Sécurité et aide à la décision

Grégory Bonnet

Université de Caen Normandie – GREYC

Plan du cours

Sécurité informatique

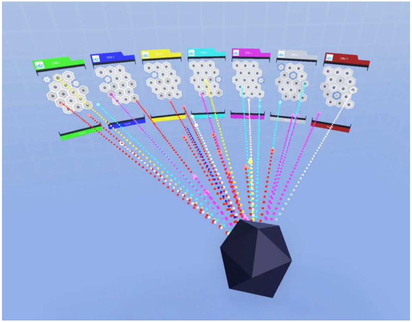
De quoi parlons-nous ?

Aide à la décision

Ensemble des techniques permettant d'opter pour la meilleure prise de décision possible. Parmi ces techniques, nous considérerons certaines techniques d'intelligence artificielle.

Sécurité informatique

Ensemble des politiques et procédures permettant d'éviter les intrusions (confidentialité), les incohérences (intégrité) et les pannes (disponibilité) des systèmes et qui définissent les règles d'authentification. Nous modéliserons ces problèmes sous forme de jeux adversariaux.



Jeux adversariaux

Problème

- États $S = \{s_0 \dots s_z\}$
- Joueurs $P = \{p_1 \dots p_y\}$
- Actions $A = \{a_1 \dots a_x\}$

- Fonction de transition $t : S \times A \rightarrow S$
- Conditions de fin de partie $v : S \rightarrow \{\top, \bot\}$
- Utilité $u : S \times P \rightarrow \mathbb{R}$

Solution

Politique $\pi : S \rightarrow A$

Objectif

Trouver une politique optimale qui maximise l'utilité du joueur qui l'applique sachant que ses adversaires vont agir de manière optimale

Question

Comment planifier une séquence d'action $\{a_i \dots a_j\}$ alors qu'un autre agent est en train de planifier contre nous et intercalle ses actions $\{a_k \dots a_l\}$ entre les notres ?

Exemple : les jeux de plateaux

Échecs, dames, go, etc.

	Déterministe	**Stochastique**
Information parfaite	Échecs, dames, go, othello	Backgammon, monopoly
Information imparfaite	Bridge, skat	Poker, scrabble, blackjack

Théorème de Harsanyi

Tout jeu stochastique peut être représenté par un jeu déterministe à information imparfaite.

Jouer c'est explorer « intelligemment » un graphe représentant un jeu

Définitions

- état : représentation s_i unique du jeu à un instant donné (situation, état)
- coup : passage d'un état s_i à un état s_j avec une action a_k
- joueur : entité qui cherche à passer d'un état initial à un état gagnant

Une représentation possible : un graphe orienté

- $\mathcal{G} = (S, E)$
- $\mathcal{E} = \{(s_i, s_j) : \exists a_k \in A, t(s_i, a_k) = s_j\}$

Problématiques informatiques

Exemple des échecs (approximé par Claude Shannon)

- entre 1 et 218 coups possibles par tour (moyenne à 30)
- environ 40 coups par partie « raisonnable »
- $\implies (30^2)^{40} = 10^{120}$ parties possibles (en fait $> 10^{6000}$ mais absurdes)
- $\implies 10^{50}$ positions légales

Impossible de mémoriser

- toutes les parties possibles
- toutes les positions possibles
- toutes les positions gagnantes
- $\implies$ une stratégie toujours gagnante

Questions

Comment calculer à la volée le meilleur coup possible ? Comment raisonner sur le jeu ?

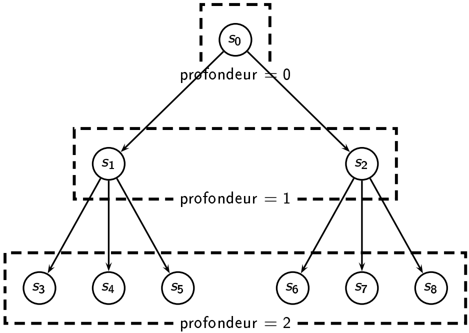
Explorer l'arbre de jeu

Principes généraux

Idée

- chercher à jouer le *meilleur* coup au regard de ce que l'on sait sur le jeu
- s'il existe un meilleur coup pour l'adversaire alors il le jouera (*symétrie*)
- choisir le *nombre de coups* à évaluer

Un arbre construit itérativement et une recherche avec une profondeur donnée



Donner une valeur aux positions pour les comparer

Fonction d'évaluation

Une fonction d'évaluation f pour un jeu donné est une application $f : S \longmapsto \mathbb{R}$.

Exemples

- États gagnants (valeur positive importante)
- États perdants (valeur négative importante)
- Somme des valeur des pièces
- Positionnement stratégique
- Mise en difficulté de l'adversaire
- ...

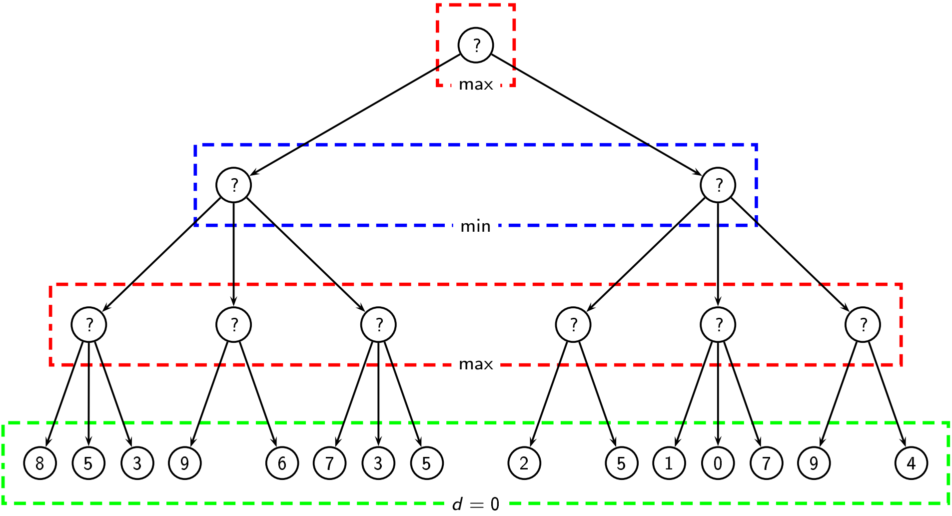
Plusieurs fonctions peuvent être combinées

Minimax

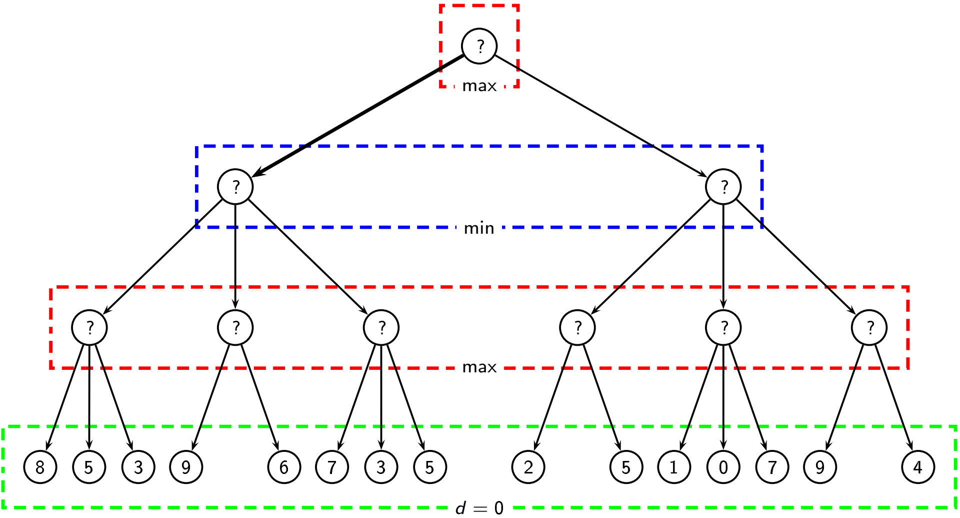
Algorithme $\mathrm{minimax}(s_i, d)$

```
if d = 0 ∨ v(s_i) then
    return f(s_i)
else
    if s_i est un nœud Max then
        b ← −∞
        for all s_j ∈ {s : ∃a_k ∈ A, t(s_i, a_k)} do
            m ← minimax(s_j, d − 1)
            if b < m then
                b ← m
            end if
        end for
    else
        b ← +∞
        for all s_j ∈ {s : ∃a_k ∈ A, t(s_i, a_k)} do
            m ← minimax(s_j, d − 1)
            if b > m then
                b ← m
            end if
        end for
    end if
    return b
end if
```

