

Chess Computing 2017

Club ESI-Mat

Algorithme Min-Max avec élagage α/β

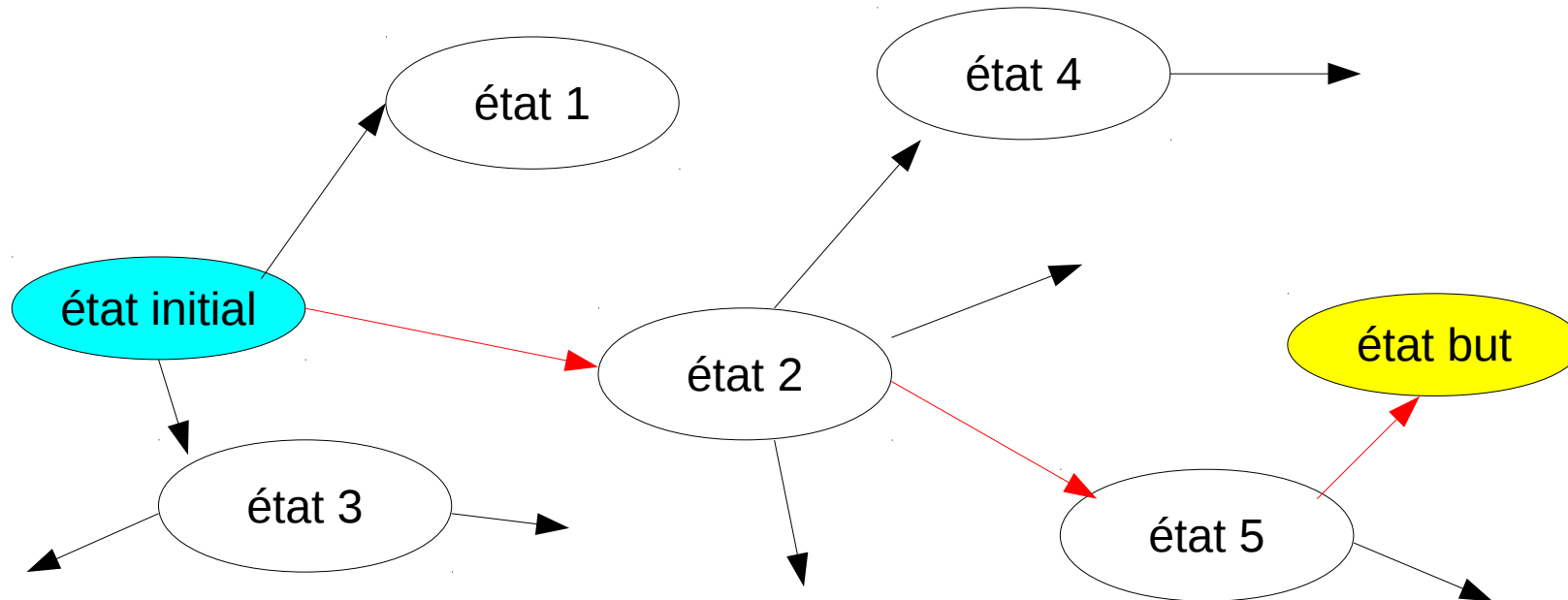
- Application au jeu d'échecs -

Plan

- Résolution de problèmes (par Backtracking)
- Problèmes de jeux (Min-Max avec coupes α/β)
- Présentation d'une petite application au jeu d'échecs

Résolution de Problèmes

« Trouver un chemin entre un état initial et un état but »



Parmi les moyens de résolution :

→ Exploration d'un **espace de recherche** à l'aide d'une recherche en profondeur (ou **Backtracking**)

Algorithme de recherche en profondeur (BackTracking)

(sur un espace de recherche connexe)

Version itérative

RP_iterative(v : état)

Init_Pile(P);

Empiler(P , v);

TQ (Non Pilevide(P) && v ≠ but)

Depiler(P, v);

Si (v n'est pas encore visité)

Visiter(v);

Pour chaque état w adjacent à v :

Si (w n'est pas encore visité)

Empiler(P, w)

Fsi

Fp

Fsi

FTQ

Si (v = but) retourner Vrai

Sinon retourner Faux

Fsi

Version récursive

RP(v : état)

Si (v = but)

Retourner Vrai

Sinon

Visiter(v);

Pour chaque w adjacent à v :

Si (w non encore visité)

trouv ← **RP**(w) ;

Si (trouv) retourner Vrai Fsi

Fsi

Fp

retourner Faux

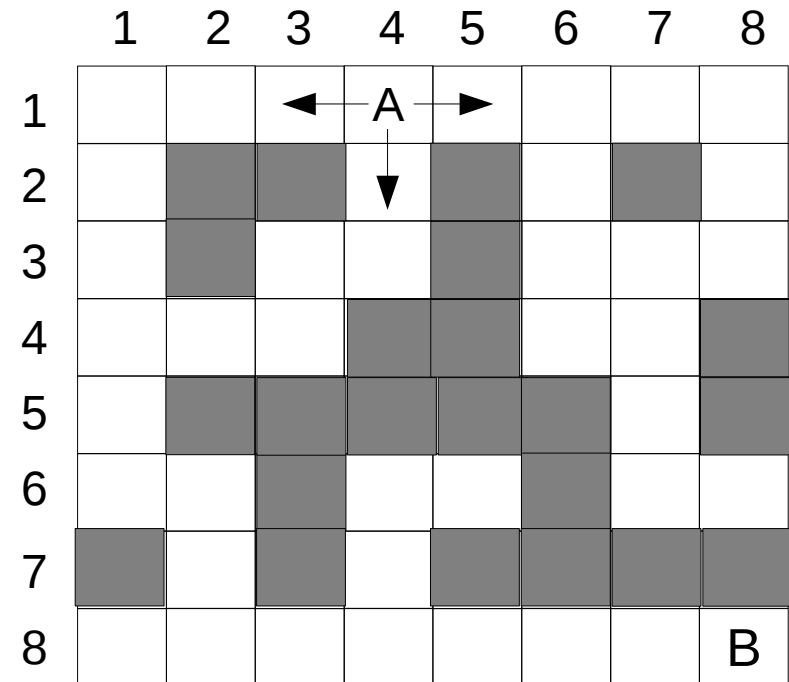
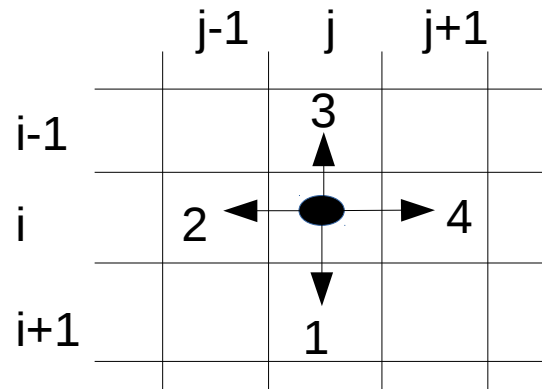
Fsi

Exemple : Problème du labyrinthe

Un état peut être représenté par les coordonnées de la position courante :

Etat initial = (1,4) / Etat but = (8,8)

Les déplacements possibles



à partir de **(i,j)** on essaye les alternatives dans cet ordre : **(i+1,j)** , **(i, j-1)** , **(i-1,j)** , **(i,j+1)**
Les cases fermées, déjà visitées ou à l'extérieur de la grille sont retirées de la liste

Algorithme du Labyrinthe (Backtracking)

```
(x,y) ← (1,4);    // l'état_initial
CasesFermées ← { (2,2), (2,3), (2,5), (2,7), (3,2), ... (7,8) };
CasesVisitées ← { };
Init_Pile( P );
Empiler( P, (x,y) );
TQ ( Non PileVide(P) et (x,y) ≠ (8,8) )
    Dépiler( P, (x,y) );
    CasesVisitées ← CasesVisitées ∪ { (x,y) };
    (a1,b1) ← (x+1, y);
    (a2,b2) ← (x, y-1);
    (a3,b3) ← (x-1, y);
    (a4,b4) ← (x, y+1);
    Pour i = 4 .. 1, par_pas_de : -1
        Si ( (ai,bi) ∉ (CasesFermées ∪ CasesVisitées) &&
            (ai ∈ [1,8]) && (bi ∈ [1,8]) )
            Empiler( P, (ai,bi) )
        Fsi
    FP
FTQ
```

Version récursive

Var globales : CasesV \leftarrow {} ; CasesF \leftarrow { (2,2), (2,3), (2,5), (2,7), ... (7,8) } ;

Labyrinthe ((x,y))

Si ((x,y) = (8,8)) succès *// On a trouvé la sortie*

Sinon

CasesV \leftarrow CasesV \cup { (x,y) } ;

(a₁,b₁) \leftarrow (x+1, y) ;

(a₂,b₂) \leftarrow (x, y-1) ;

(a₃,b₃) \leftarrow (x-1, y) ;

(a₄,b₄) \leftarrow (x, y+1) ;

Pour i = 1 .. 4

Si ((ai,bi) \notin (CasesF \cup CasesV) && (ai \in [1,8]) && (bi \in [1,8]))

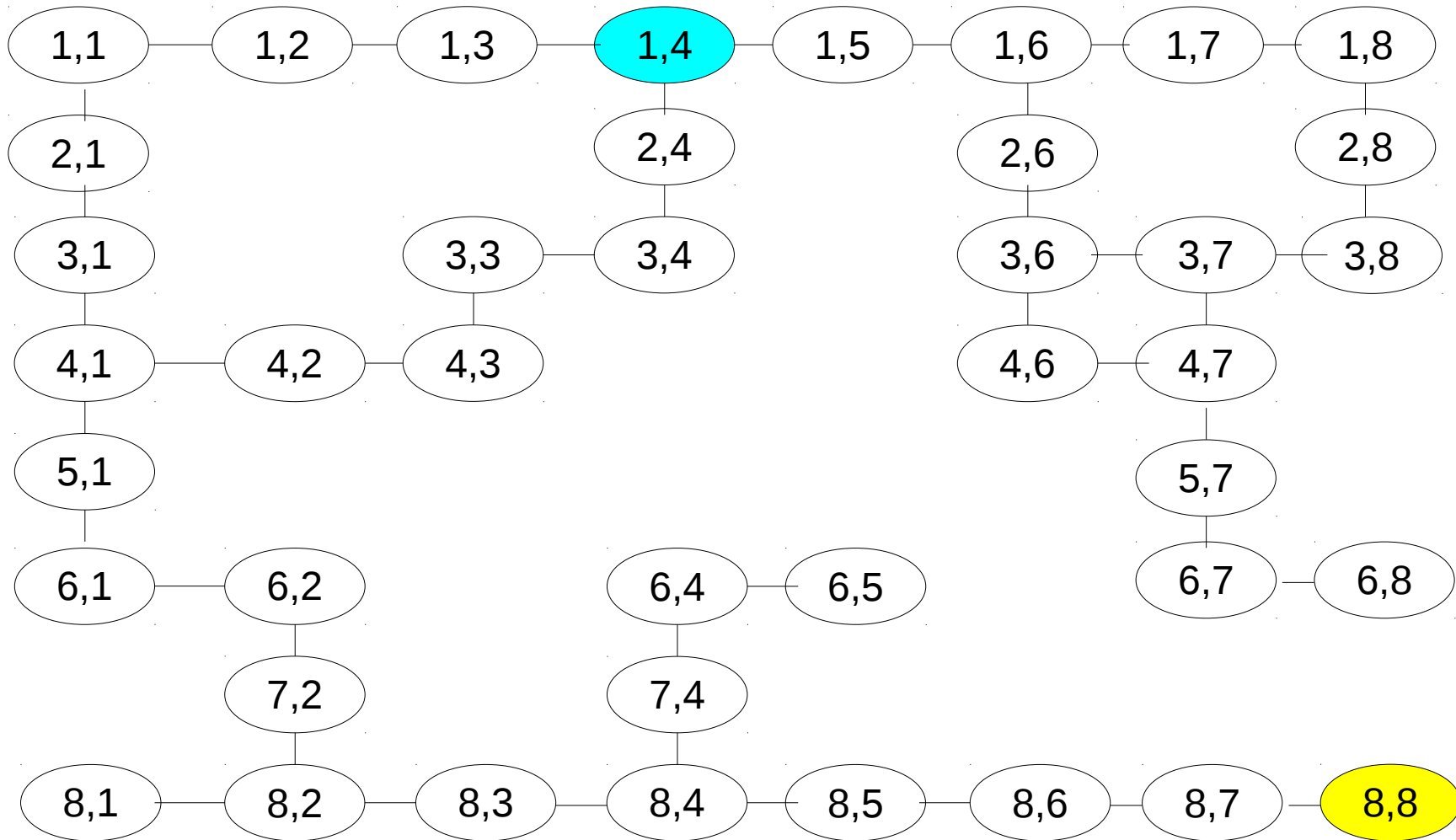
Labyrinthe((ai,bi))

Fsi

FP

Fsi

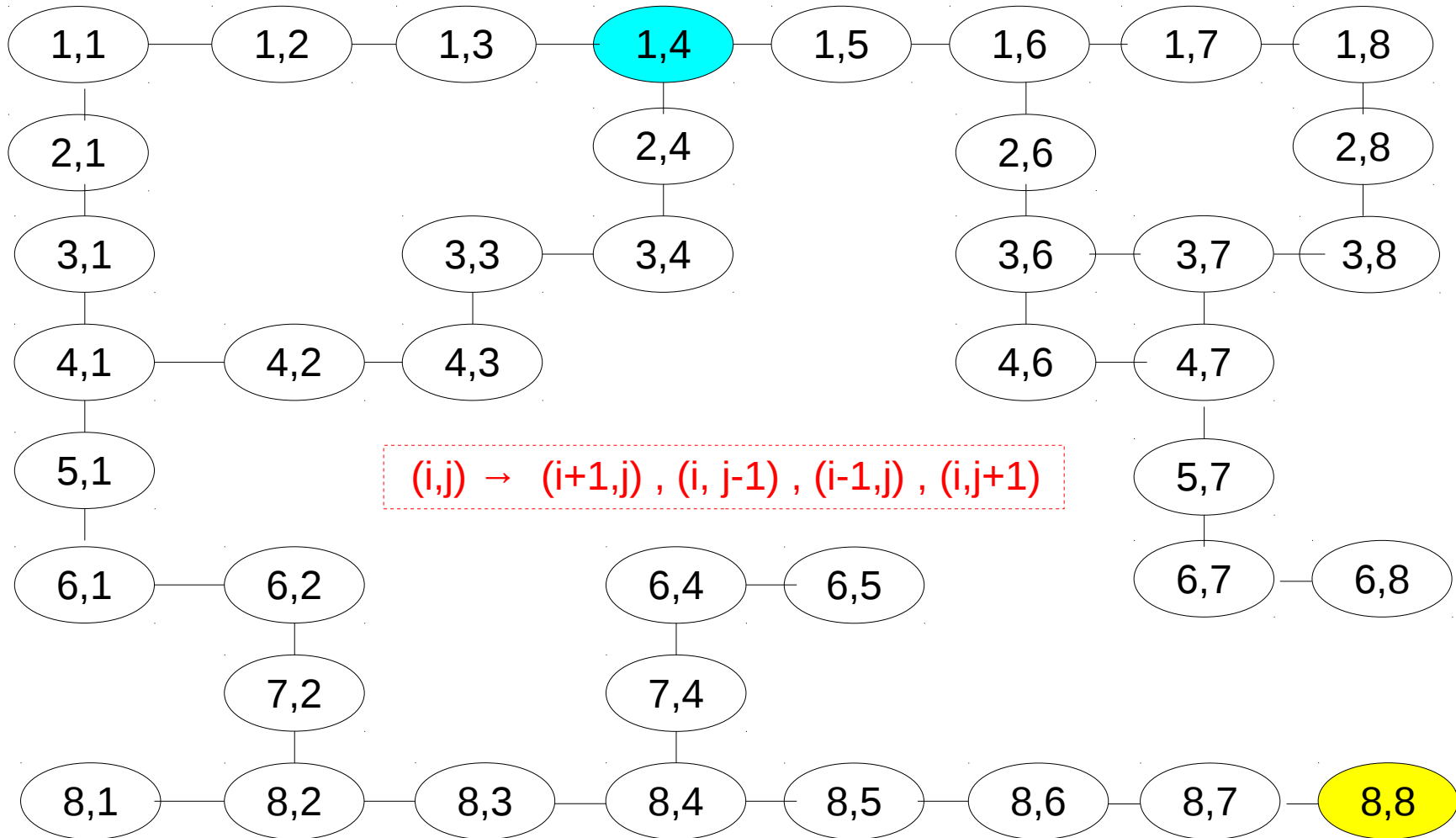
L'espace de recherche associé



à partir de (i,j) on essaye les alternatives dans cet ordre : $(i+1,j)$, $(i, j-1)$, $(i-1,j)$, $(i,j+1)$

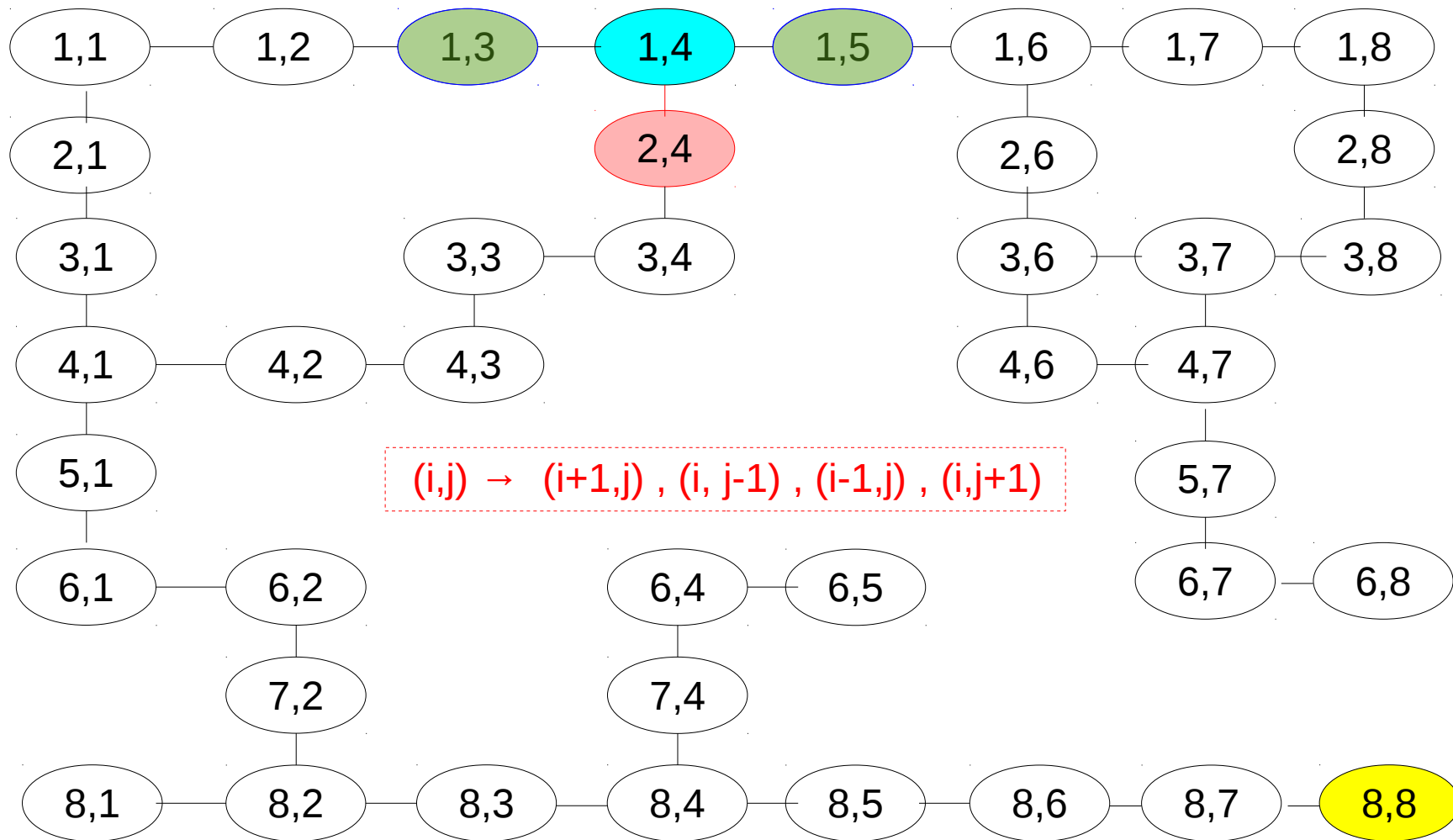
état courant : $(1,4)$ / états successeurs : $(2,4)$, $(1,3)$, , $(1,5)$

état courant : **(1,4)** / états successeurs : **(2,4)** , **(1,3)** , **(1,5)**



La pile

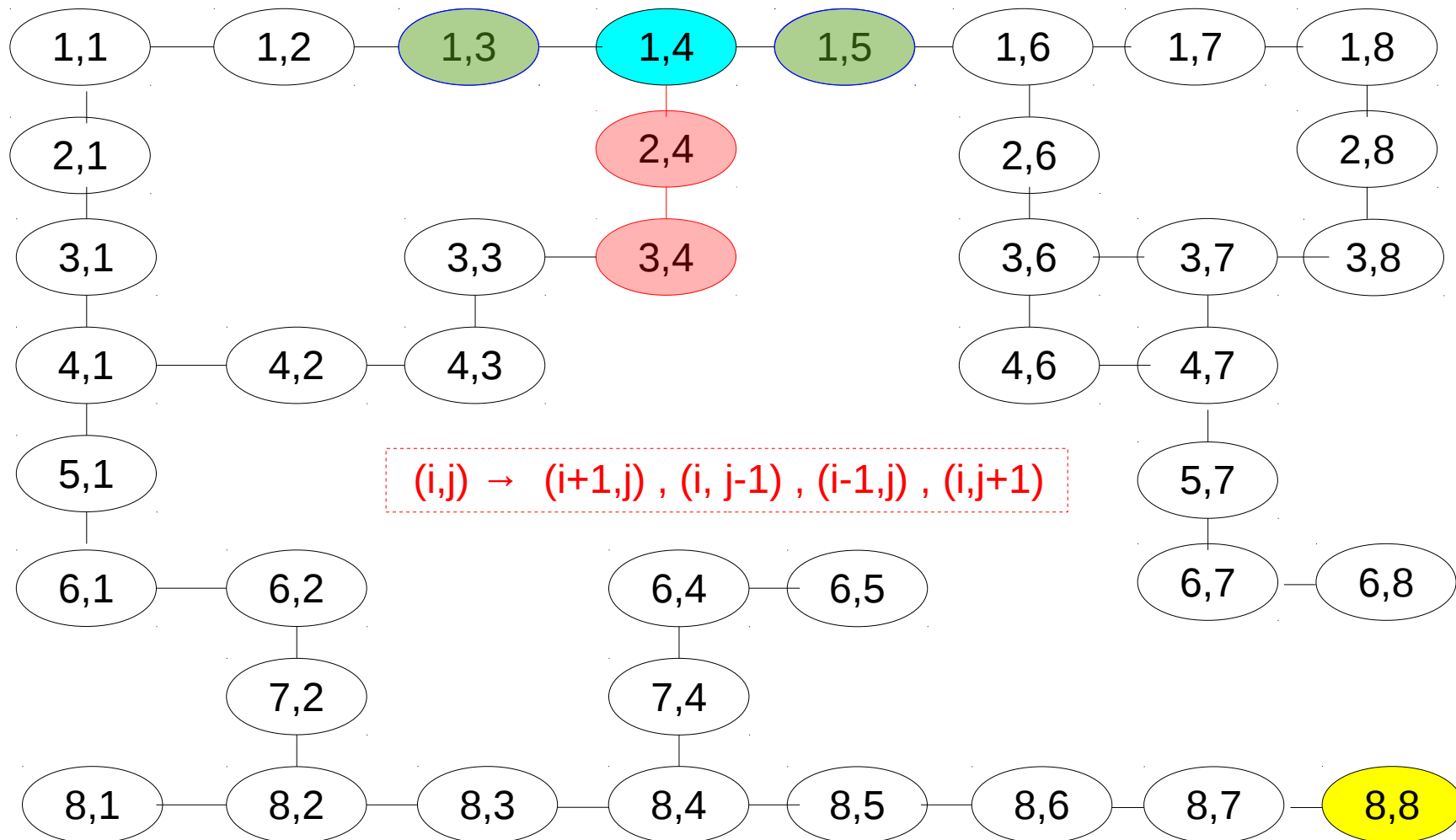
état courant : **(2,4)** / états successeurs : **(3,4)**



La pile

(1,5) | (1,3) |

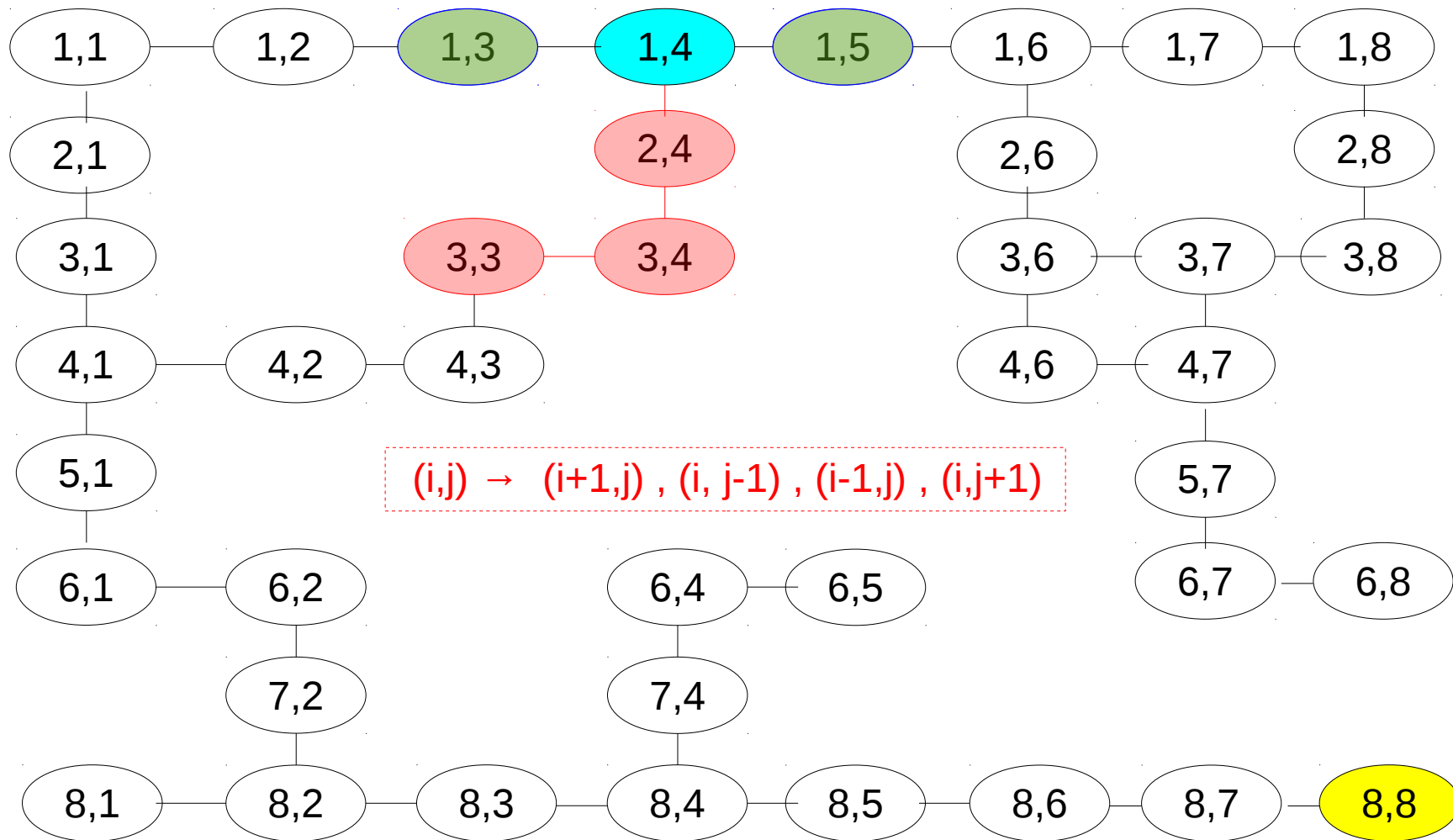
état courant : **(3,4)** / états successeurs : **(3,3)**



