Fiches algorithmes et structures de données

Julien Romero

Algorithmes de tri - CI2

- Entrée: Un tableau d'éléments comparables
- Sortie: Rien (tri sur place) ou nouveau tableau avec les éléments ordonnés

Algorithme	Pire des cas	Moyenne	Spatiale Moyenne
Tri bulle	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Tri par insertion	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Tri fusion	$\mathcal{O}(n \cdot log(n))$	$\mathcal{O}(n \cdot log(n))$	$\mathcal{O}(n)$
Tri rapide	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot log(n))$	$\mathcal{O}(log(n))$

- En Java: `Arrays.sort` (tri rapide)

Tableau extensible - CI3

- Un tableau qui grandit quand il est plein
- En Java: `ArrayList`

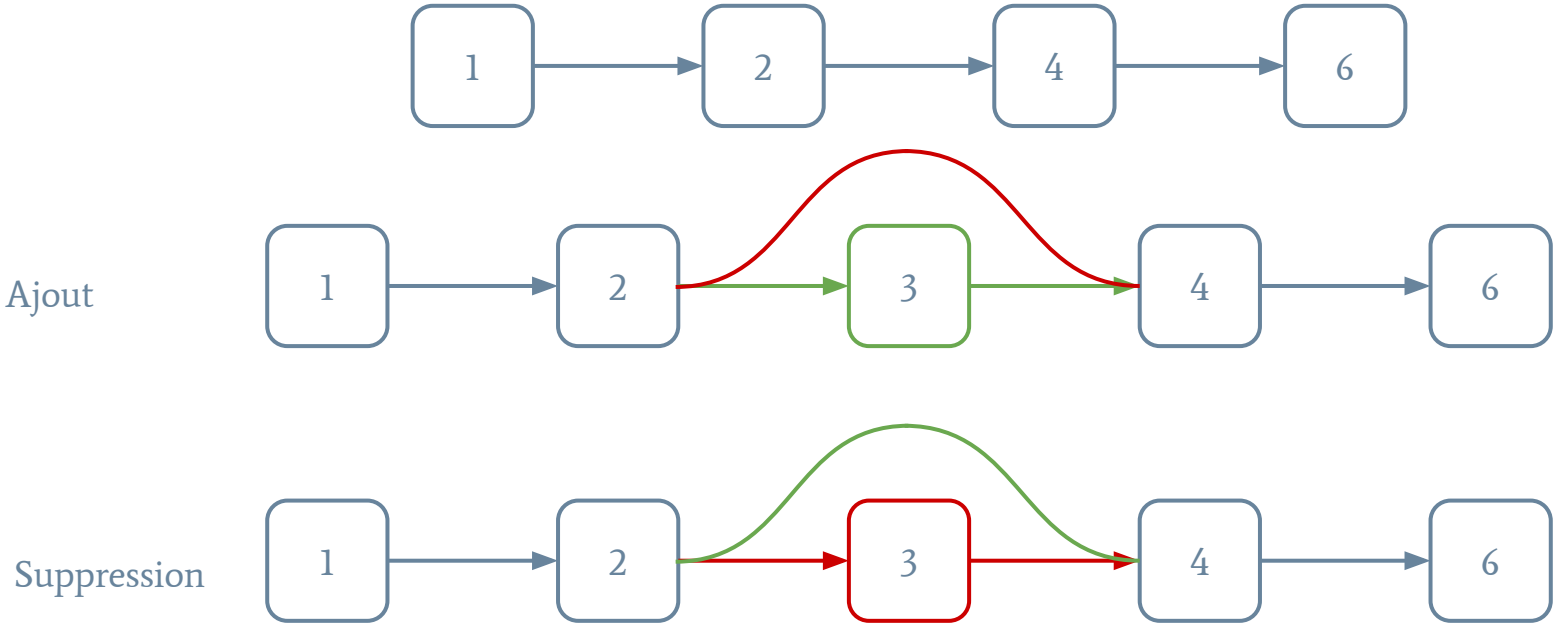
Opération	Pire des cas	Moyenne/Amortie
add(element)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
add(index, element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
get(index)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
remove(index or element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
contains(element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$

Liste chainée - CI3

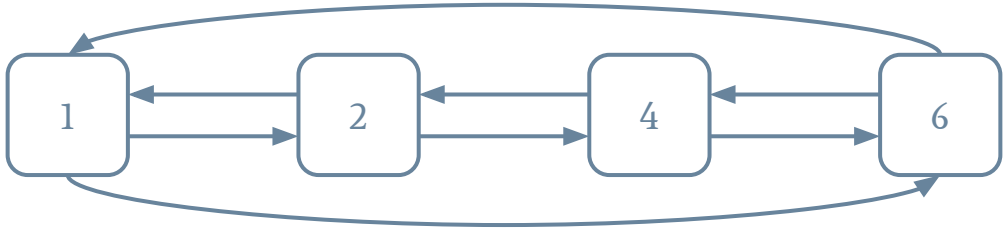
- Chaque élément de la liste a une valeur et ne connaît que ses voisins.
- En Java: `LinkedList`

Opération	Pire des cas	Moyenne/Amortie
add(element)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add(index, element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
get(index)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
remove(index or element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
contains(element)	$\mathcal{O}(n)$	$\mathcal{O}(n)$