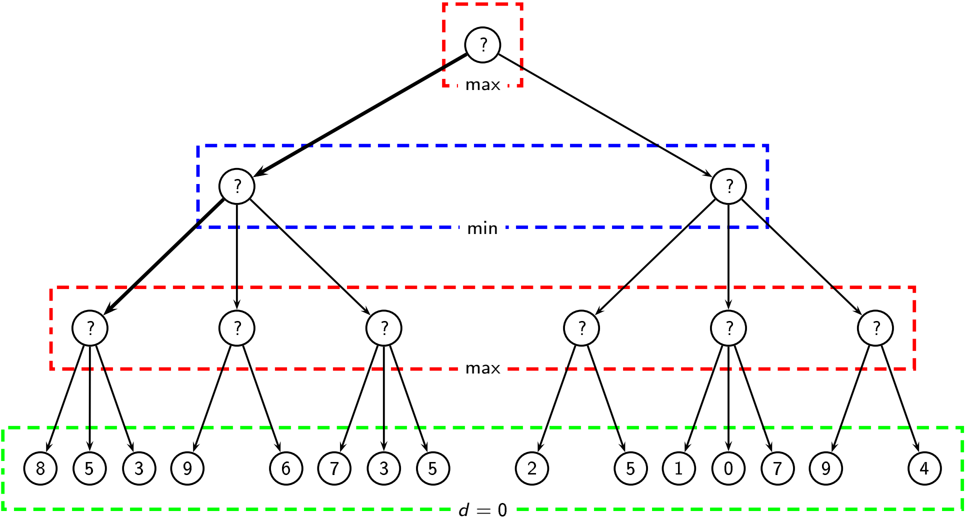
Exemple



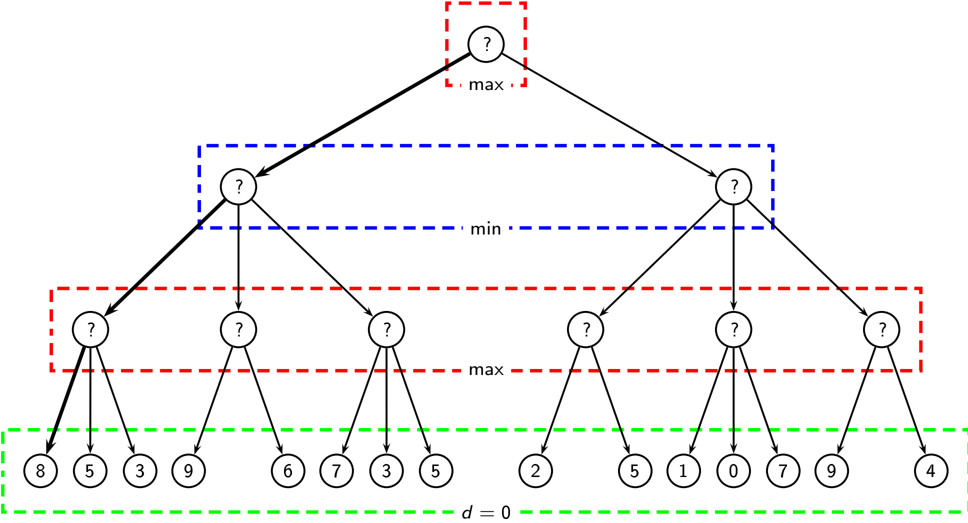
Exemple



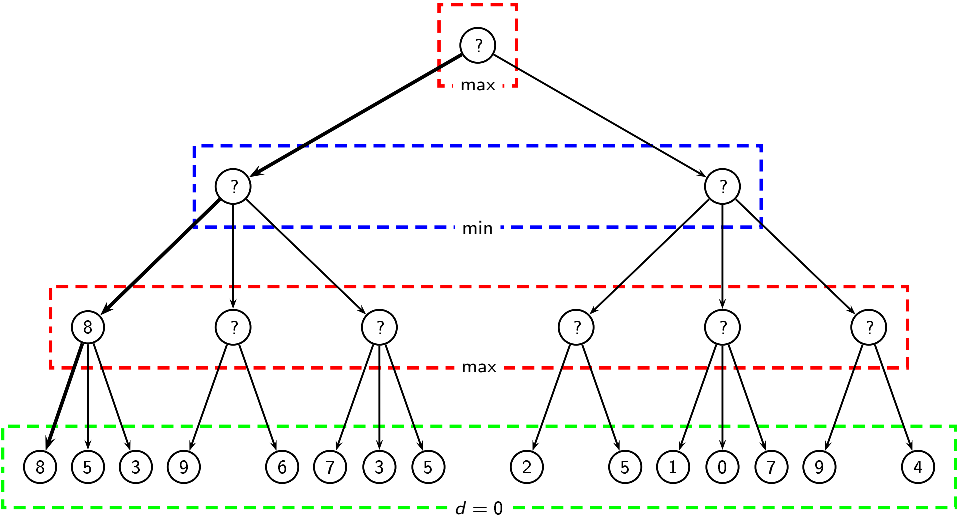
Exemple



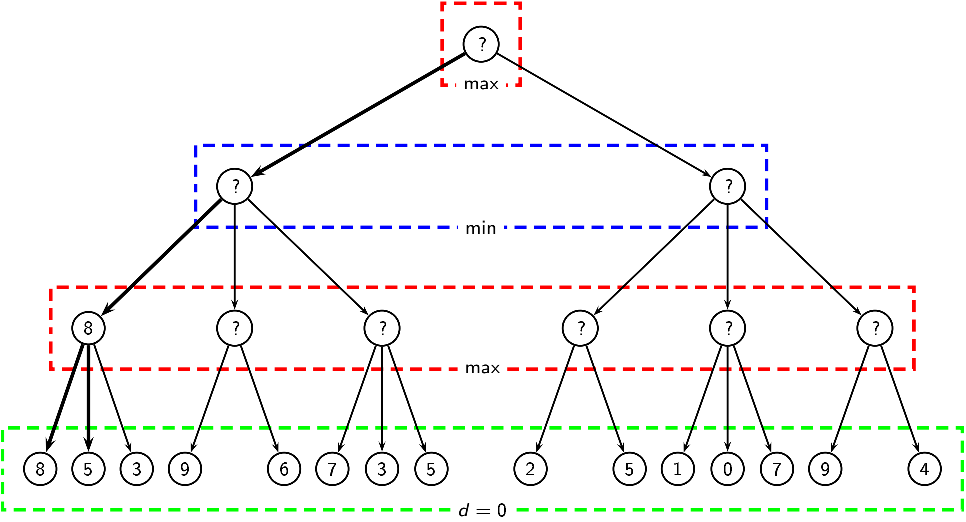
Exemple



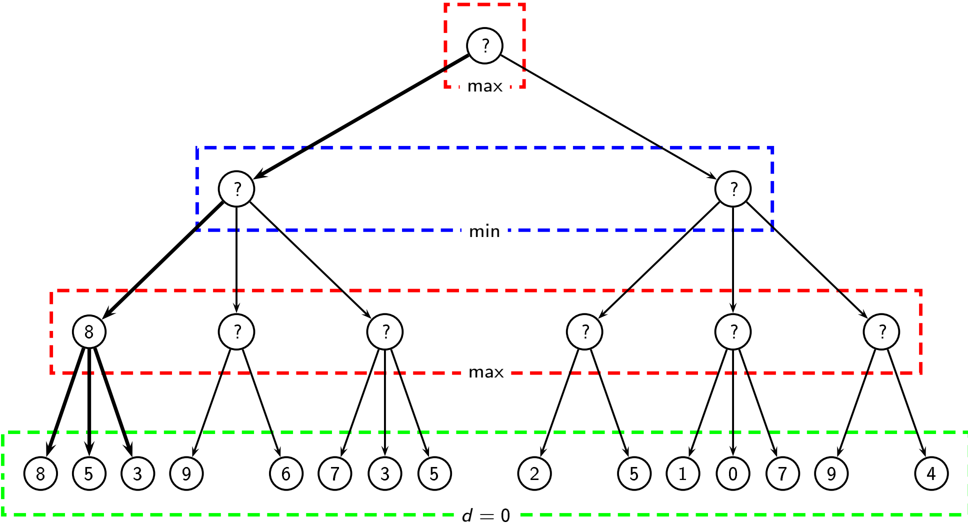
Exemple



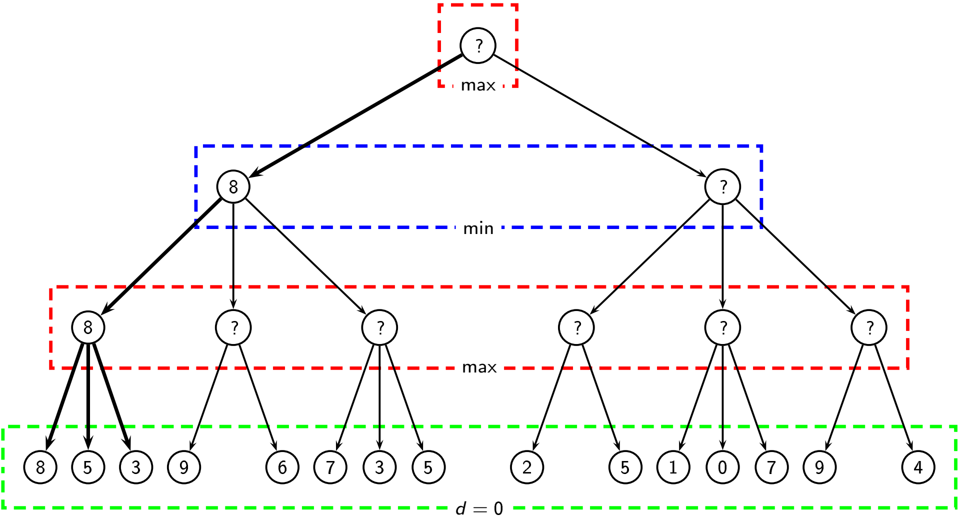
Exemple



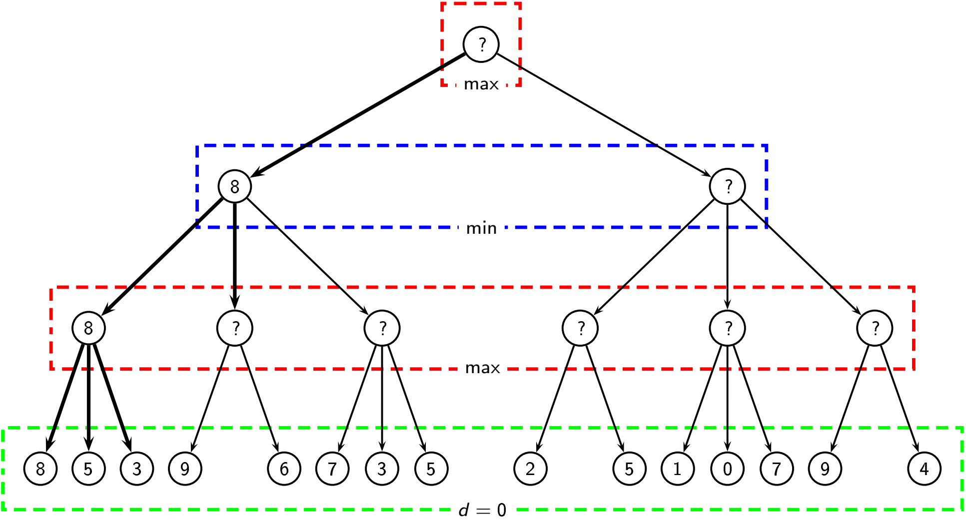
Exemple



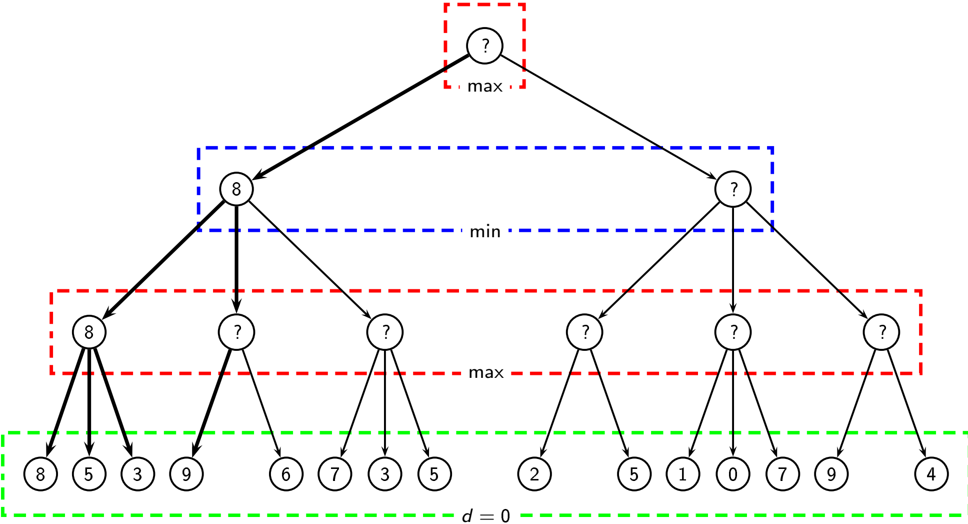
Exemple



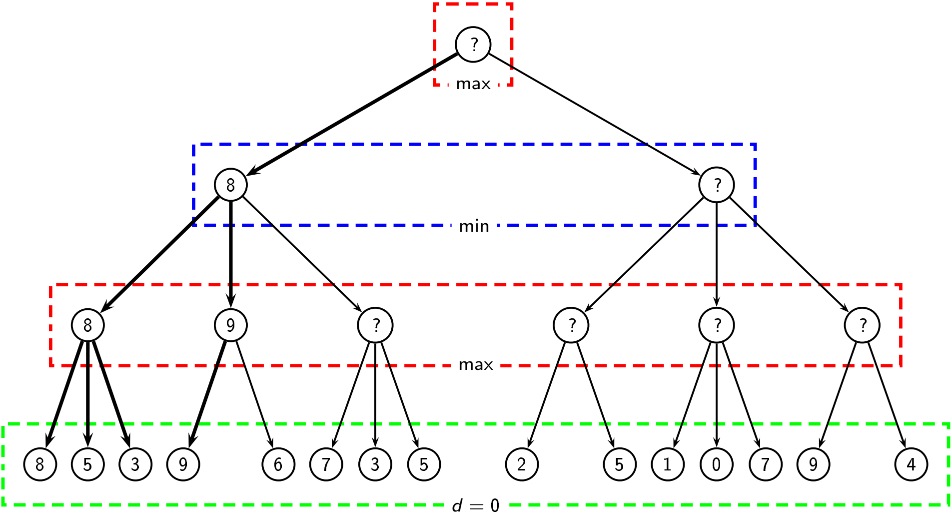
Exemple



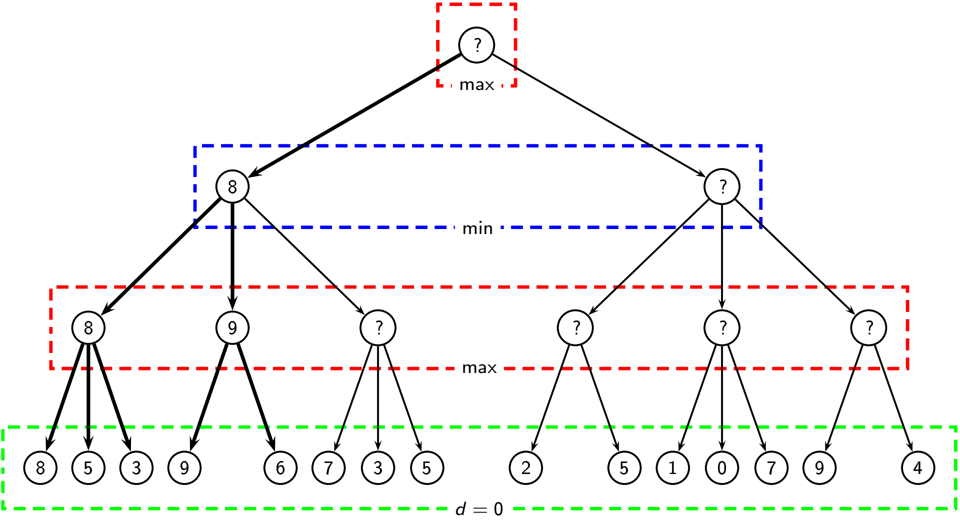
Exemple



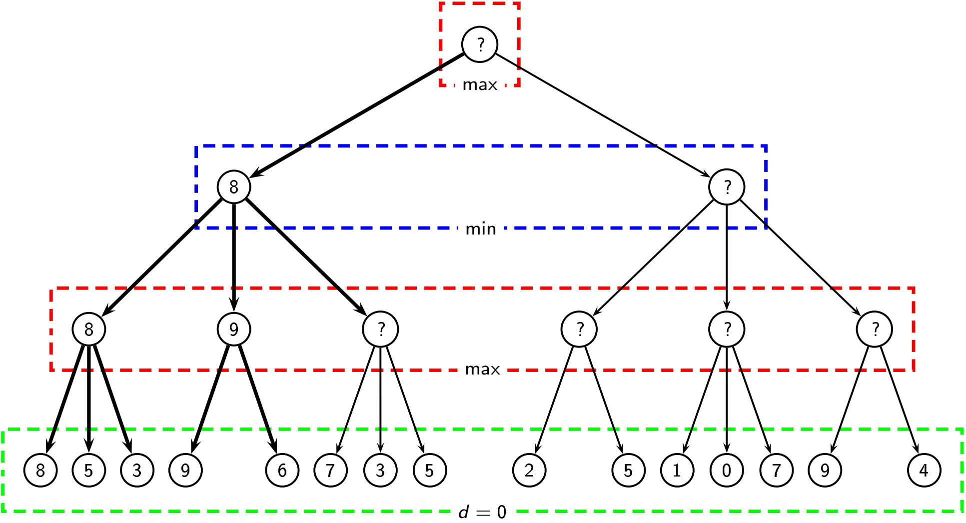
Exemple



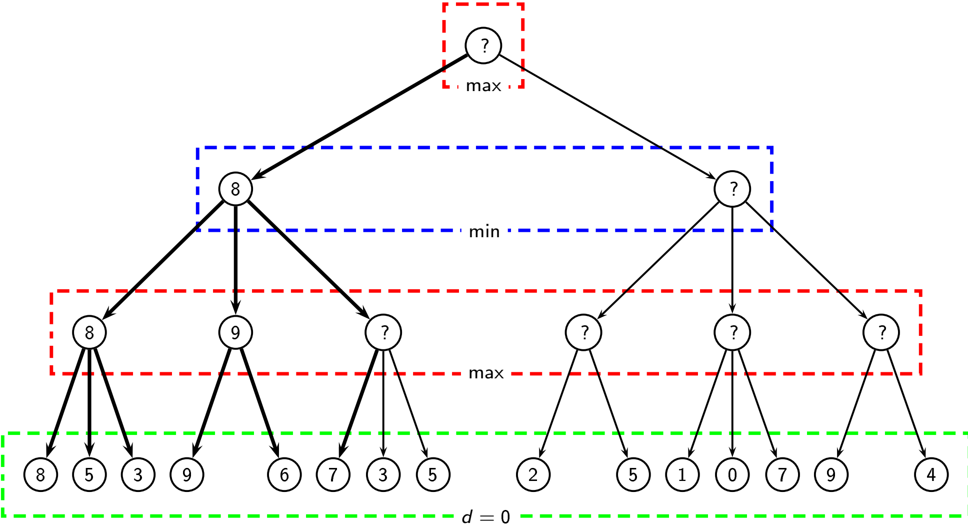
Exemple



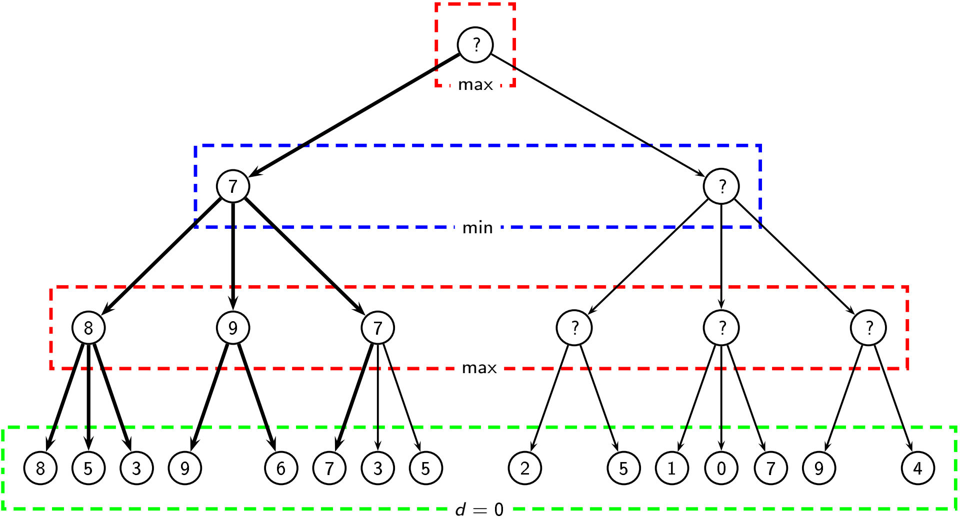
Exemple



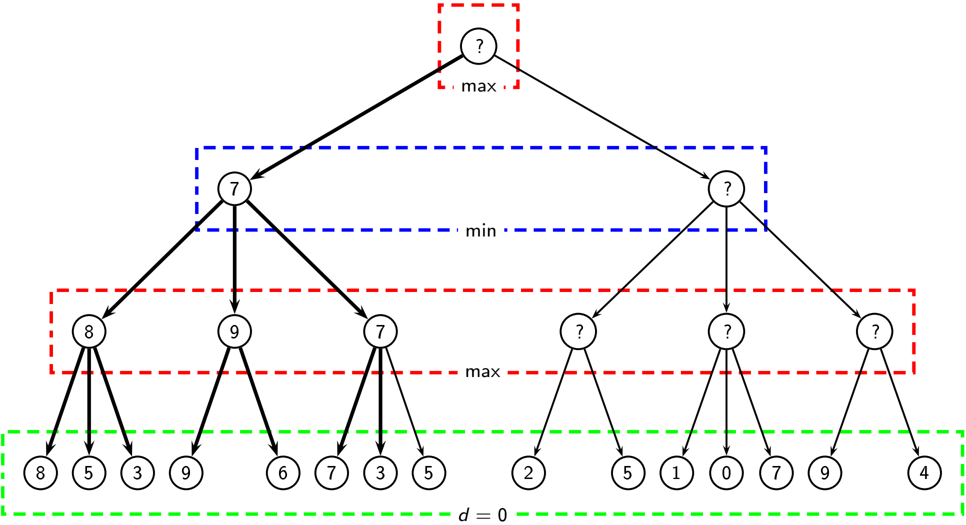
Exemple



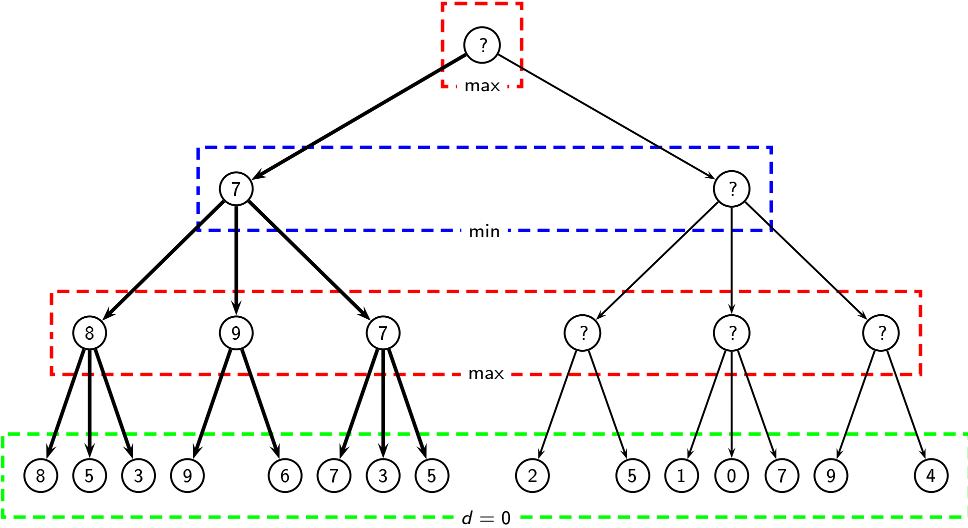
Exemple