La pile

(1,5) | (1,3) |

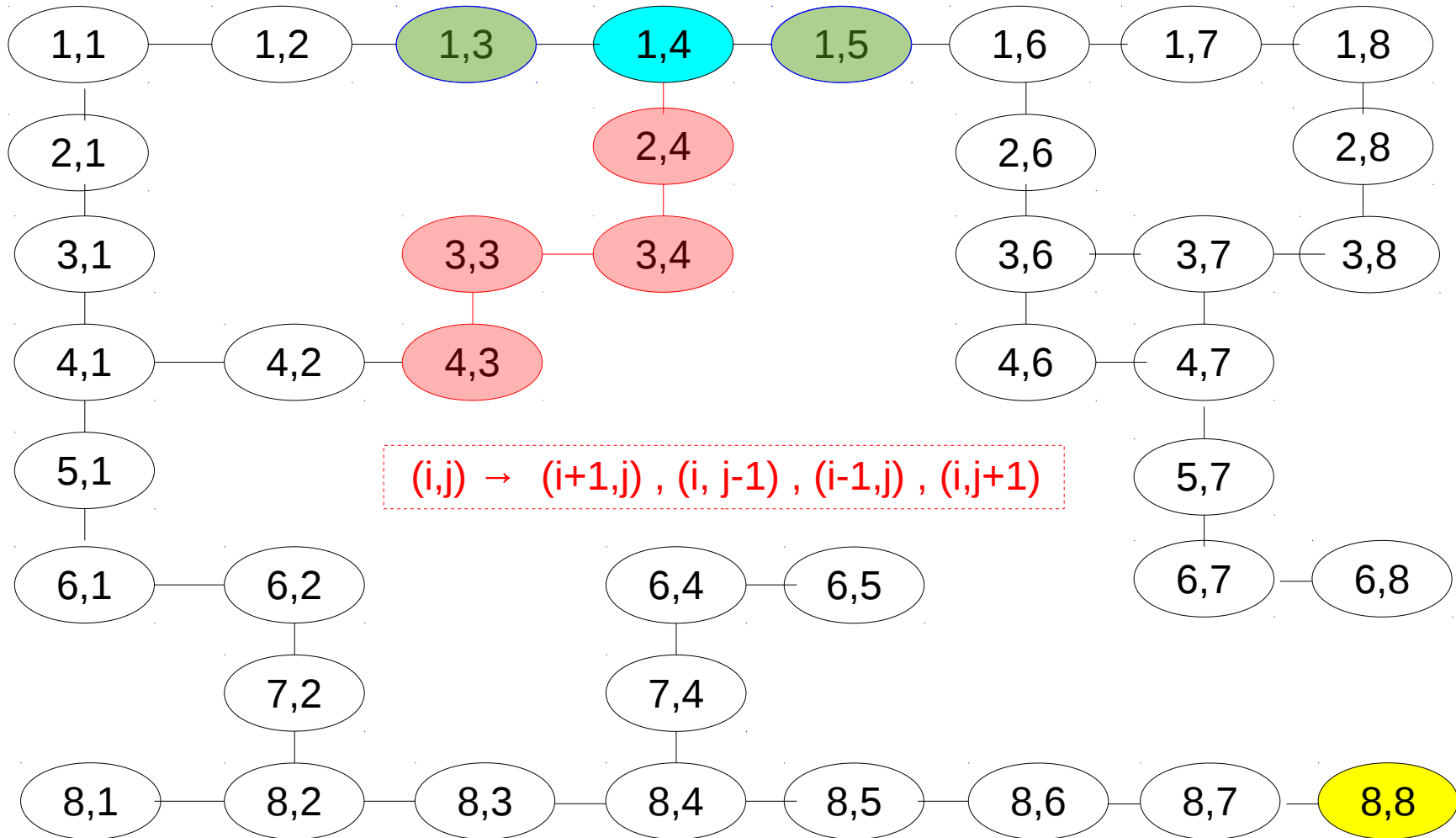
état courant : **(3,3)** / états successeurs : **(4,3)**



La pile

(1,5) | (1,3) |

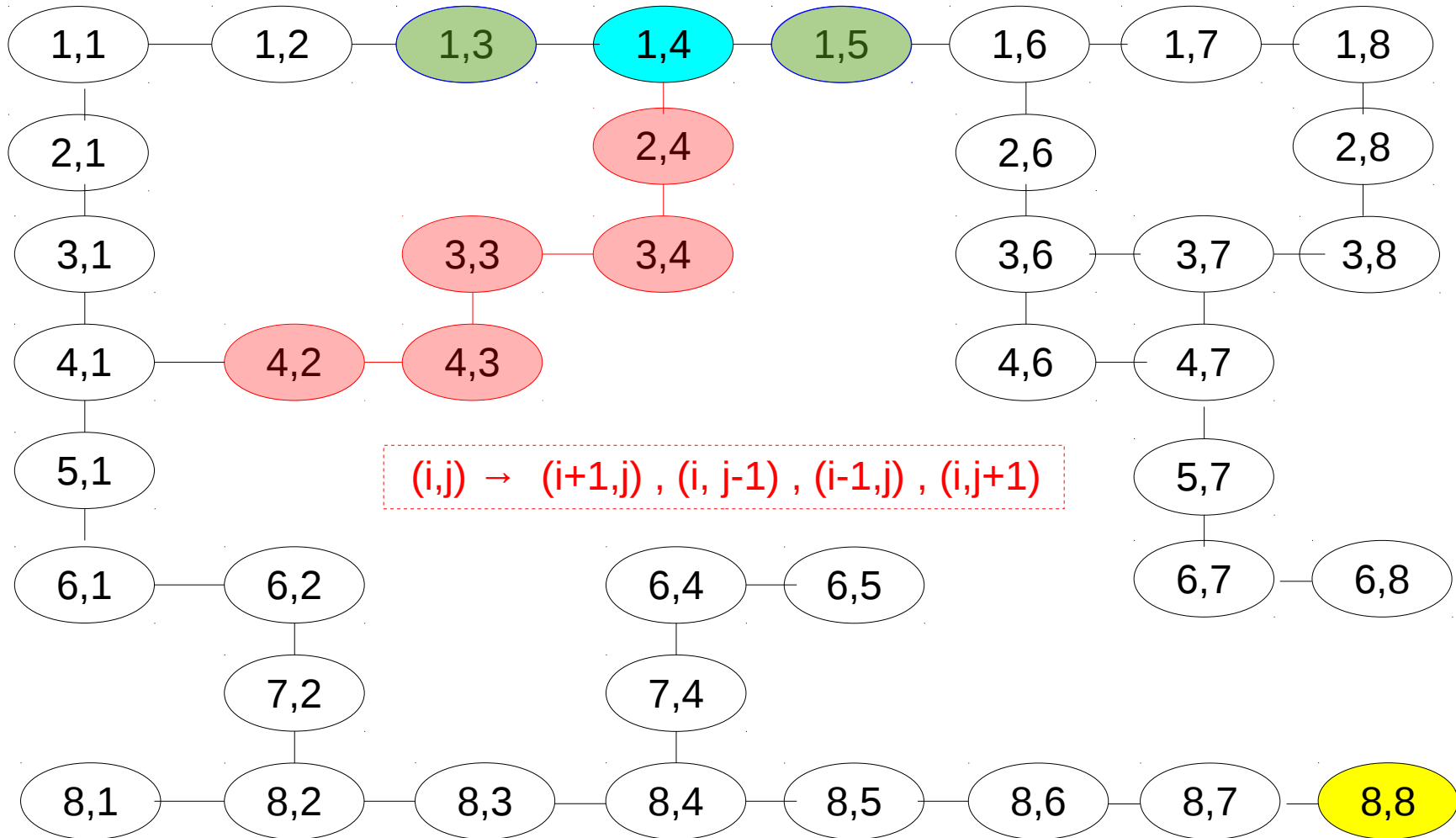
état courant : **(4,3)** / états successeurs : **(4,2)**



La pile

(1,5) | (1,3) |

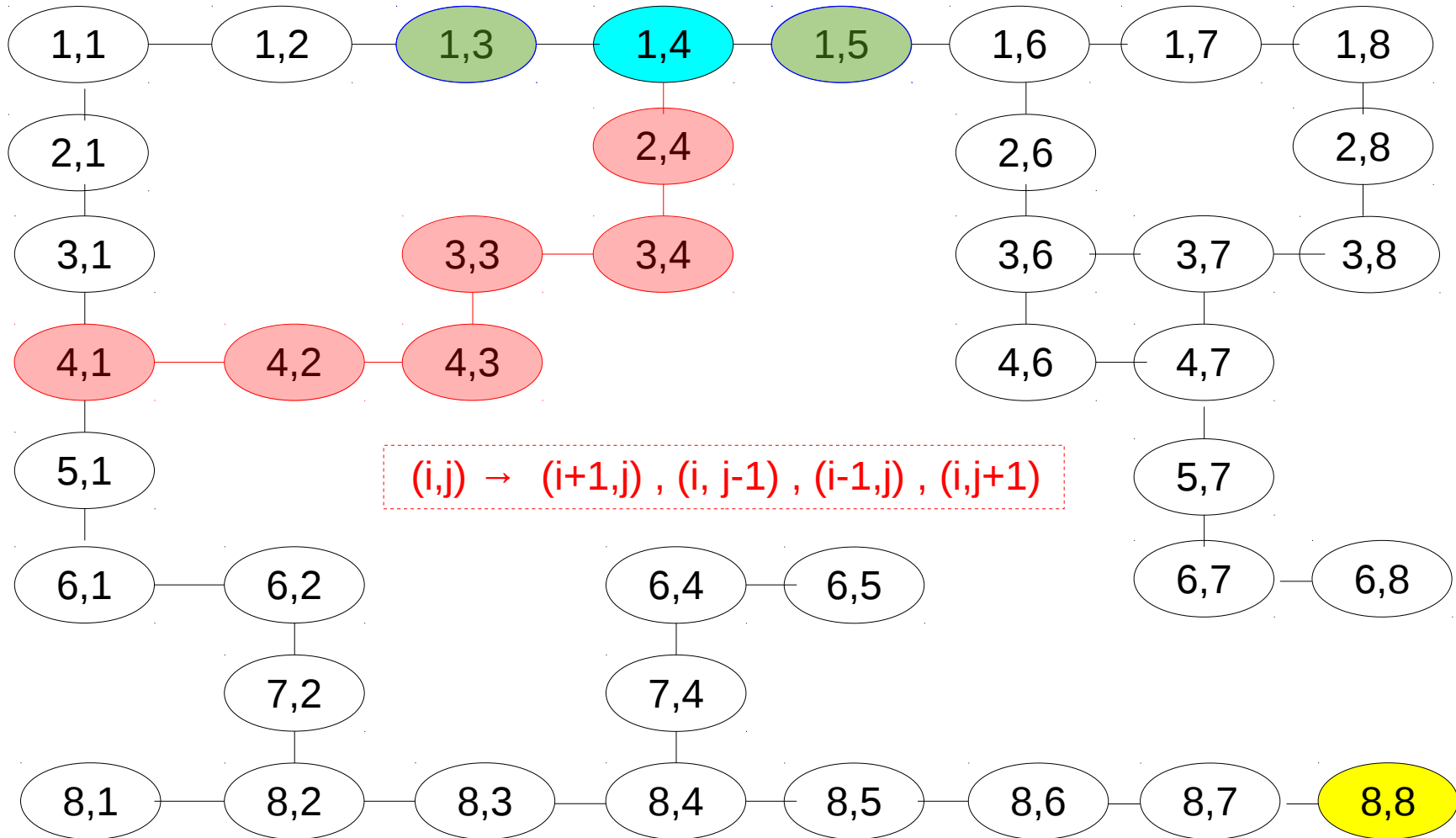
état courant : **(4,2)** / états successeurs : **(4,1)**



La pile

(1,5) | (1,3) |

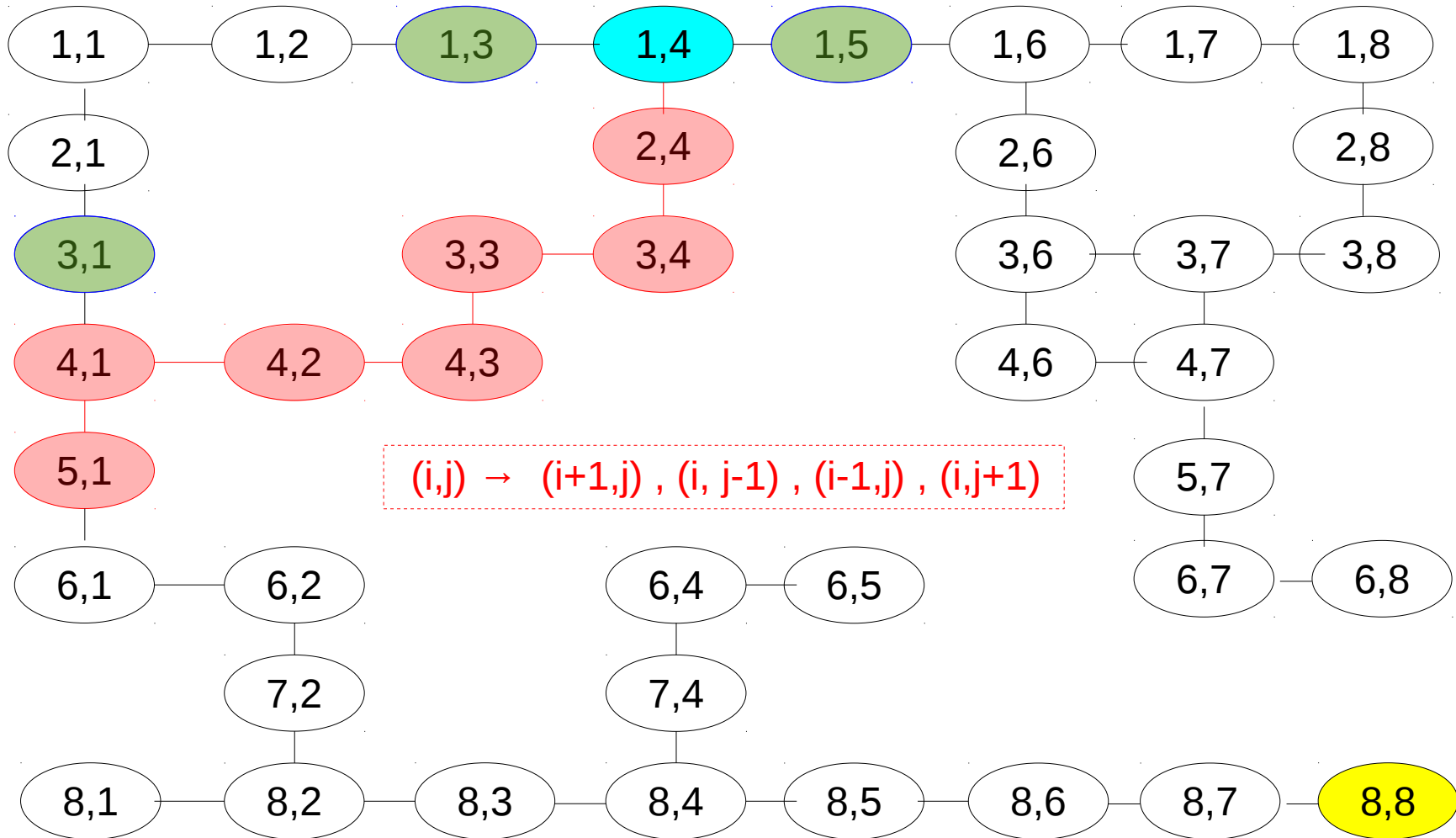
état courant : **(4,1)** / états successeurs : **(5,1)** , **(3,1)**



La pile

(1,5) | (1,3) |

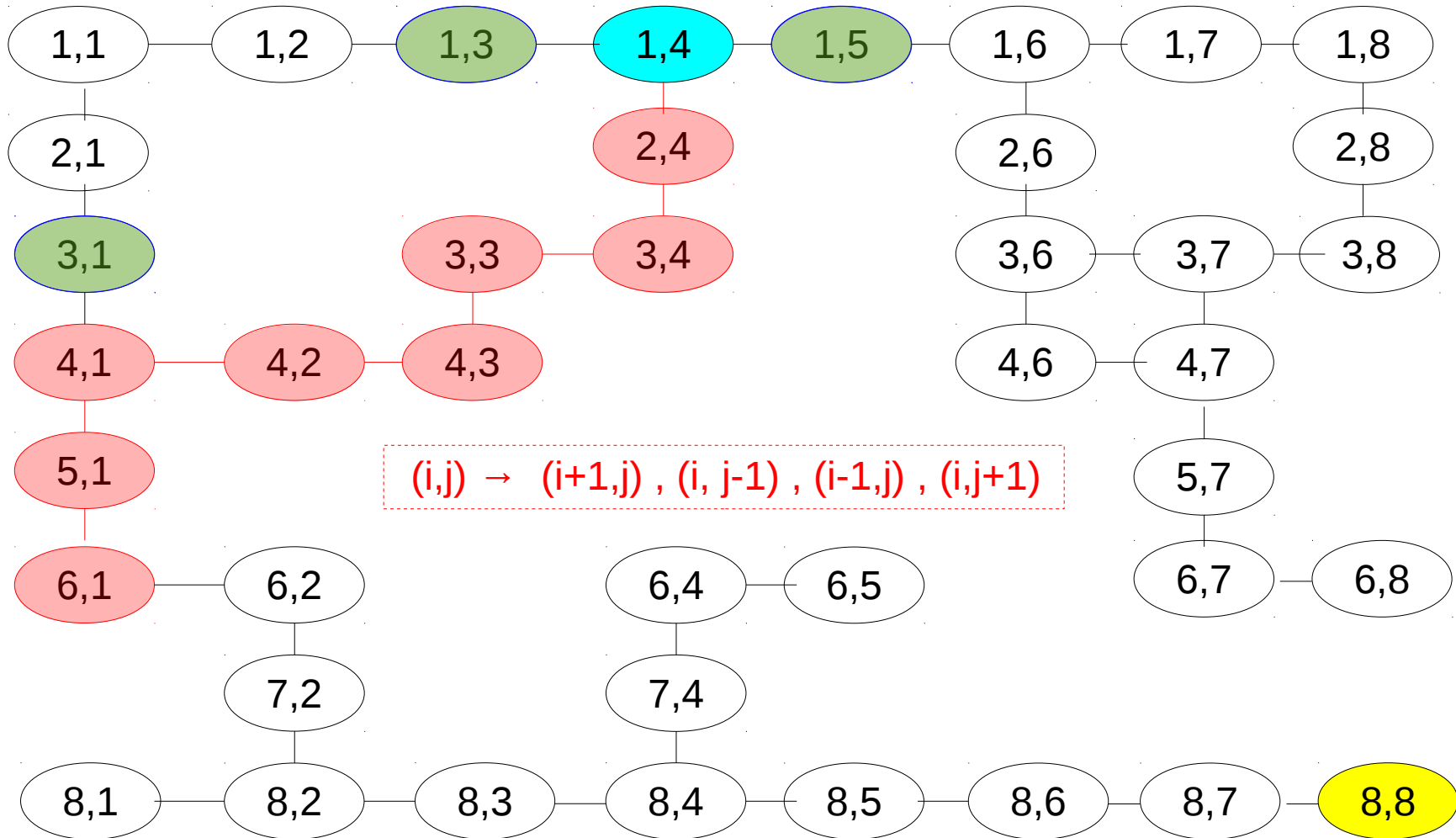
état courant : **(5,1)** / états successeurs : **(6,1)**



La pile

(1,5) | (1,3) | (3,1) |

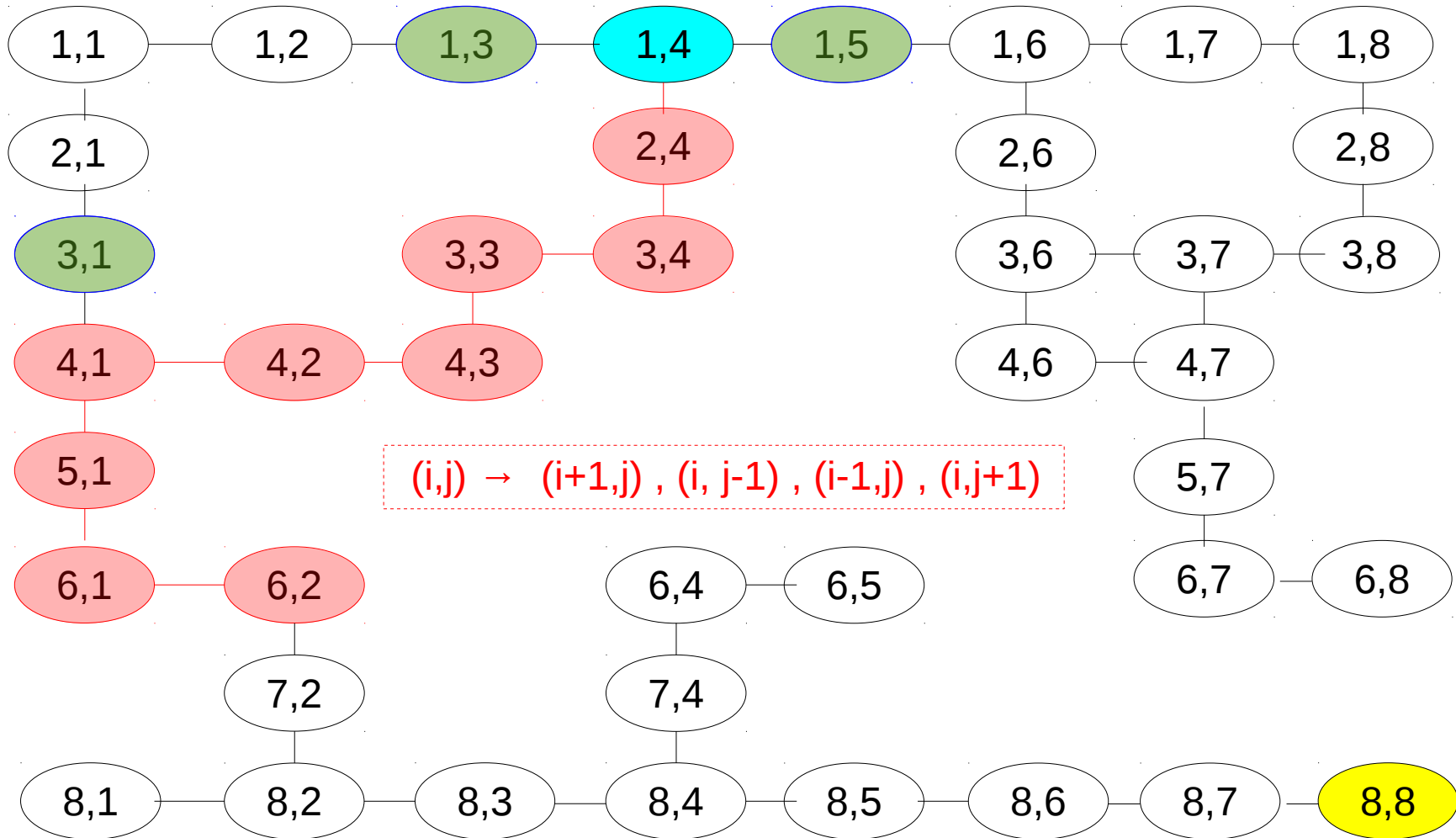
état courant : **(6,1)** / états successeurs : **(6,2)**



La pile

(1,5) | (1,3) | (3,1) |

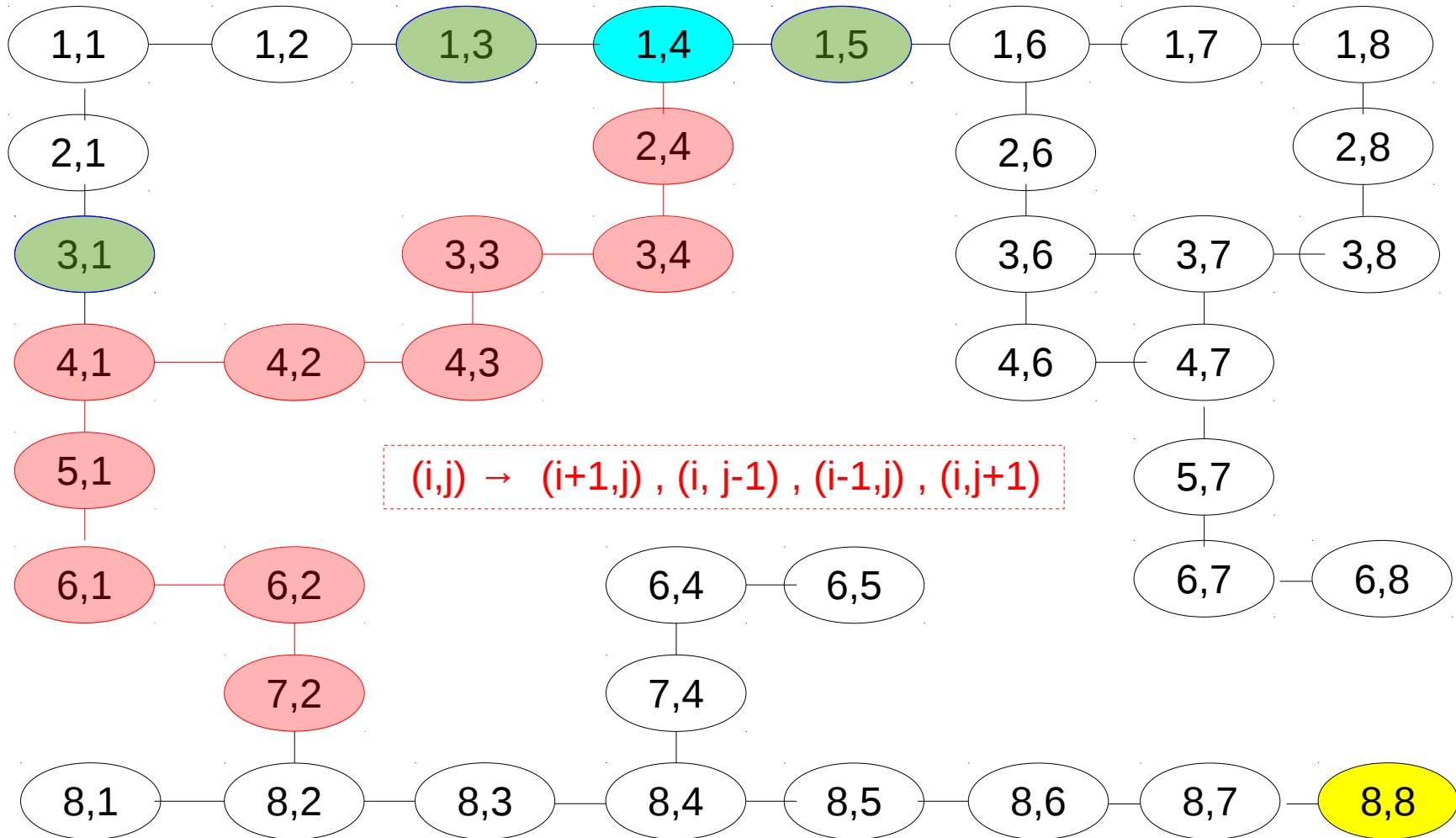
état courant : **(6,2)** / états successeurs : **(7,2)**



La pile

(1,5) | (1,3) | (3,1) |

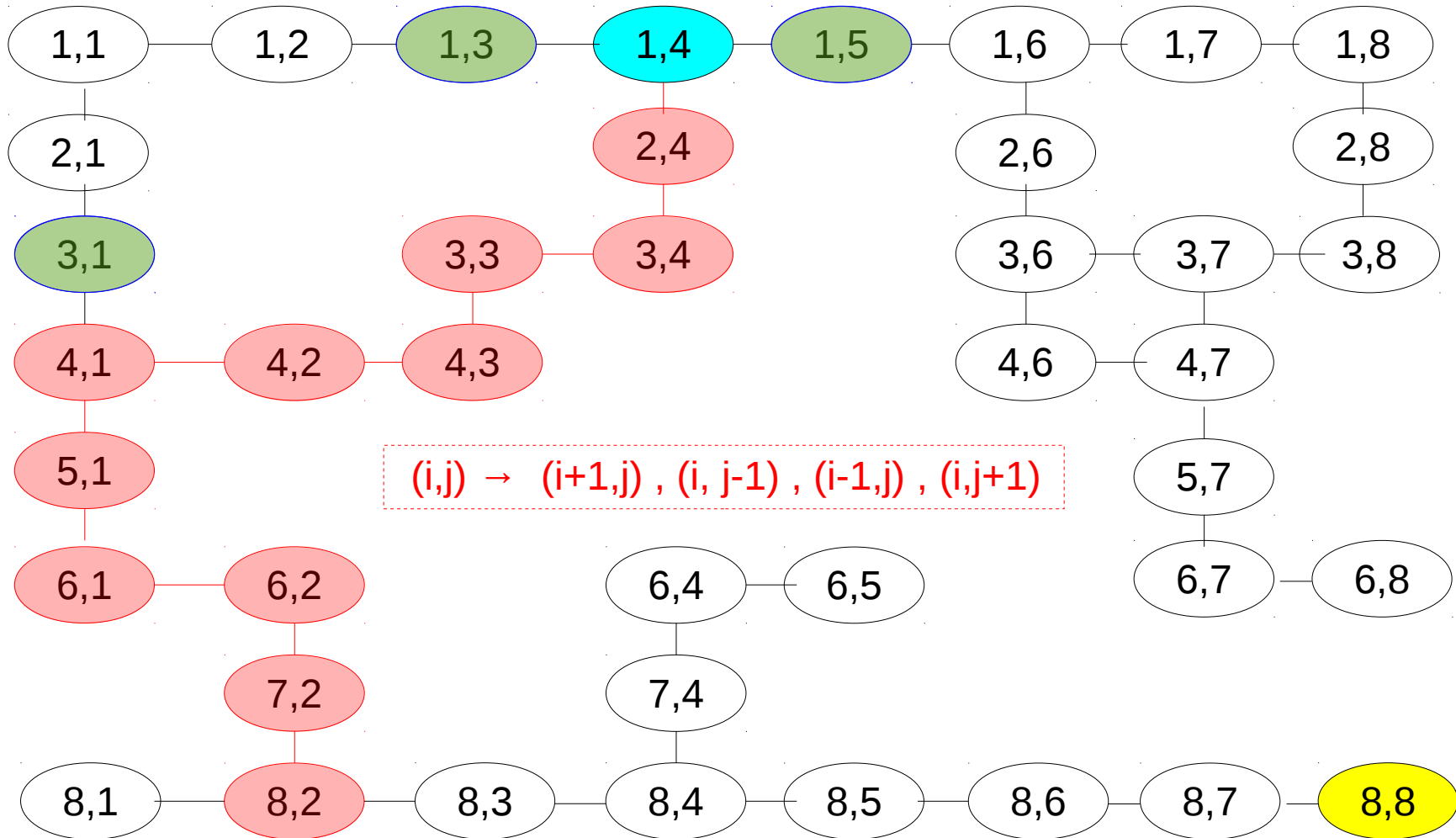
état courant : **(7,2)** / états successeurs : **(8,2)**



