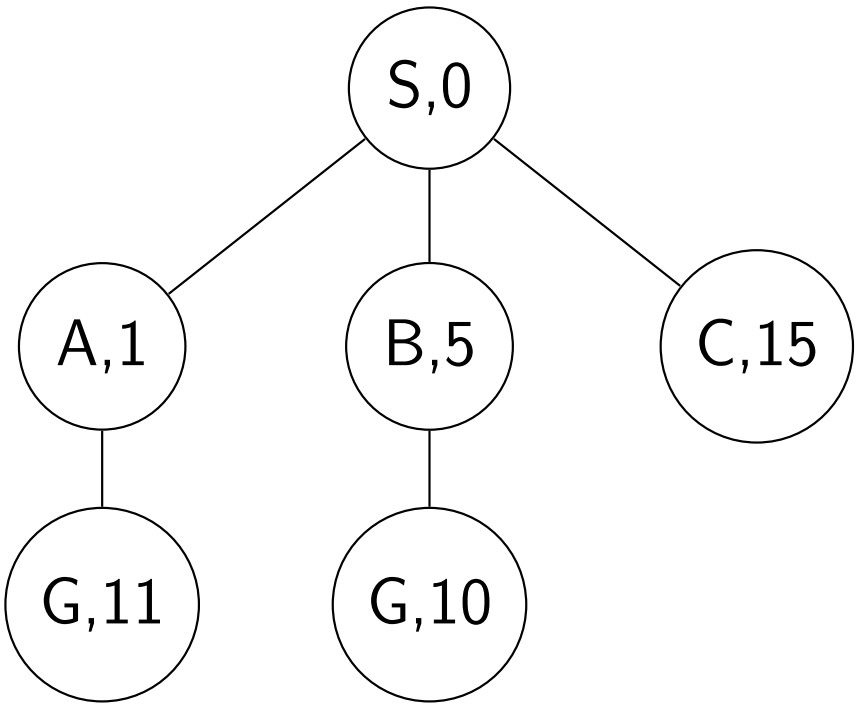
Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)

(B,5) (G,11) (C,15)

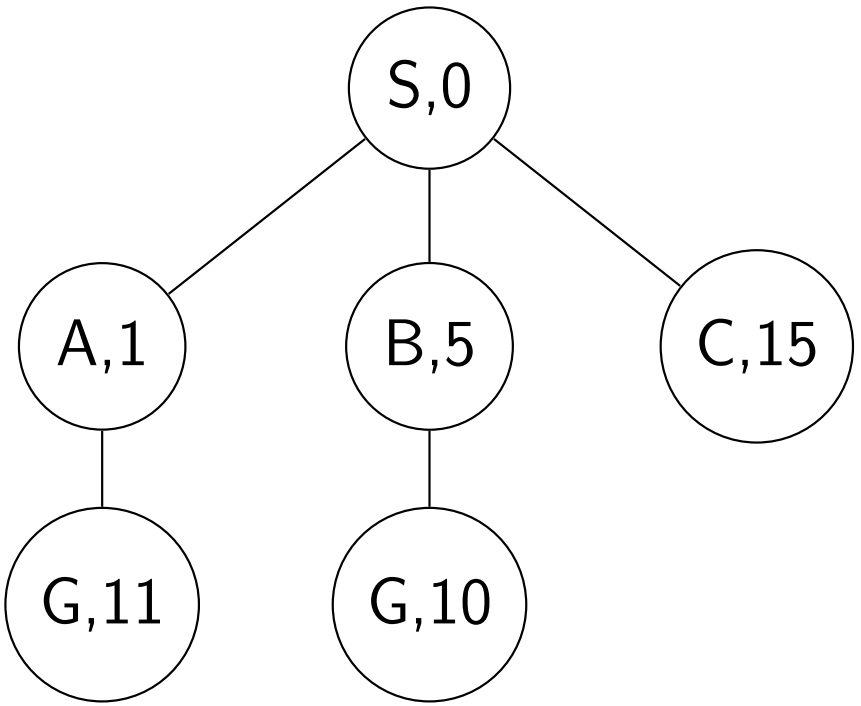
Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)