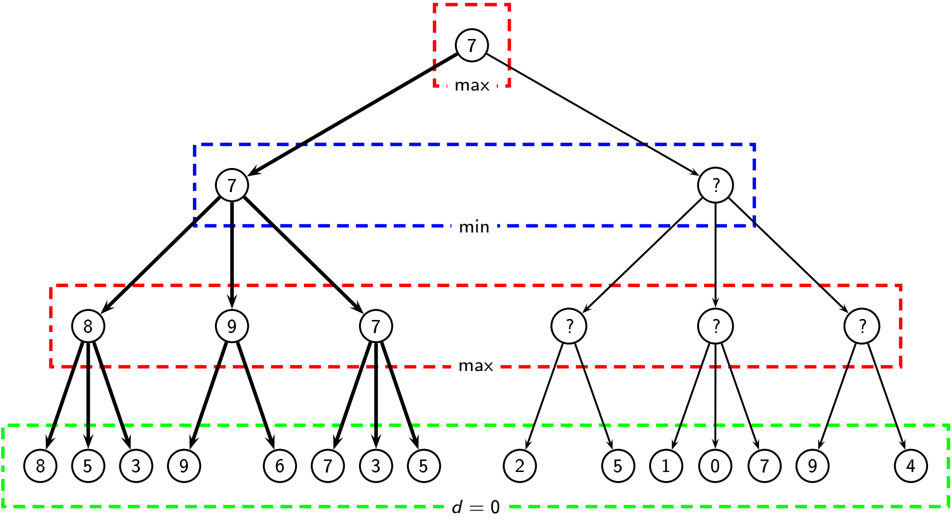
Exemple



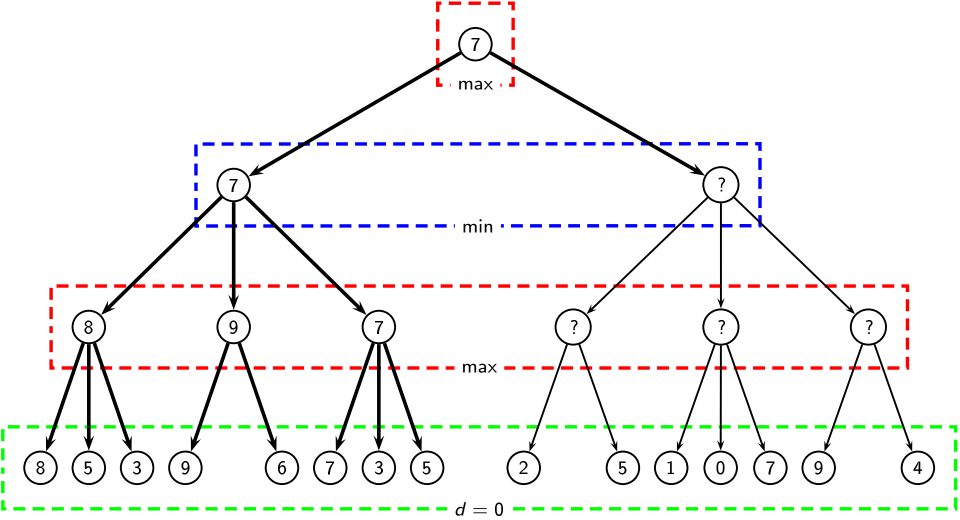
Exemple



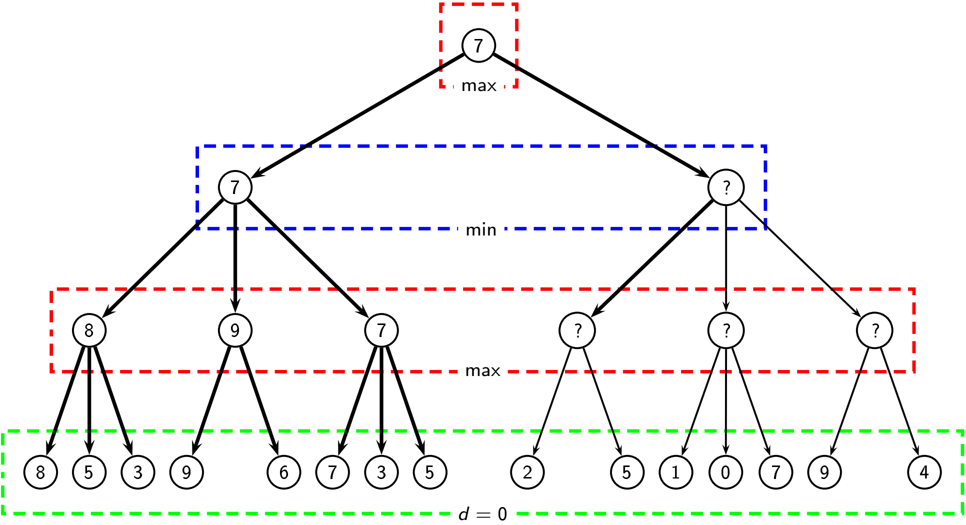
Exemple



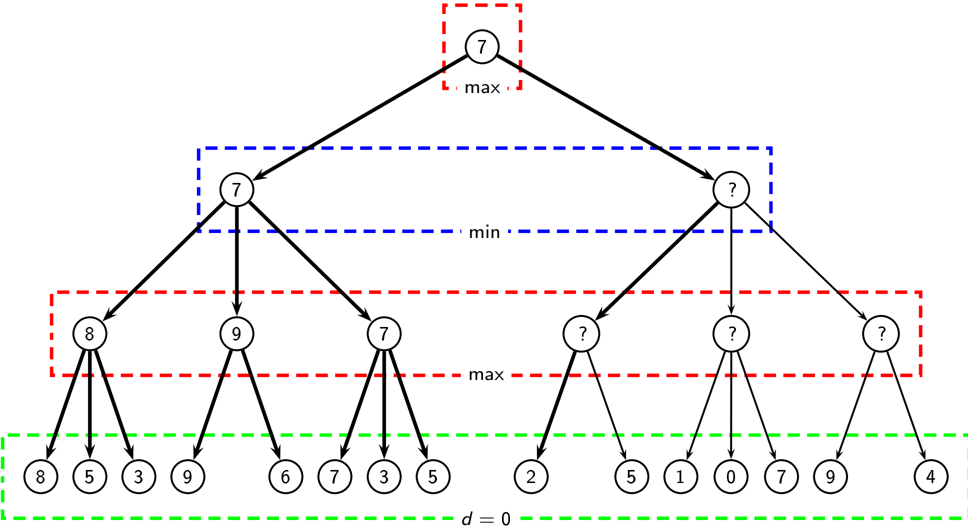
Exemple



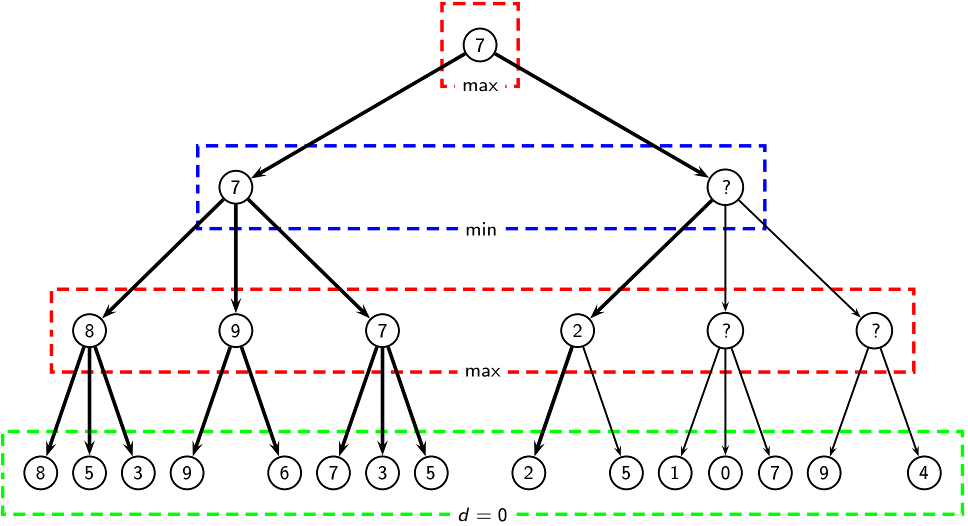
Exemple



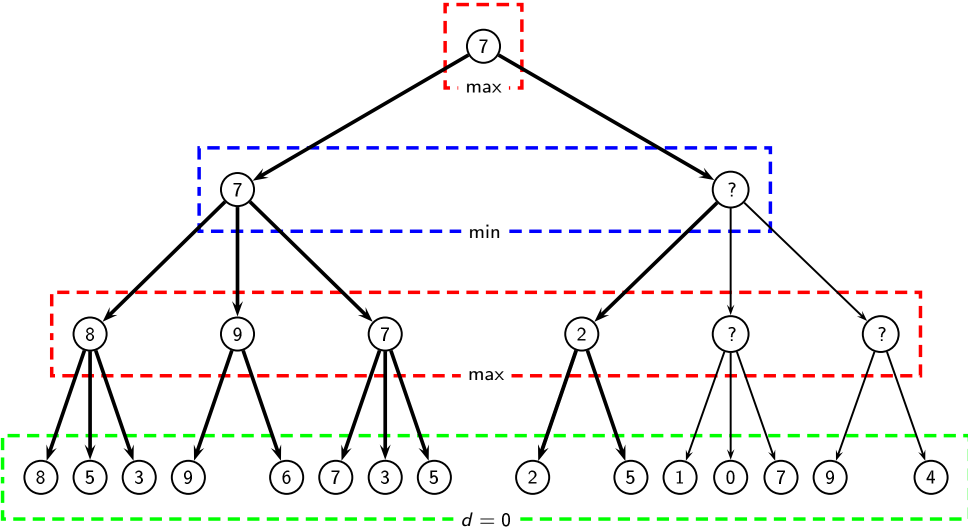
Exemple



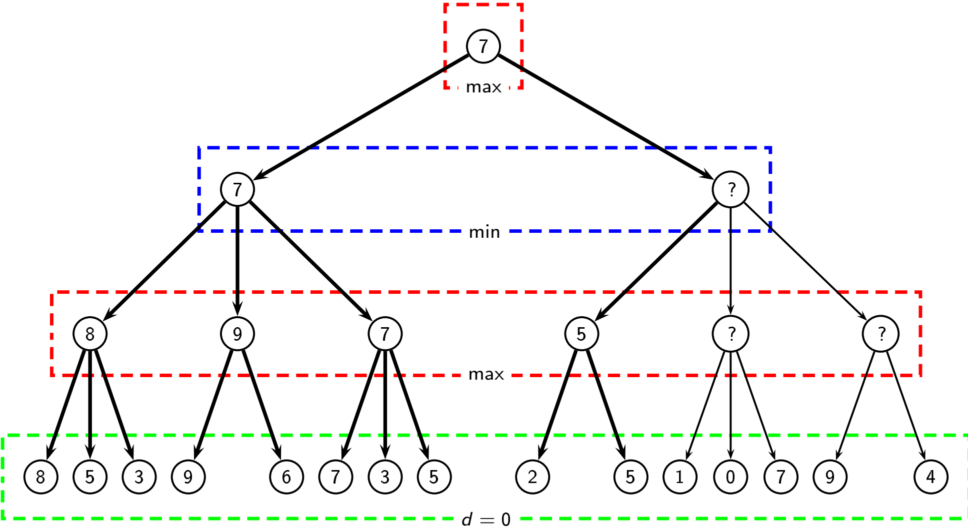
Exemple



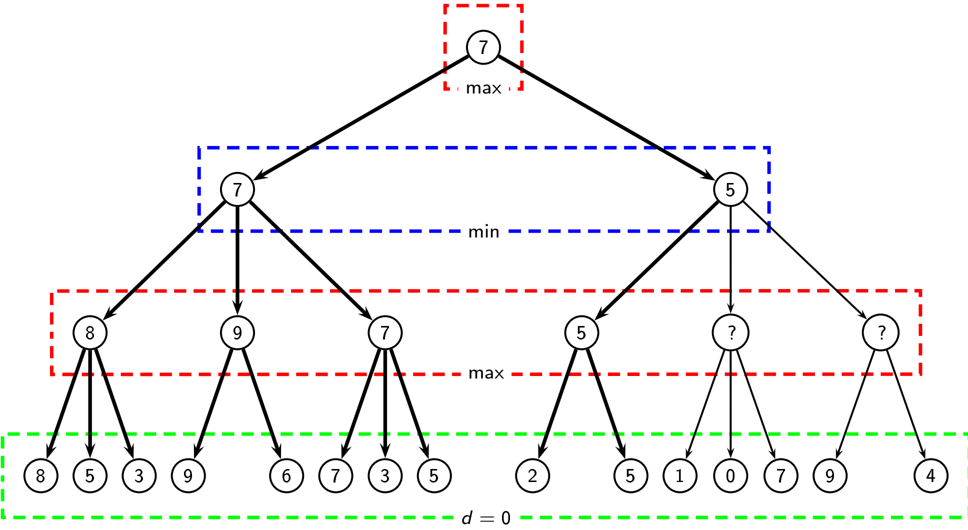
Exemple



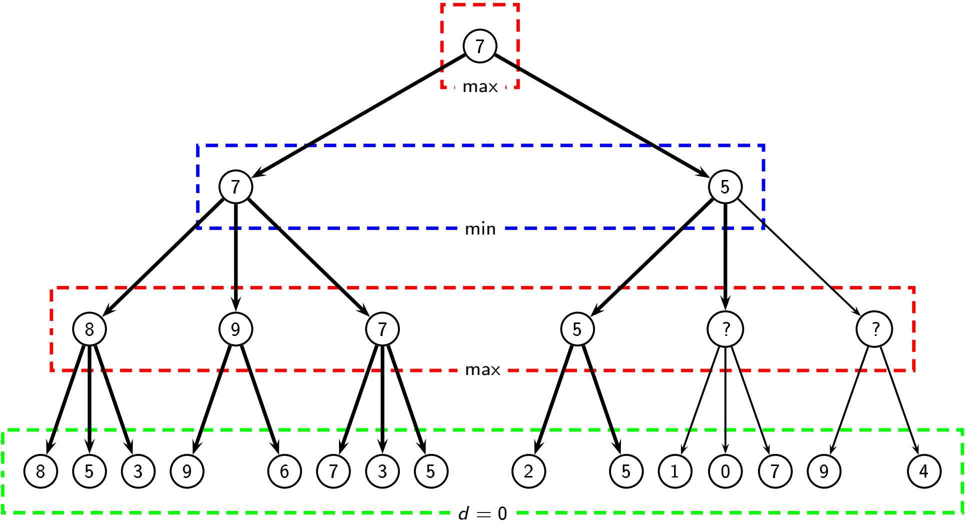
Exemple



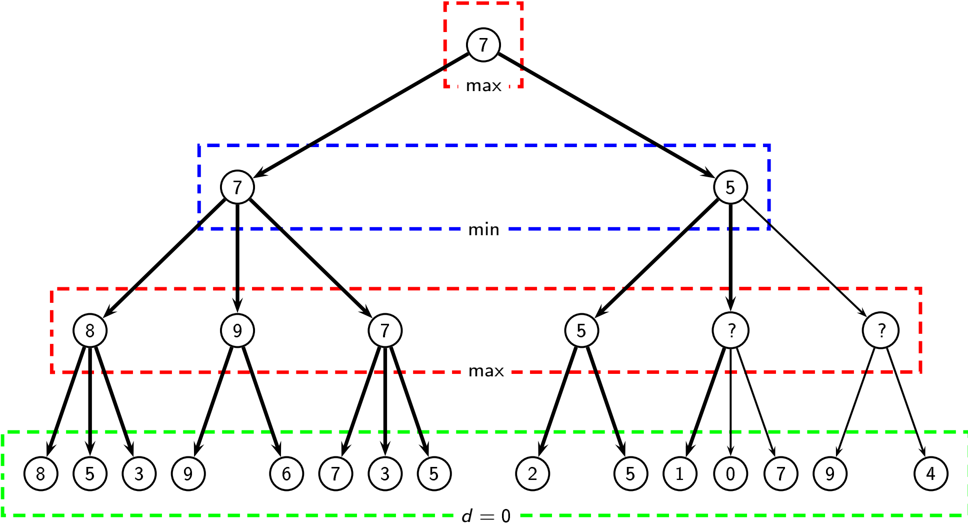
Exemple



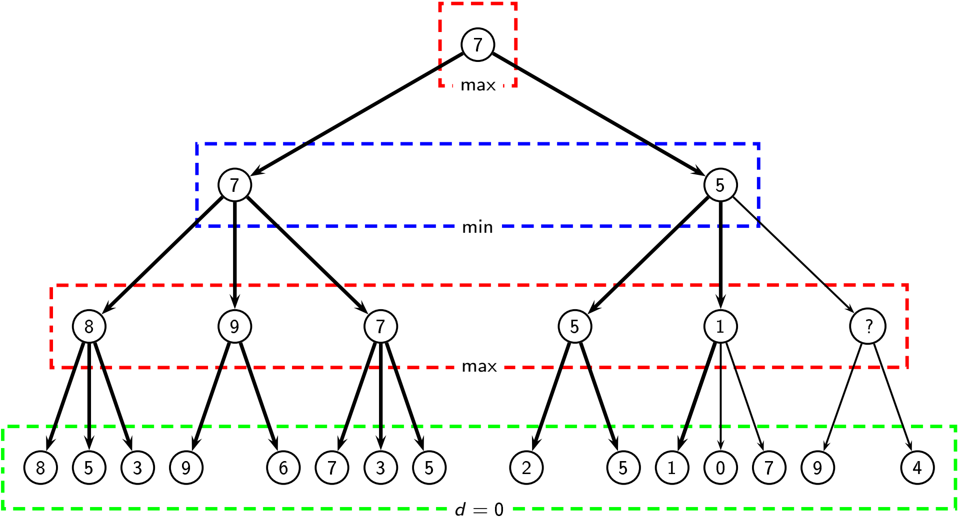
Exemple



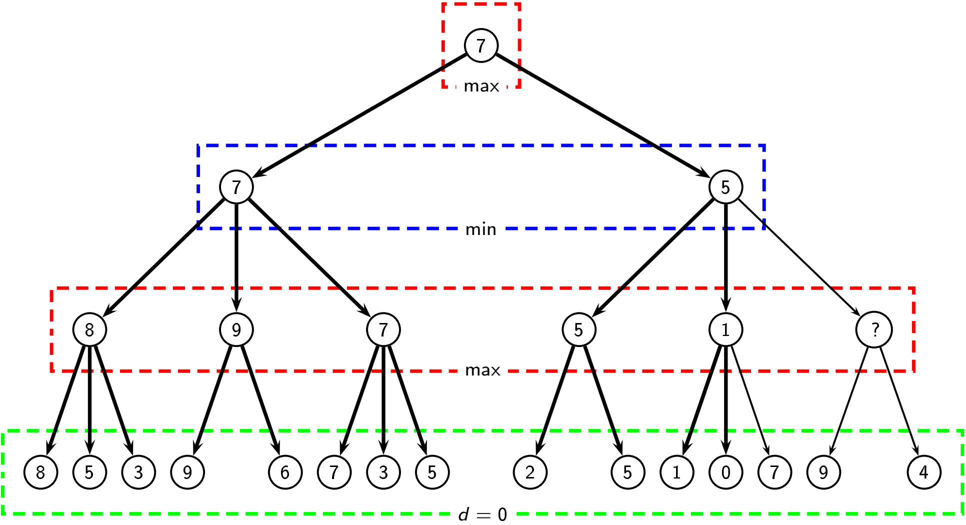
Exemple



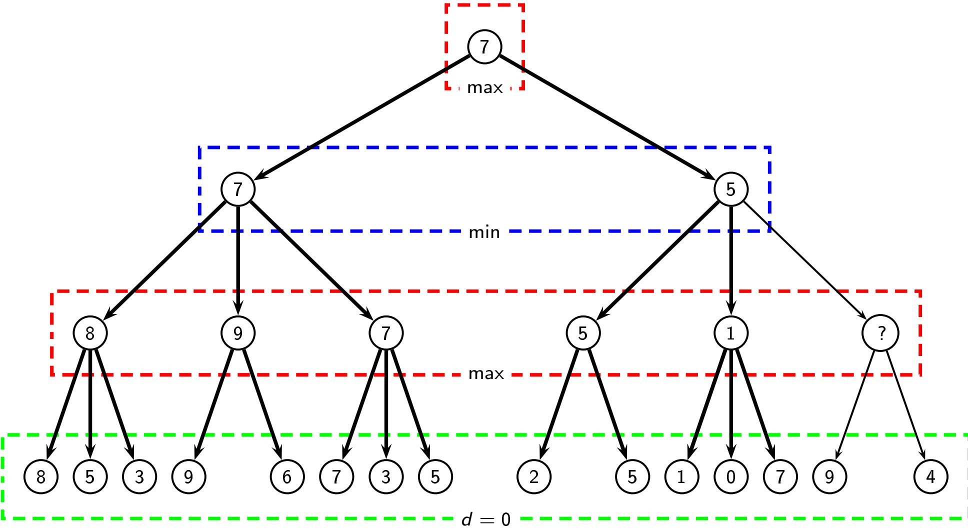
Exemple



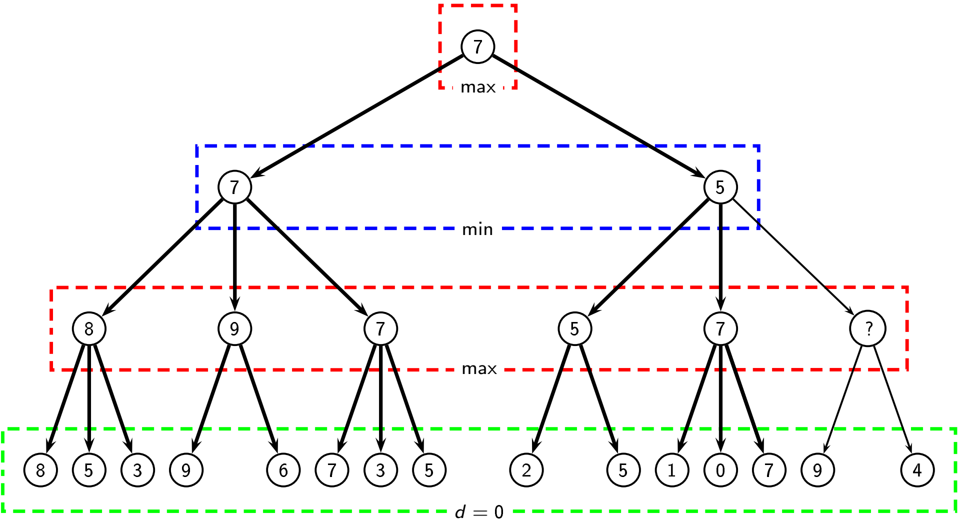
Exemple



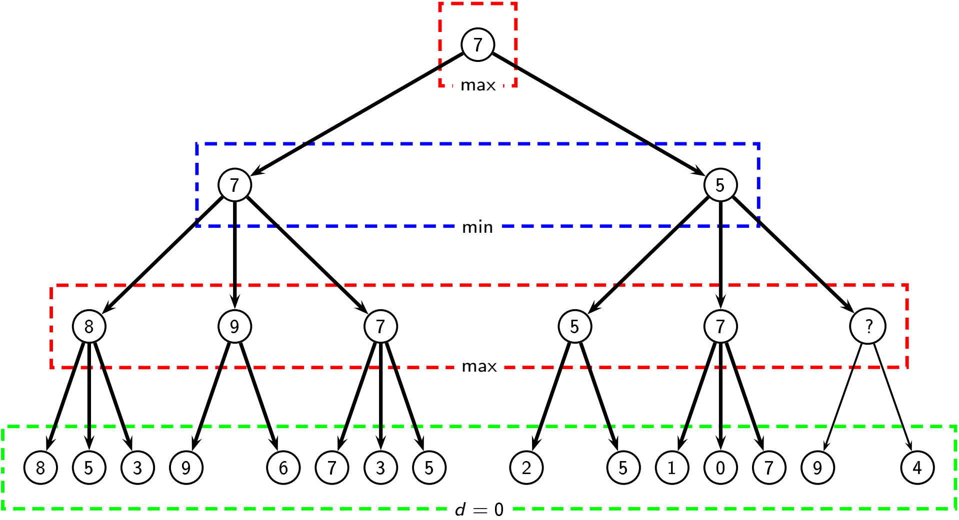
Exemple



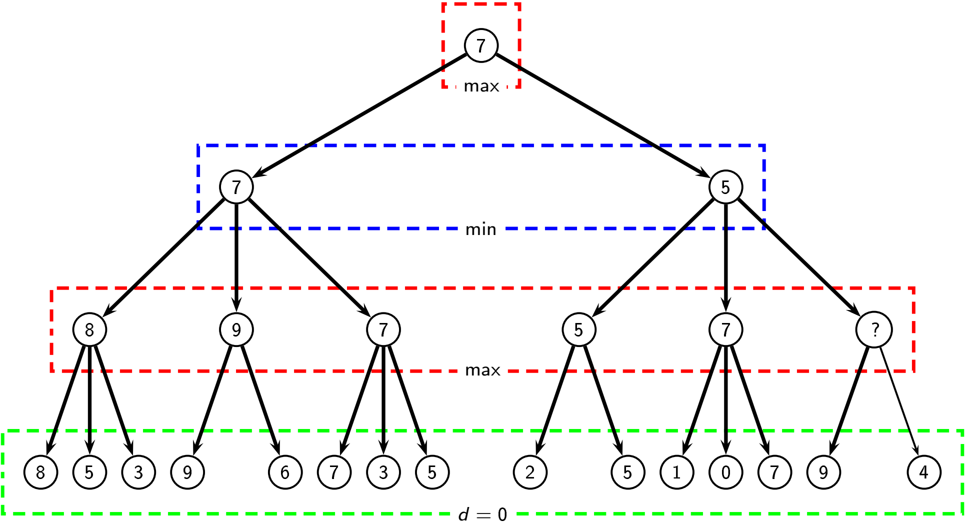