La pile

(1,5) | (1,3) | (3,1) |

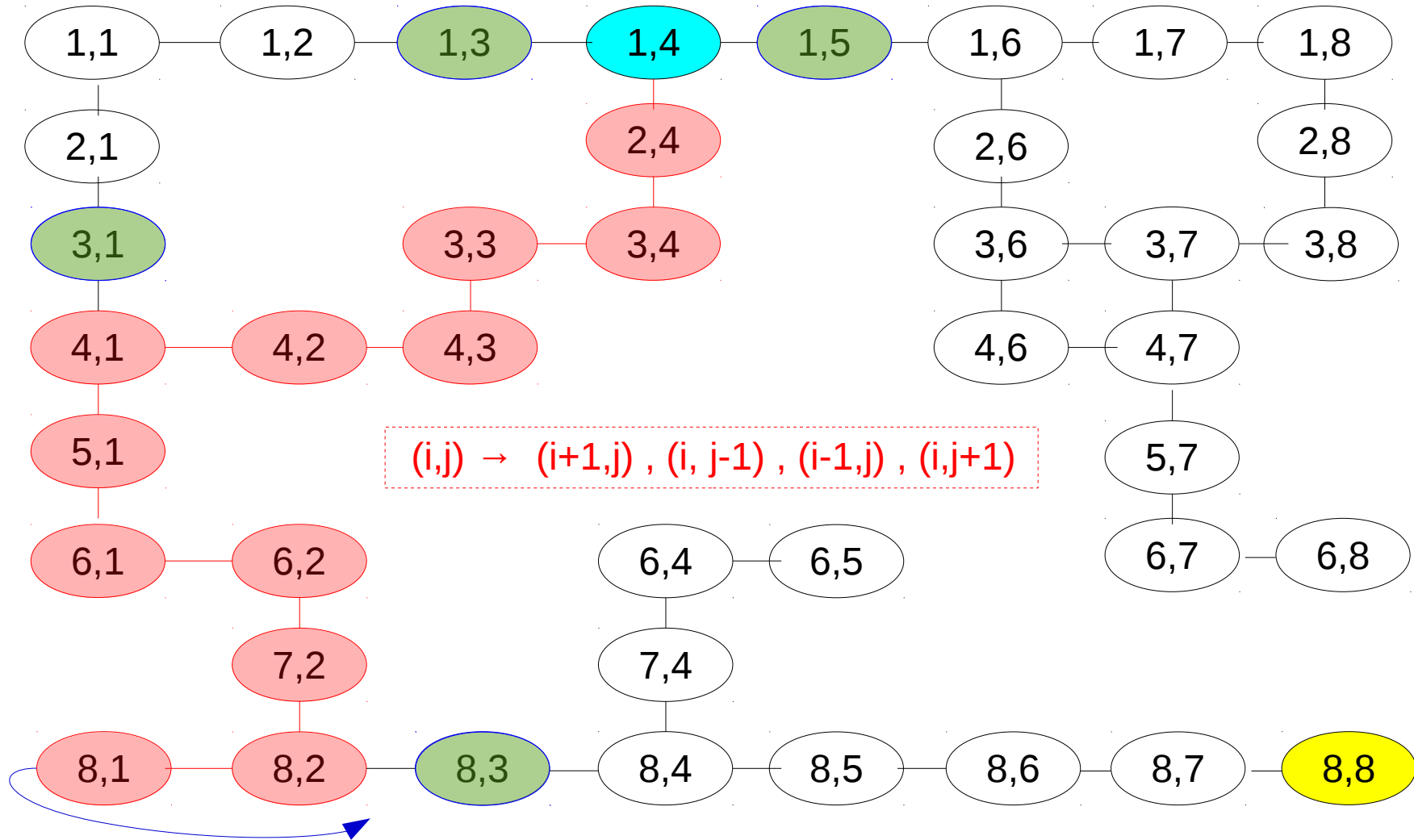
état courant : **(8,2)** / états successeurs : **(8,1)** , **(8,3)**



La pile

(1,5) | (1,3) | (3,1) |

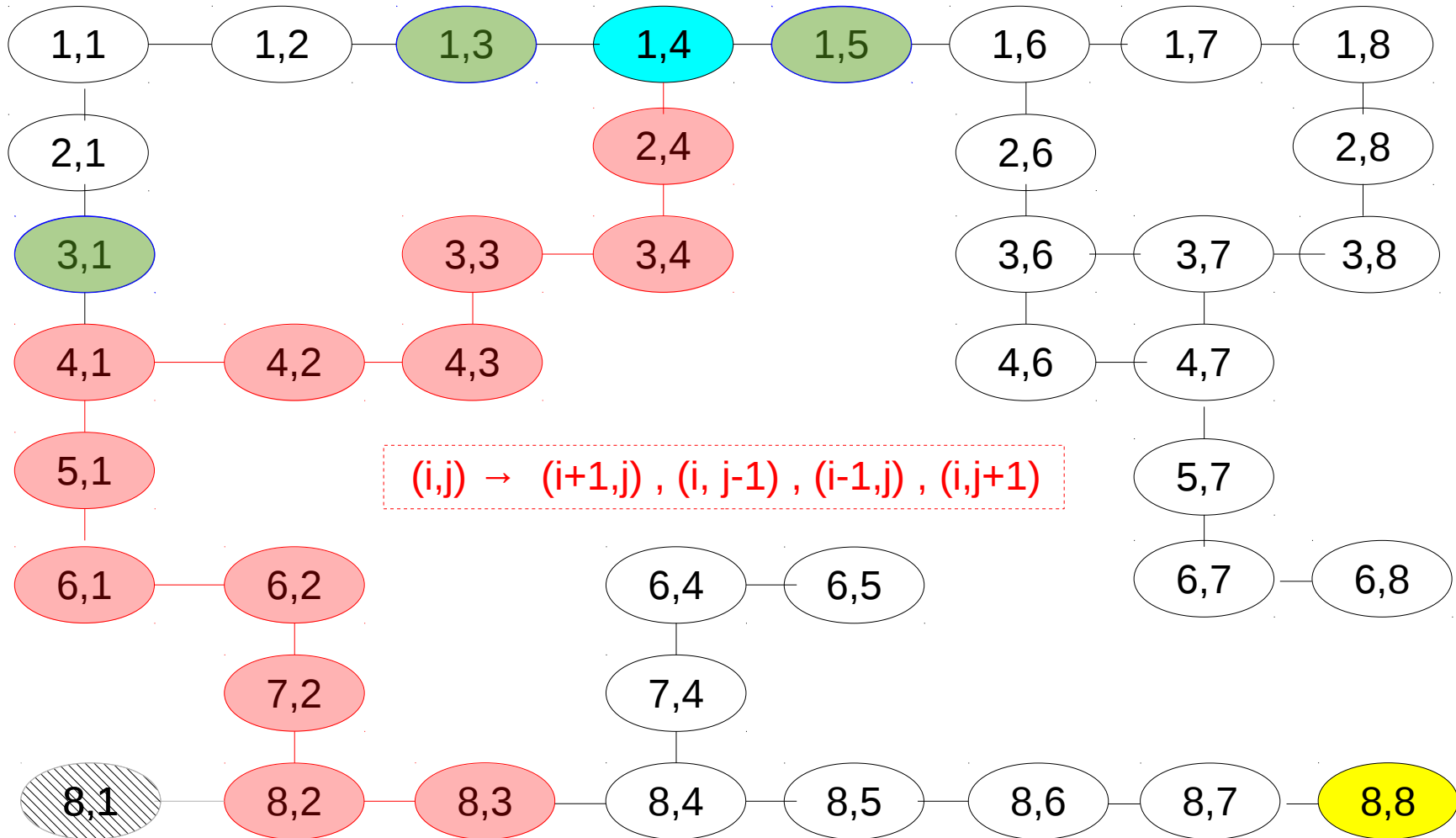
état courant : **(8,1)** / états successeurs : **-vide-** Donc : [Retour arrière](#)



La pile

(1,5) | (1,3) | (3,1) | **(8,3)** |

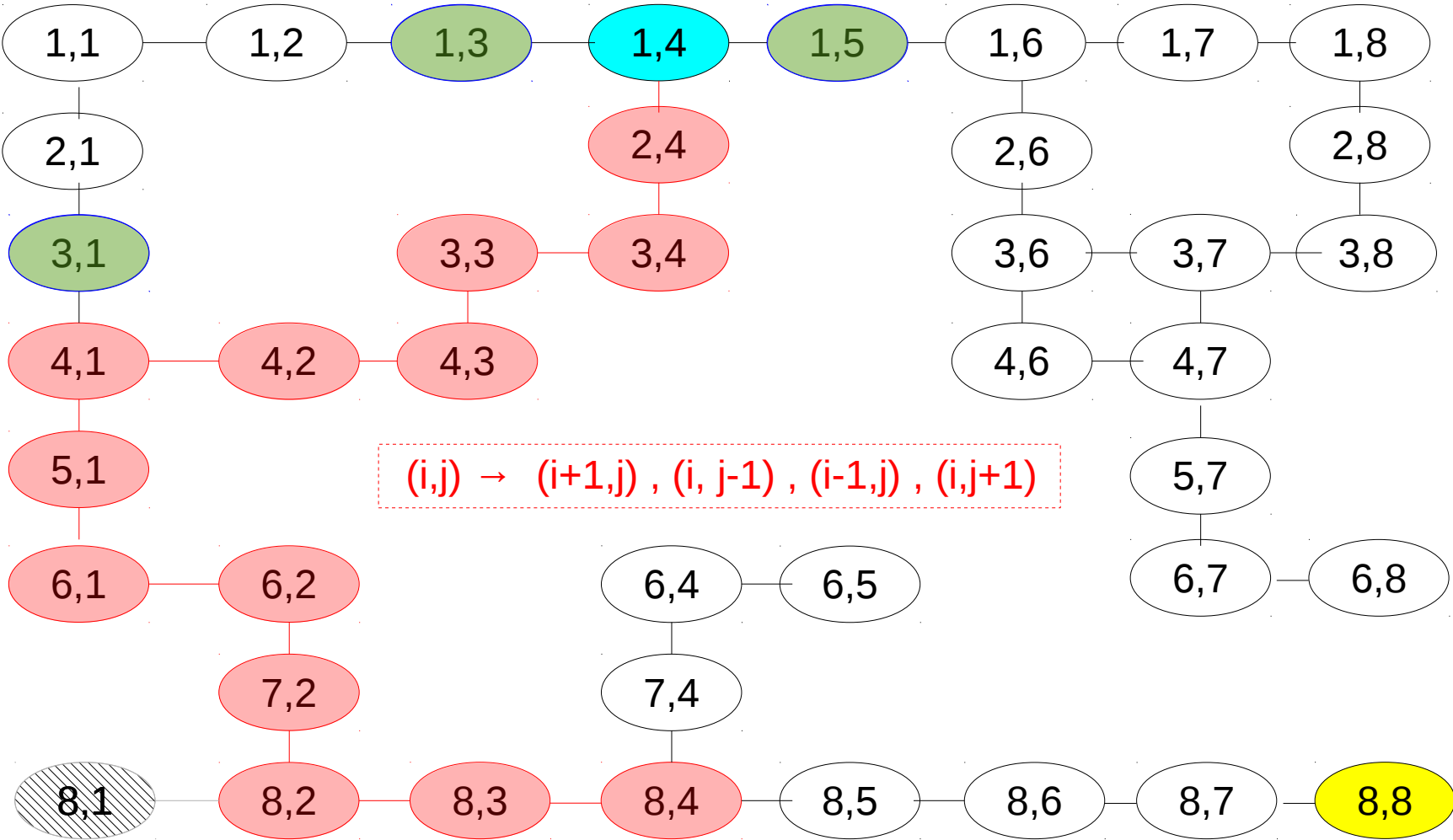
état courant : **(8,3)** / états successeurs : **(8,4)**



La pile

(1,5) | (1,3) | (3,1) |

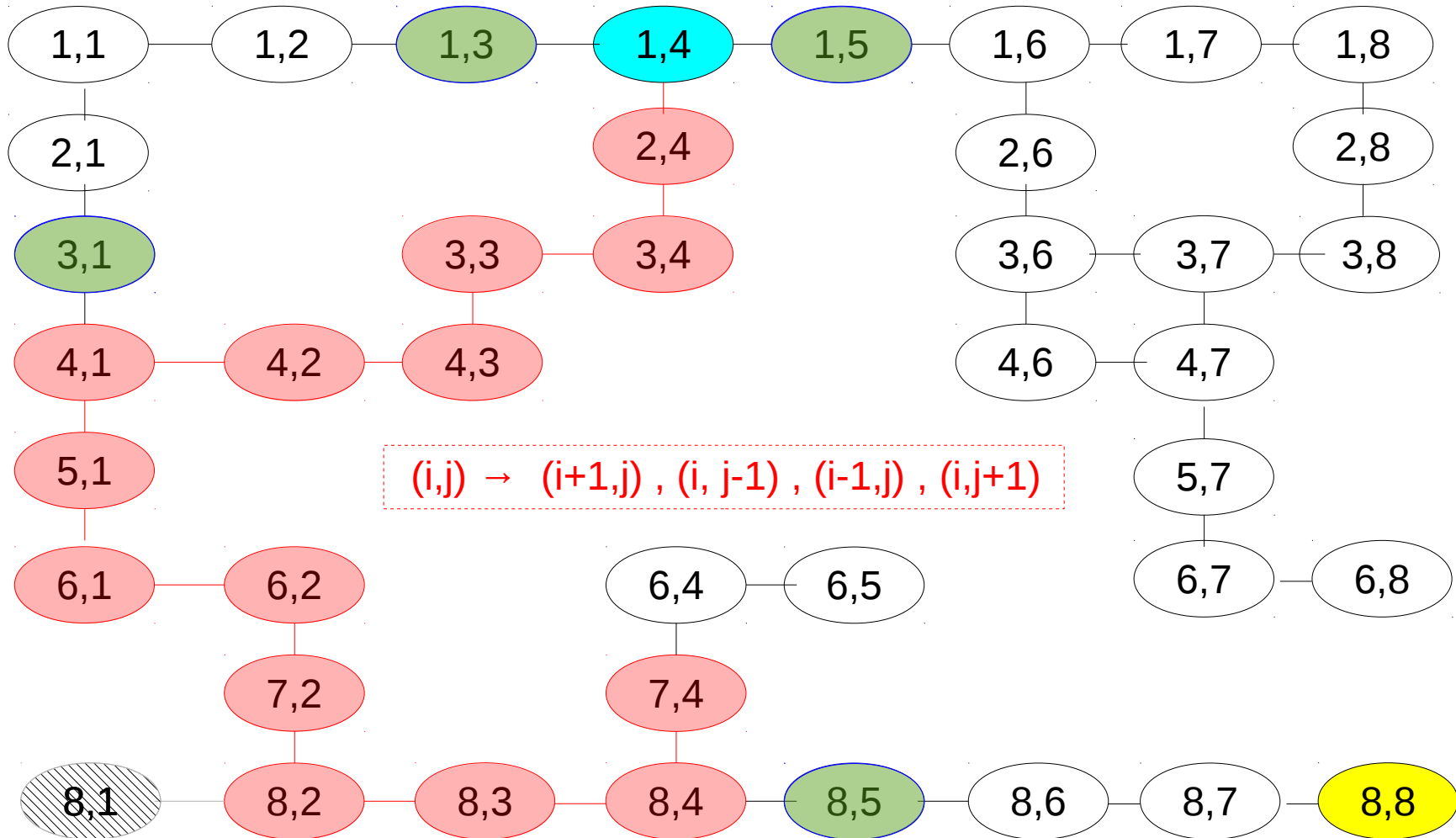
état courant : **(8,4)** / états successeurs : **(7,4)** , **(8,5)**



La pile

(1,5) | (1,3) | (3,1) |

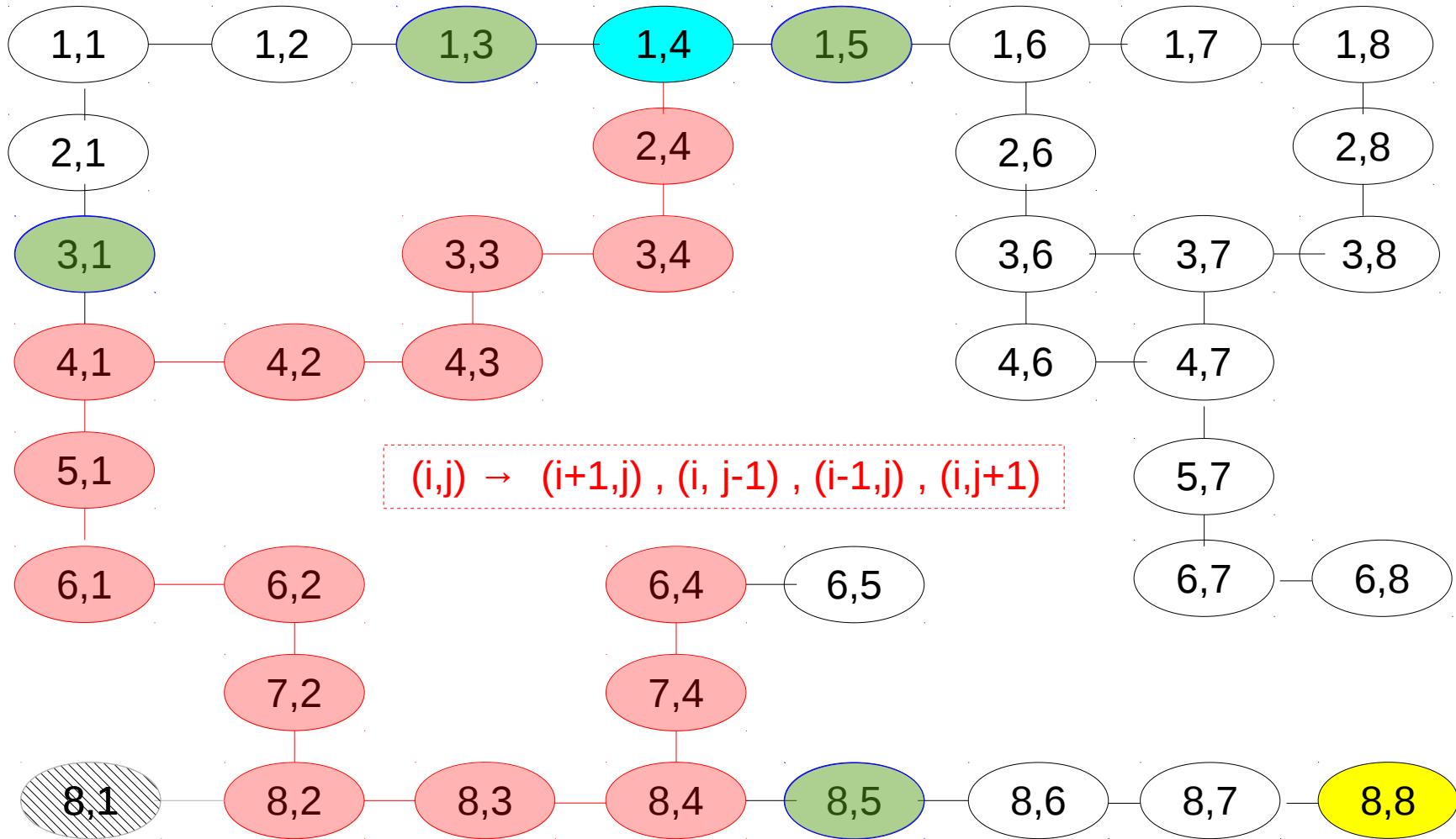
état courant : **(7,4)** / états successeurs : **(6,4)**



La pile

(1,5) | (1,3) | (3,1) | (8,5) |

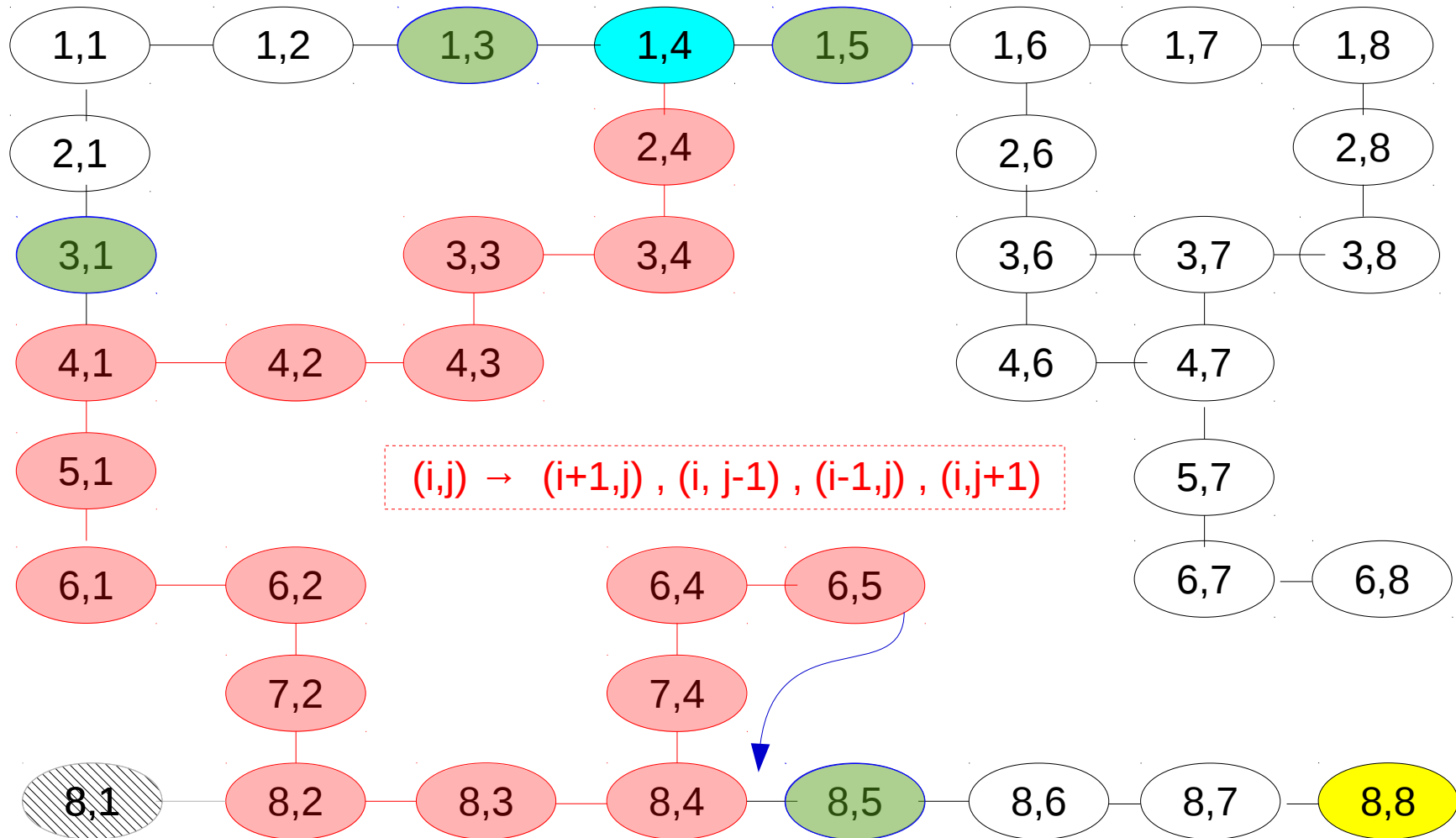
état courant : **(6,4)** / états successeurs : **(6,5)**



La pile

(1,5) | (1,3) | (3,1) | (8,5) |

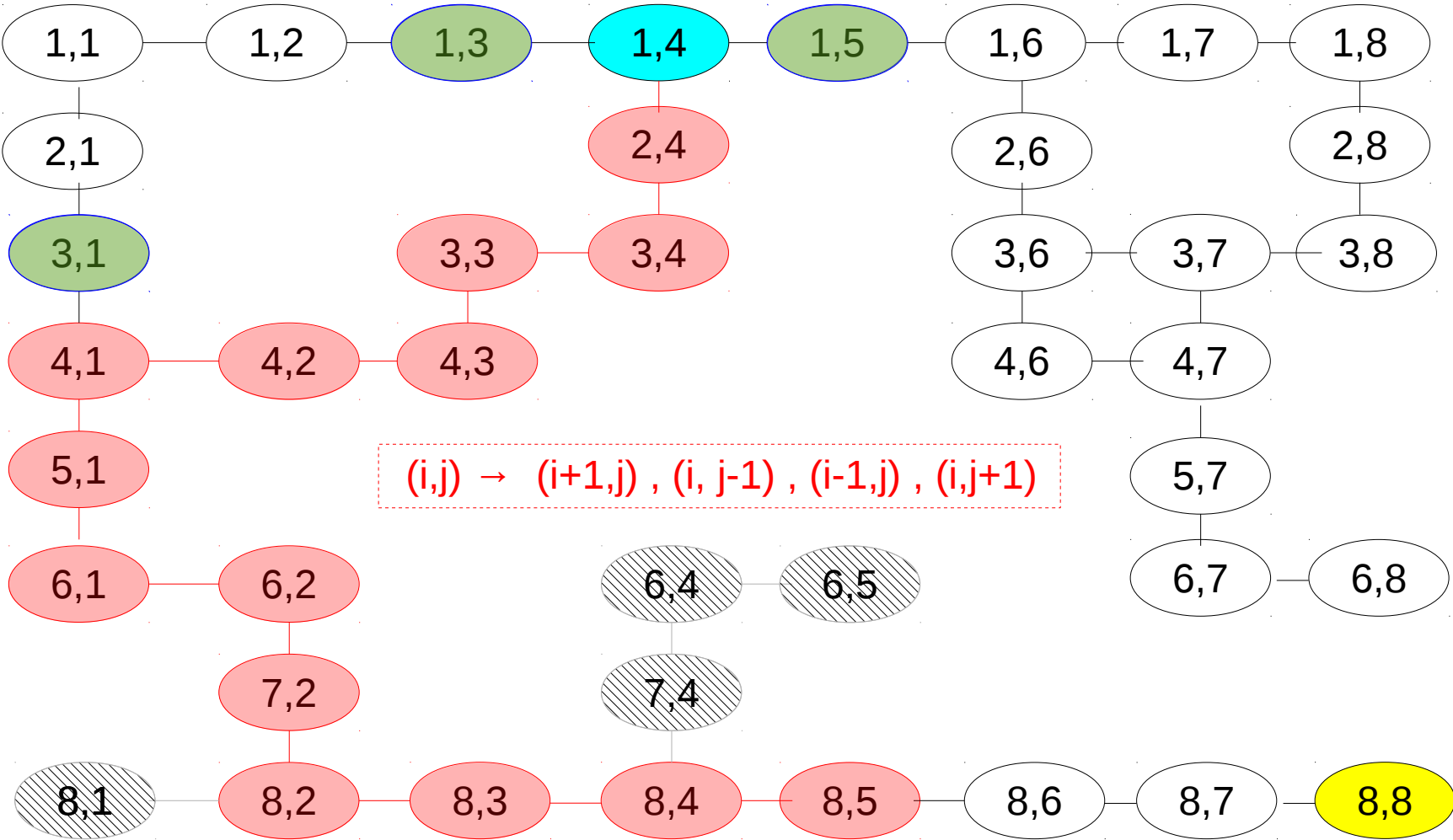
état courant : **(6,5)** / états successeurs : **-vide-** Donc [Retour arrière](#)



La pile

(1,5) | (1,3) | (3,1) | (8,5) |

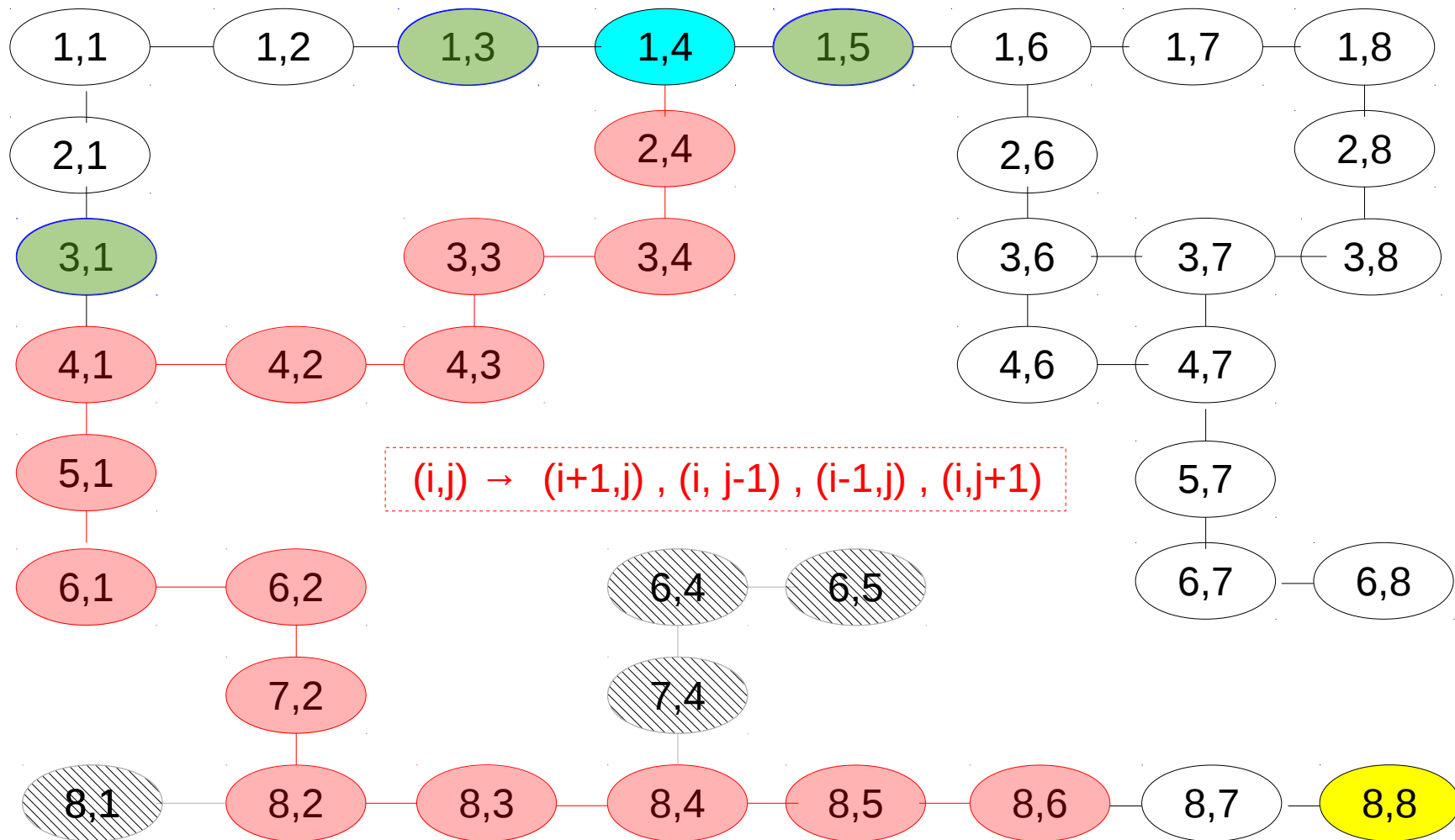
état courant : **(8,3)** / états successeurs : **(8,6)**