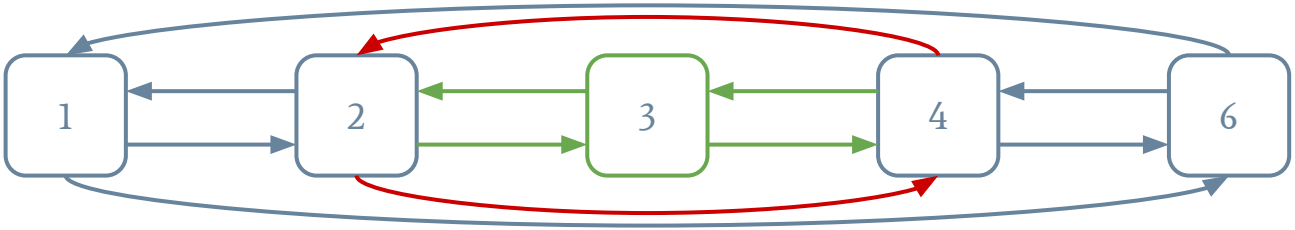
Liste chainée - CI3



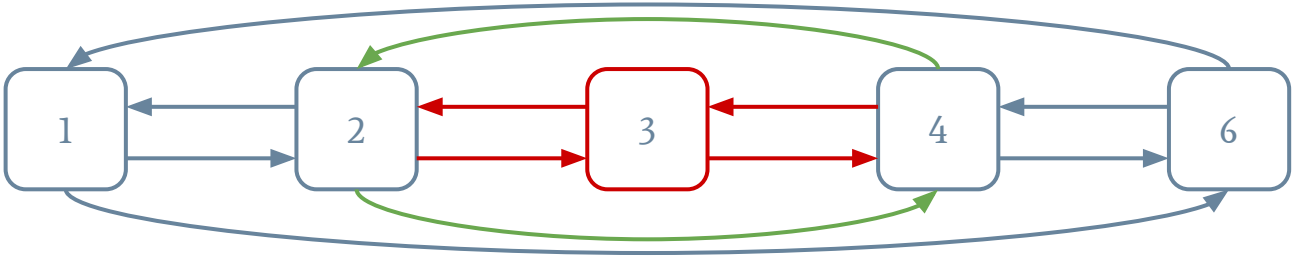
Liste doublement chainée circulaire - Cl7



Ajout

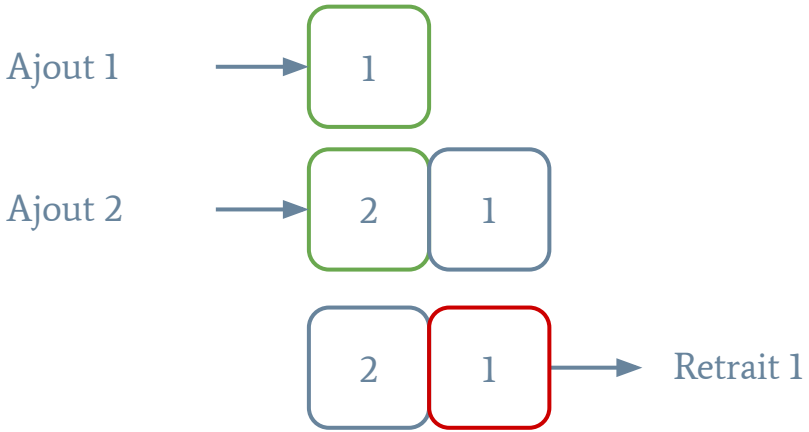


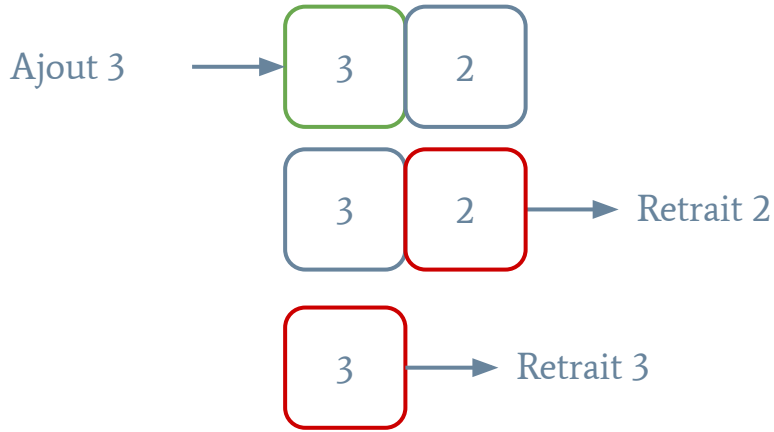
Suppression



File (Queue) - CI7 (optionnel)

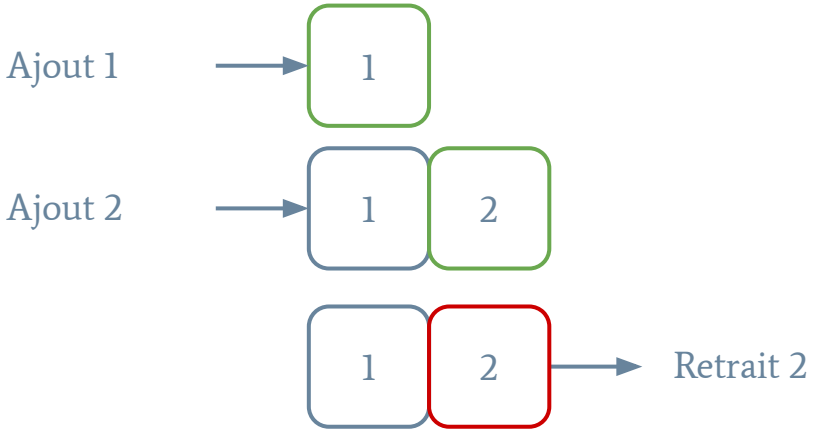
- Structure permettant de récupérer les éléments dans l'ordre d'insertion, le tout en temps constant.
- En Java: `LinkedList` (interface `Queue`)

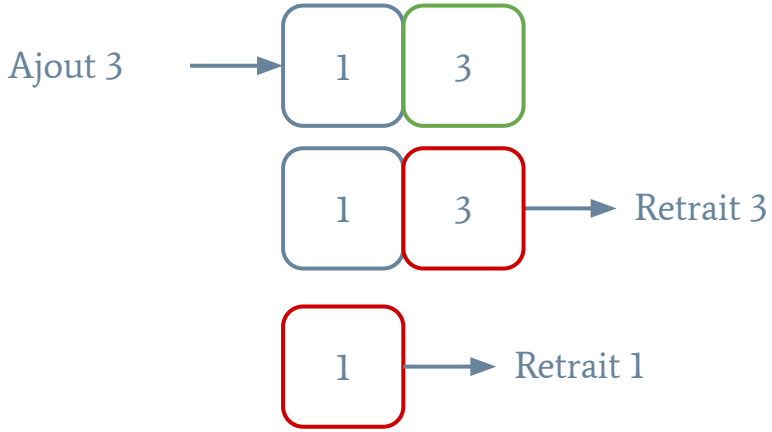




Pile (Stack) - CI7 (optionnel)