Exemple



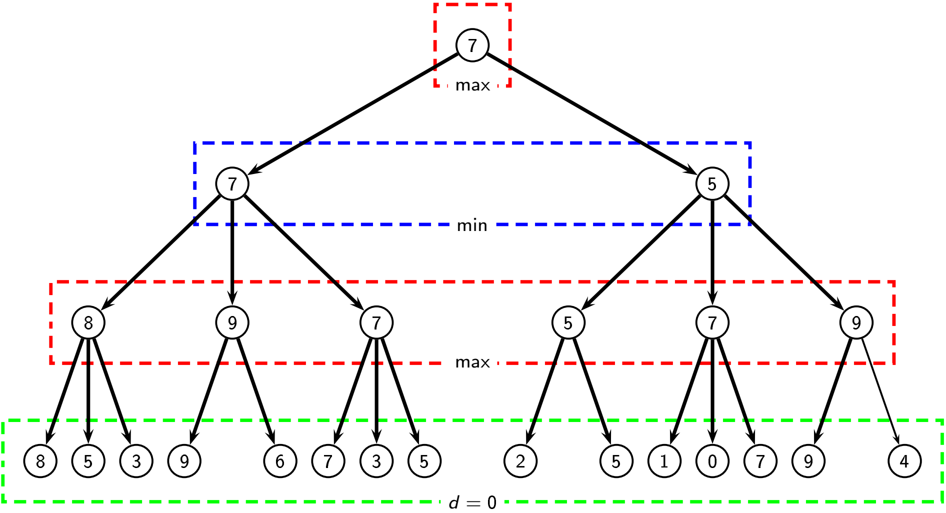
Exemple



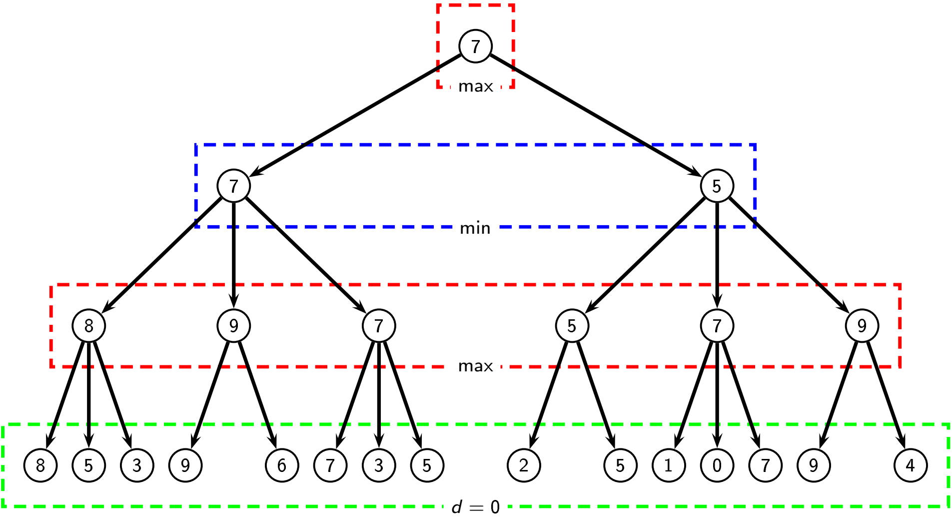
Exemple



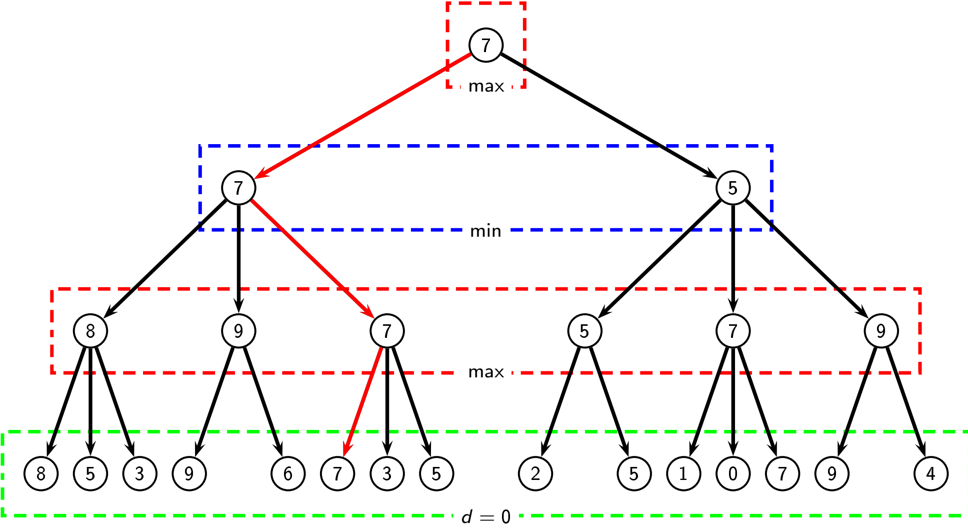
Exemple



Exemple



Exemple



Negamax

Hypothèses et principes géneraux

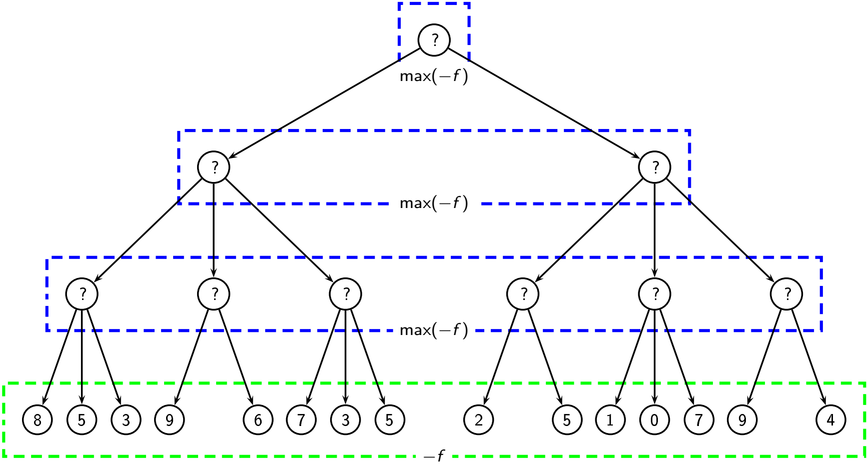
- minimiser f est équivalent à maximiser $-f$ (*dualité*)
- on définit $g : S \longmapsto \mathbb{R}$ tel que :

$$g(s_i) = f(s_i) \text{ si } s_i \text{ est un nœud du joueur, } g(s_i) = -f(s_i) \text{ sinon}$$

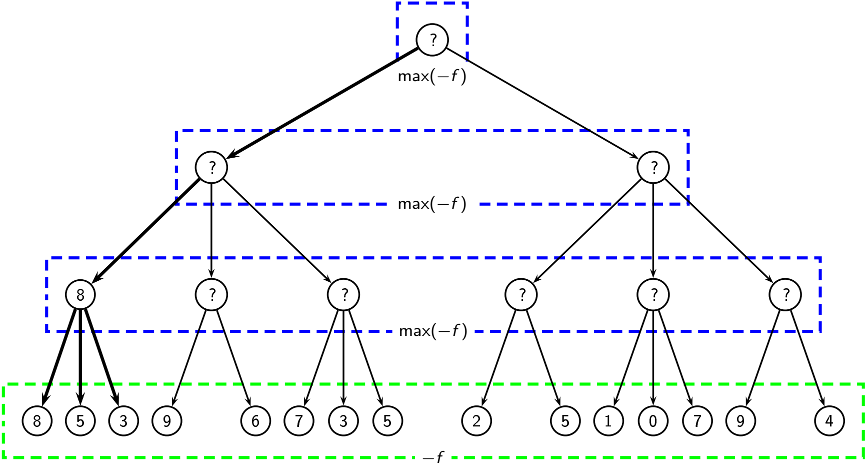
Algorithme $\mathrm{negamax}(s_i, d)$

if $d = 0 \vee v(s_i)$ **then**
  return $g(s_i)$
else
  $m \leftarrow -\infty$
  for all $s_j \in \{s : \exists a_k \in A, t(s_i, a_k)\}$ **do**
    $m \leftarrow \max(m, -\mathrm{negamax}(s_j, d-1))$
  end for
  return m
end if