La pile

(1,5) | (1,3) | (3,1) |

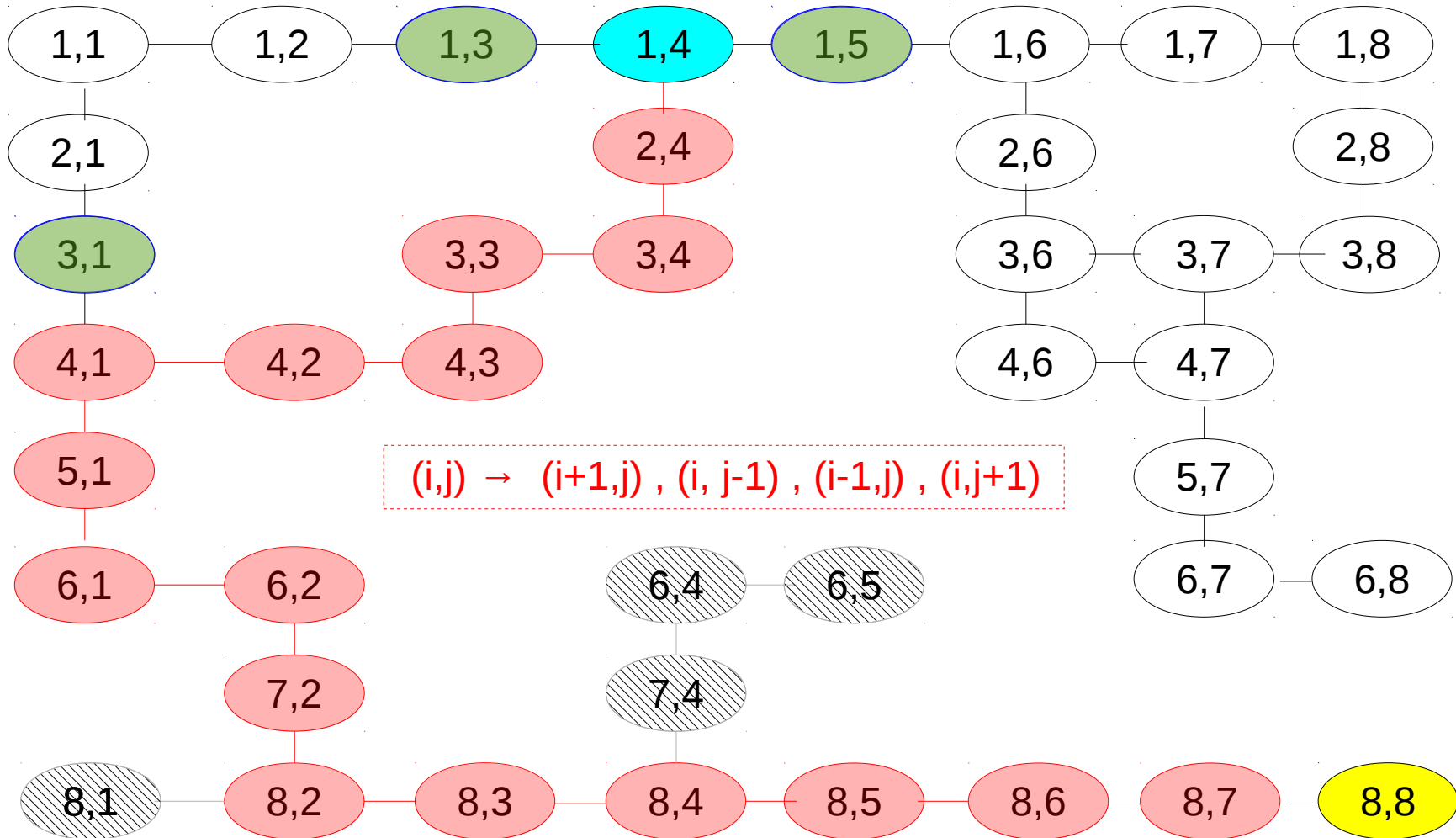
état courant : **(8,6)** / états successeurs : **(8,7)**



La pile

(1,5) | (1,3) | (3,1) |

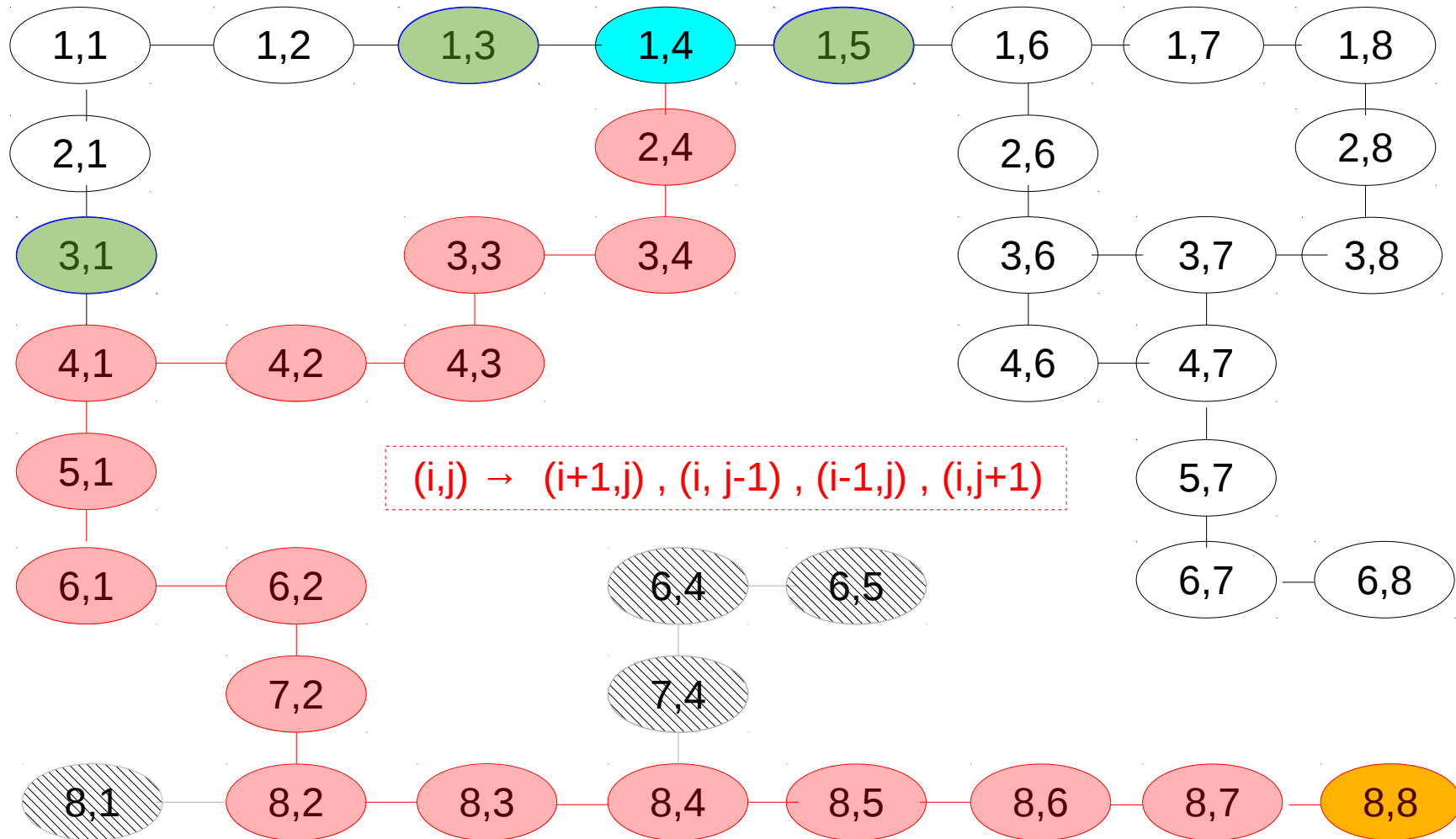
état courant : **(8,7)** / états successeurs : **(8,8)**



La pile

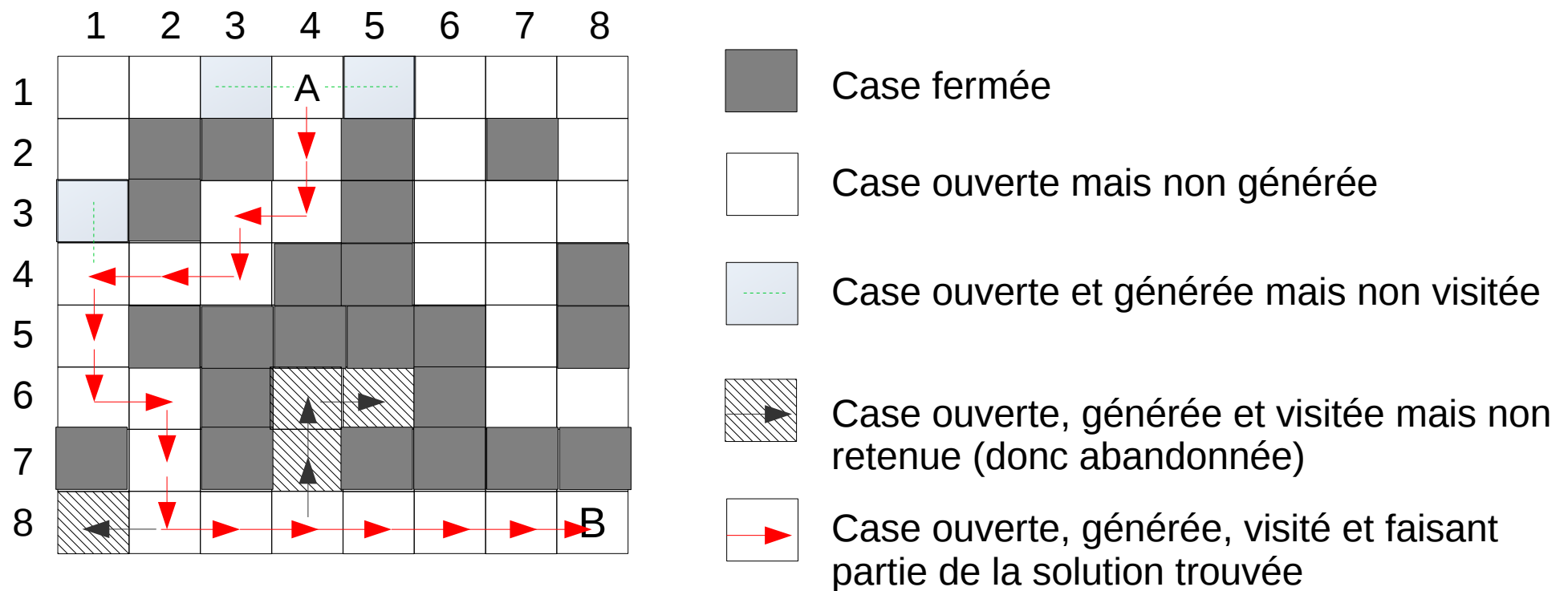
(1,5) | (1,3) | (3,1) |

état courant : **(8,8)** / états successeurs : — **(recherche terminée)**



La pile

(1,5) | (1,3) | (3,1) |



Longueur du chemin solution = 17 cases

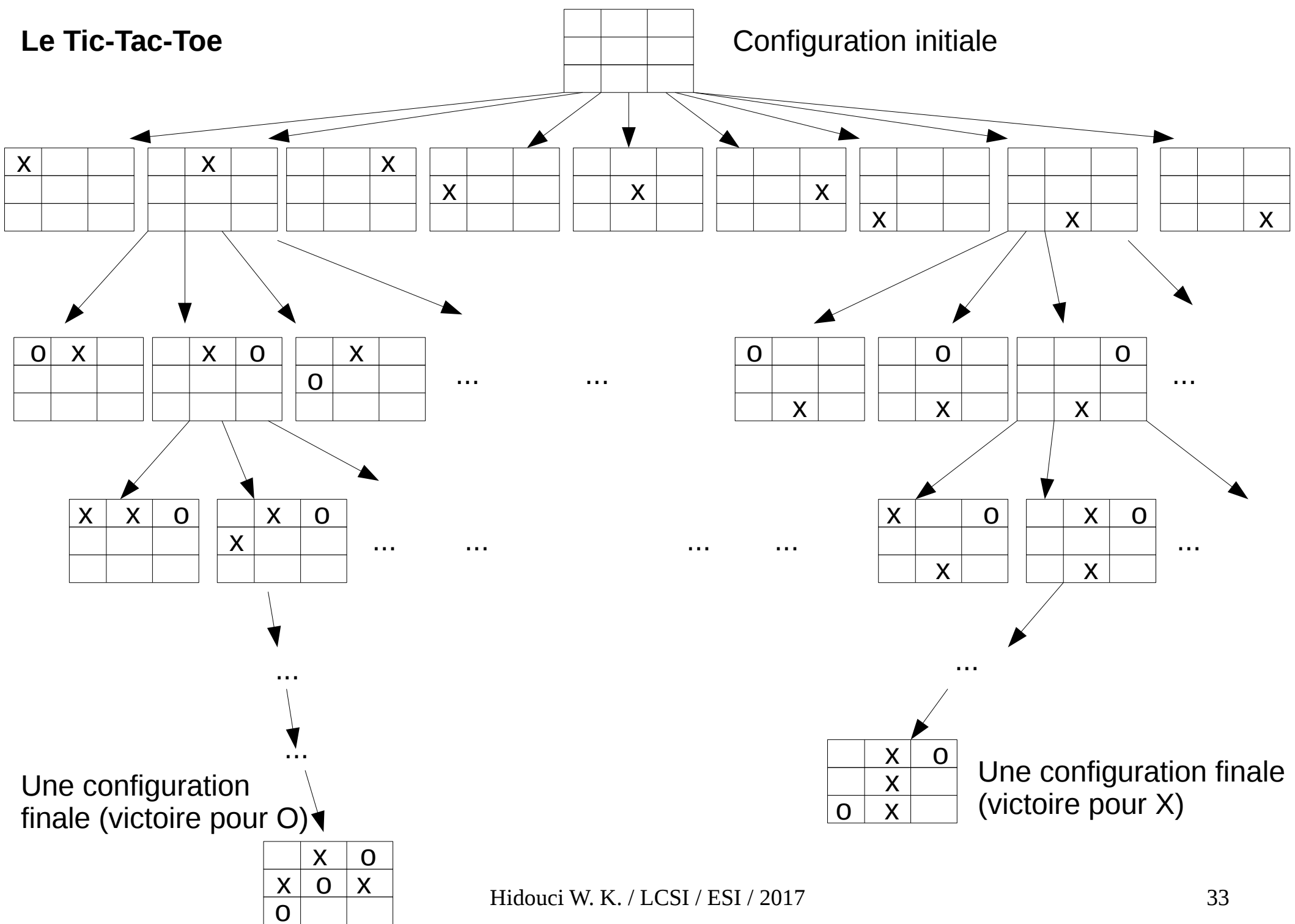
Nombre de cases visitées = 21 cases

Nombre de cases générées (visitées ou dans la pile) = 24 cases

Problèmes de jeux

- Le backtracking s'adapte facilement pour les problèmes de jeux à deux joueurs opposés
 - **Algorithme du Min-Max (ou MiniMax)**
- Les coups des 2 joueurs sont **effectués de manière alternée.**
- L'espace de recherche est composé de toutes les configurations possibles obtenues par les différents coups des 2 joueurs
- Une partie est un chemin dans l'espace de recherche entre la configuration initiale du jeu et une des configurations finales possibles (victoire, perte ou match nul)
- La résolution de problème (dans ce cas) consiste à trouver une suite de coups permettant au programme de jouer le mieux possible contre un adversaire quelconque.

Le Tic-Tac-Toe



MinMax

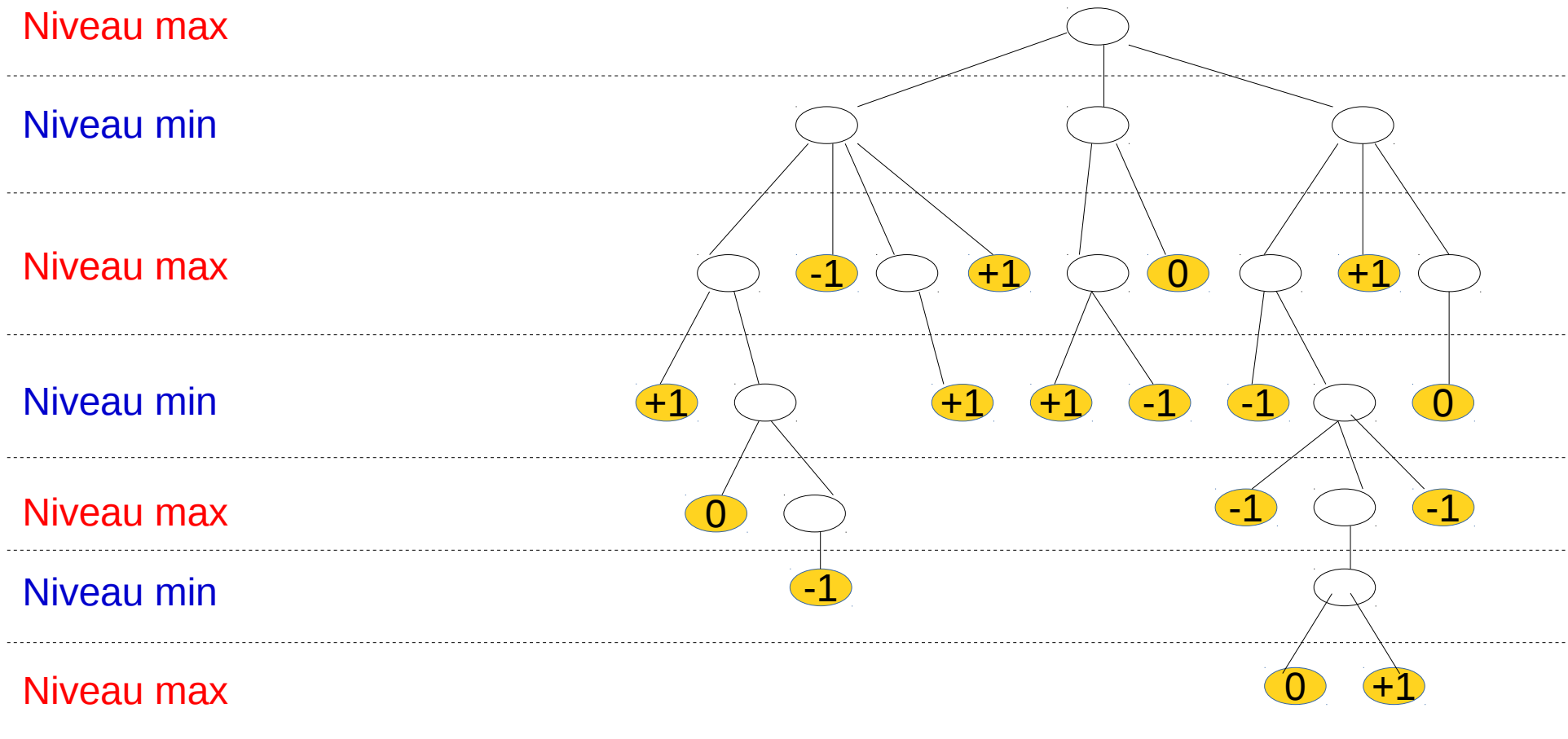
Objectif : Donner une valeur à une configuration

+1 : victoire du joueur A

-1 : victoire du joueur B

0 : match nul

L'évaluation des configurations terminales dépend des règles du jeu



MinMax

Evaluation d'un nœud **J** appartenant à un niveau **Mode** {min ou max}

MinMax(J, Mode)

Si (J est une feuille)

Retourner (coût(J)) // -1 (victoire pour min),
 // 0 (match nul) ou
 // +1 (victoire pour max)

Sinon

Si (Mode = 'max') Val $\leftarrow -\infty$ Sinon Val $\leftarrow +\infty$ Fsi

Générer_tous_les_successeurs(J)

Pour chaque successeur K de J :

 Si (Mode = 'max')

 Val \leftarrow Max(Val, MinMax(K, 'min'))

 Sinon

 Val \leftarrow Min(Val, MinMax(K, 'max'))

 Fsi

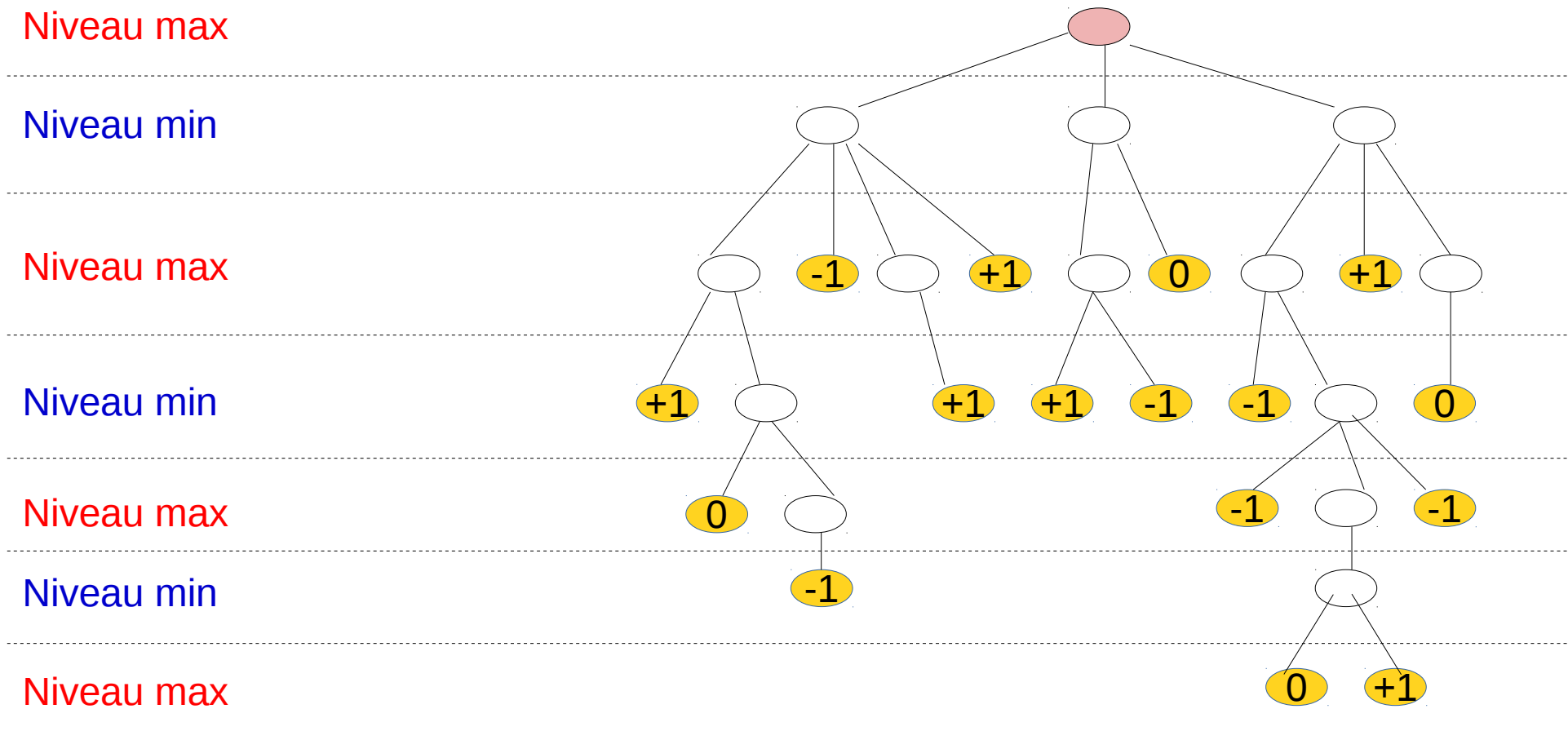
FP

Retourner Val

Fsi

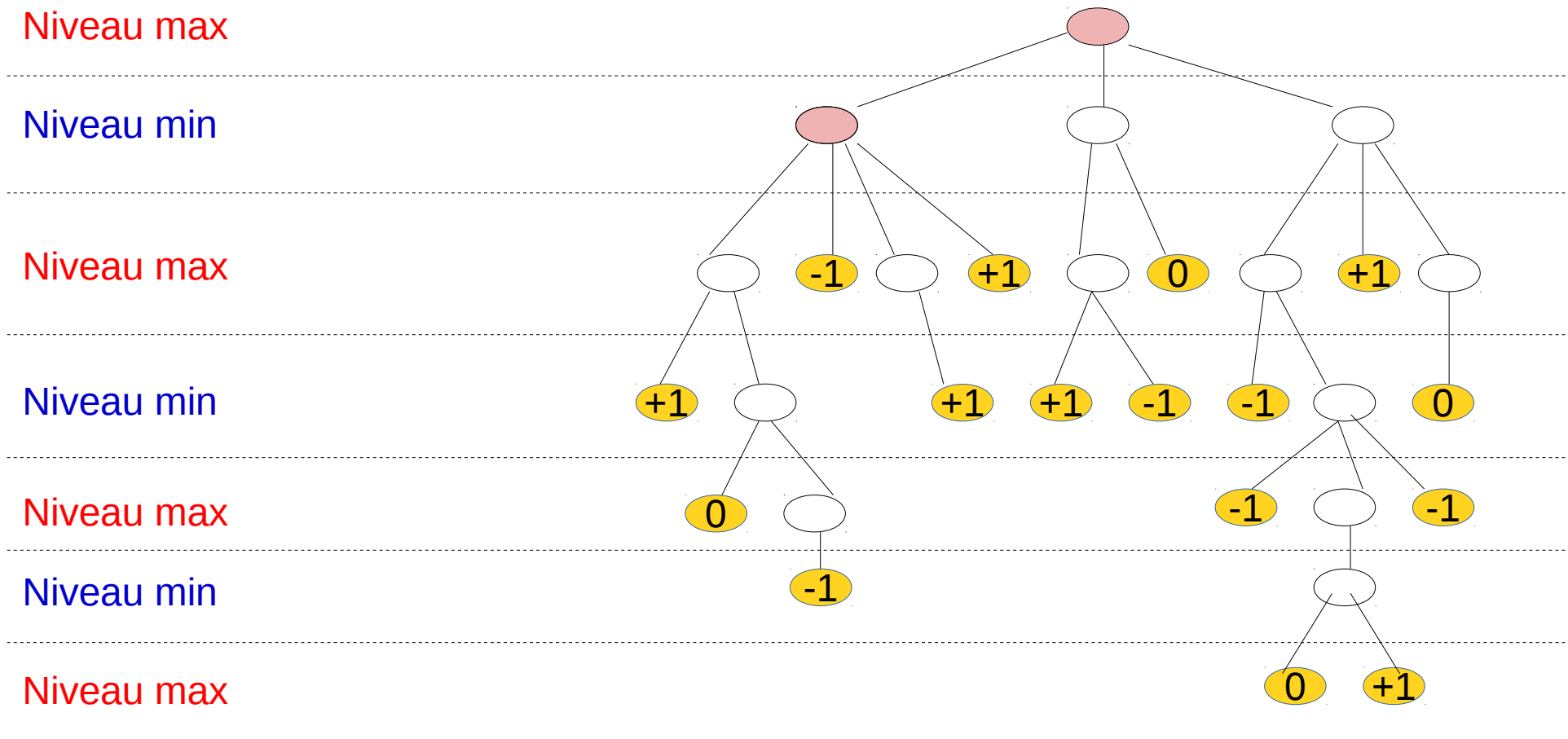
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