(G,10) (G,11) (C,15)

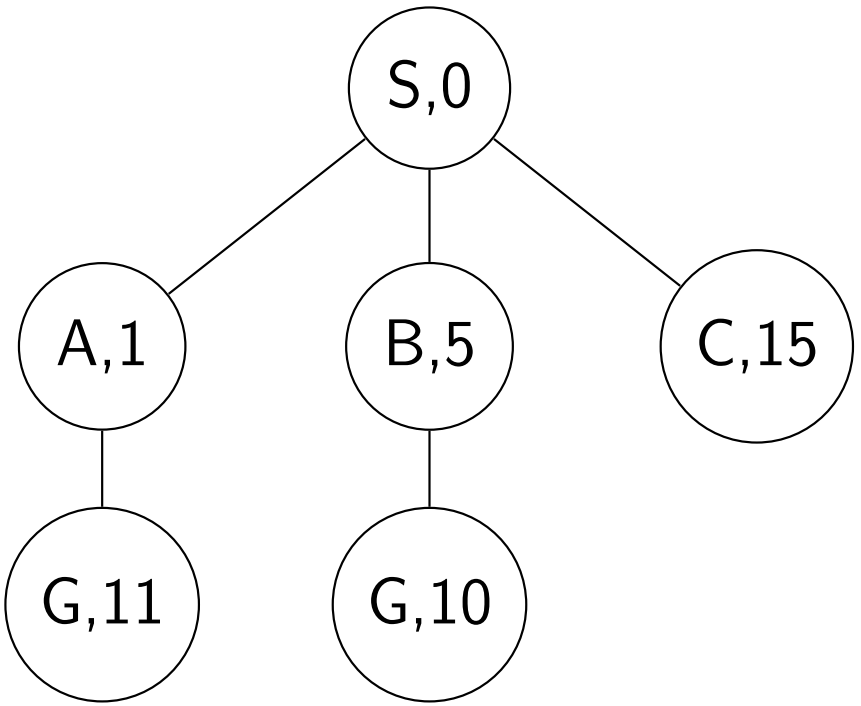
Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)

(G,10) (G,11) (C,15)

Recherche sur l'exemple



Utilisation d'une file de priorité (*priority queue*)

(G,10) (G,11) (C,15)

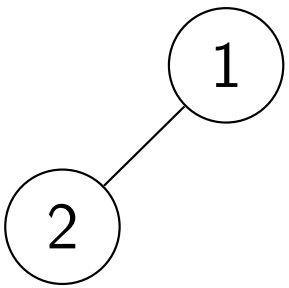
Si coût égal partout, on retrouve la recherche en largeur d'abord