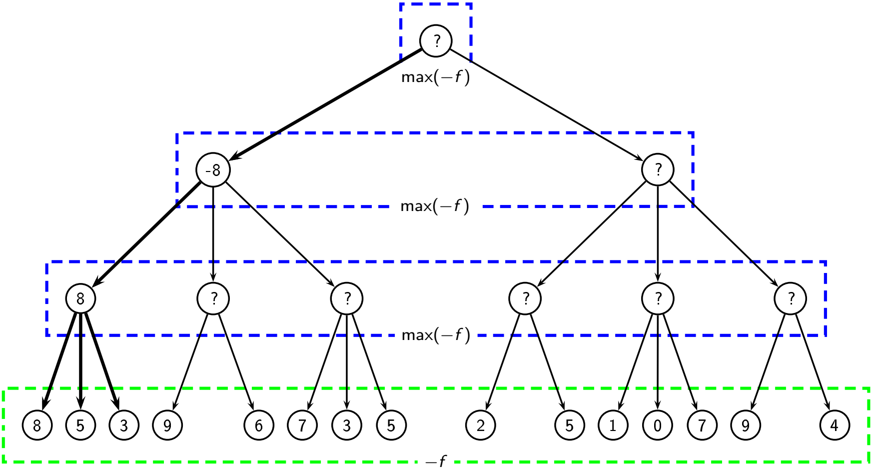
Sur le même exemple que précédemment



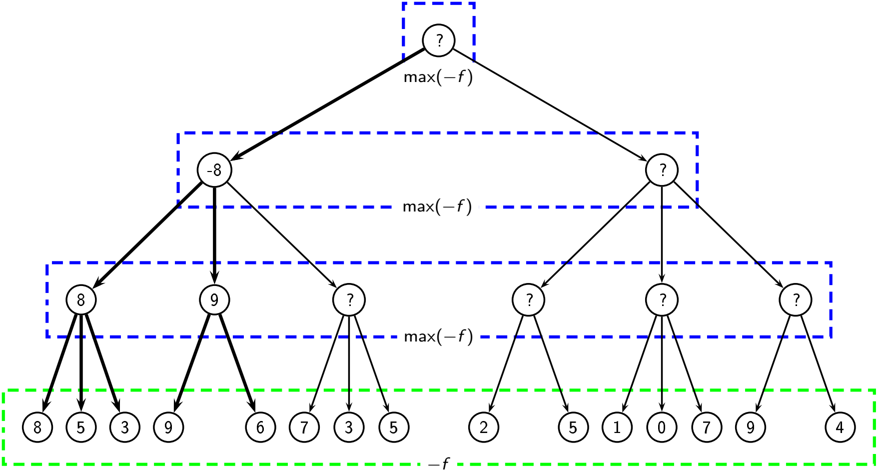
Sur le même exemple que précédemment



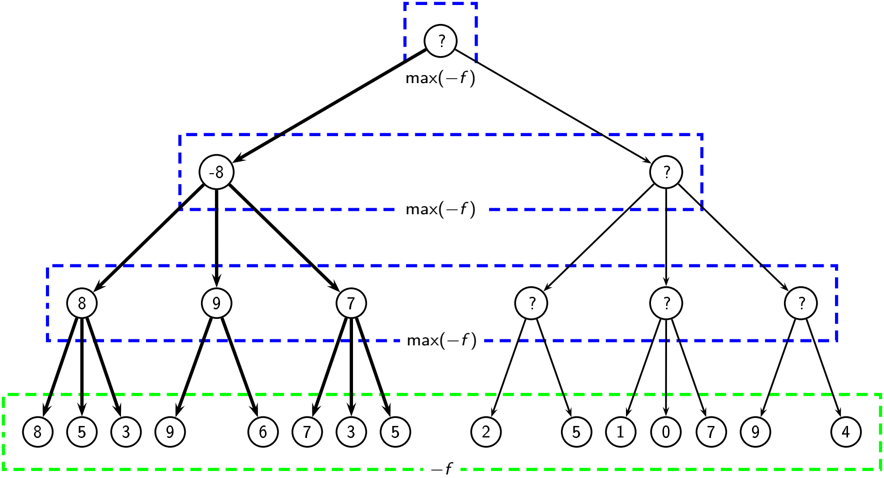
Sur le même exemple que précédemment



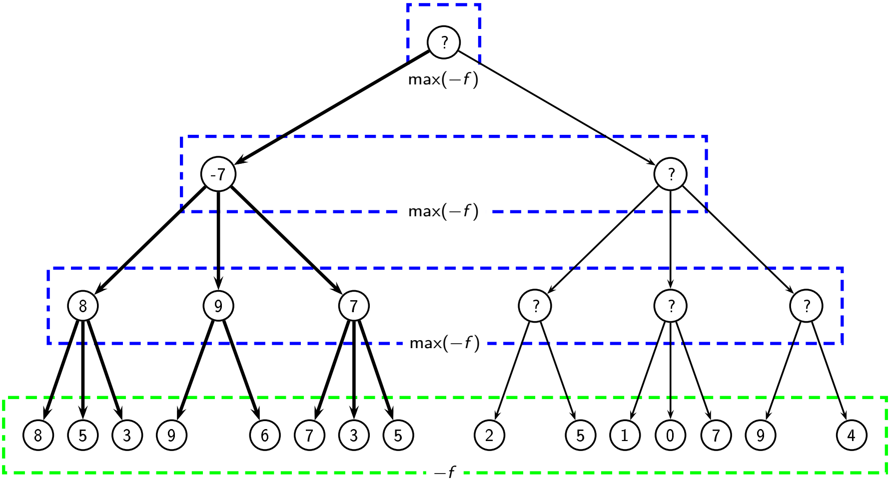
Sur le même exemple que précédemment



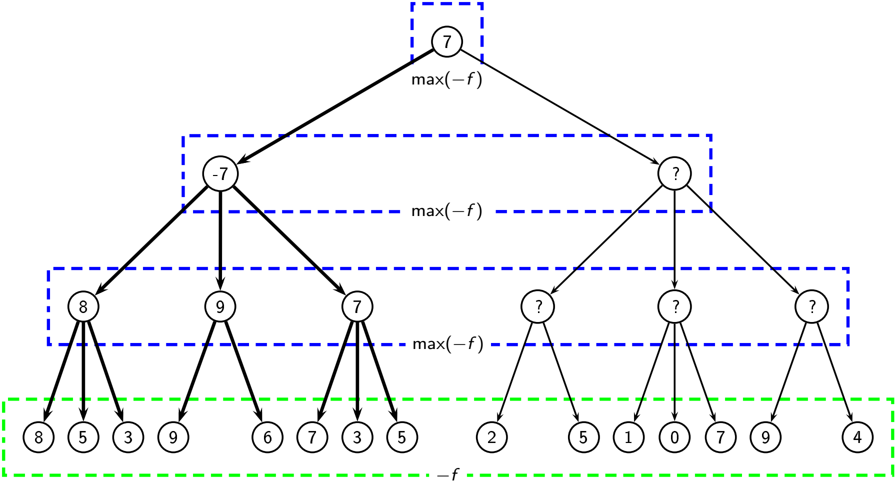
Sur le même exemple que précédemment



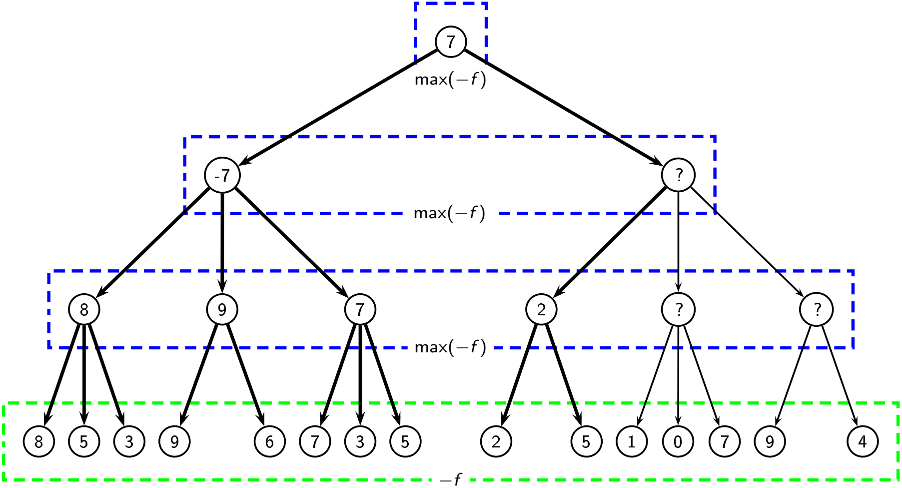
Sur le même exemple que précédemment



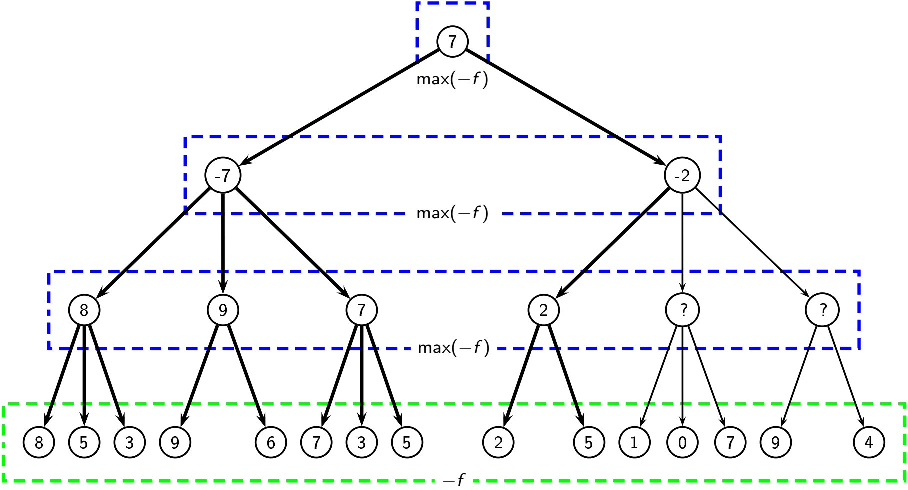
Sur le même exemple que précédemment



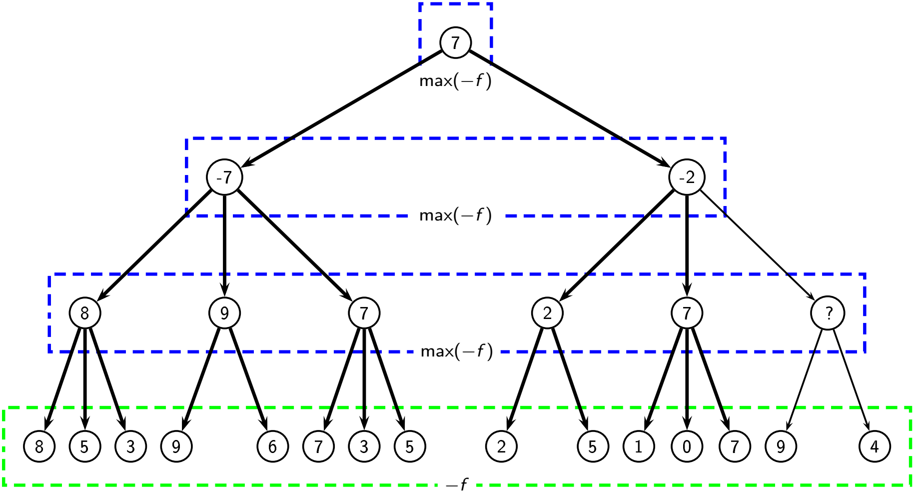
Sur le même exemple que précédemment



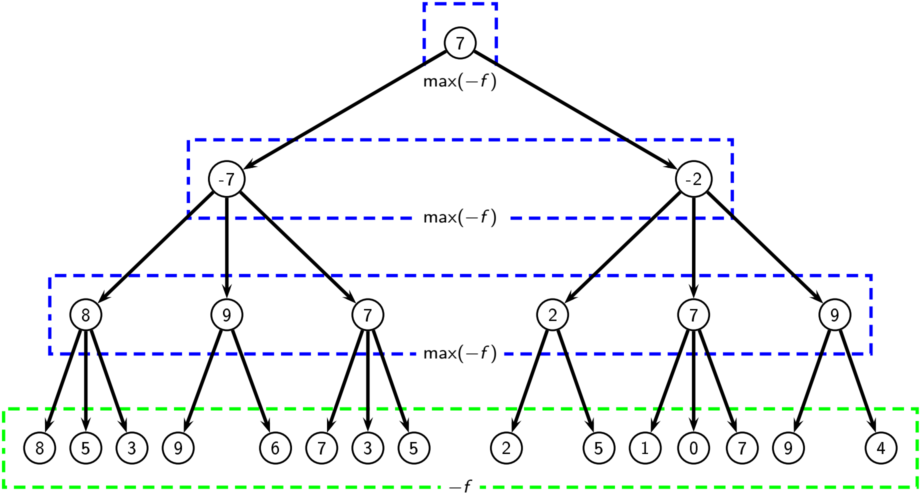
Sur le même exemple que précédemment



Sur le même exemple que précédemment



Sur le même exemple que précédemment



Techniques d'élagage

« Il est inutile d'évaluer des états dont nous sommes sûr que la valeur est inférieure à celle des états déjà évalués »

La coupe α et son symétrique (coupe β)

Imaginons sur un jeu d'échecs

- notre fonction d'évaluation prend 10^{-6} secondes pour un état
- minimax de profondeur 5 $\implies$ évaluer 30^5 état $\implies$ 24 *secondes*
- minimax de profondeur 6 $\implies$ évaluer 30^6 état $\implies$ 12 *minutes*
- minimax de profondeur 7 $\implies$ évaluer 30^7 état $\implies$ 6 *heures*

Notations

- α : valeur la plus basse que le joueur Max sait pouvoir obtenir
- β : valeur maximale que le joueur Min autorisera Max à obtenir

Propriété

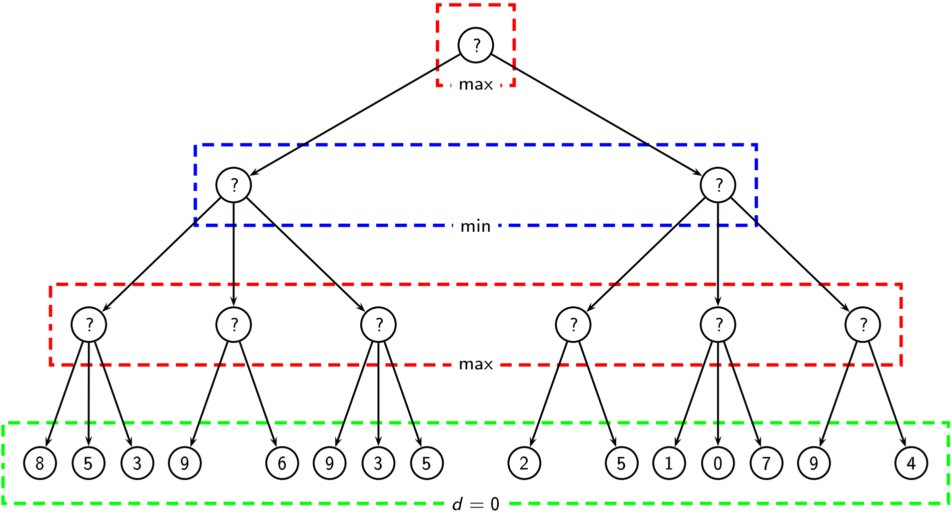
Si `minimax` trouve un coup en n itérations, `alphabeta` le fait en $2\sqrt{n} - 1$ si les coups sont ordonnés du meilleur au moins bon.

Dans l'algorithme minimax

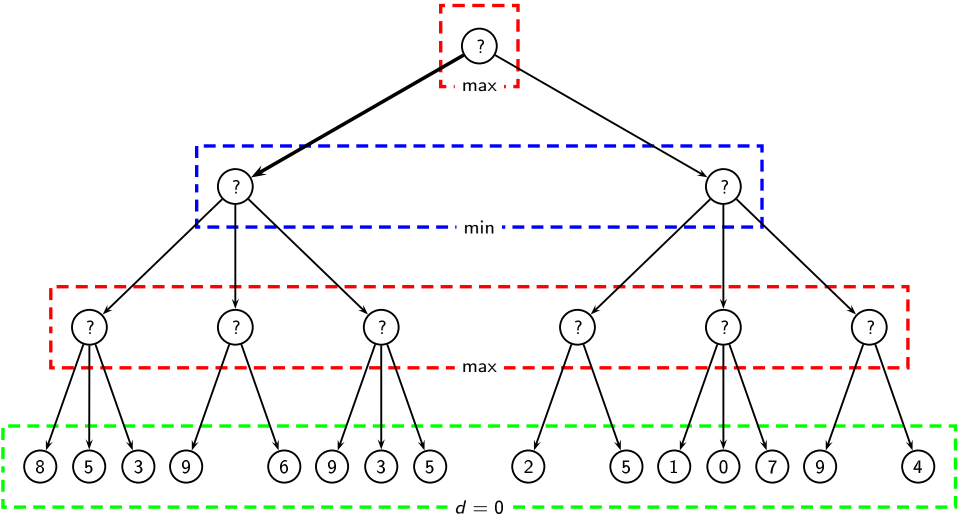
Algorithme alphabeta(s_i, α, β, d)

```
if d = 0 ∨ v(s_i) then
    return f(s_i)
else
    if s_i est un noeud Max then
        for all s_j ∈ {s : ∃a_k ∈ A, t(s_i, a_k)} do
            α ← max(α, alphabeta(s_j, α, β, d − 1))
            if α ≥ β then
                return α
            end if
        end for
        return α
    else
        for all s_j ∈ {s : ∃a_k ∈ A, t(s_i, a_k)} do
            β ← min(β, alphabeta(s_j, α, β, d − 1))
            if α ≥ β then
                return β
            end if
        end for
        return β
    end if
end if
```