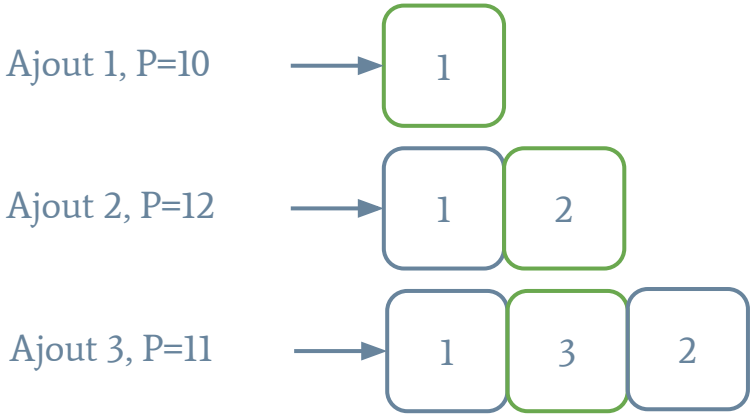
- Structure permettant de récupérer les éléments dans l'ordre **inverse** d'insertion, le tout en temps constant.
- En Java: `Stack`





File à priorité (Priority Queue) - CI5 (optionnel)

- Variante de la file avec un ordre de sortie dépend d'une priorité.
- En Java: `PriorityQueue`



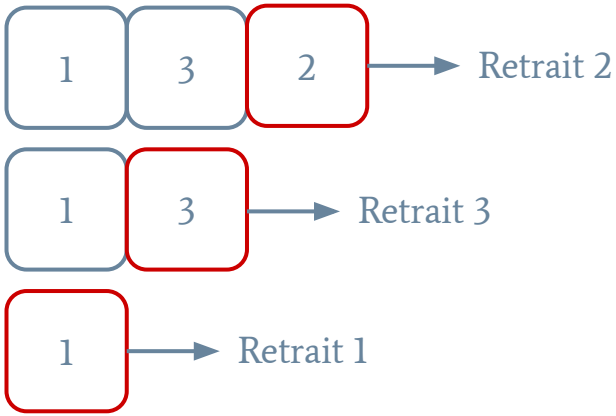


Table de hachage - CI7

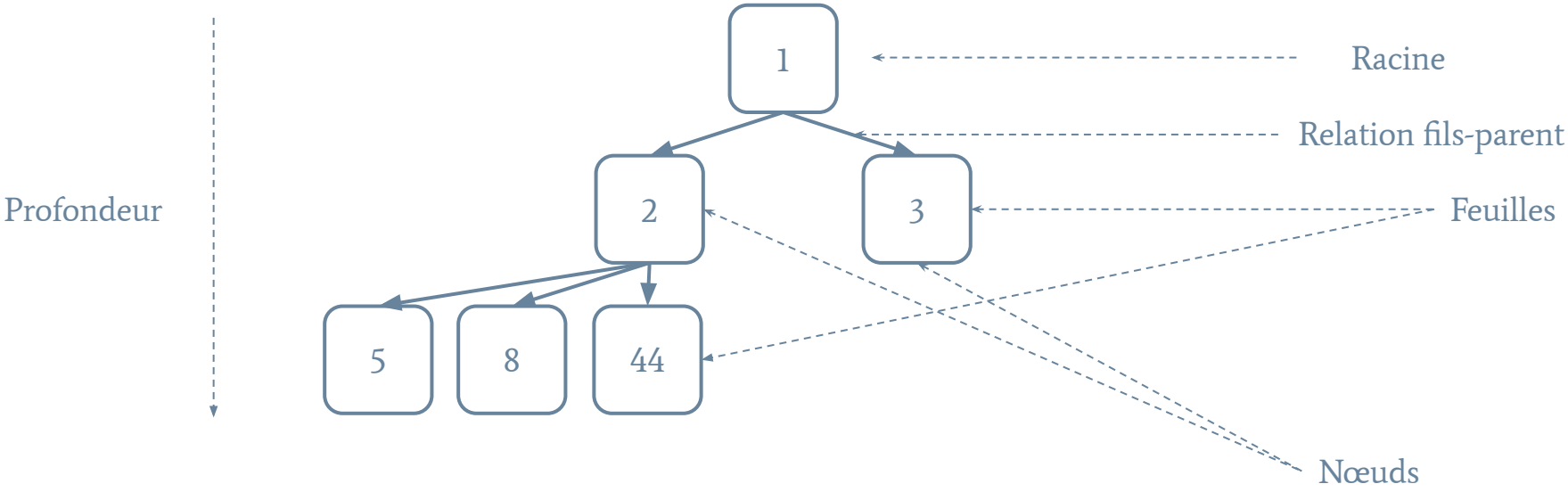
- Structure algorithmique associant des clefs à des valeurs
- L'ajout de paire de clef/valeur ou l'accès à la valeur associée à une clef se fait en *temps constant*.
- En Java: `HashMap`

Ensemble - CI7 (optionnel)

- Structure algorithmique permettant de rapidement ajouter un élément et de rapidement vérifier que l'ensemble contient un élément.
- Ajouter un élément et vérifier sa présence se fait en temps constant. Les autres opérations peuvent être plus coûteuses par rapport à une liste ou un tableau.
- En Java: Interface `Set`
  - En pratique : `HashSet`

Arbre - CI4

- Structure hiérarchique utilisée dans de nombreux algorithmes.
- En Java: À réécrire suivant les cas d'usage