Recherche en profondeur d'abord

depth-first search

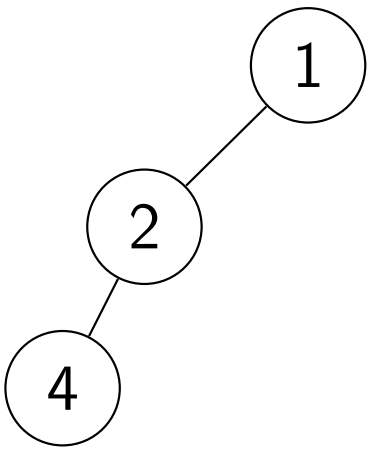
Recherche en profondeur d'abord

depth-first search



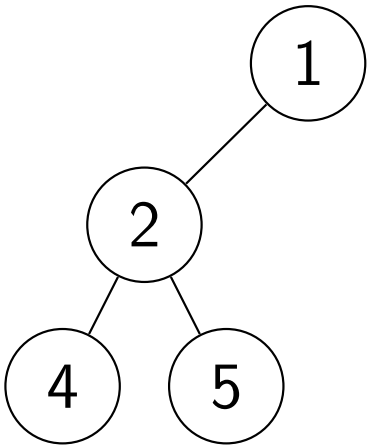
Recherche en profondeur d'abord

depth-first search



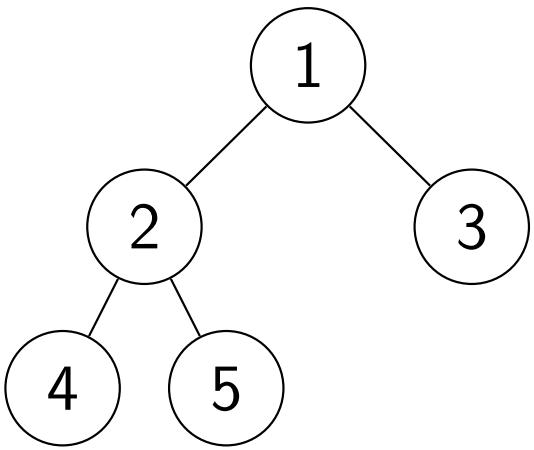
Recherche en profondeur d'abord

depth-first search



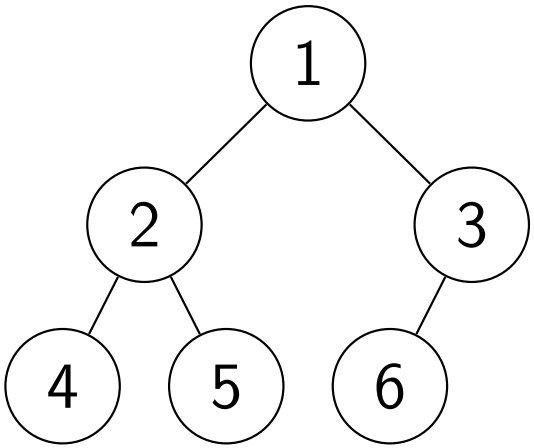
Recherche en profondeur d'abord

depth-first search



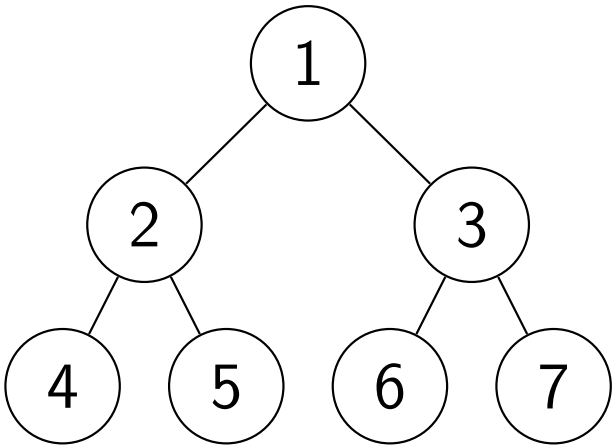
Recherche en profondeur d'abord

depth-first search



Recherche en profondeur d'abord

depth-first search



Recherche en profondeur d'abord

depth-first search

Utilisation d'une pile (LIFO, *stack*)
1

Recherche en profondeur d'abord

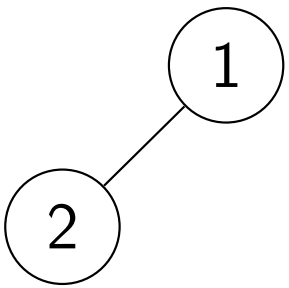
depth-first search

Utilisation d'une pile (LIFO, *stack*)

2 3

Recherche en profondeur d'abord

depth-first search

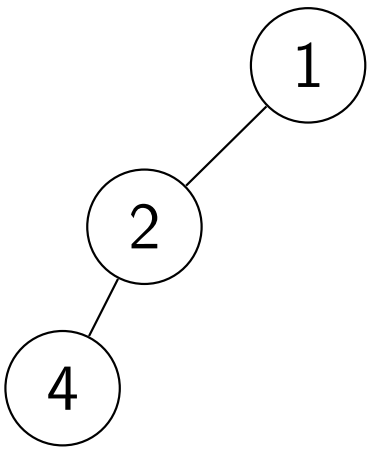


Utilisation d'une pile (LIFO, *stack*)