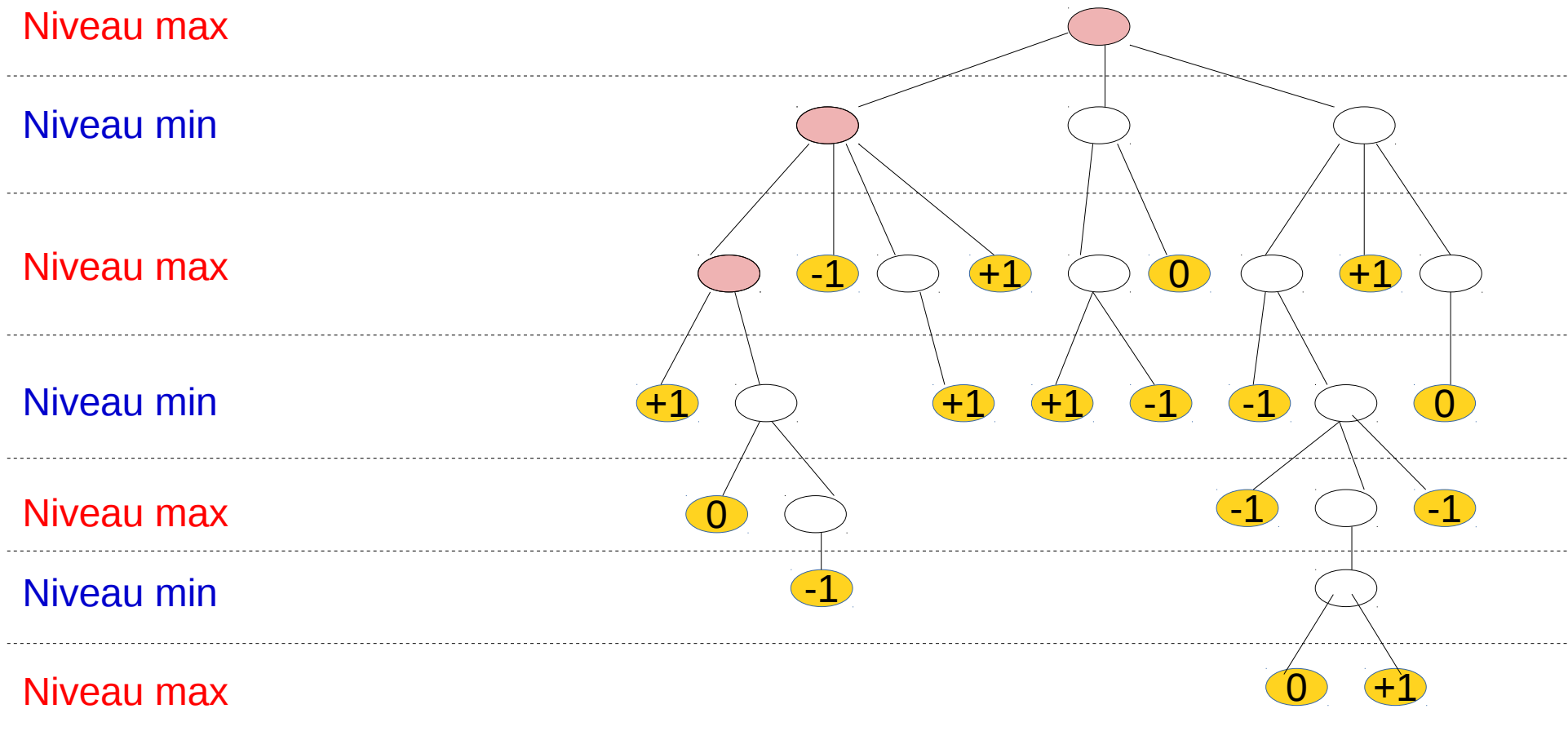
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



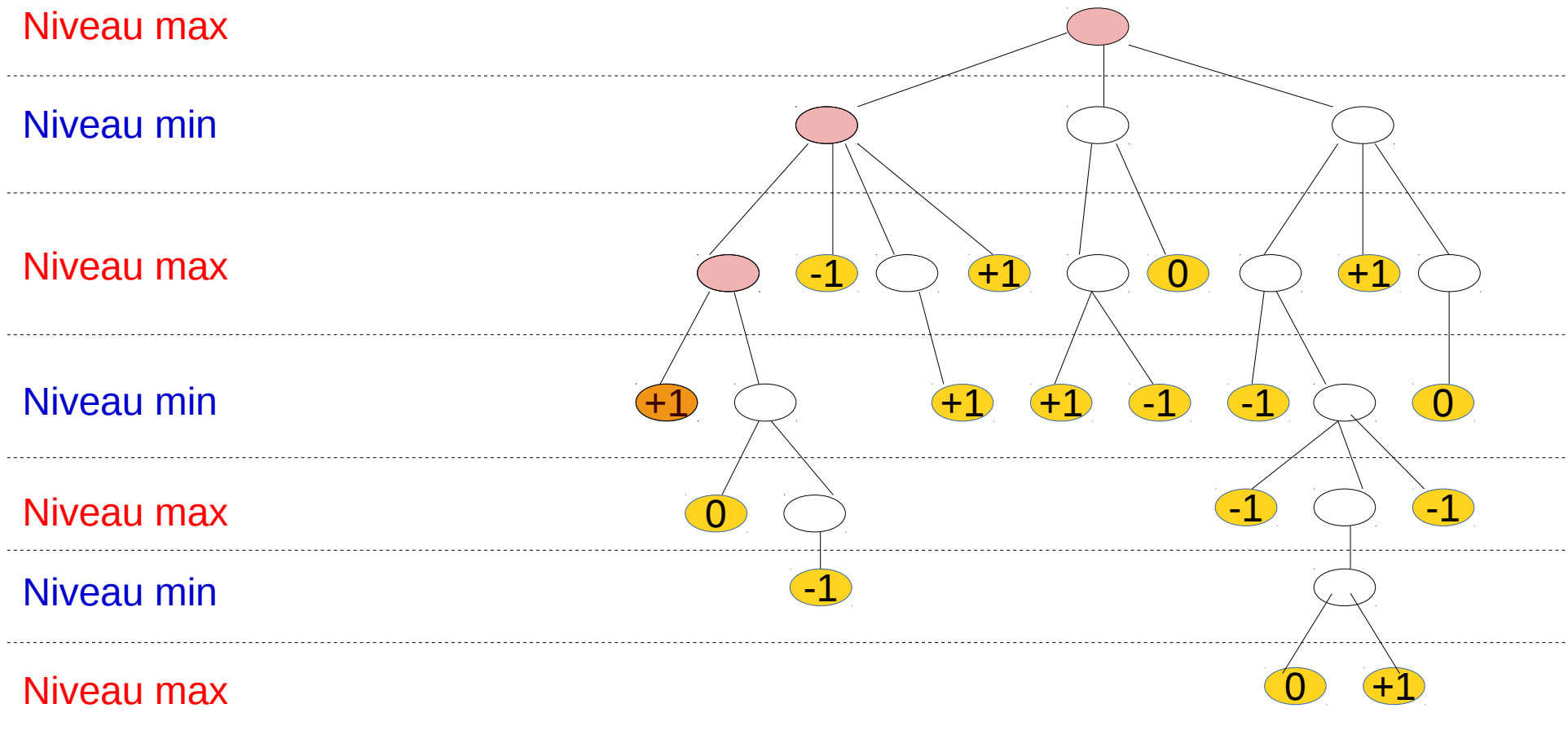
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



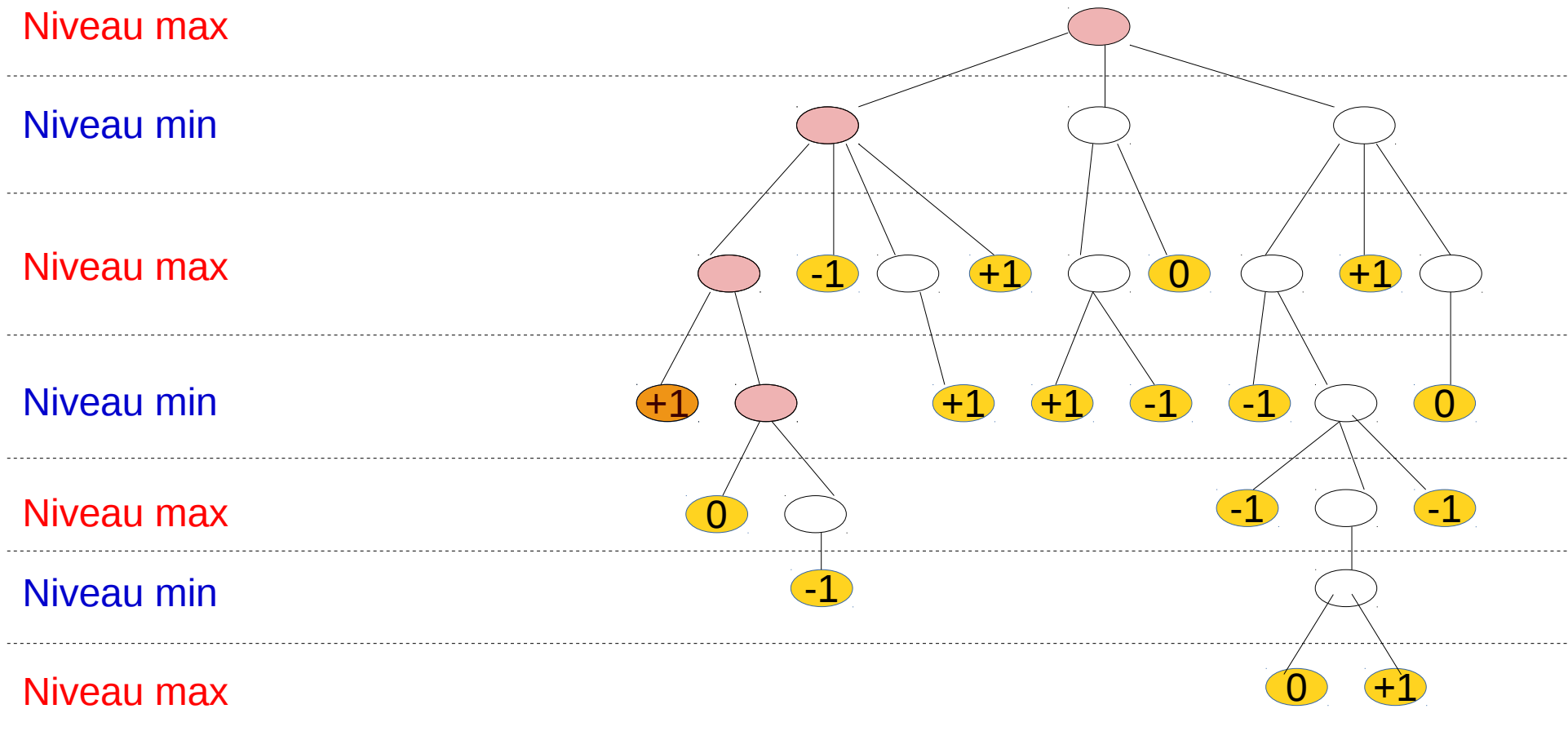
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



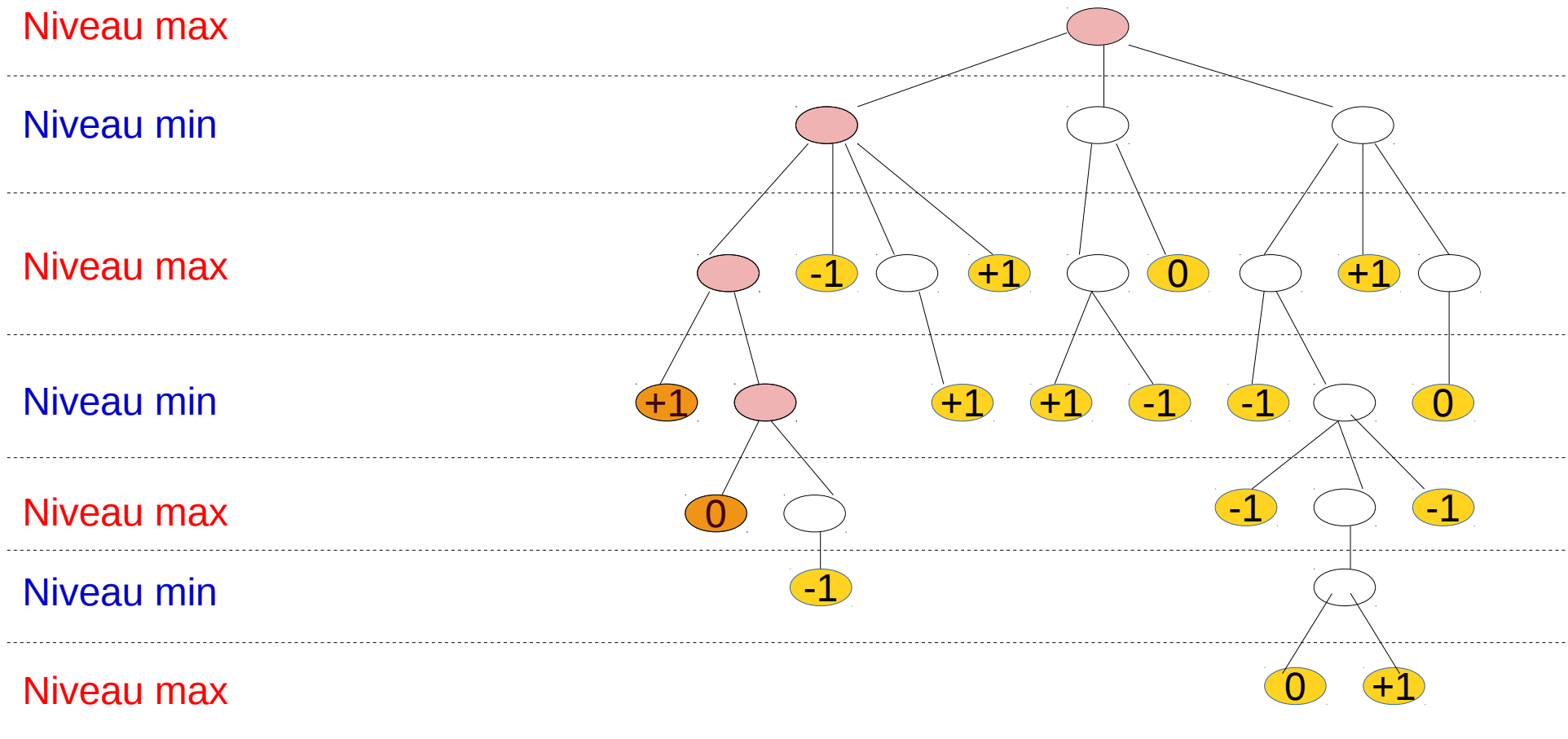
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



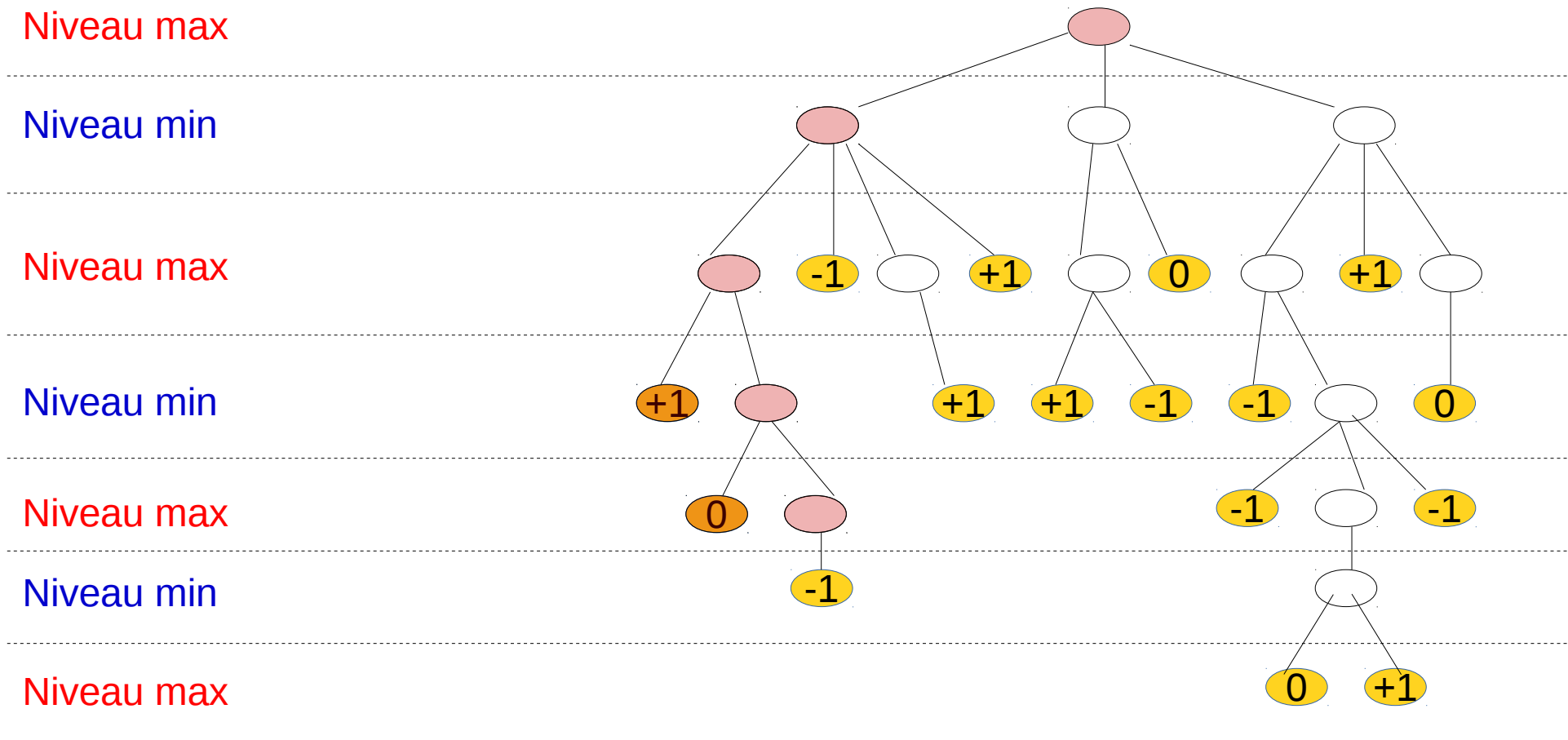
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



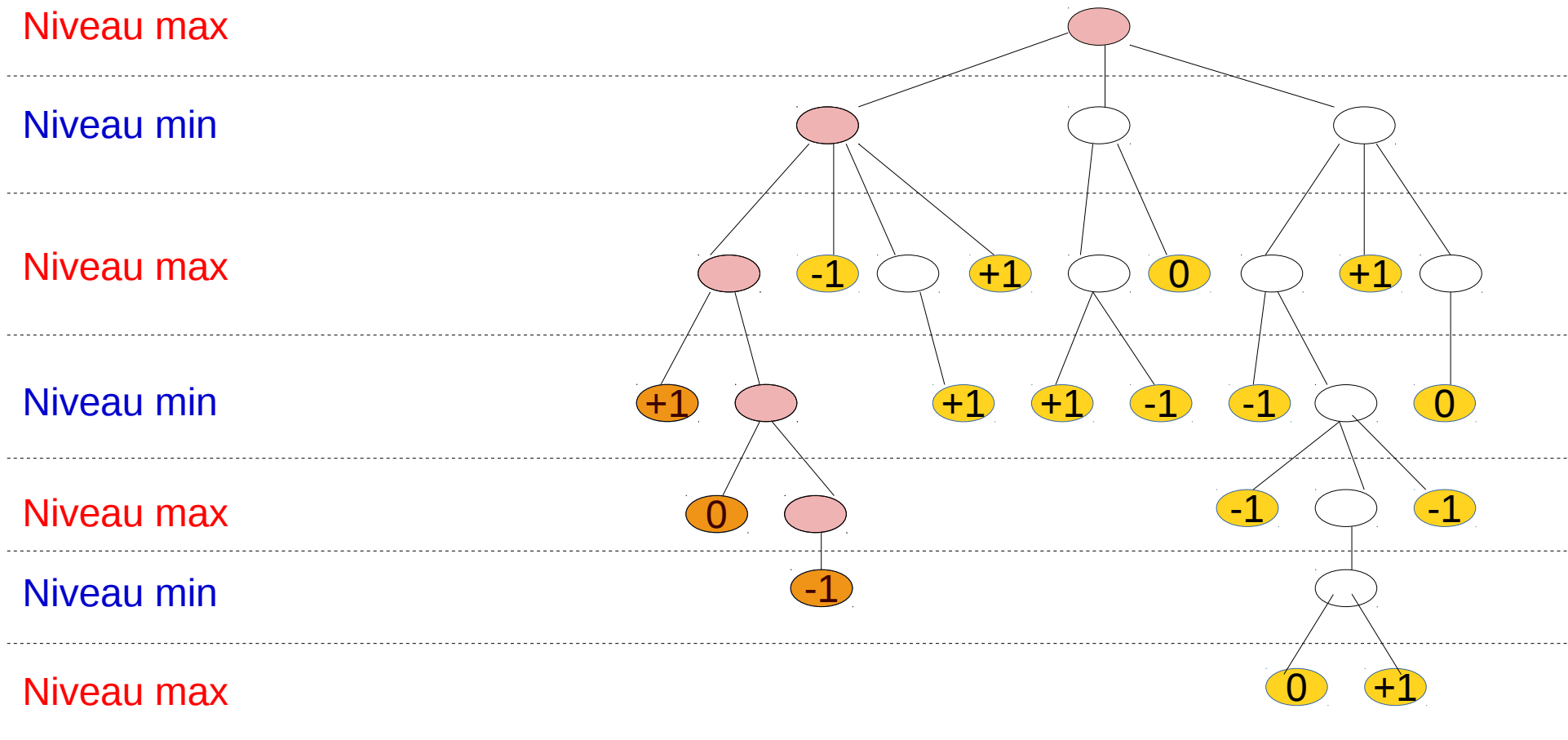
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)

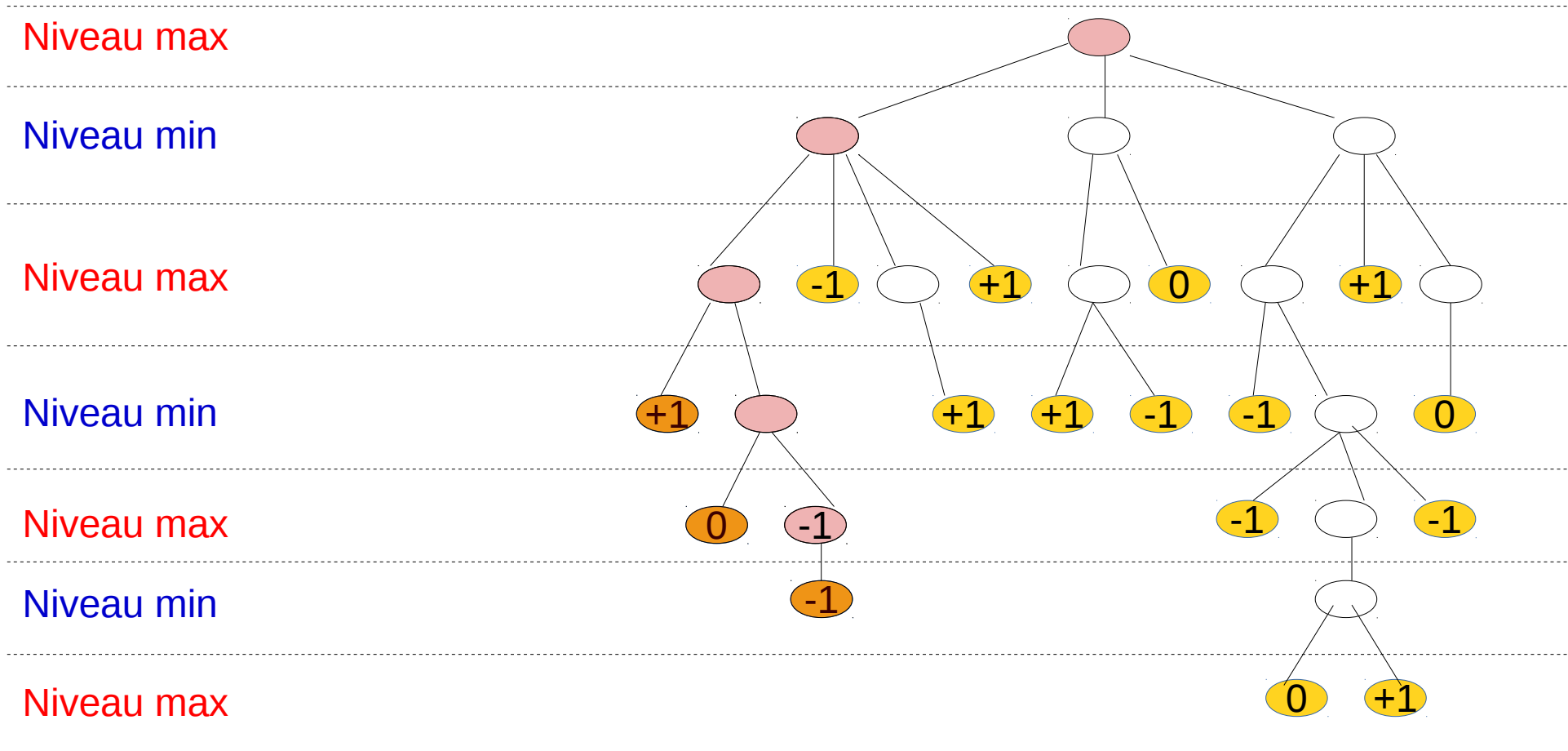


MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils

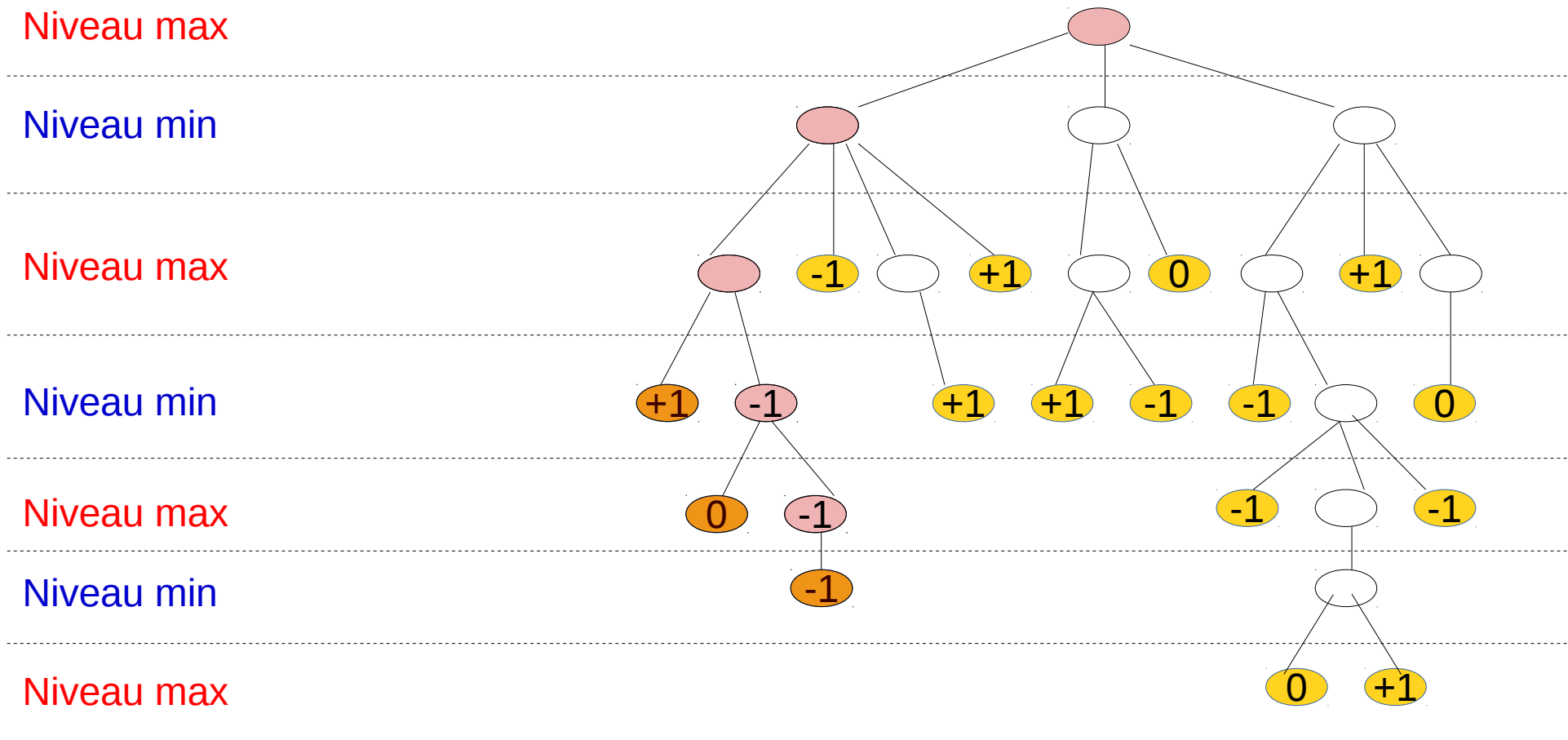
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils

si le nœud courant est terminal, sa valeur sera donnée par coût(J)