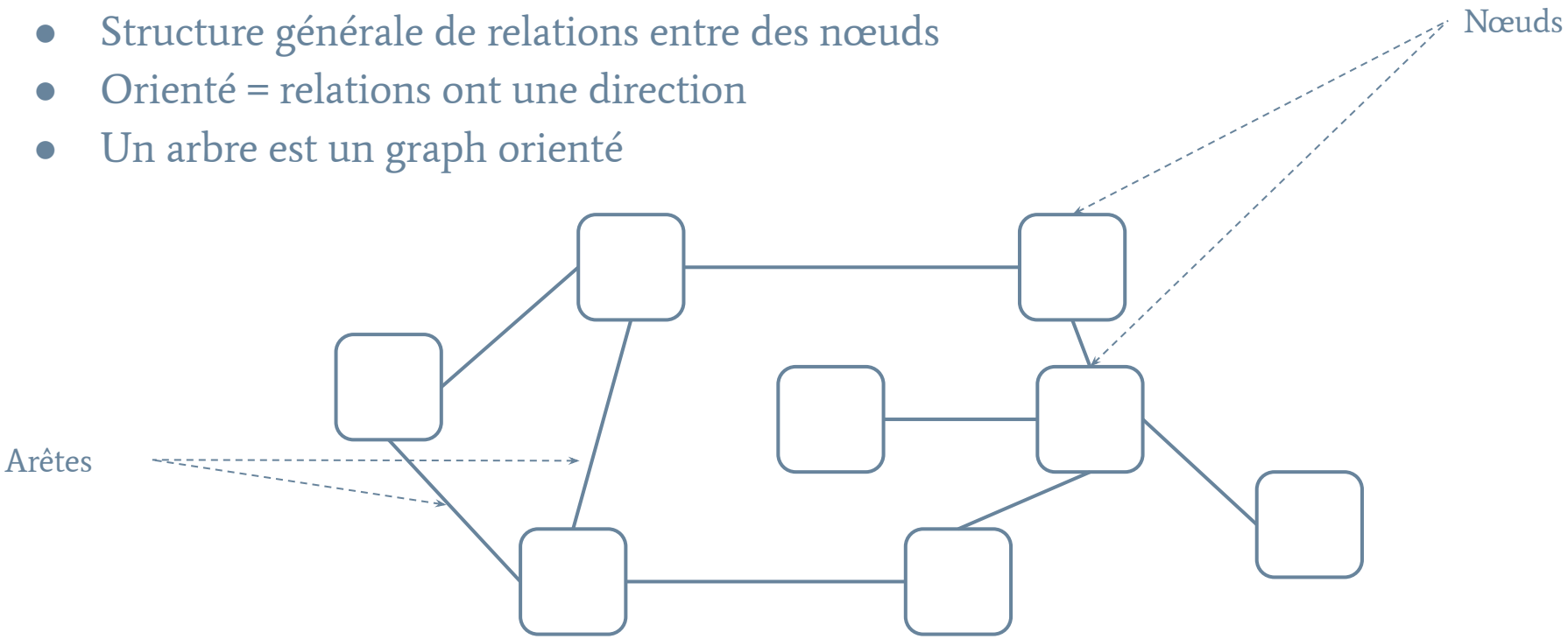
Arbre binaire de recherche - CI4

- Une structure d'arbre permettant de rapidement ajouter, rechercher et supprimer des éléments.
- Arbre binaire = nombre de fils est 0 (feuille) ou 2.

Opération	Pire des cas	Moyenne/Amortie
add	$\mathcal{O}(n)$	$\mathcal{O}(log(n))$
delete	$\mathcal{O}(n)$	$\mathcal{O}(log(n))$
search	$\mathcal{O}(n)$	$\mathcal{O}(log(n))$

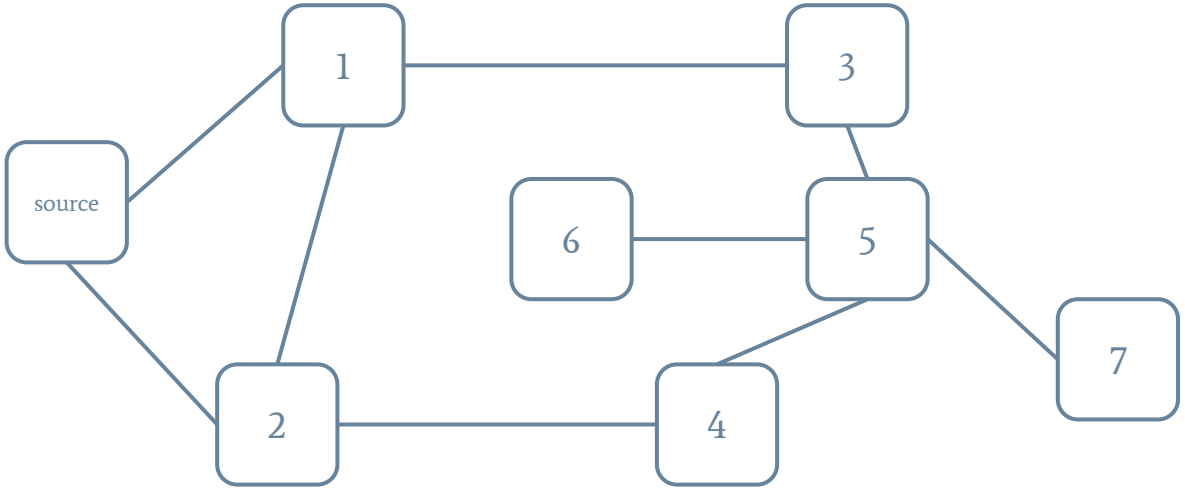
Graphe - CI5

- Structure générale de relations entre des nœuds
- Orienté = relations ont une direction
- Un arbre est un graph orienté