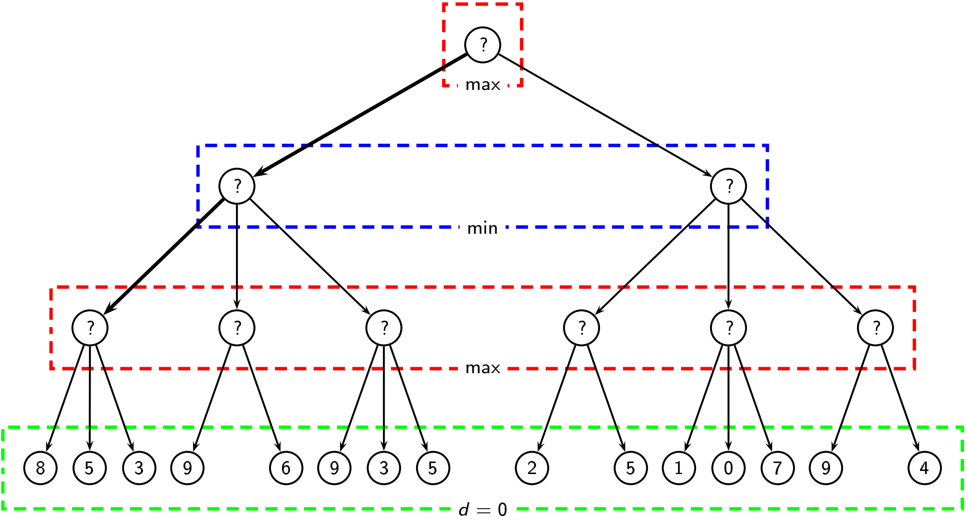
Exemple



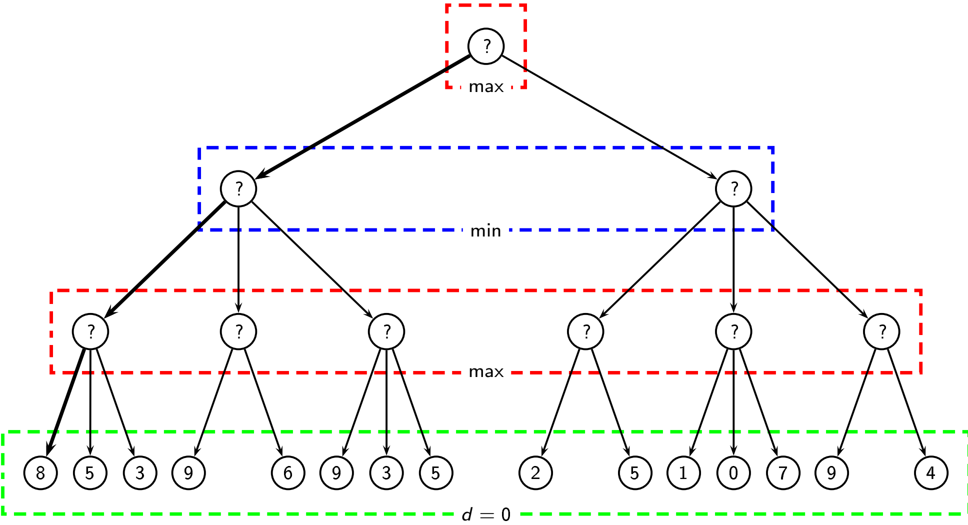
Exemple



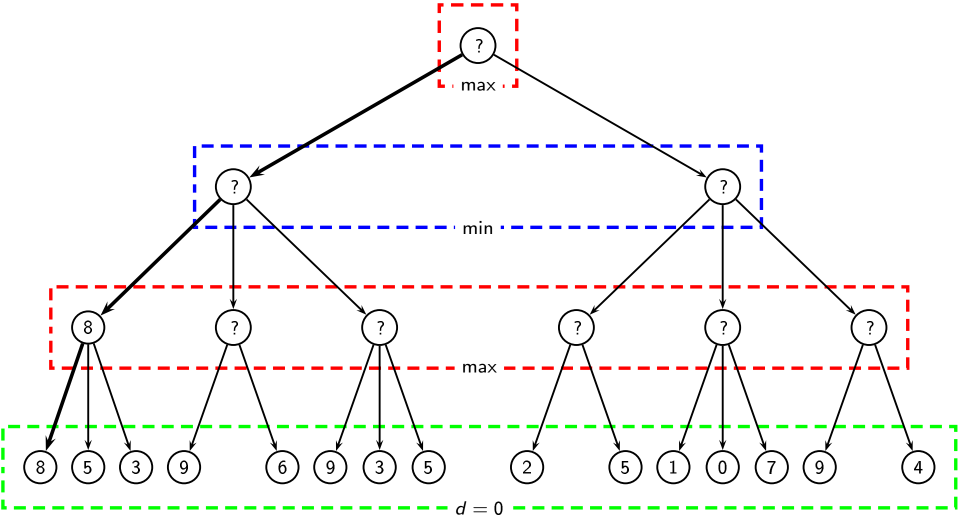
Exemple



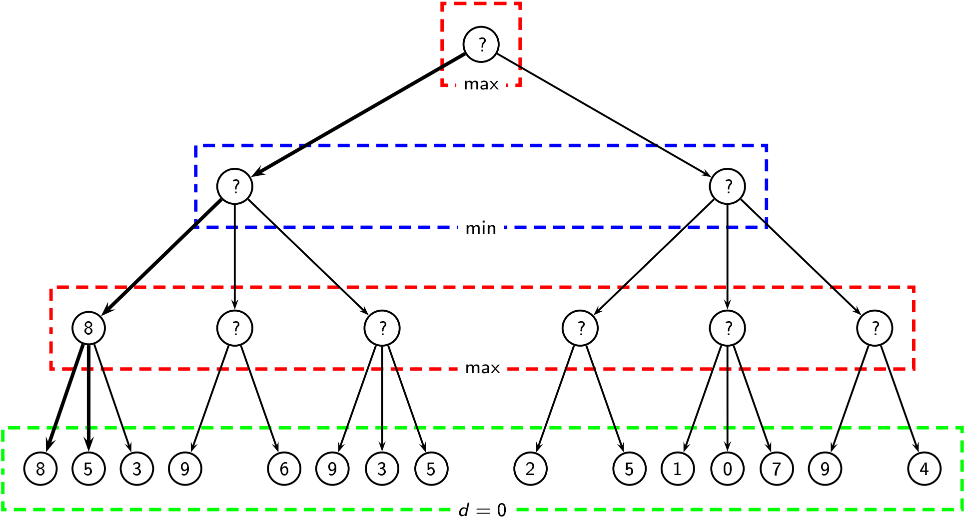
Exemple



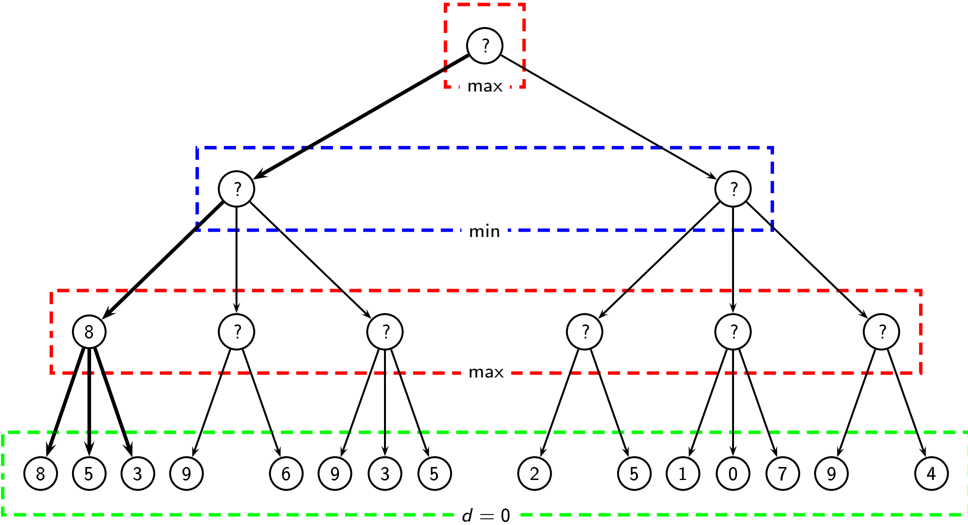
Exemple



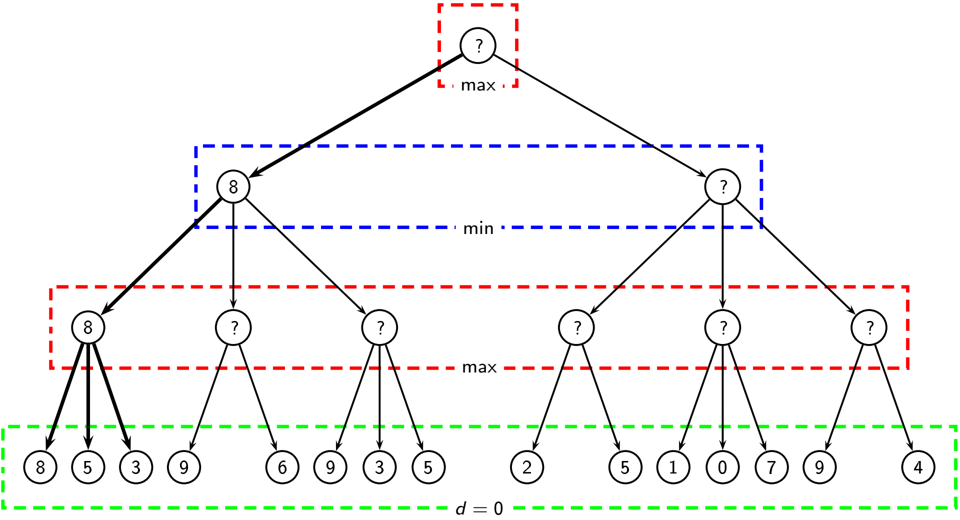
Exemple



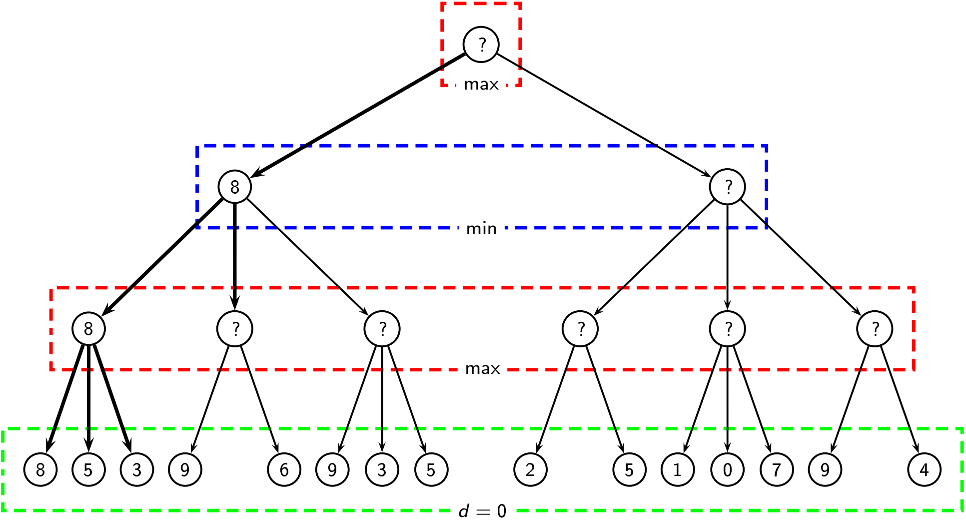
Exemple



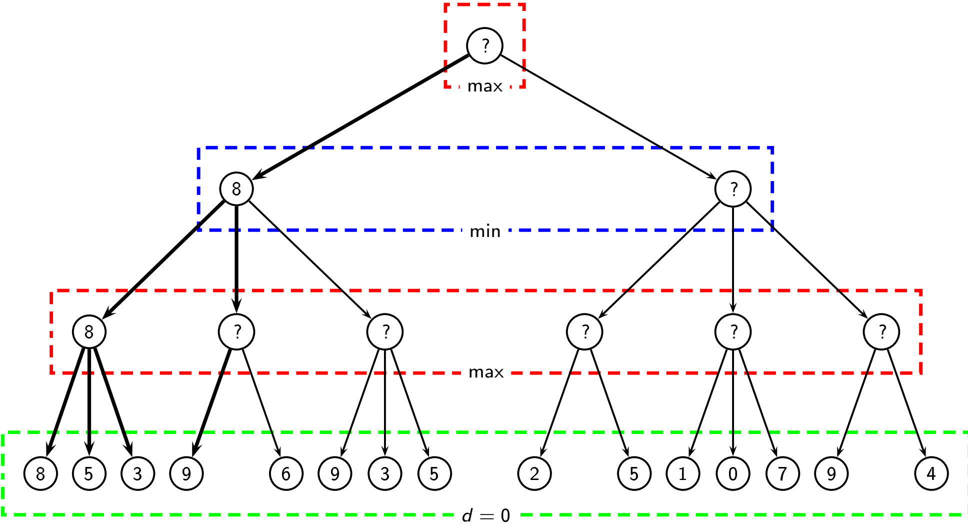
Exemple



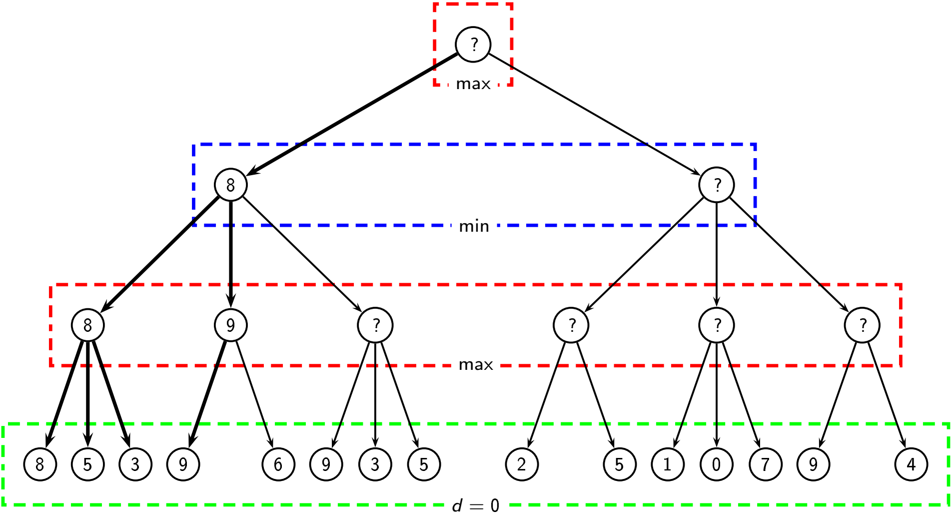
Exemple



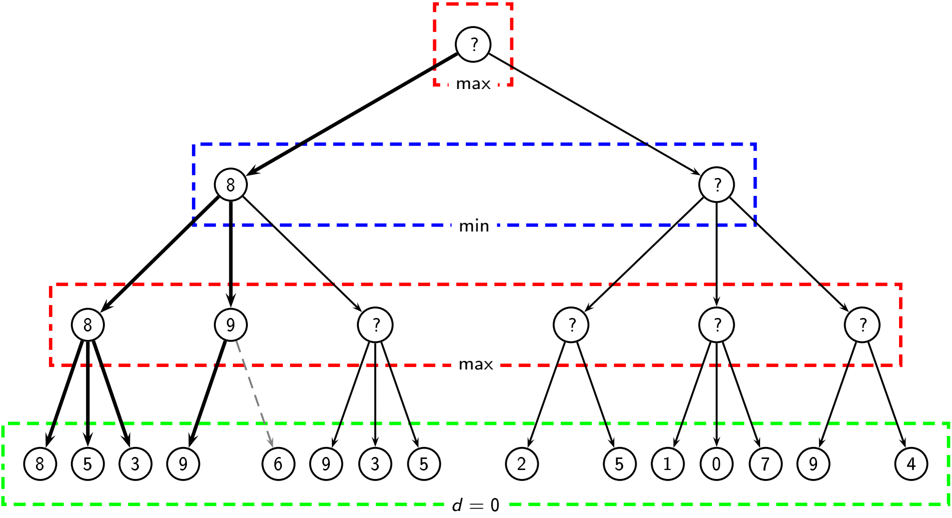
Exemple



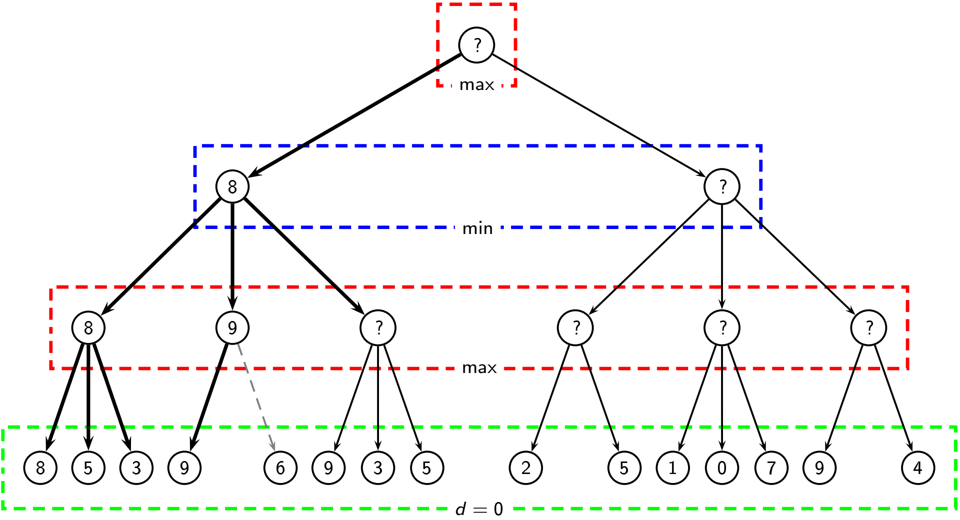
Exemple



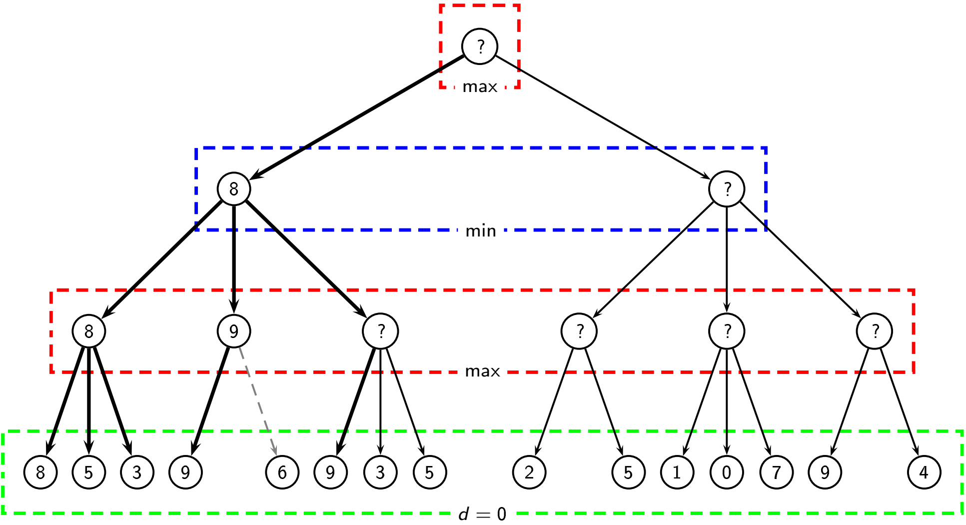
Exemple



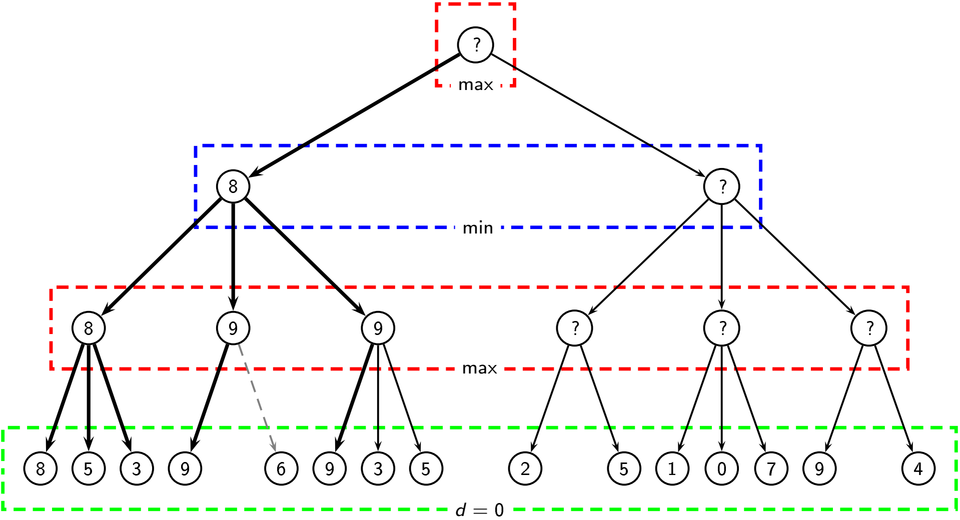
Exemple



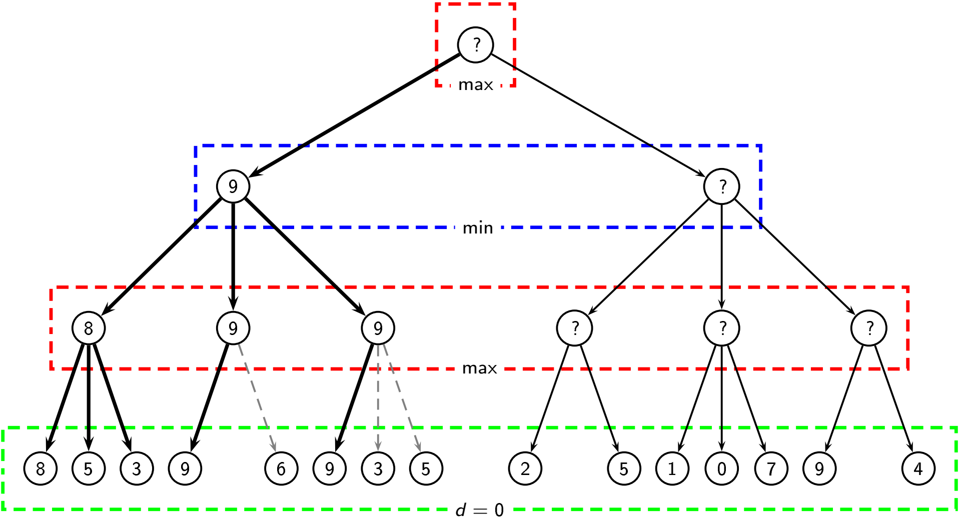
Exemple



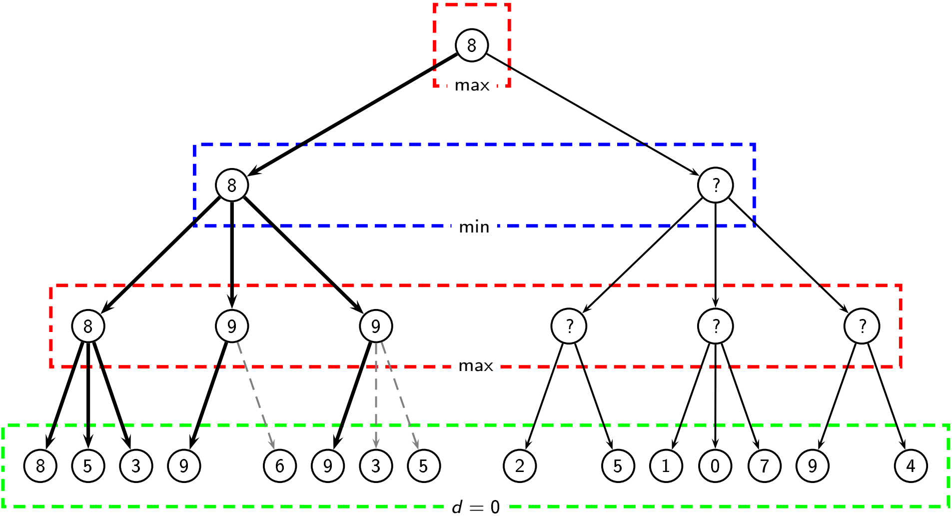
Exemple



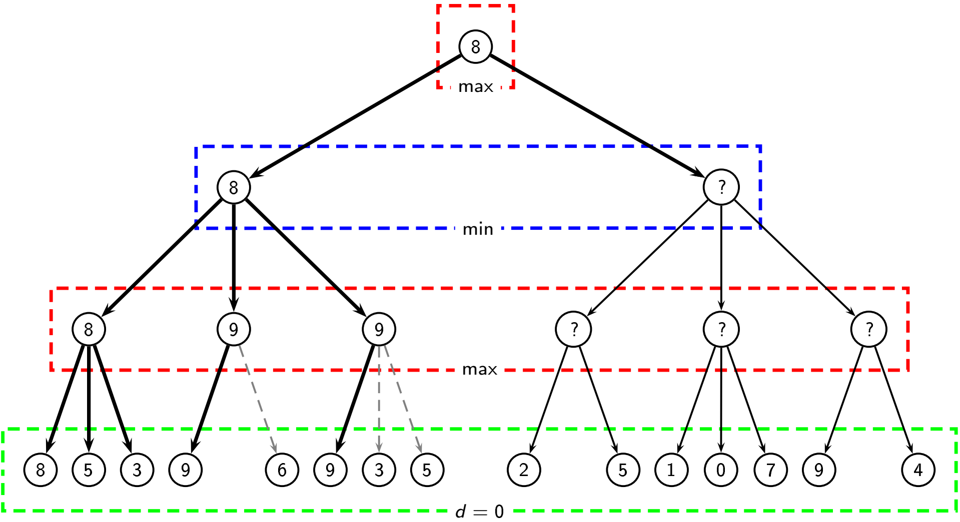
Exemple



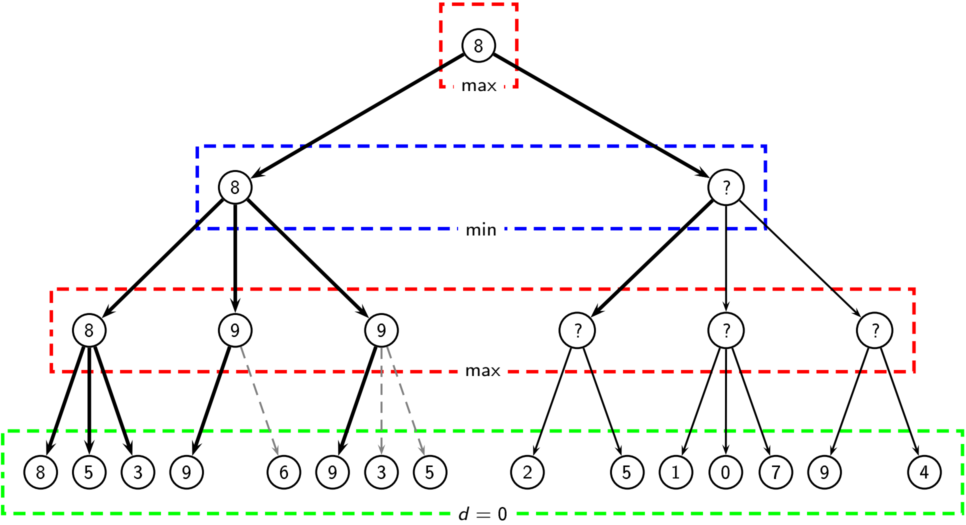
Exemple



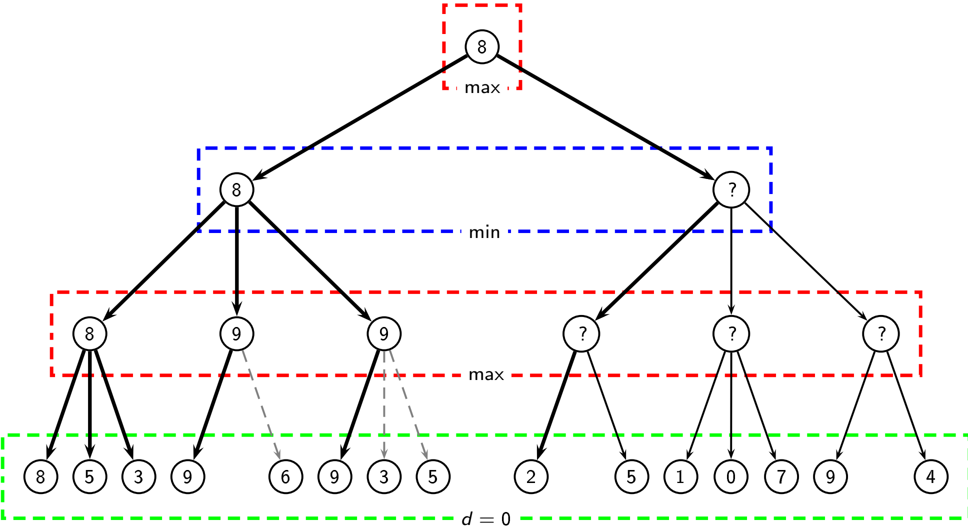
Exemple



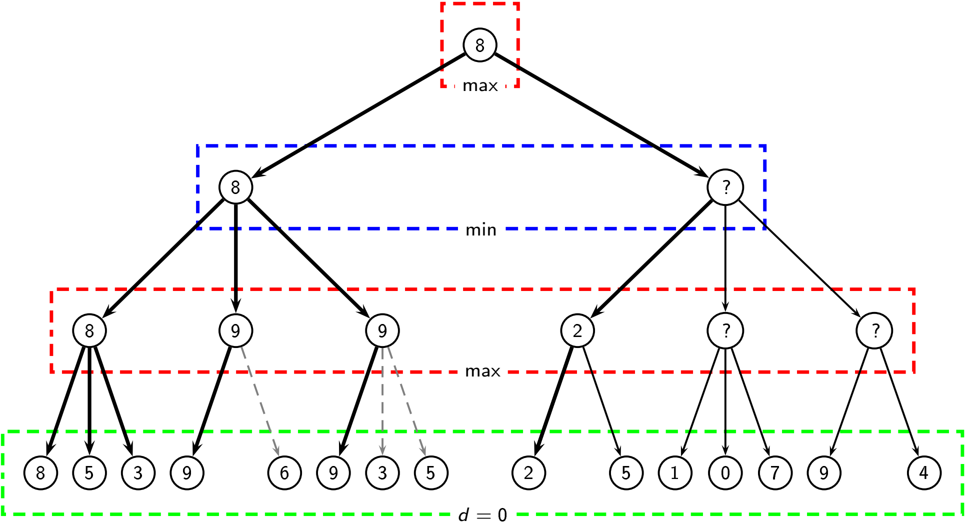
Exemple



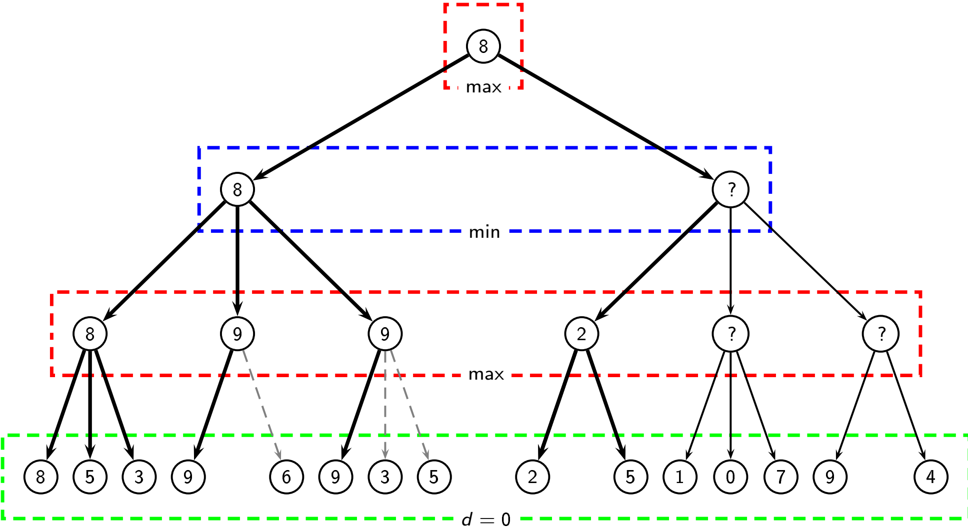
Exemple



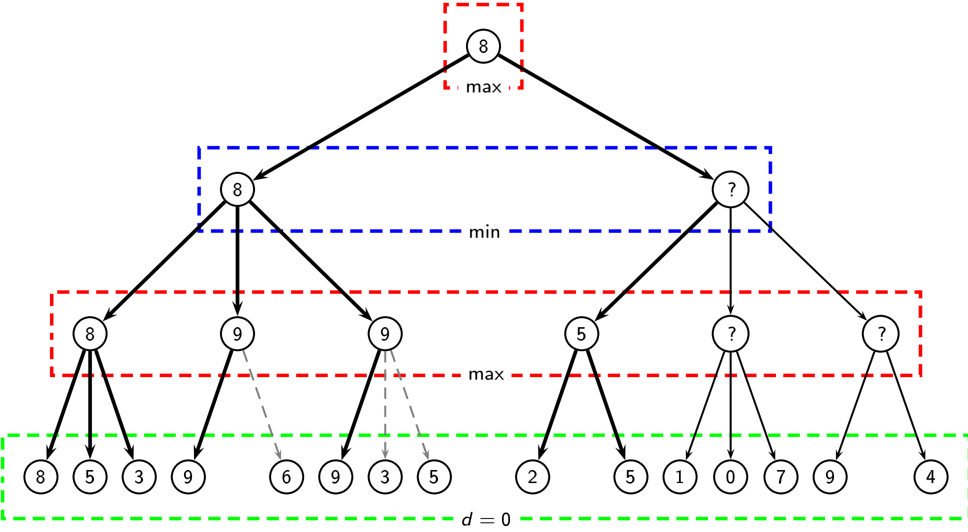
Exemple



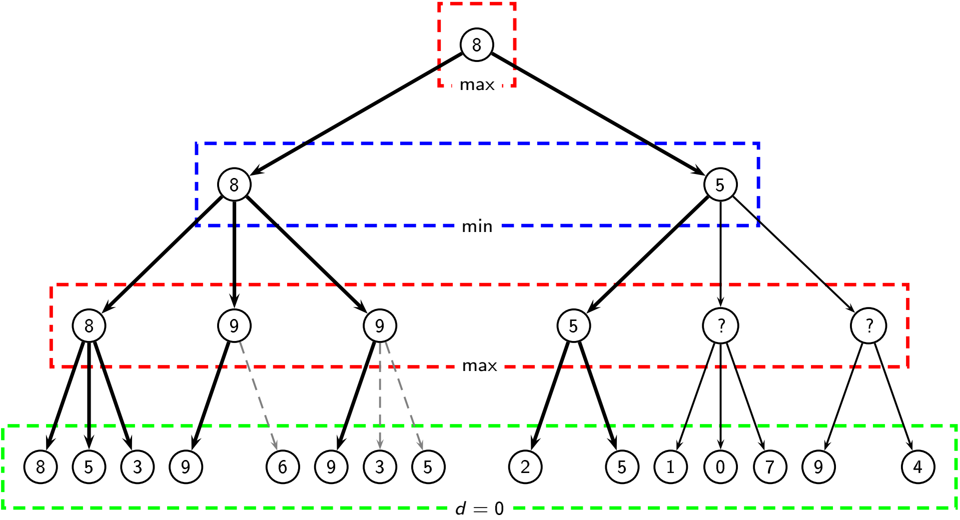
Exemple



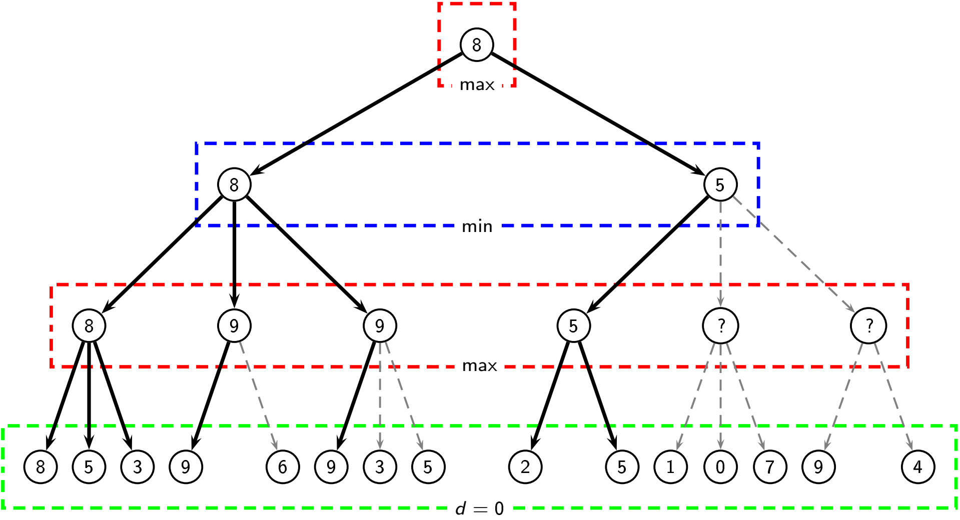
Exemple



Exemple



Exemple



Dans l'algorithme negamax

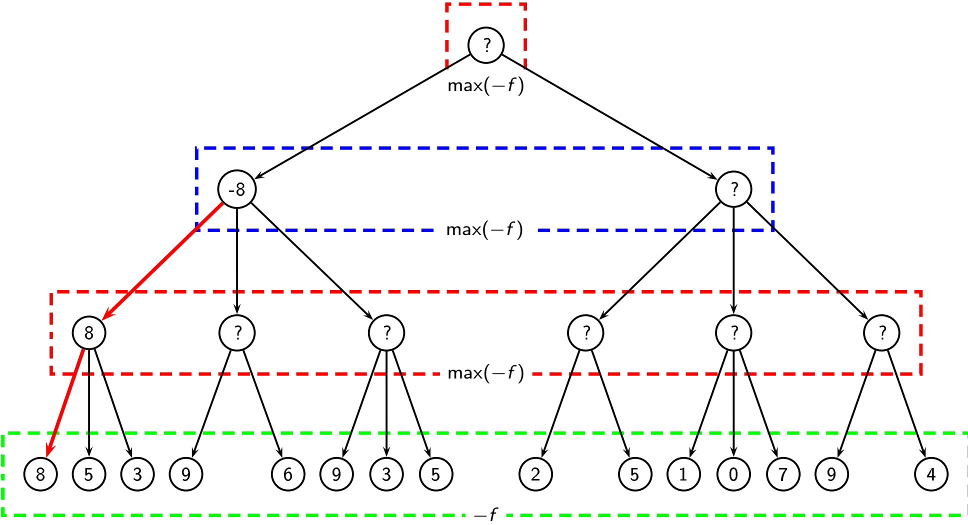
Initialisation

- $\alpha \leftarrow -\infty$
- $\beta \leftarrow +\infty$

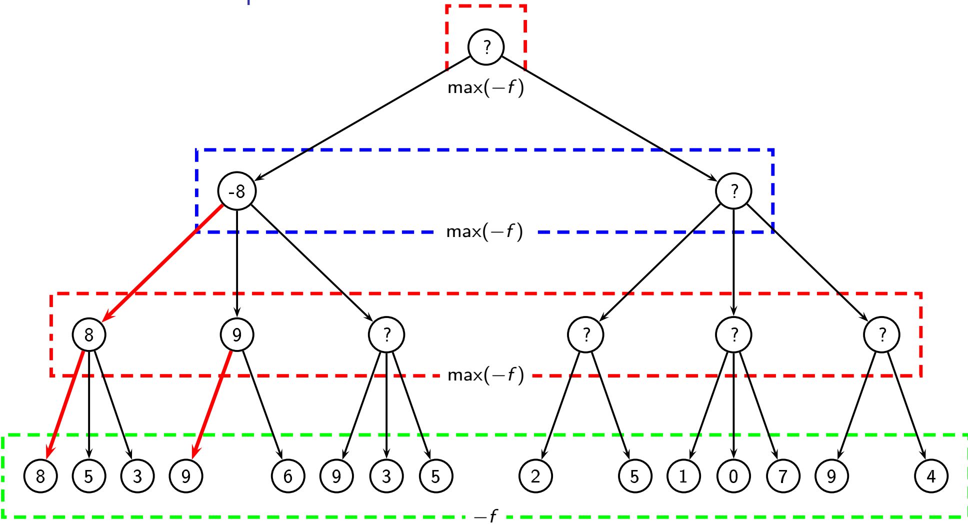
Algorithme $\mathrm{alphabeta}(s_i, \alpha, \beta, d)$

if $d = 0 \vee v(s_i)$ **then**
    return $g(s_i)$
else
    for all $s_j \in \{s : \exists a_k \in A, t(s_i, a_k)\}$ **do**
        $\alpha \leftarrow \max(\alpha, -\mathrm{alphabeta}(s_j, -\beta, -\alpha, d-1))$
        if $\alpha \geq \beta$ **then**
            return α
        end if
    end for
    return α
end if

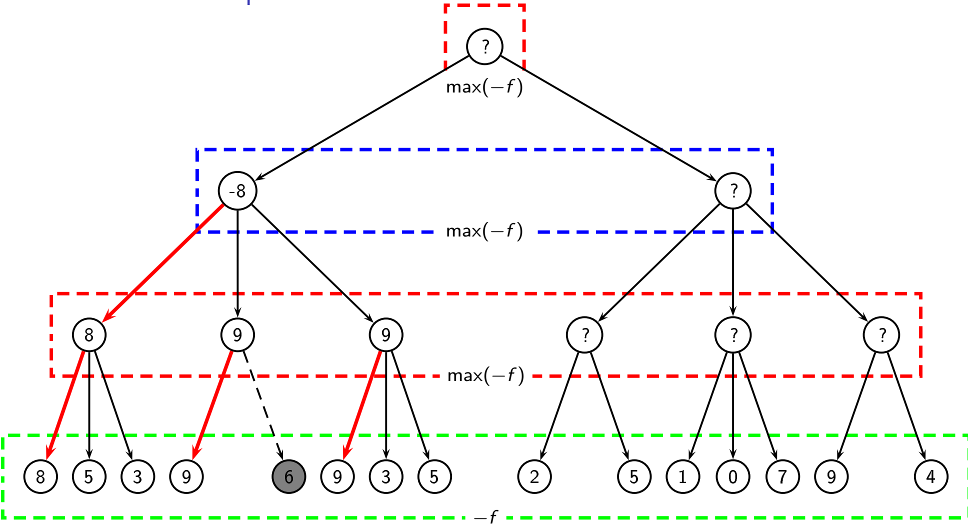
Sur le même exemple



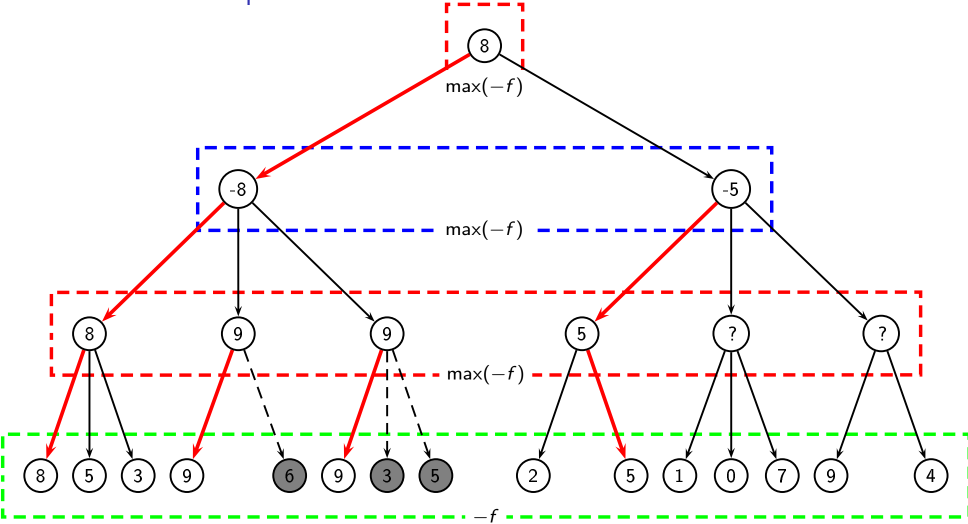
Sur le même exemple



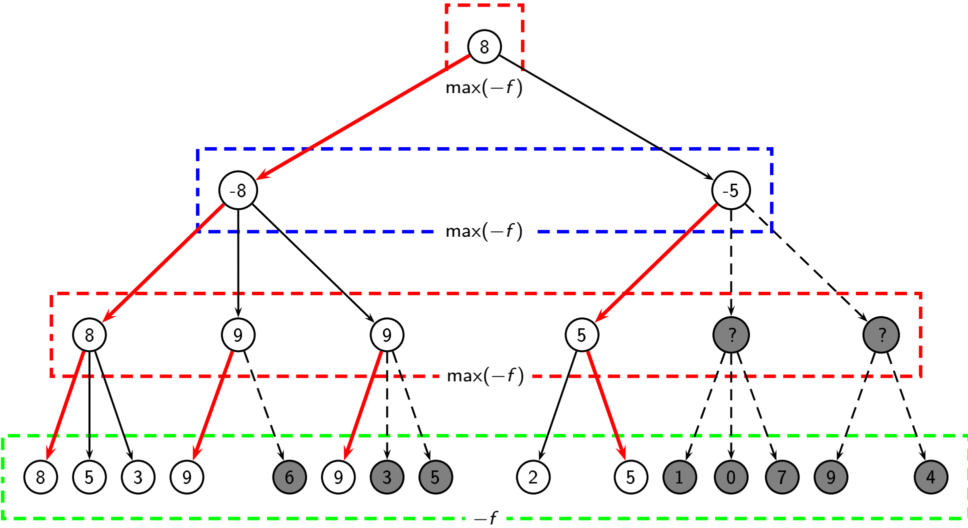
Sur le même exemple



Sur le même exemple



Sur le même exemple



Recherche aspirante

Intuition

Si des branches sont coupées lors d'une recherche alors il a y de fortes chances qu'elles soient coupées lors de la recherche suivante

Algorithme de recherche aspirante

$\alpha \leftarrow -\infty \wedge \beta \leftarrow +\infty$
while not fini **do**
    $(m, \alpha, \beta) \leftarrow \text{alphabeta}(s_i, \alpha - 1, \beta + 1, d))$
    if $m = \emptyset$ **then**
        $\alpha \leftarrow \alpha - \Delta \wedge \beta \leftarrow \beta + \Delta$
    else
        $\text{jouer}(m)$
    end if
end while

Propriété