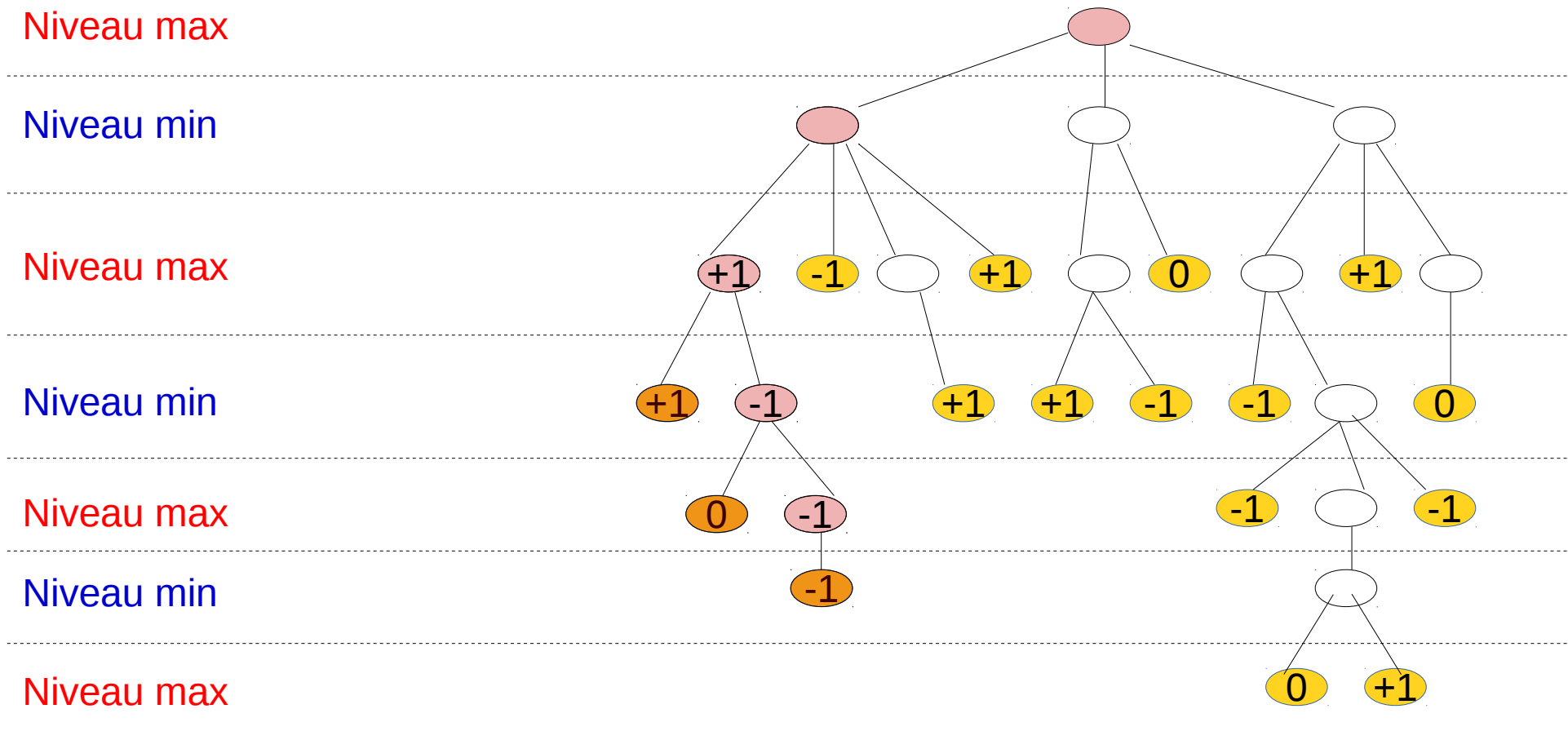
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



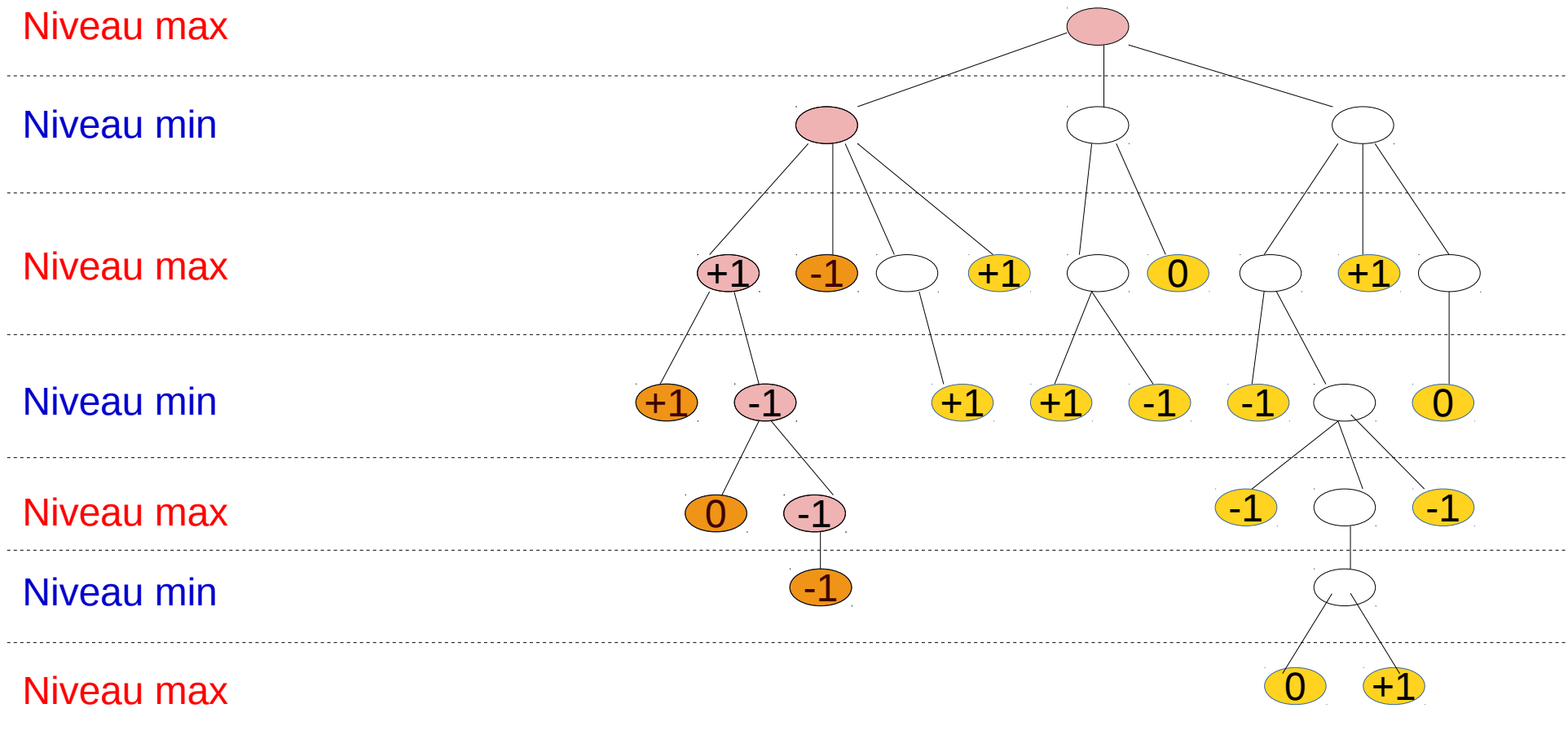
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



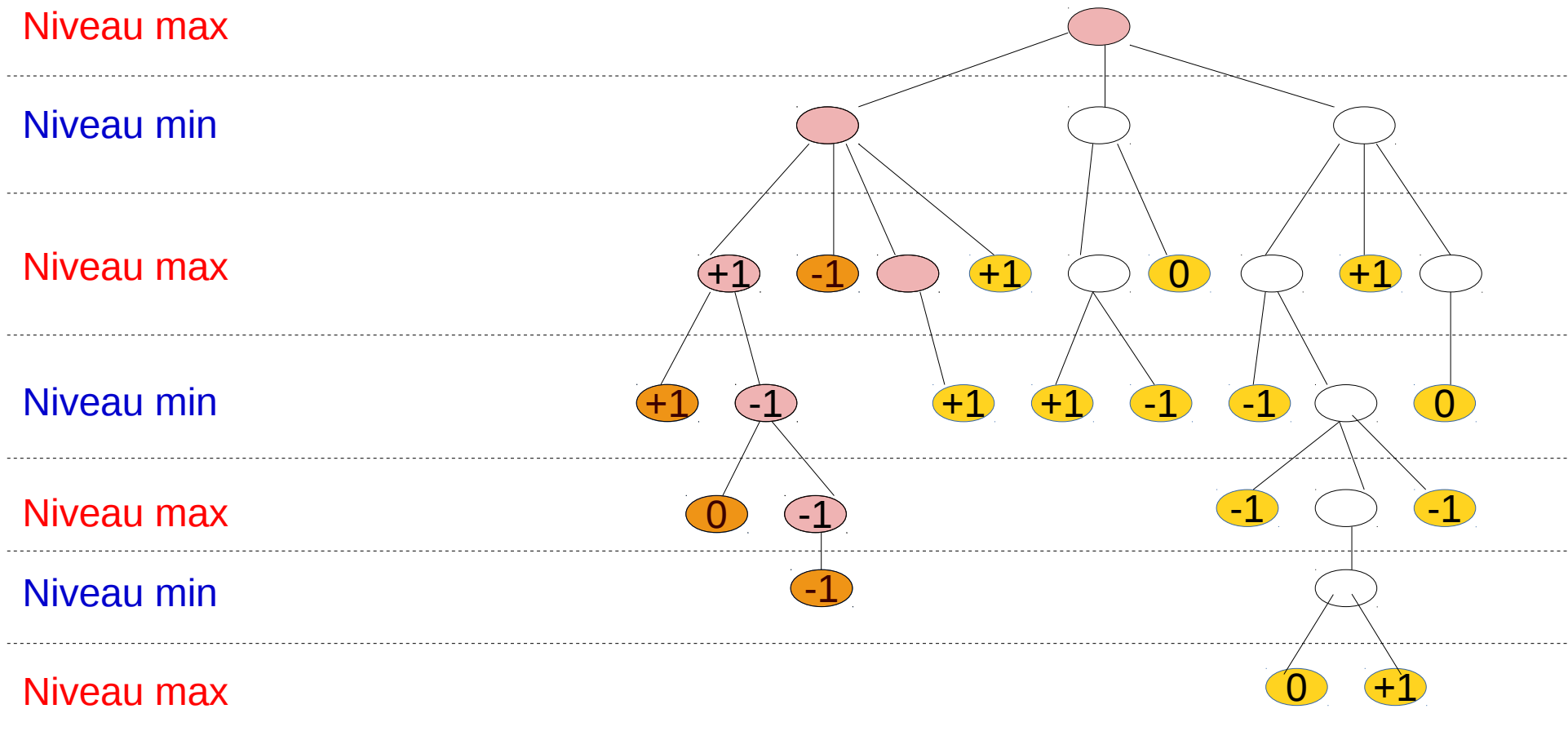
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



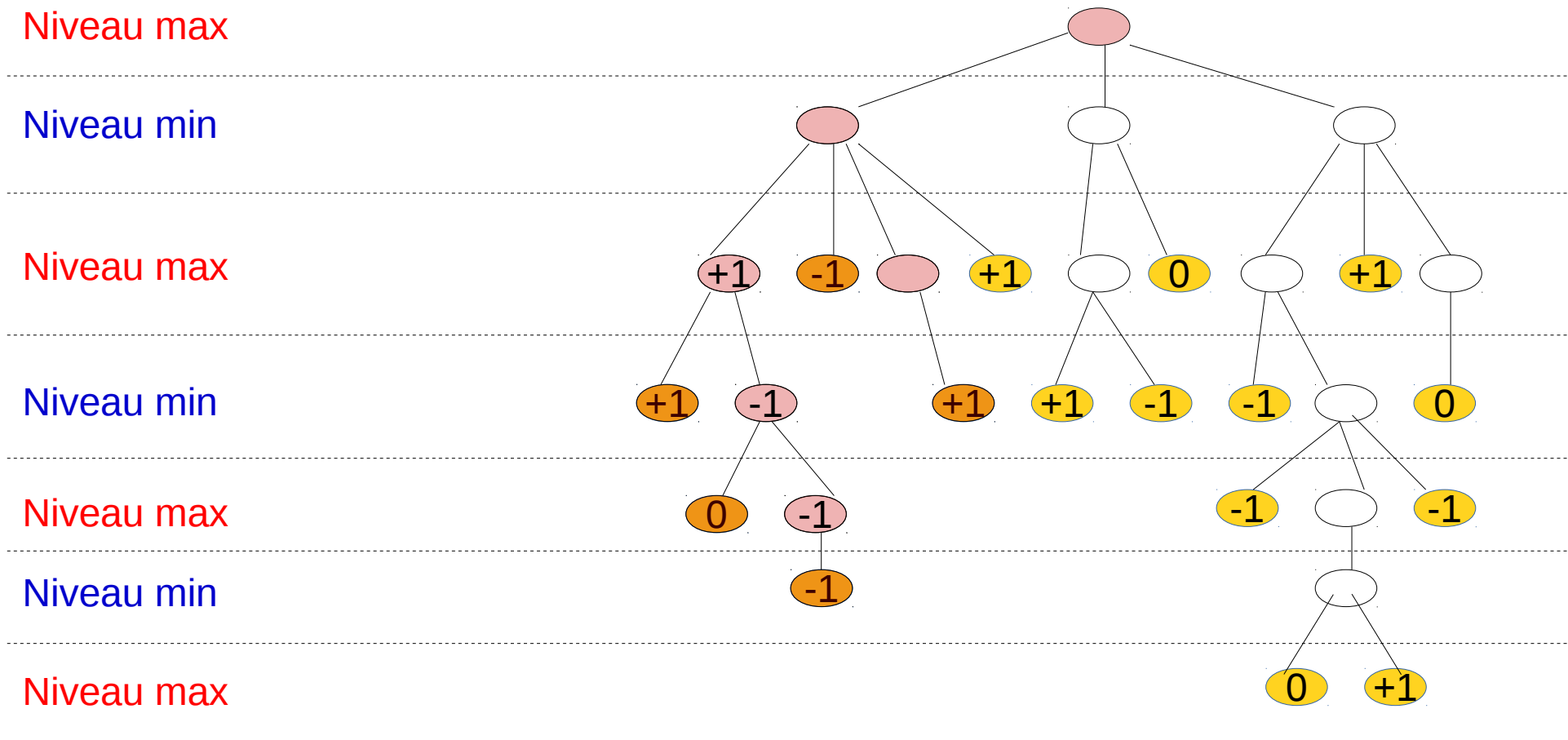
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



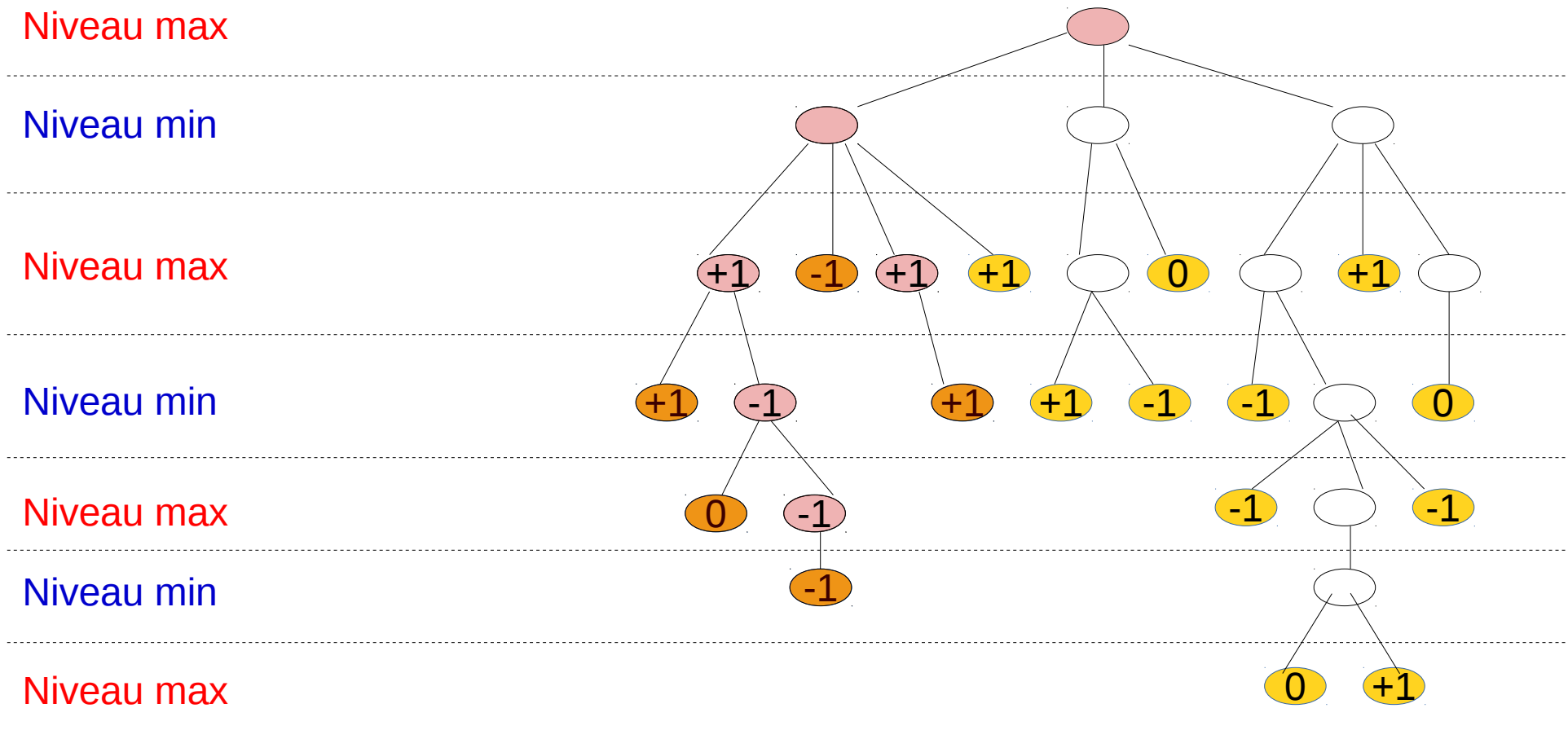
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



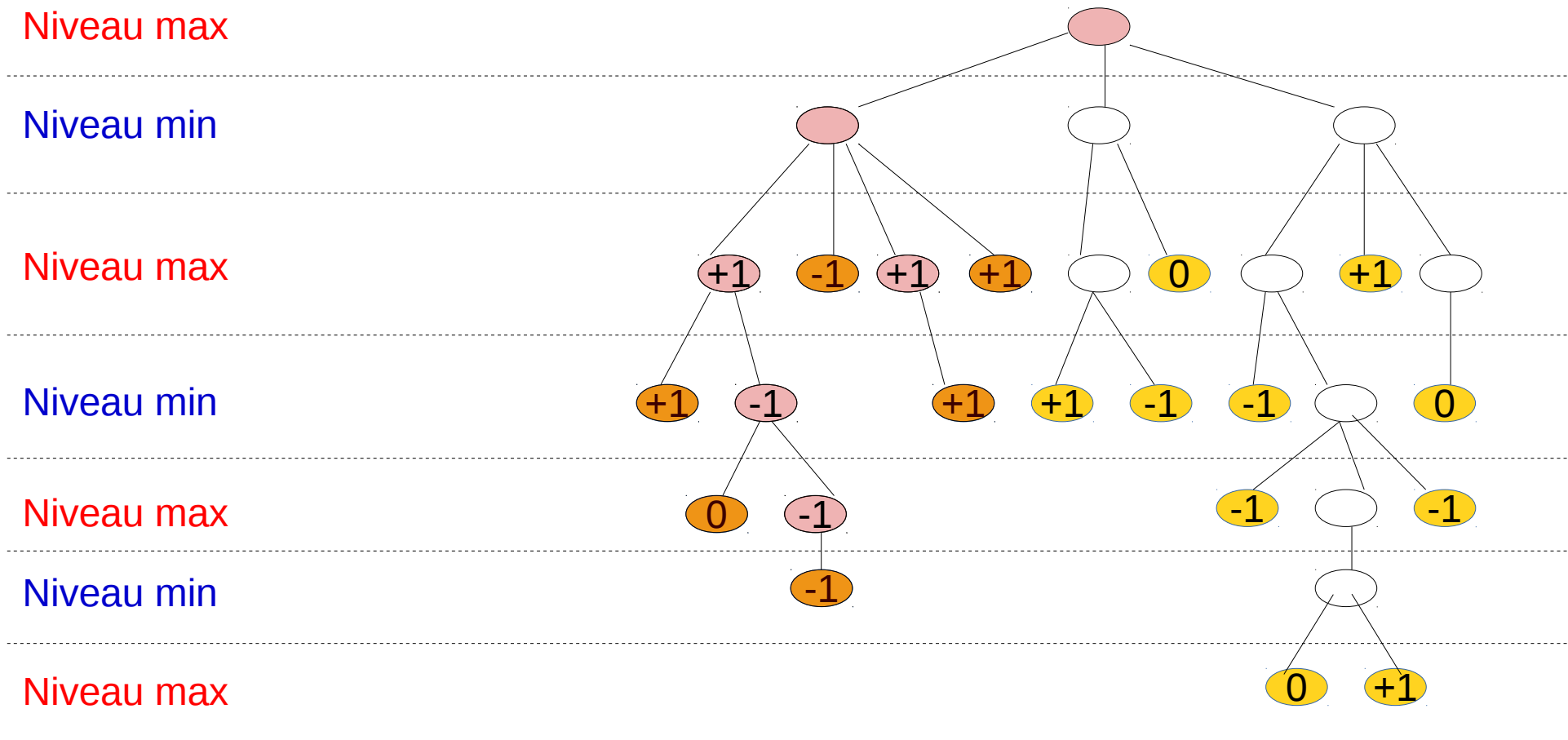
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



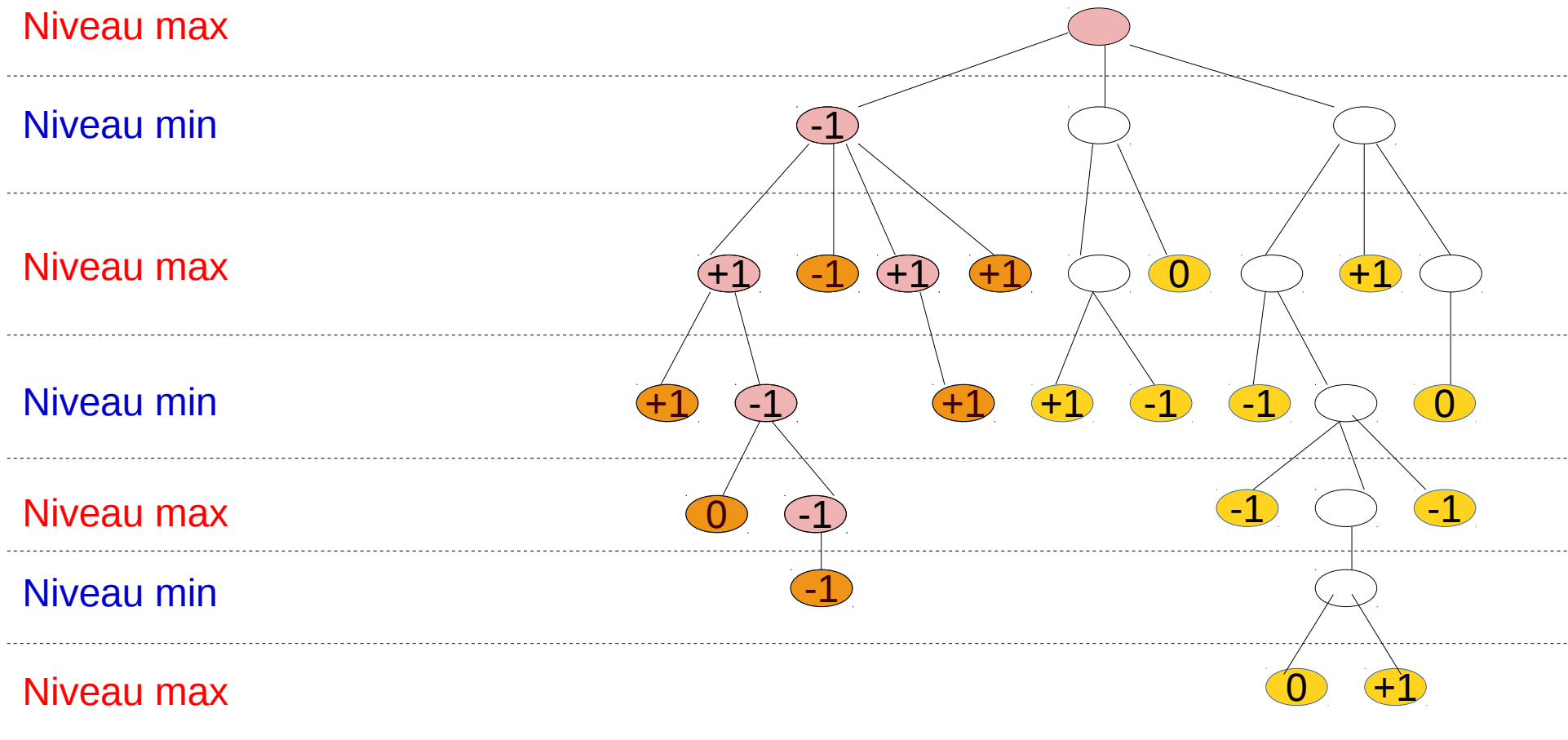
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



MinMax Parcours en profondeur (type post-ordre)

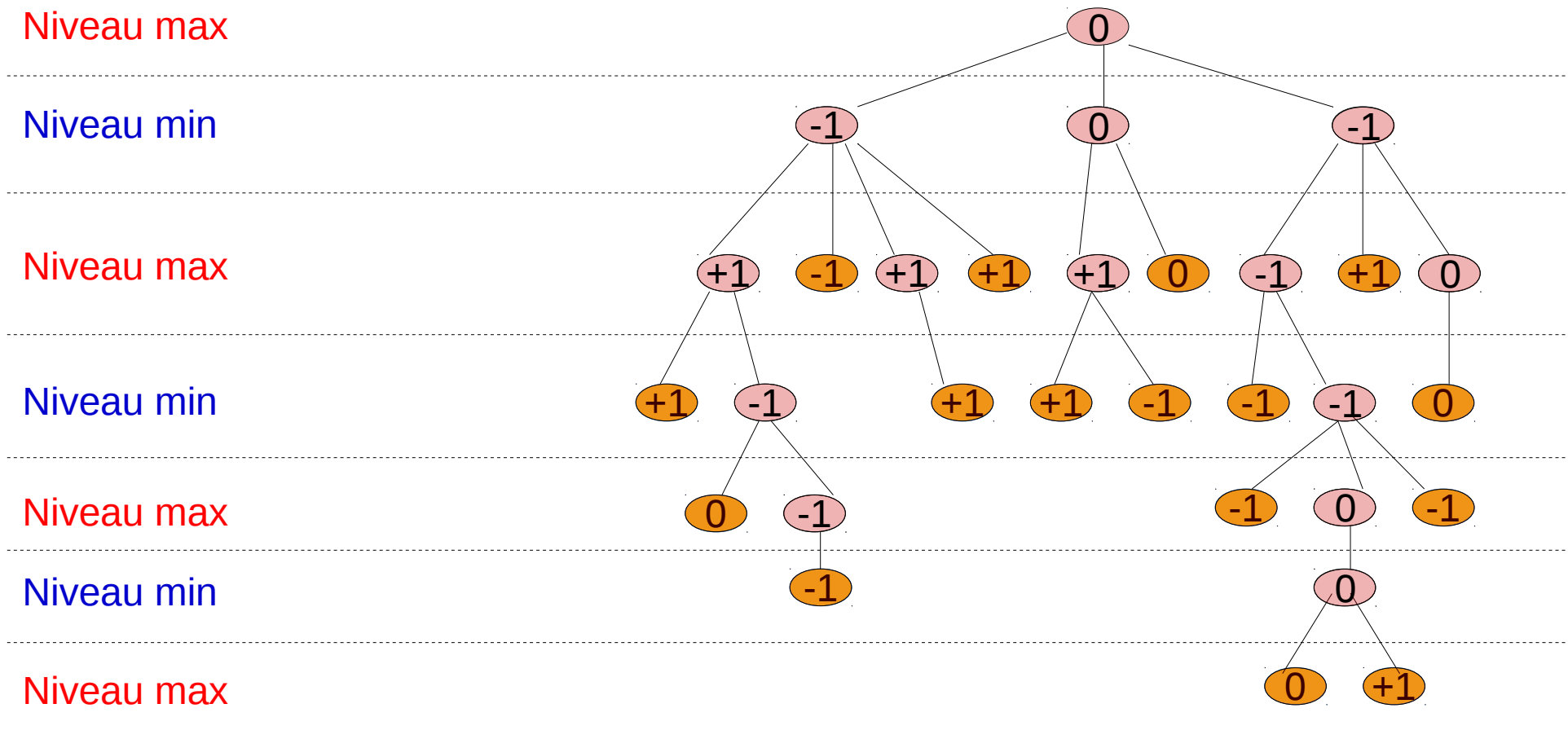
si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)



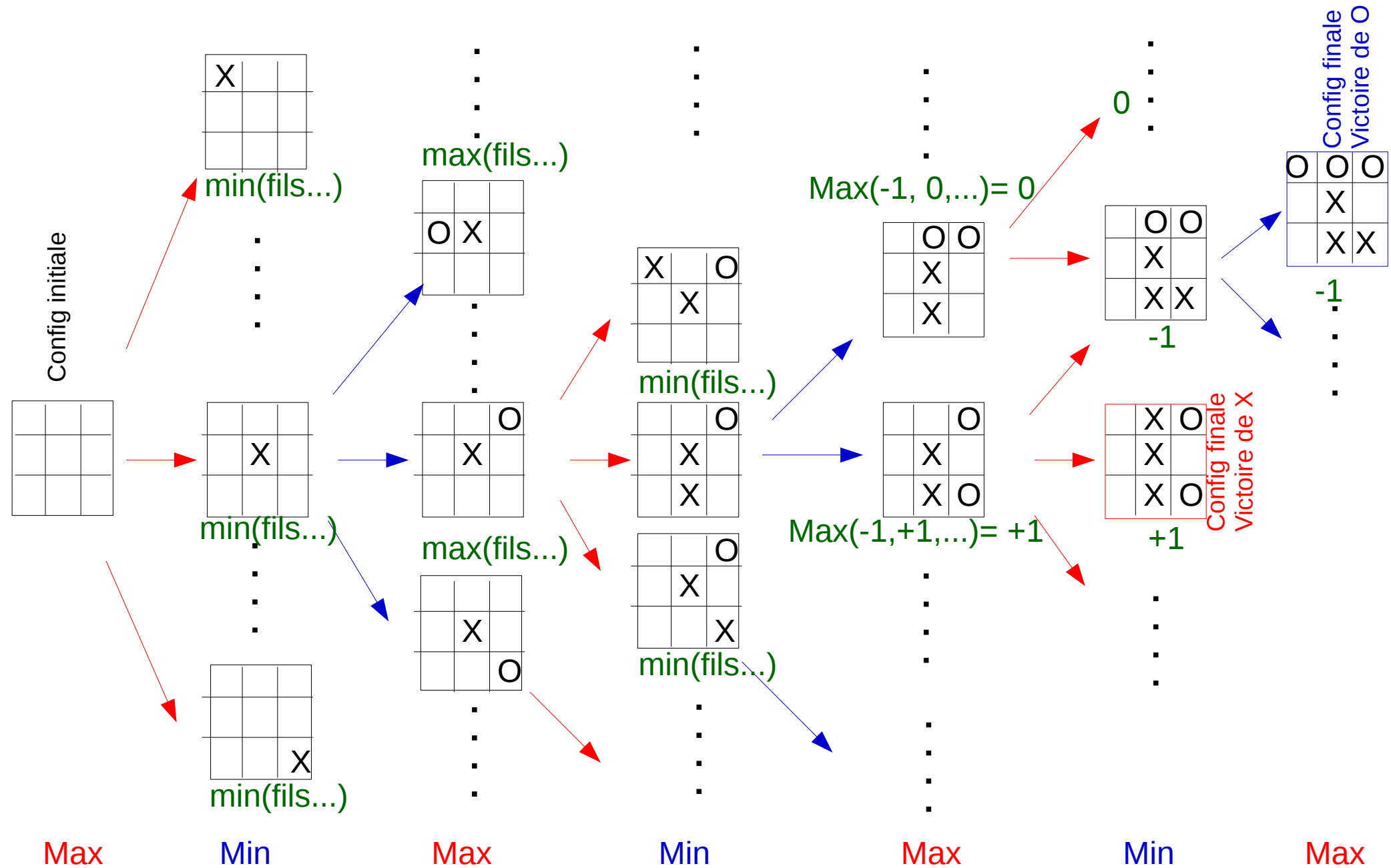
MinMax Parcours en profondeur (type post-ordre)

si le nœud courant est 'maximisant', sa valeur sera le max de ses fils
si le nœud courant est 'minimisant', sa valeur sera le min de ses fils
si le nœud courant est terminal, sa valeur sera donnée par coût(J)