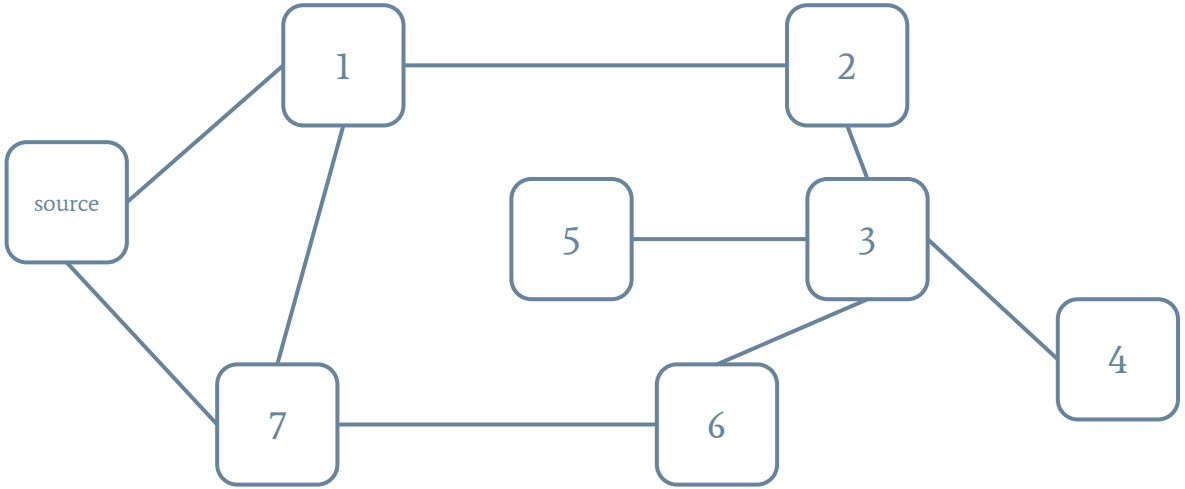
Parcours en largeur/profondeur - CI5 + CI9

- Algorithme visant à visiter tous les nœuds accessibles depuis un nœud source.
- Parcours en largeur: On explore d'abord les nœuds à un distance 1, puis 2, ..., puis k, puis k+1, ...



Parcours en largeur/profondeur - CI5 + CI9

- Algorithme visant à visiter tous les nœuds accessibles depuis un nœud source.
- Parcours en profondeur: On suit un chemin jusqu'au bout puis on le remonte pour explorer les bifurcations.



Algorithme de Dijkstra - CI5

- Permet de trouver le plus court chemin entre deux nœuds d'un graphe pondéré.
- Graph pondéré = graphe avec des poids sur chaque arête.
- Complexité (A = nombre d'arêtes, N = nombre de nœuds):
  - Avec un tas binaire: $\mathcal{O}((\mathcal{A} + \mathcal{N}) \cdot log(\mathcal{N}))$
  - Avec un tas de Fibonacci: $\mathcal{O}(\mathcal{A}+\mathcal{N} \cdot log(\mathcal{N}))$

Parcours A* - CI9

- Variante du parcours en profondeur et en largeur, mais l'ordre d'exploration dépend d'une fonction d'exploration appelée heuristique.
- Exemple d'heuristique : la distance de Manhattan entre deux points.
- Souvent utilisé dans des intelligences artificielles de jeux vidéos.

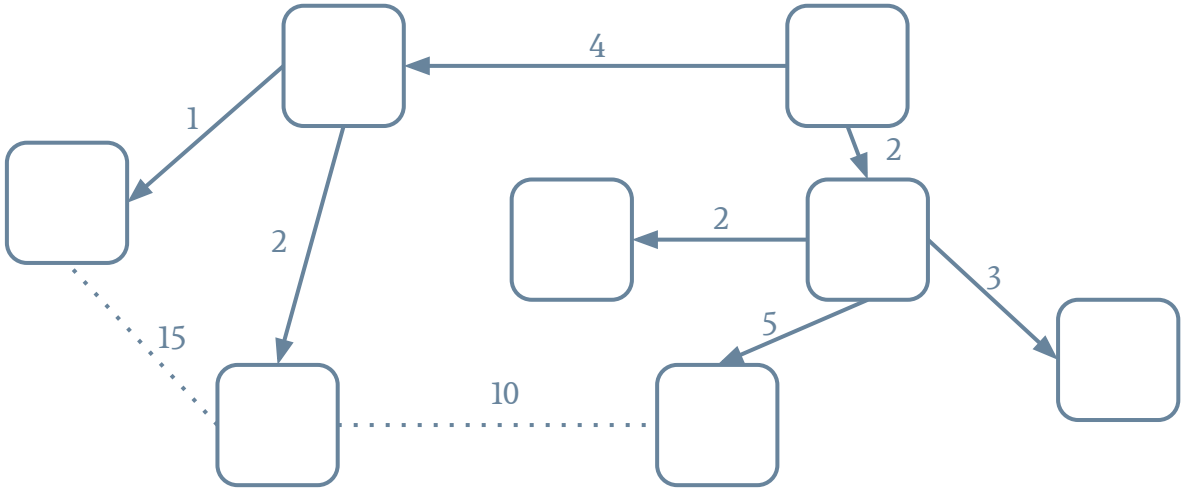
Tas binaire - Cl5 (optionnel)

- Structure de donnée combinant un arbre binaire complet et un tas (un nœud est plus grand/petit que ses fils) utilisée pour rapidement trouver le minimum/maximum d'un ensemble.

Opération	Pire des cas	Moyenne/Amortie
add	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$
removeMin/removeMax	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$
update	$\mathcal{O}(log(n))$	$\mathcal{O}(log(n))$

Arbre couvrant minimal - CI6 (optionnel)

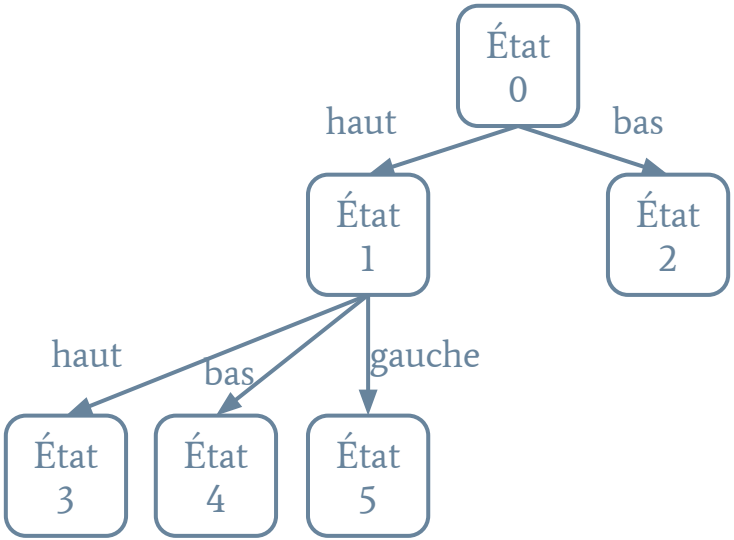
- Problème consistant à trouver un arbre “couvrant” tous les nœuds d’un graphe de façon continue de telle sorte que le poids des arêtes soit minimal.
- Utile pour de nombreux algorithmes
- Algorithme: Kruskal