4 5 3

Recherche en profondeur d'abord

depth-first search

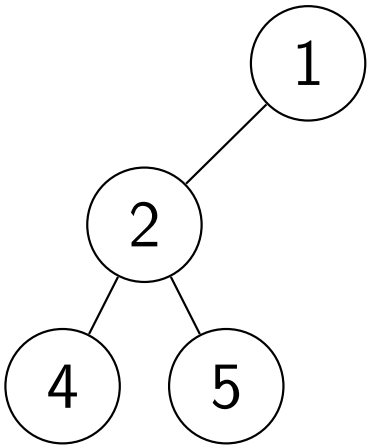


Utilisation d'une pile (LIFO, *stack*)

5 3

Recherche en profondeur d'abord

depth-first search

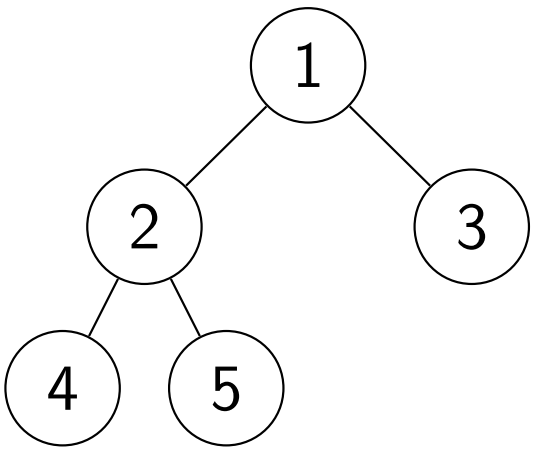


Utilisation d'une pile (LIFO, *stack*)

3

Recherche en profondeur d'abord

depth-first search

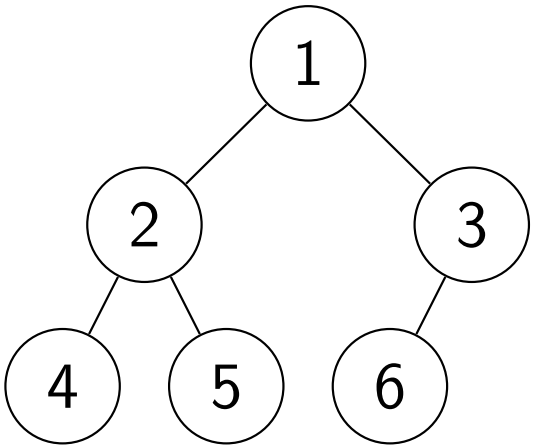


Utilisation d'une pile (LIFO, *stack*)

6 7

Recherche en profondeur d'abord

depth-first search

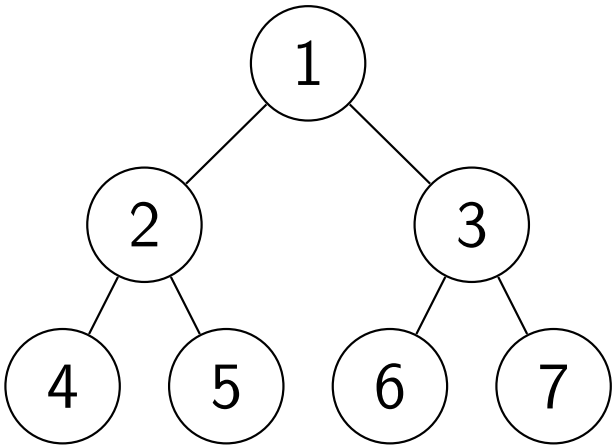


Utilisation d'une pile (LIFO, *stack*)

7

Recherche en profondeur d'abord

depth-first search



Utilisation d'une pile (LIFO, *stack*)

Caractéristiques de la recherche en profondeur d'abord

- Non complète si espace d'états infini
- Complexité en temps : $O(b^m)$
- Complexité en espace : $O(b \times m)$
- Non optimale

Recherche en profondeur limitée

Comme la recherche en profondeur d'abord, mais on ne va pas plus qu'une profondeur donnée

Recherche en profondeur itérative

iterative deepening
On itère la recherche en profondeur limitée en augmentant petit à petit la borne

```
p := 0
res := faux
tant que res = faux
  res := recherche_en_profondeur_limitee(p)
  p := p + 1
retourne res
```