Si la valeur finale est comprise entre les α et β initiaux alors la même valeur aurait été trouvée pour $-\infty$ et $+\infty$

Heuristique du coup meurtrier

Intuition

- `alphabeta` est efficace si les coups sont ordonnés du meilleur au moins bon
- mais un coup meurtrier ne l'est-il pas dans de nombreuses situations ?
- plus un coup est bien évalué, plus il a de chance de provoquer une coupe $\alpha\beta$
- dès qu'une coupe est identifiée, il faut l'essayer sur toutes les autres branches

Qu'est-ce qu'un coup meurtrier ?

Un coup meurtrier est un coup qui réduit la fenêtre $[\alpha, \beta]$. Pour une position donnée, il existe deux meilleurs coups : celui du joueur max et celui du joueur min.

Dans n'importe quel algorithme fondé sur le `minimax`

Require: meilleur-coup est le premier élément de $\{s : \exists a_k \in A, t(s_i, a_k)\}$ si possible

1: **if** $\alpha \geq \beta$ **then**
2: meilleur-coup $\leftarrow s_i$
3: ...
4: **end if**

Généralisation de l'heuristique

Idée

Pourquoi ne considérer qu'une unique coup meurtrier ?

Historique

Require: $\{s : \exists a_k \in A, t(s_i, a_k)\}$ est ordonné selon $\mathrm{score}(s_k)$

if $\alpha \geq \beta$ **then**
  $\mathrm{score}(s_i) \leftarrow \mathrm{score}(s_i) + d^2$
  $\ldots$
end if

Historique relatif

Require: $\{s : \exists a_k \in A, t(s_i, a_k)\}$ est ordonné selon $\frac{\text{hh-score}(s_k)}{\text{bf-score}(s_k)}$

if $\alpha \geq \beta$ **then**
  $\mathrm{score}(s_i) \leftarrow \text{hh-score}(s_i) + d^2$
  $\ldots$
else
  $\mathrm{score}(s_i) \leftarrow \text{bf-score}(s_i) + d^2$
end if

Effet d'horizon

« Un état peut avoir une valeur élevée mais rien ne nous dit que les états successeurs ne sont pas très mauvais »

Recherche de quiescence

Qu'est-ce que la quiescence ?

Une fois atteint la profondeur maximale d'évaluation, l'algorithme poursuit le calcul en continuant à examiner tous les successeurs des états produisant un important changement de valeur.

Recherche de quiescence $\text{quiescence-search}(s_i, s_j, d)$

```
if (d ≤ 0 ∧ quiescent(s_i, s_j)) ∨ v(s_i) then
    return f(s_i)
else
    m ← −∞
    for all s_j ∈ {s : ∃a_k ∈ A, t(s_i, a_k)} do
        m ← évaluation minimax ou alphabeta
        quiescence-search(s_j, s_i, d − 1)
    end for
    return m
end if
```

Test de quiescence $\text{quiescent}(s_i, s_j)$

$f(s_i) > f(s_j) + \Delta \vee f(s_i) < f(s_j) - \Delta$.

Approfondissement itératif

Supposons qu'un ordinateur dispose d'un temps t pour jouer

- estimer la profondeur que l'on peut atteindre en un temps t ? (*très dur*)
- utiliser l'approfondissement itératif (Depth First Iterative Deepening)

Algorithme $\mathrm{dfid}(s_i, t)$