Résultat final du parcours récursif



Espace de recherche (jeu du Tic-Tac-Toe)



Min-Max avec limitation de profondeur

Si l'espace de recherche est trop grand :

Limiter la hauteur d'exploration à une certaine profondeur et

Utiliser des **fonctions d'estimations** pour évaluer les nœuds de la frontière

MinMax(J, Mode, hauteur)

Si (J est une feuille) Retourner (coût(J))

Sinon

Si (hauteur = 0) Retourner(**Estimation(J)**)

Sinon

Si (Mode = 'max') Val $\leftarrow -\infty$ Sinon Val $\leftarrow +\infty$ Fsi

Générer_tous_les_successeurs(J)

Pour chaque successeur K de J :

 Si (Mode = 'max')

 Val \leftarrow Max(Val, **MinMax**(K, 'min', hauteur - 1))

 Sinon

 Val \leftarrow Min(Val, **MinMax**(K, 'max', hauteur - 1))



 Fsi

FP

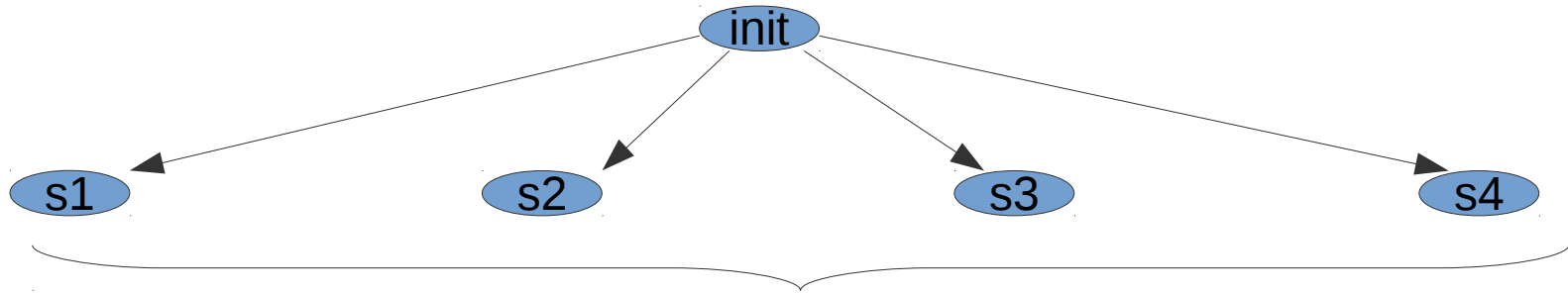
Retourner Val

Fsi

Déroulement d'une partie

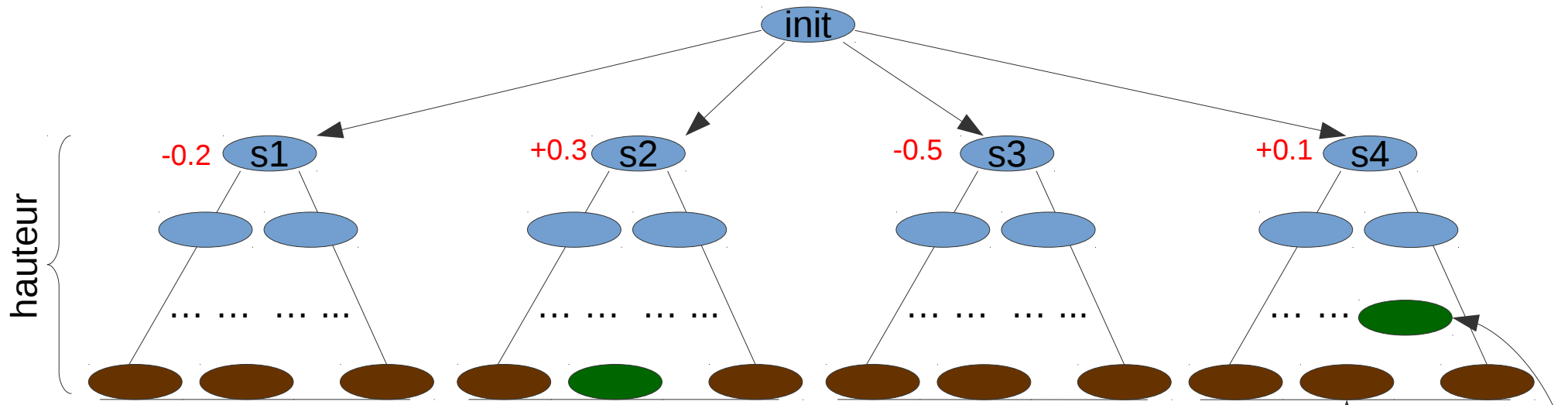
A partir d'une configuration initiale  

Déroulement d'une partie



Générer tous les successeurs
s1, s2, ...

Déroulement d'une partie



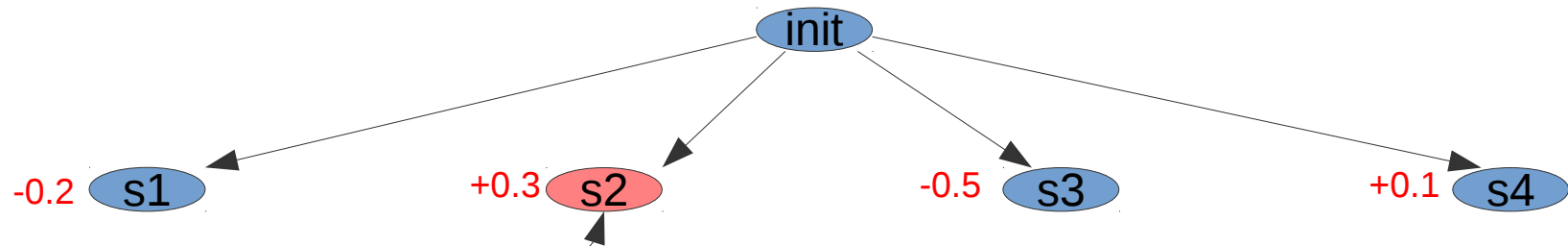
Lancer **MinMax** sur tous les successeurs s1, s2, ... (avec une hauteur spécifiée)

La fonction Estimation(...) est utilisée pour les **nœuds de la frontière**

La fonction Coût(...) pour les éventuels **nœuds terminaux (feuilles)**

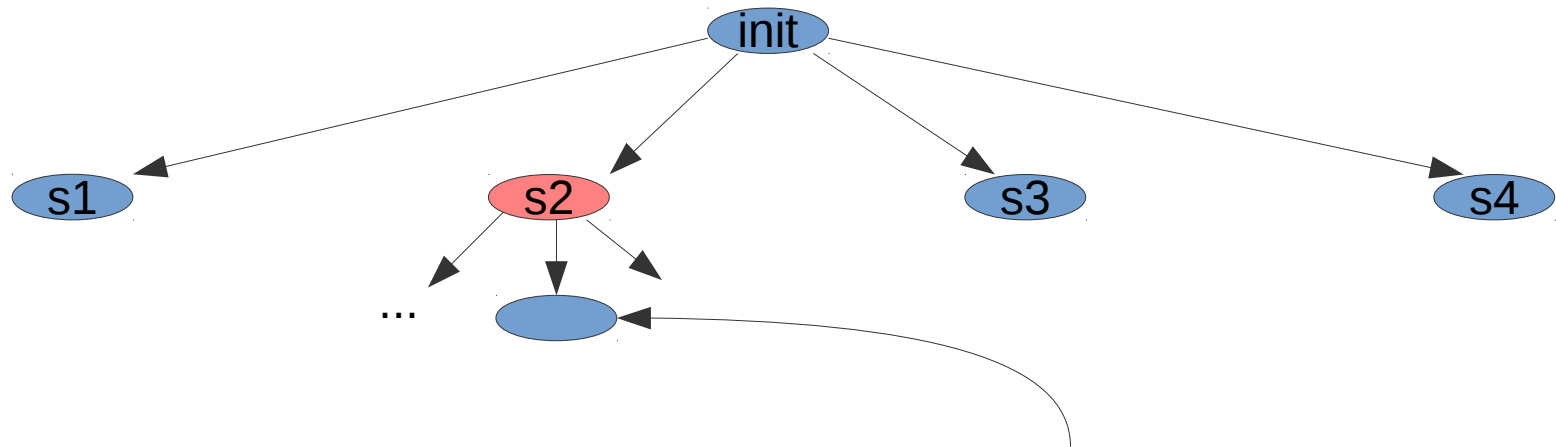
Cela permet d'avoir **une valeur** (estimée) pour chaque successeur s1, s2, ...

Déroulement d'une partie



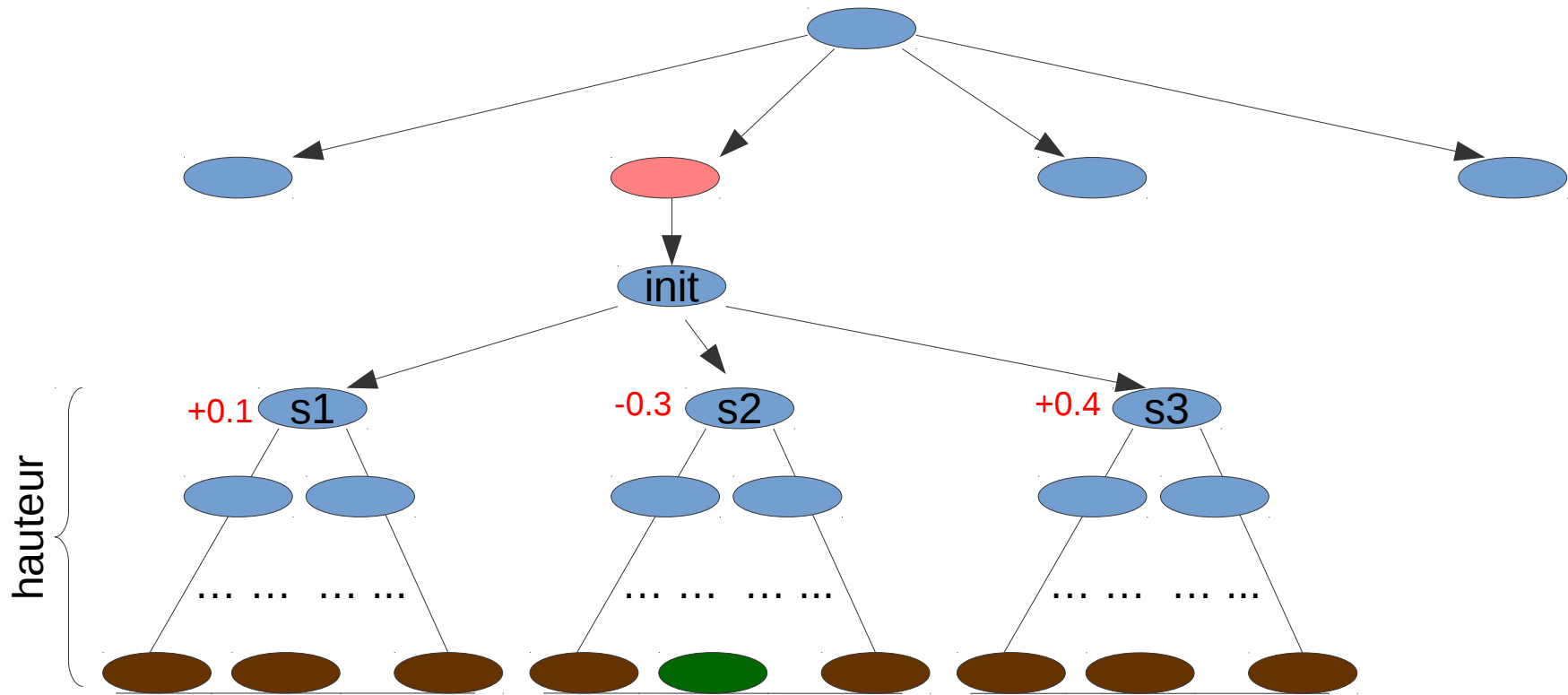
Le **coup à jouer** représente le successeur ayant la plus grande (ou la plus petite) des valeurs retournées (selon le type du joueur Maximisant ou Minimisant)

Déroulement d'une partie



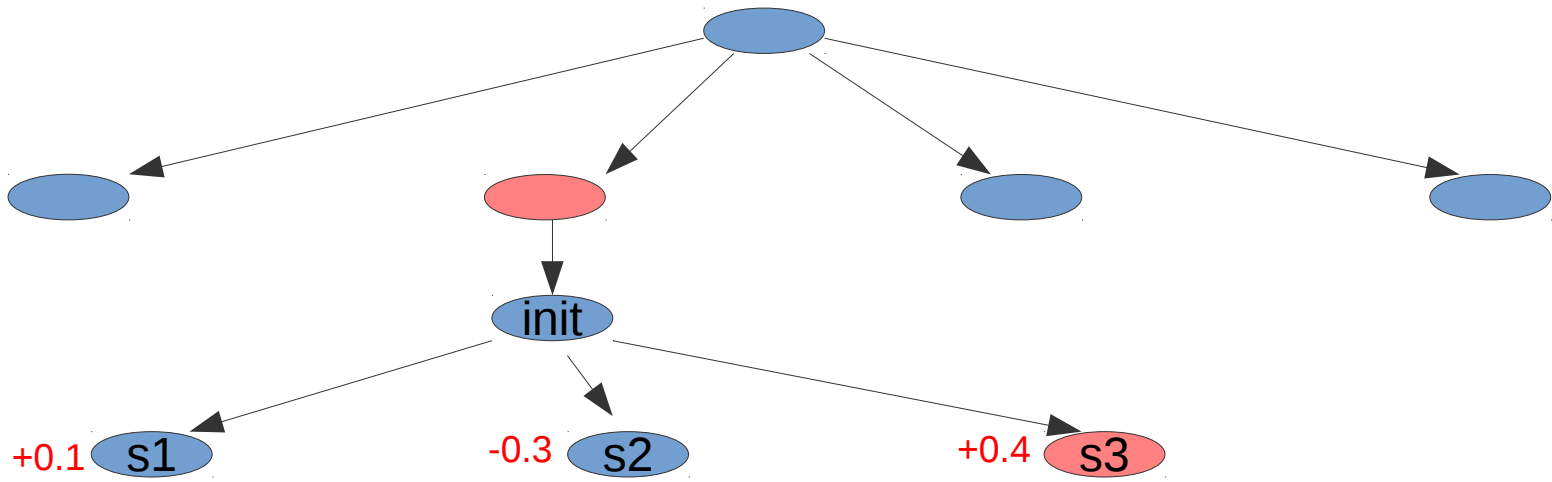
La main est donnée pour le joueur adverse qui **choisira un des successeurs** du coup joué.

Déroulement d'une partie



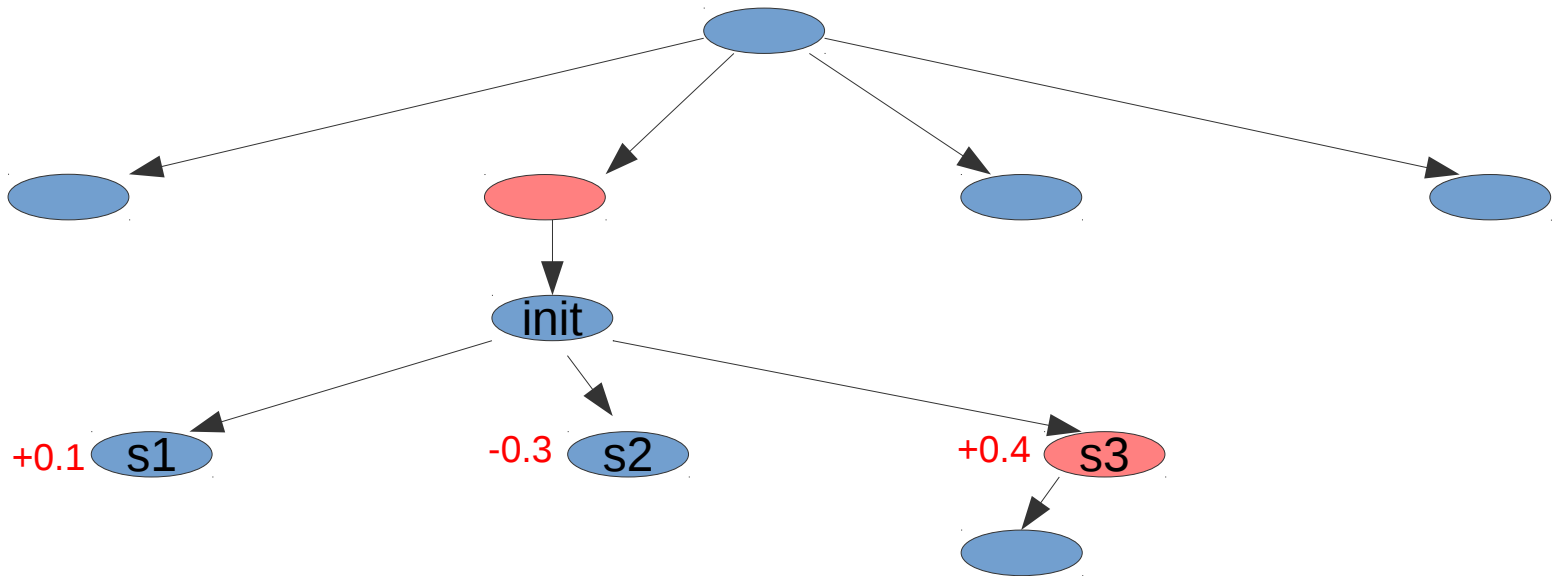
A partir de là on refait une nouvelle itération en prenant comme nœud initial le nœud choisi par l'adversaire ...

Déroulement d'une partie



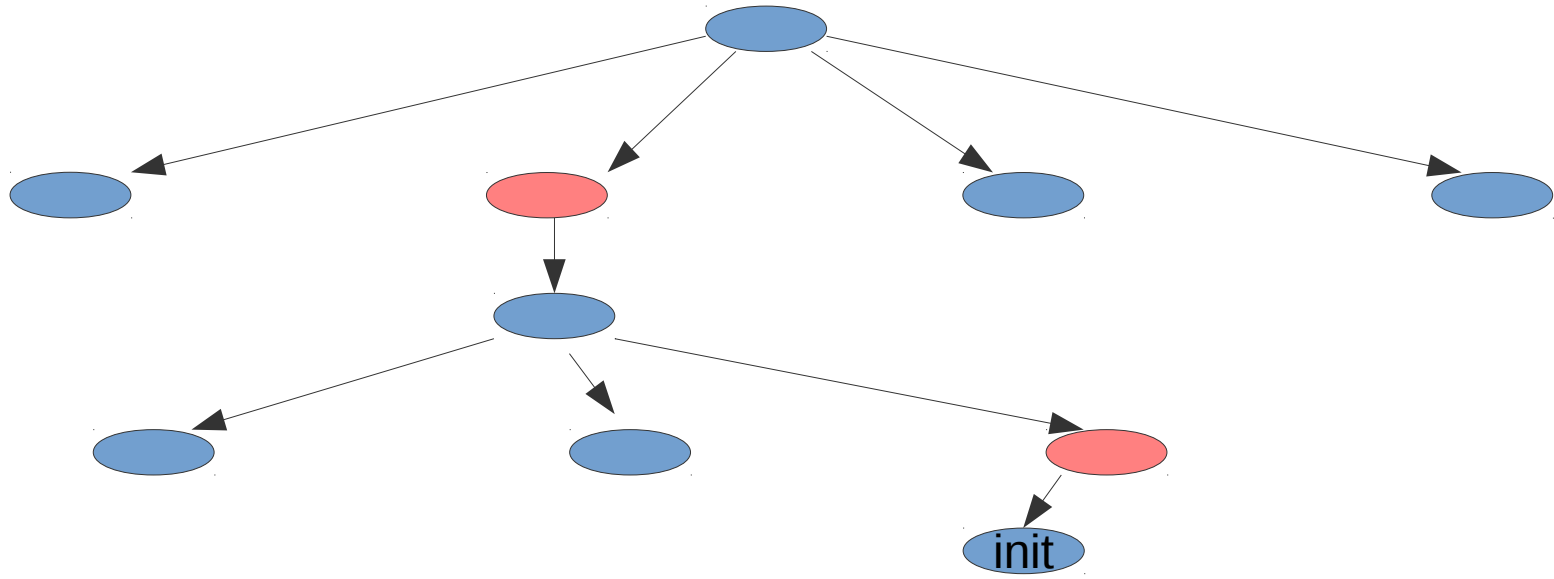
A partir de là on refait une nouvelle itération en prenant comme nœud initial le nœud choisi par l'adversaire ...

Déroulement d'une partie

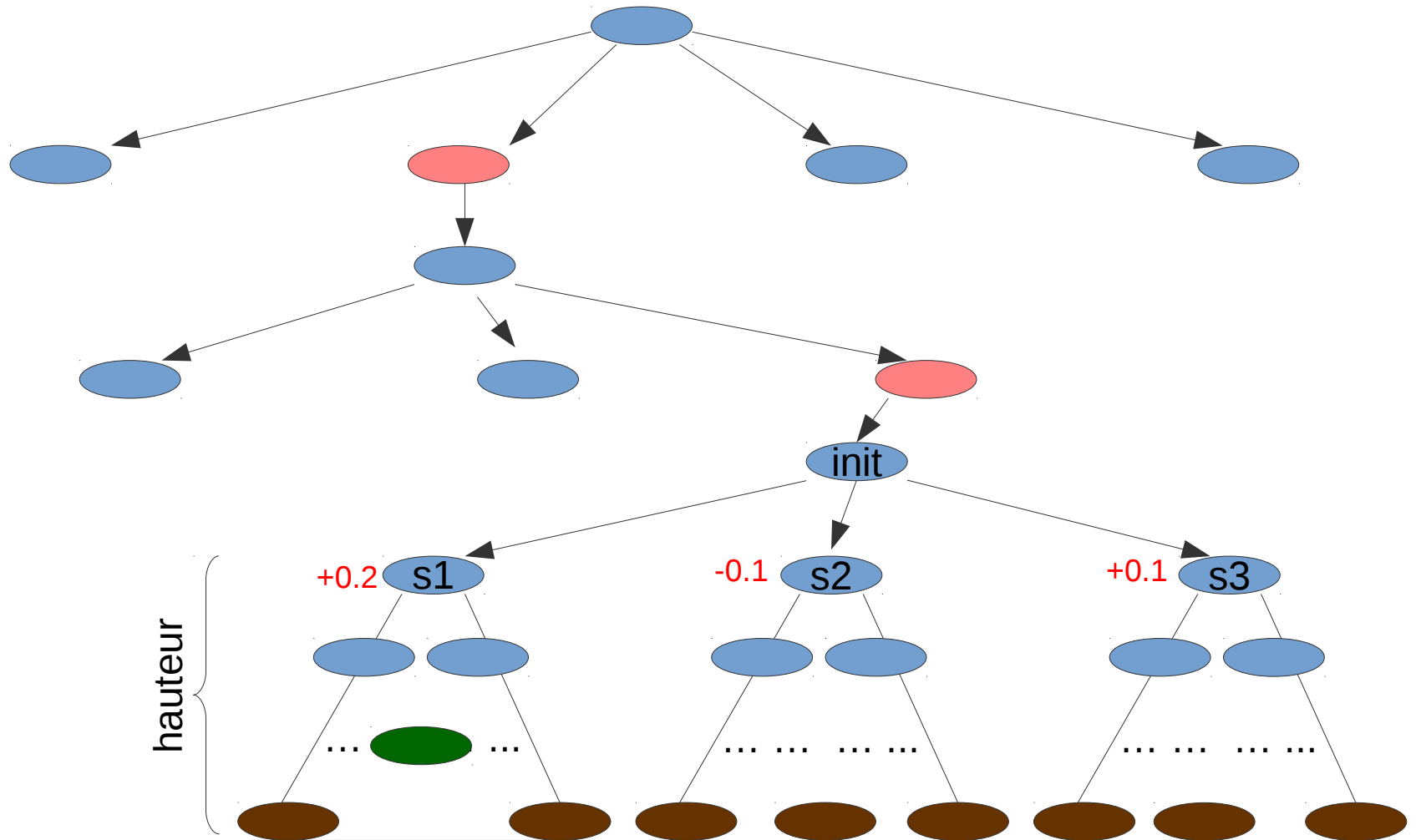


A partir de là on refait une nouvelle itération en prenant comme nœud initial le nœud choisi par l'adversaire ...