Intelligence artificielle pour des jeux - CI9

- Représentation du monde avec un état
  - Exemple : pour Pacman, l'aire de jeu avec la position de la nourriture et de Pacman.
- Dans chaque état, un agent (i.e., une entité du jeu) choisit une action parmi un choix d'actions légales
  - Exemple d'agents : Pacman et les fantômes, les noirs et les blancs aux échecs
  - Exemple d'actions : Pacman : Haut, bas, gauche, droite. Échecs : déplacement de pièce.
- Il est souvent nécessaire de simuler l'état suivant à partir d'une action
- On peut former un graphe (ou plus simplement un arbre) dont les nœuds sont états et les arêtes les actions possibles.

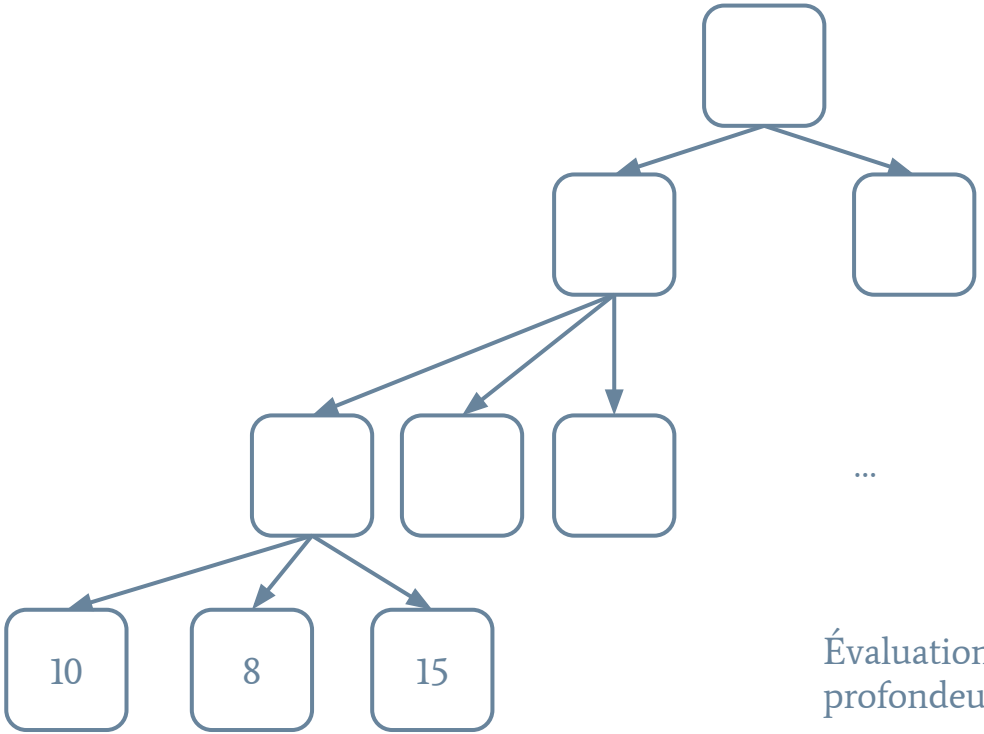
Intelligence artificielle pour des jeux - CI9



Minimax - CI9 (optionnel)

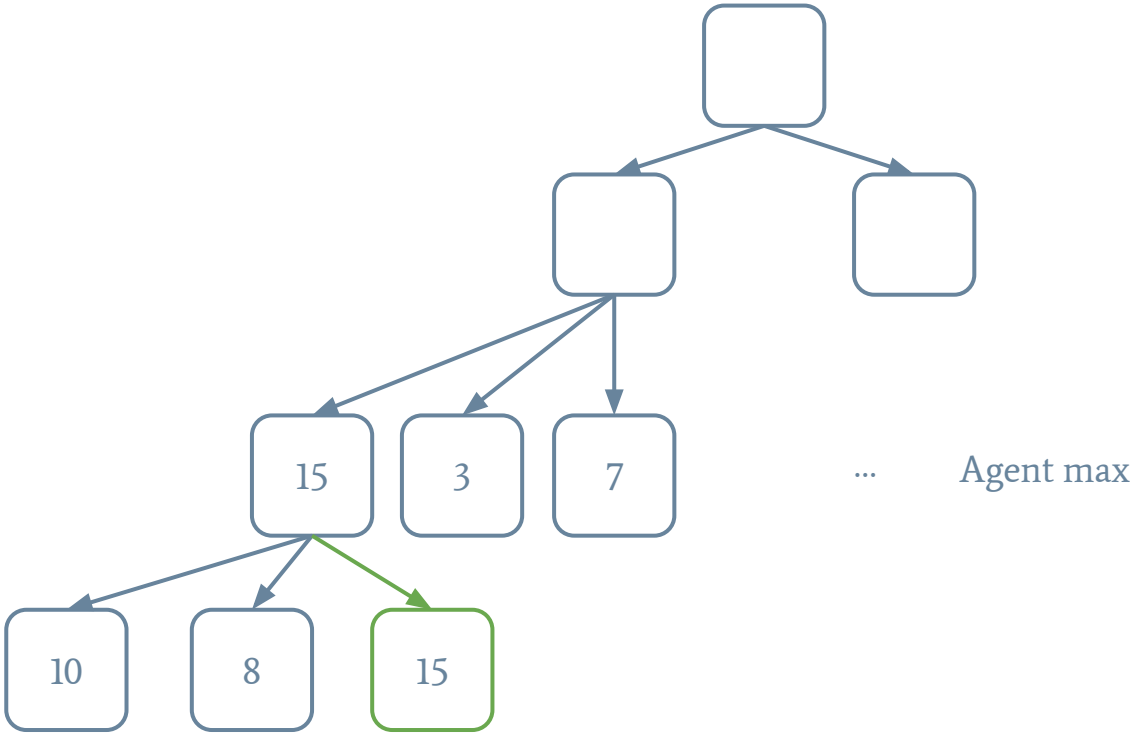
- Algorithme d'exploration souvent utilisé dans les intelligences artificielles qui consiste à simuler toutes les actions possibles de tous les agents jusqu'à une certaine profondeur.
- Les états sont évalués avec une fonction d'évaluation à la profondeur maximale.
- Deux types d'agents:
  - Agent max : choisit toujours l'évaluation maximale (le "gentil")
  - Agent min : choisit toujours l'évaluation minimale (le "méchant")
- Complexité exponentielle en la profondeur.