Caractéristiques de la recherche en profondeur itérative

- Complète
- Complexité en temps : $1 + b + b^2 + ... + b^d = O(b^d)$
- Complexité en espace : $O(b \times d)$
- Optimale si coût $= 1$

Recherche bi-directionnelle

On part à la fois de l'état initial (dans le sens du graphe) et de l'état final (dans le sens inverse)
On s'arrête quand on trouve un état en commun

Résumé

- Recherche en largeur d'abord :
  - solution optimale [1], mémoire importante
- Recherche en coût uniforme :
  - solution optimale, mémoire importante
- Recherche en profondeur d'abord :
  - pas optimal, économe en place
- Recherche itérative en profondeur :
  - optimal [1], économe en place
- Recherche bidirectionnelle :
  - optimal [1], mémoire importante, plus rapide

Dans tous les cas :

- explosion combinatoire

1. si coût égaux

Heuristique

Informations supplémentaires pour orienter la recherche
Utilité

- Choisir le nœud à développer : le plus prometteur
- Déterminer les successeurs à engendrer : quels opérateurs
- Décider des nœuds à ignorer

Heuristique

Informations supplémentaires pour orienter la recherche
Utilité

- Choisir le nœud à développer : le plus prometteur
- Déterminer les successeurs à engendrer : quels opérateurs
- Décider des nœuds à ignorer

Algorithme de recherche meilleur en premier

Même principe que la recherche à coût uniforme,
mais on n'ordonne pas la file de priorité en fonction du coût passé
mais en fonction d'une mesure d'utilité

- heuristique qui indique si l'état semble amener au but

Recherche gloutonne

La mesure d'utilité ne dépend pas du chemin parcouru jusque là mais uniquement de l'état considéré
Par exemple dans recherche du plus court chemin entre deux villes,
distance à vol d'oiseau entre les villes

Caractéristiques de la recherche gloutonne

- Incomplète (car boucles)
- Complexité en temps : $O(b^m)$
- Complexité en espace : $O(b^m)$
- Non optimale.

Algorithme A*

Améliorer la recherche gloutonne en prenant en compte le coût déjà dépensé

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

- $g(n)$ coût de l'état initial jusqu'à n
- $h(n)$ heuristique estimant le coût de n à un état final
- $f(n)$ fonction pour prioriser les états

Admissibilité de l'heuristique

h est admissible si pour tout état n on a

$$0 \leq h(n) \leq h^*(n)$$

où $h^*(n)$ est le vrai coût pour aller de n à un état final

Exemple d'heuristique admissible :
distance à vol d'oiseau (inégalité triangulaire)

Caractéristiques de l'algorithme A*

On suppose l'heuristique admissible

- Complète
- Complexité en temps : $O(b^m)$
- Complexité en espace : $O(b^m)$
- Optimale

Exemples d'heuristiques pour le taquin

- $h_1(n) =$ nombre de pièce mal placées
- $h_2(n) =$ sommes des distances de chaque pièce à sa position finale
- $h_3(n) = h_2(n) + 3s(n)$
  avec $s(n) =$ somme des scores pour chaque case :
  - case périphérique : 2 si pas suivie par son successeur
    0 sinon
  - case centrale : 1 si non vide, 0 sinon

Comparaison des heuristiques

h domine h' (noté $h \sqsupseteq h'$) si

$$h(n) \geq h'(n) \qquad \text{pour tout } n$$

(h est mieux informé que h')

Exemple : $h_3 \sqsupseteq h_2 \sqsupseteq h_1$

Si $h \sqsupseteq h'$, alors en utilisant h on ne développera aucun nœud que h' ne développe pas.

- utiliser une heuristique la plus dominante possible (mais qui reste admissible)

Trouver une fonction heuristique

- Extraction de traits descriptifs
- Relâcher les restrictions des opérateurs
  - relaxed problem
- Utilisation de statistiques
  - Tests avec une fonction modifiée après étude des résultats

Compromis entre calcul de l'heuristique (temps, espace) et sa pertinence