$d \leftarrow 1$
while $t > 0$ **do**
    $m \leftarrow \mathrm{alphabeta}(s_i, -\infty, +\infty, d))$
    $\mathrm{update}(t)$
    $d \leftarrow d + 1$
end while
return m

Propriété

- la dernière itération parcourt n^d nœuds où n est le facteur de branchement
- donc DFID est $\mathcal{O}(n^d + 2n^{d-1} + 3n^{d-2} + \ldots + dn)$
- si d est grand, les termes en $(d - i)$ sont négligeables
- **bonus** : trier les noeuds de profondeurs $d - 1$ pour améliorer la recherche en d

SEX (Search EXtension)

Classiquement

La profondeur de recherche est fixée à d et chaque descente d'un niveau dans l'arbre la diminue de 1. Lorsque $d = 0$, la recherche est terminée.

Dans n'importe quel algorithme fondé sur le `minimax`

- on définit $S_X \approx 10 \times d$
- d est remplacé par S_X
- S_X est diminuée à chaque appel récursif en fonction de l'intéret du coup joué
- la recherche se termine lorsque $S_X \leq 0$

Heuristique : intéret d'un coup

- coup « quelconque » ≈ -10
- prise ou mise en échec ≈ -2 ou -3
- retraite ≈ -20 à -40

Conclusion

L'intelligence se fonde sur l'heuristique

Donald Michie (1927 – 2007)

La recherche sur le jeu d'échecs est le champ le plus important de la recherche cognitive. Les échecs seront pour nous ce que la drosophile a été pour les généticiens : un moyen simple et pratique de développer de nouvelles techniques.

Principe fondamental

- construire itérativement un arbre de jeu
- le prochain coup est celui qui mène aux coups les plus prometteurs
- éviter d'avoir à évaluer tous les nœuds de l'arbre

Cas des échecs : un travail de moine copiste

Jonathan Schaeffer (1957 – ?)

Malheureusement, la plupart des travaux sur les échecs relèvent de l'ingénierie et non pas de la science. [...] À mon avis, les méthodes informatiques conventionnelles pour les échecs n'ont plus d'intérêt pour la recherche en intelligence artificielle.

« Tuning » de fonctions d'évaluation

- doit être rapide à calculer
- doit produire une stratégie (globale) à partir de la tactique (locale)

Développement de processeurs dédiés

- fonctions implantées en « dur » sur des processeurs VLSI
- CHEOPS (Chess-Oriented Processing System)

Construction de bibliothèques de coups

- les ouvertures et leurs probabilités de succès
- les finales élémentaires ou non
- tous les coups des grands maîtres

Bibliographie

Nils J. Nilsonn.
The Quest for Artificial Intelligence.
Cambridge University Press , 2009.

Tristan Cazenave.
Intelligence artificielle : une approche ludique.
Ellipses, 2011.

Tristan Cazenave.
Intelligence artificielle et jeux.
Hermès Sciences Lavoisier, 2006.