Minimax - CI9 (optionnel)

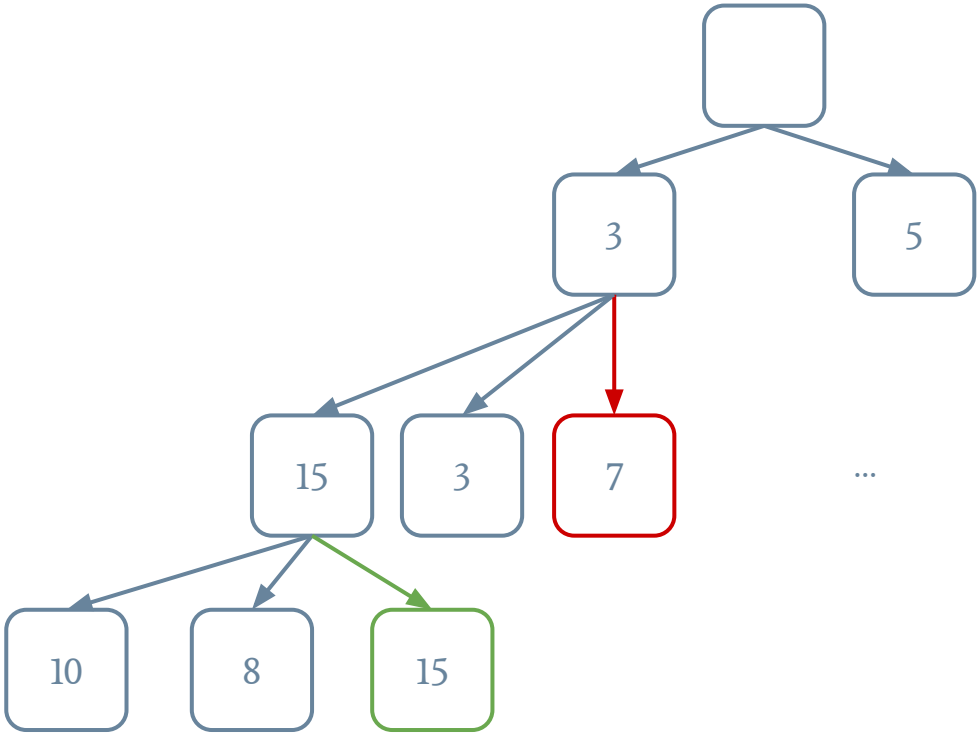


Évaluation des états à la profondeur maximale

Minimax - CI9 (optionnel)

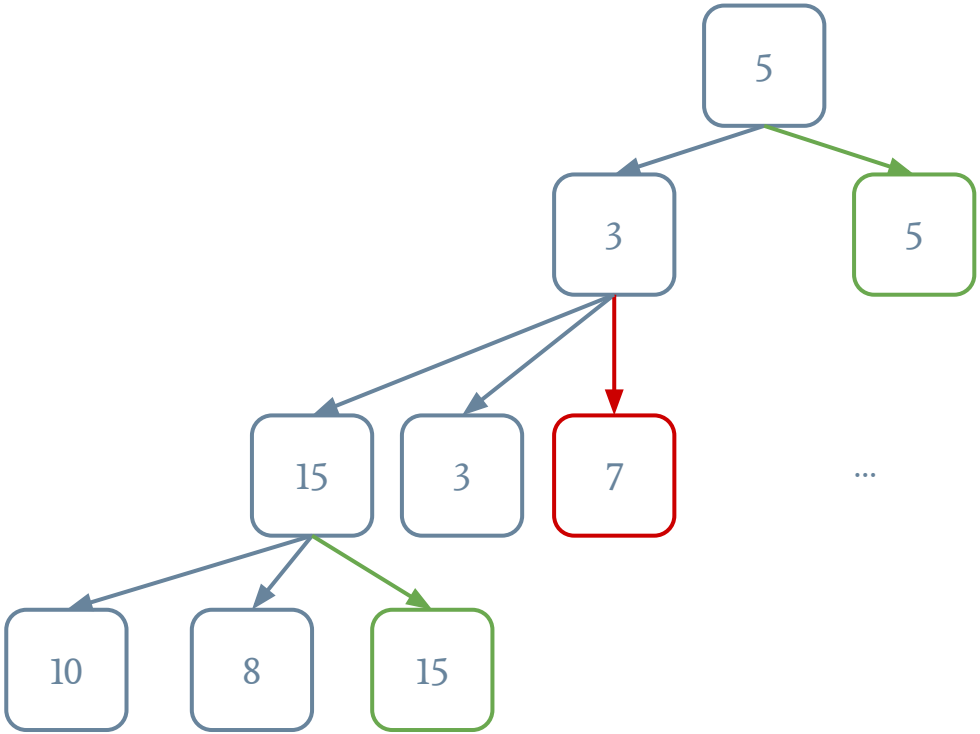


Minimax - CI9 (optionnel)



Agent min

Minimax - Cl9 (optionnel)



Agent max

Élagage AlphaBeta - CI9 (optionnel)

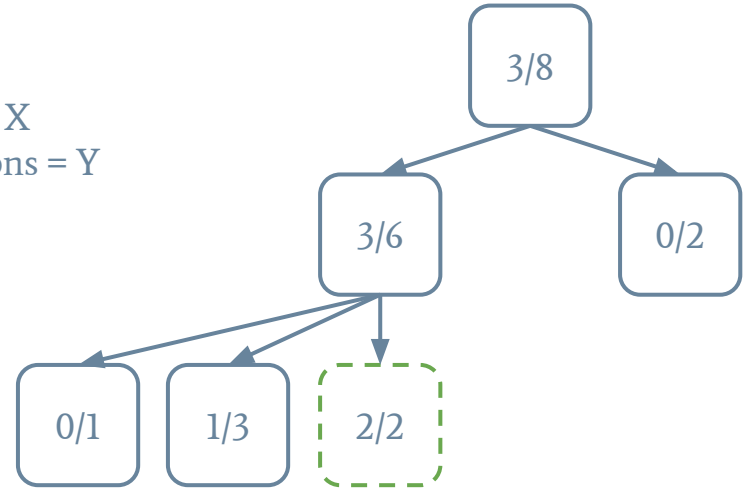
- Variante de Minimax qui supprime de manière intelligente des branches de l'arbre
- Accélération des calculs mais pas de changement de complexité

Monte Carlo Tree Search - CI9 (optionnel)

- Algorithme d'exploration qui approxime la valeur d'un état à l'aide de simulations aléatoires.
- Construit un arbre d'états au fur et à mesure
- Quatre actions principales :
  - La sélection : On choisit la feuille de l'arbre d'où partira la simulation aléatoire. La sélection se fait avec une métrique, UCB (Upper Confidence Bound) qui mélange désir de choisir la meilleure solution et d'explorer de nouveaux états
  - L'expansion : On ajoute les fils de la feuille précédemment sélectionnée à l'arbre
  - La simulation : On simule le jeu de manière aléatoire jusqu'à atteindre un état final ou une profondeur max.
  - La backpropagation : On évalue l'état à la fin de la simulation et on fait remonter son score dans l'arbre d'exploration.

Monte Carlo Tree Search - CI9 (optionnel) - Sélection

X/Y =>
Somme des scores = X
Nombre d'explorations = Y

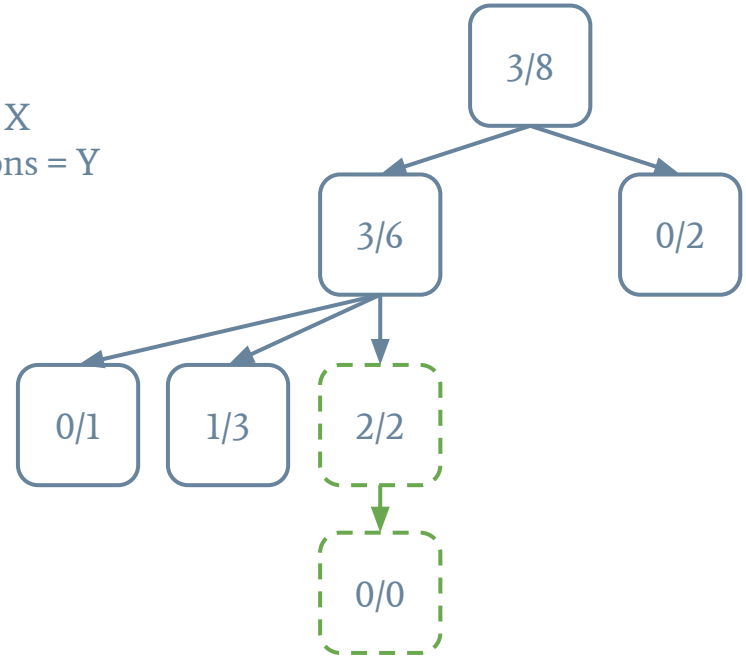


La sélection dépend de la métrique (e.x. UCB)

Pour simplifier, nous considérons qu'il n'y a qu'un seul agent.

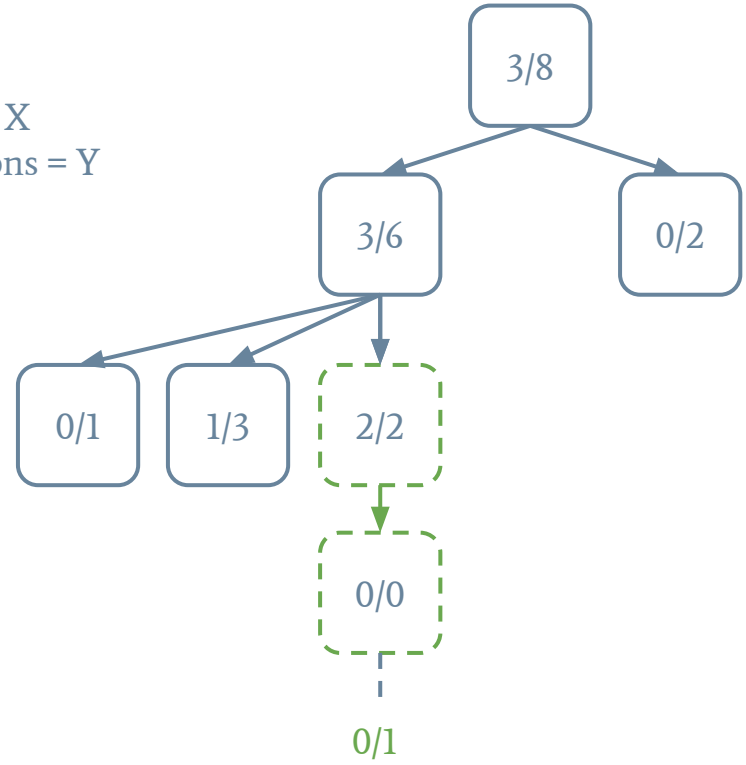
Monte Carlo Tree Search - CI9 (optionnel) - Expansion

X/Y =>
Somme des scores = X
Nombre d'explorations = Y



Monte Carlo Tree Search - CI9 (optionnel) - Simulation

X/Y =>
Somme des scores = X
Nombre d'explorations = Y



Monte Carlo Tree Search - CI9 (optionnel) - Backpropagation

X/Y =>
Somme des scores = X
Nombre d'explorations = Y