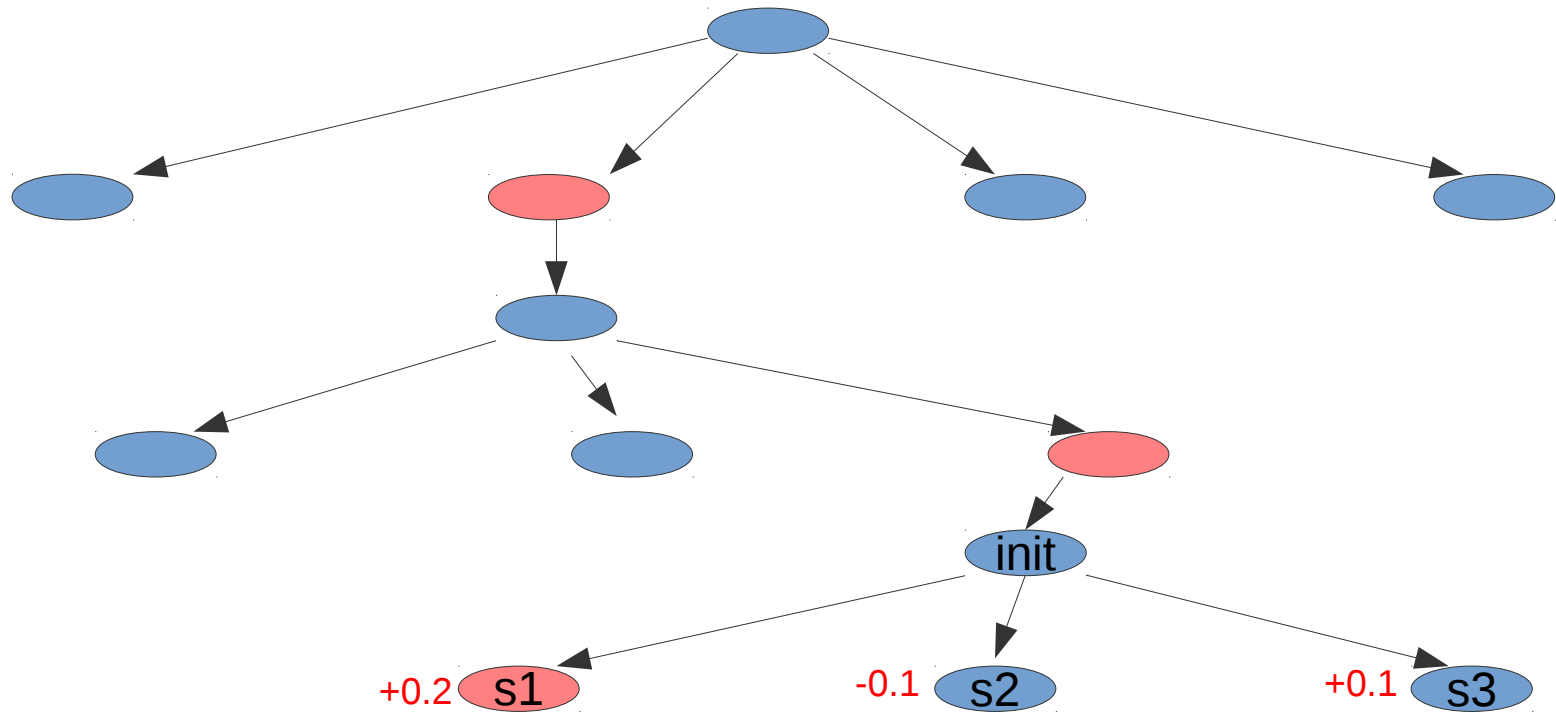
Déroulement d'une partie



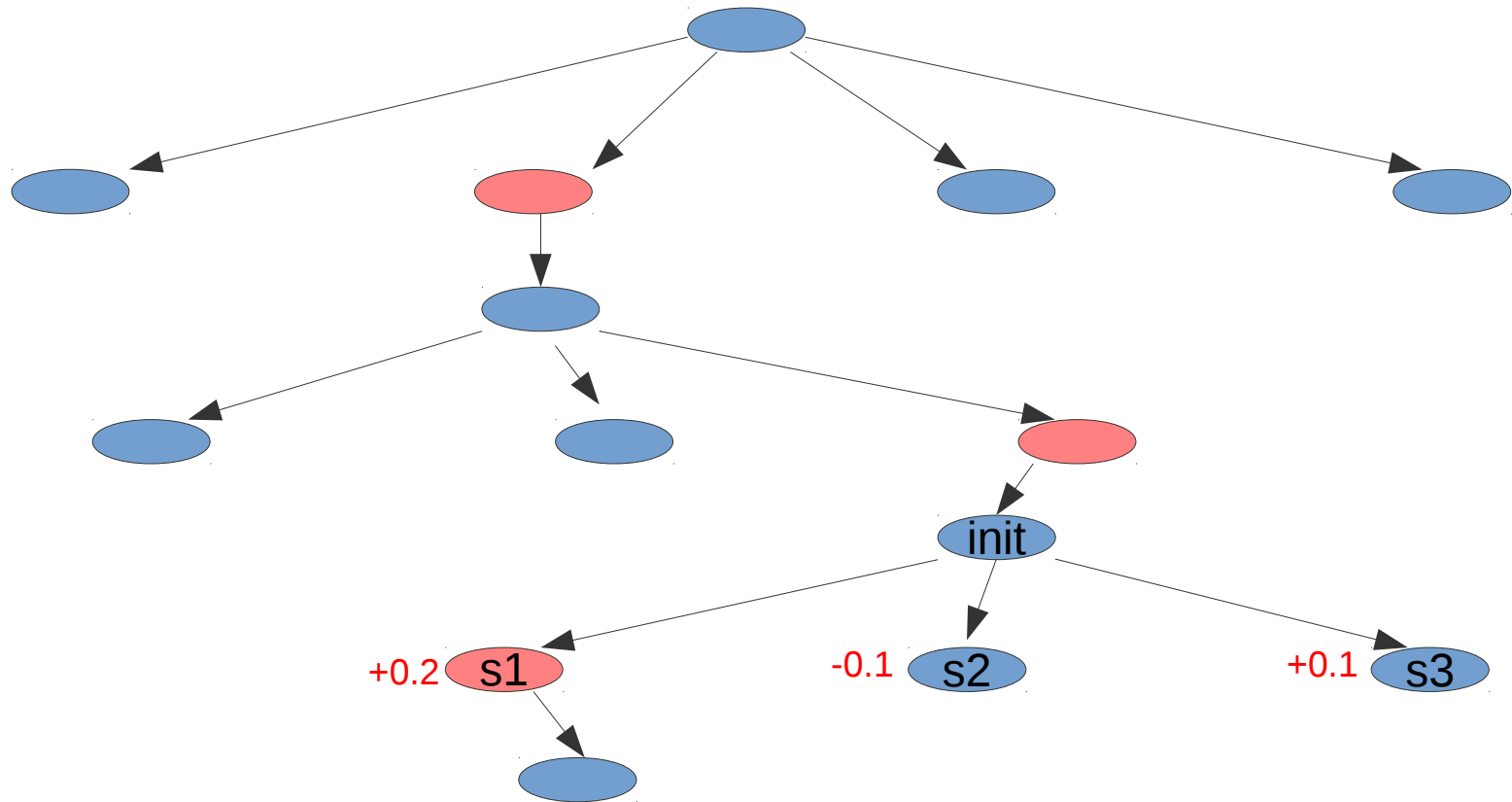
Déroulement d'une partie



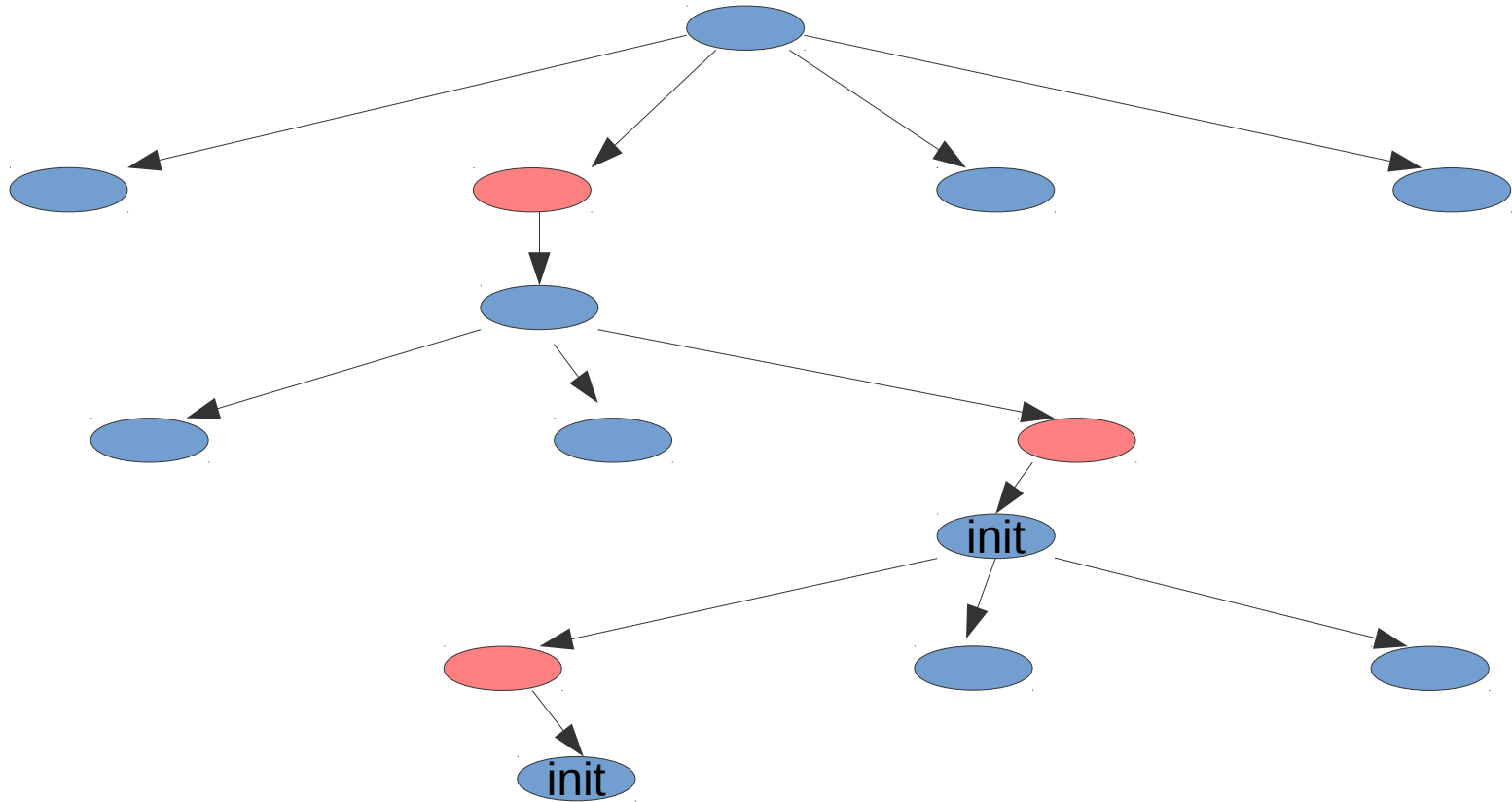
Déroulement d'une partie



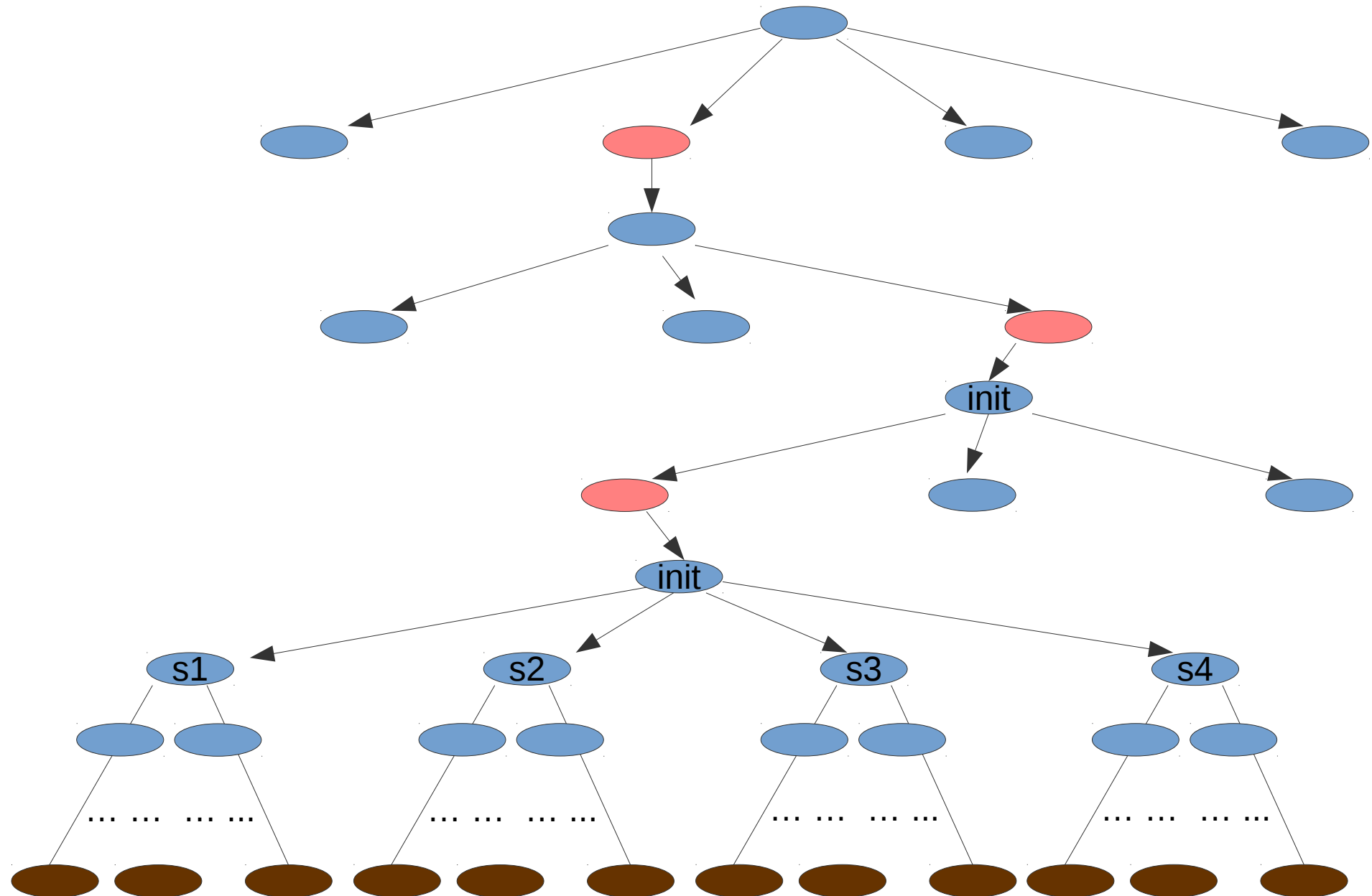
Déroulement d'une partie



Déroulement d'une partie



Déroulement d'une partie



Les états explorés durant une partie

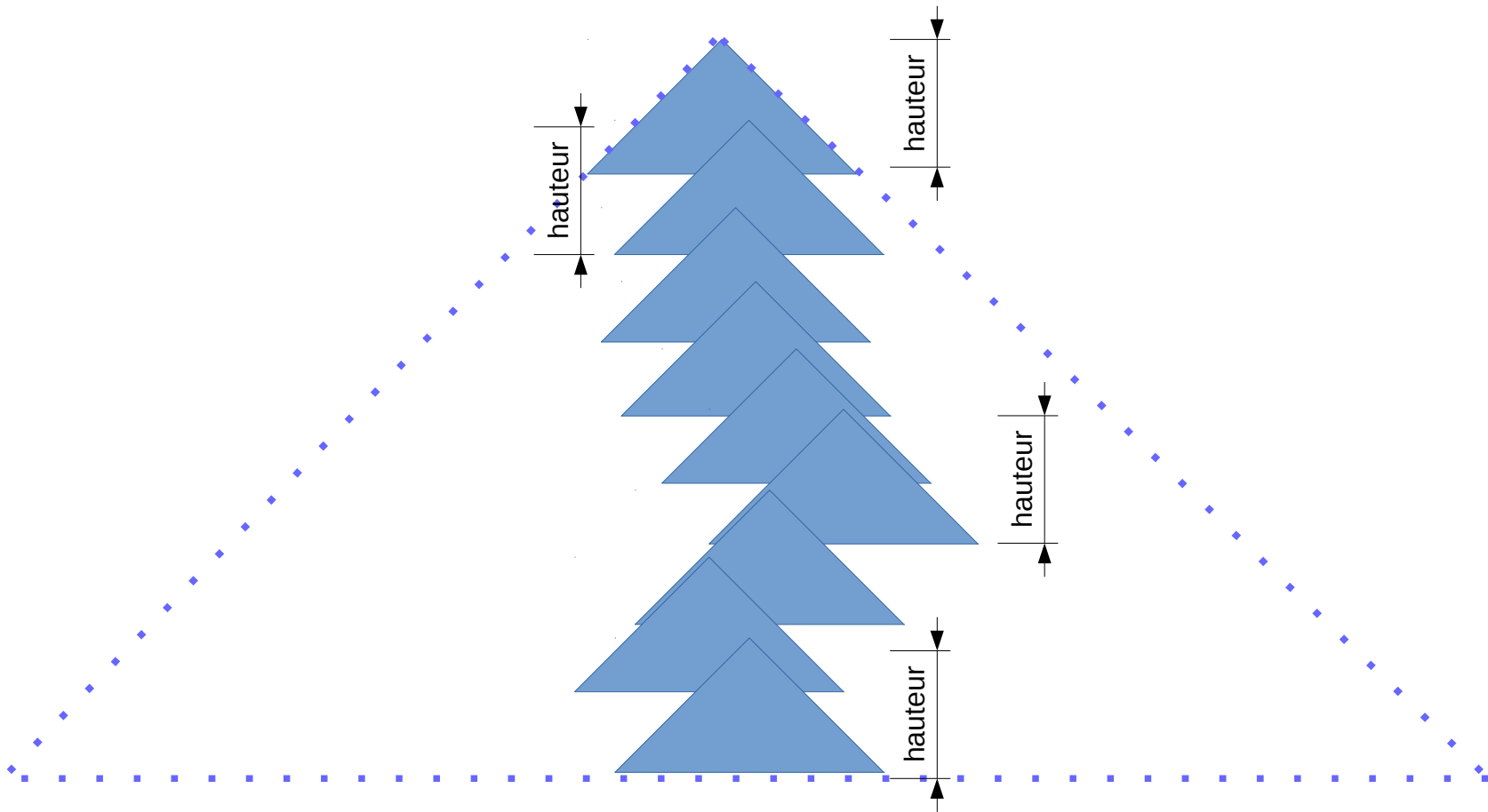


Schéma général d'une partie avec un adversaire de type 'min'

$J \leftarrow \text{Configuration_initiale} ;$

$\text{Afficher}(J) ;$

TQ (J n'est pas une configuration_finale)

$\text{Générer_successeurs}(J, 'max') \rightarrow s1, s2, \dots sk$

$v1 \leftarrow \text{MinMax}(s1, 'min', \text{hauteur}) ;$

$v2 \leftarrow \text{MinMax}(s2, 'min', \text{hauteur}) ;$

 ...

$vk \leftarrow \text{MinMax}(sk, 'min', \text{hauteur}) ;$

Soit S le meilleur successeur de J (c-a-d le $\text{Max}\{v1, v2, \dots vk\}$)

$\text{Afficher}(S) ;$

 Si (S n'est pas une configuration_finale)

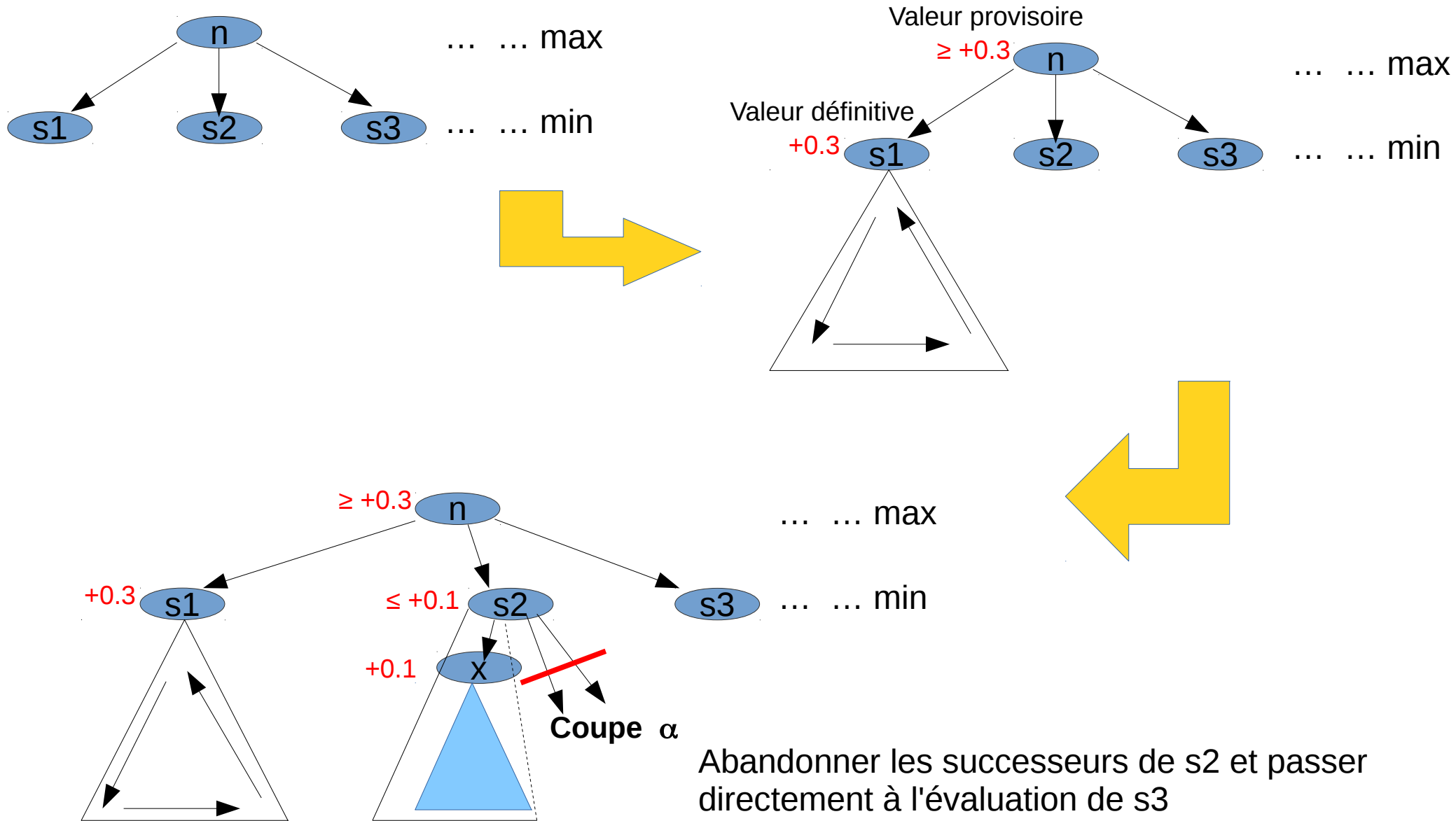
Récupérer le coup joué par l'adversaire, soit S la nouvelle config

 Fsi ;

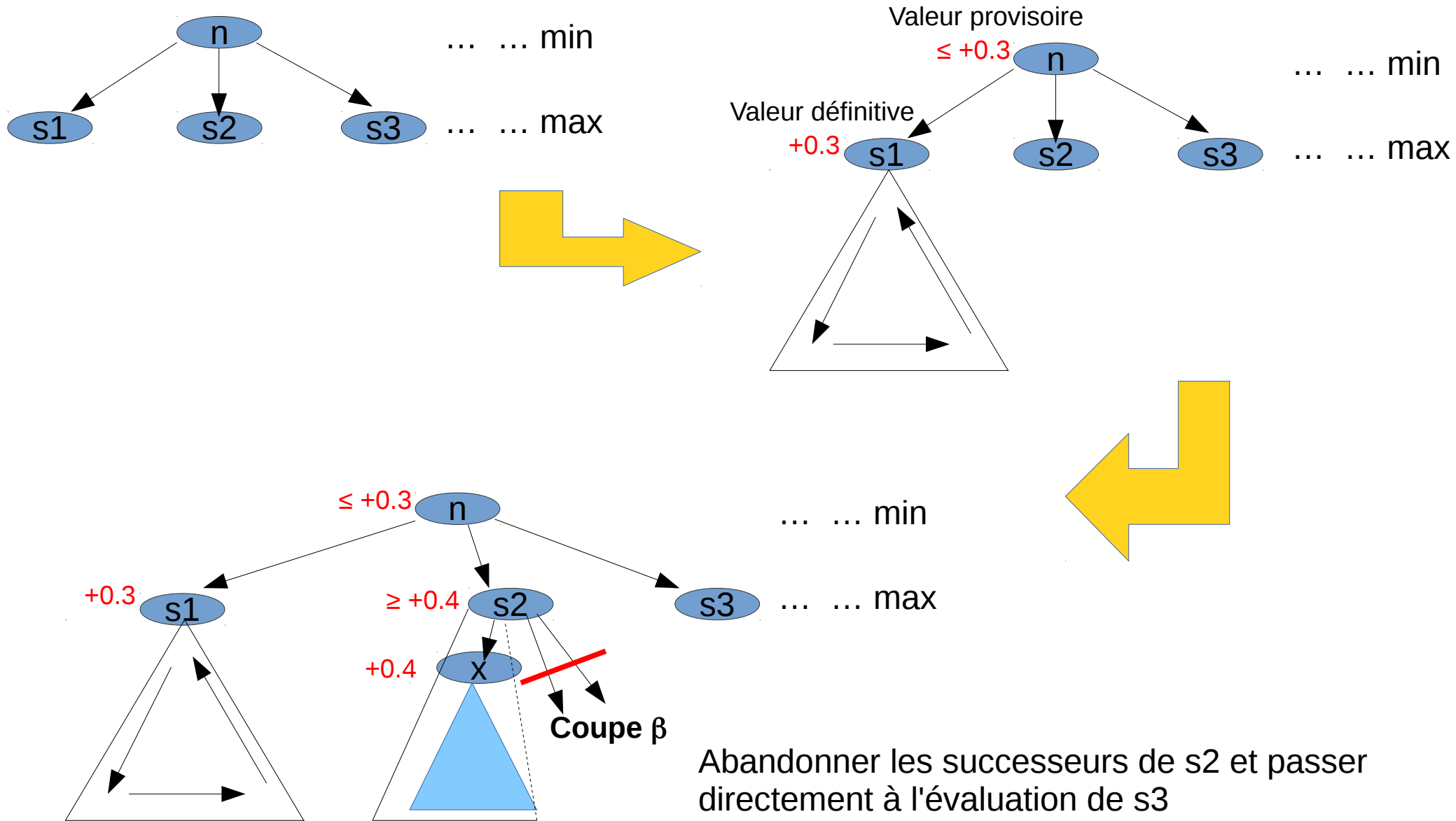
$J \leftarrow S ;$

FTQ

Élagage (coupes) α/β



Élagage (coupes) α/β



Min-Max avec élagage α/β

```
MinMaxAB(J, Mode, hauteur, alpha, beta )
  Si (J est une feuille)      Retourner ( coût( J ) )
  Sinon
    Si ( hauteur = 0 )      Retourner( Estimation( J ) )
    Sinon
      Si (Mode = 'max') Val ← alpha  Sinon Val ← beta  Fsi
      Générer_tous_les_successeurs( J ) ;
      Pour chaque successeur K de J :
        Si (Mode = 'max')
          Val ← Max( Val, MinMaxAB(K, 'min', hauteur – 1, Val, beta ) ;
          Si ( Val ≥ beta )
            Retourner beta ; // Coupe de type  $\beta$ 
          Fsi
        Sinon
          Val ← Min( Val, MinMaxAB(K, 'max', hauteur – 1, alpha, Val ) ;
          Si ( Val ≤ alpha )
            Retourner alpha ; // Coupe de type  $\alpha$ 
          Fsi
        Fsi
      FP
      Retourner Val
    Fsi
  Fsi
```

Min-Max avec élagage α/β

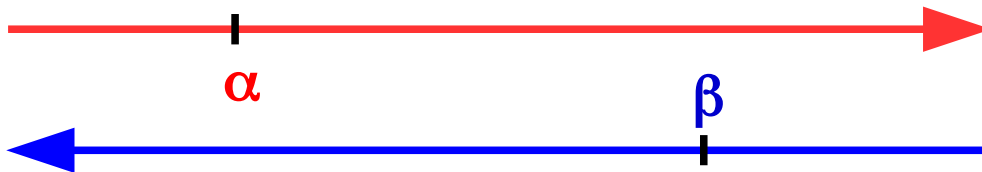
Les paramètres α et β représentent un intervalle d'intérêt. Tant que la valeur provisoire d'un nœud (en cours d'évaluation) reste dans cet intervalle, il y a un intérêt à continuer son évaluation.

L'intervalle $[\alpha, \beta]$ est dynamique

Au départ : $\alpha = -\infty$ et $\beta = +\infty$

A chaque fin d'évaluation d'un nœud de type 'min' (évaluation complète ou coupure), la borne α de son père ('max') peut (éventuellement) **augmenter**.

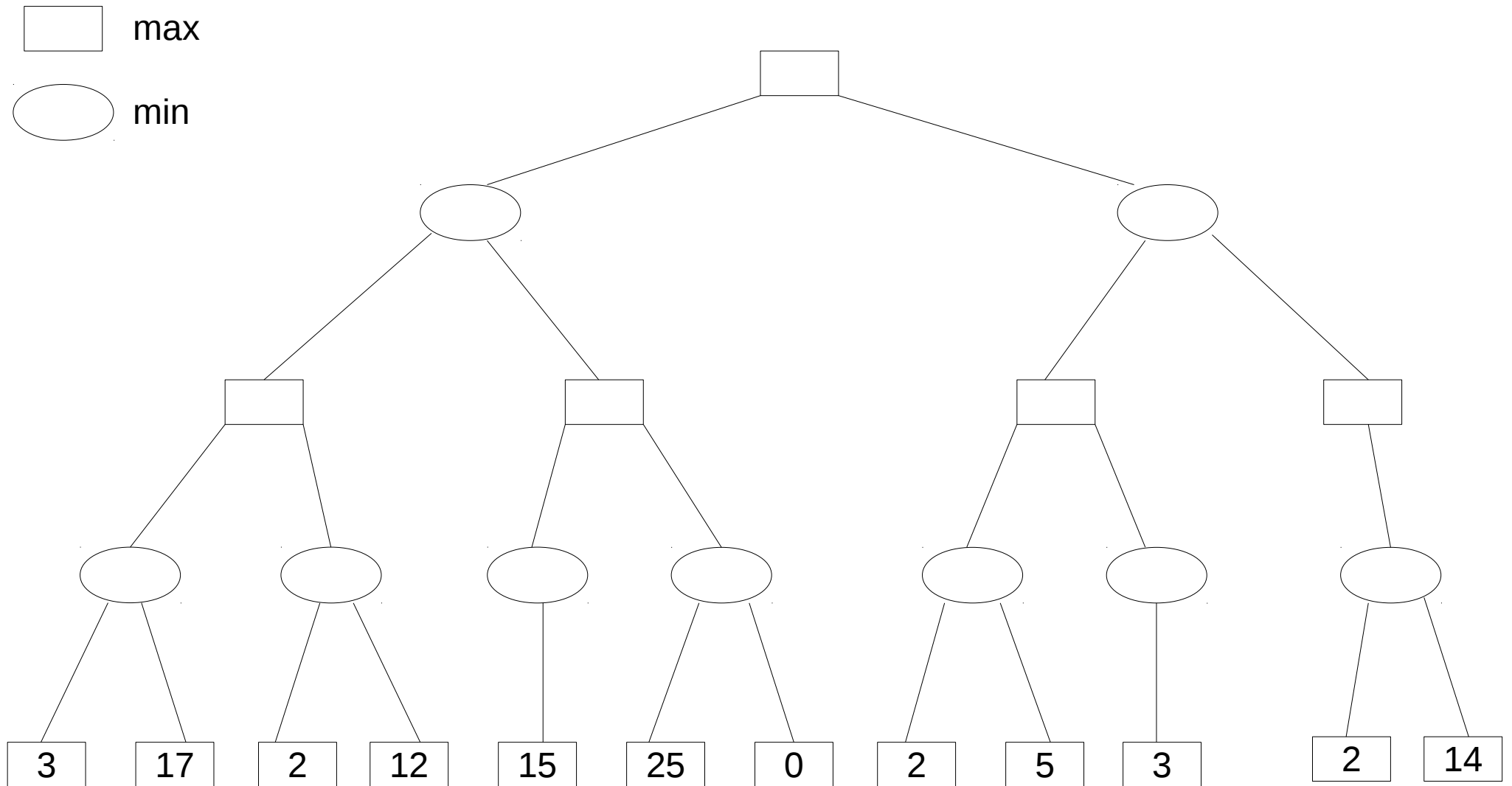
A chaque fin d'évaluation d'un nœud de type 'max' (évaluation complète ou coupure), la borne β de son père ('min') peut (éventuellement) **diminuer**.



Lorsque $\alpha \geq \beta$, il n'y a plus d'intérêt à continuer l'évaluation du nœud courant
→ **coupure** α ou β

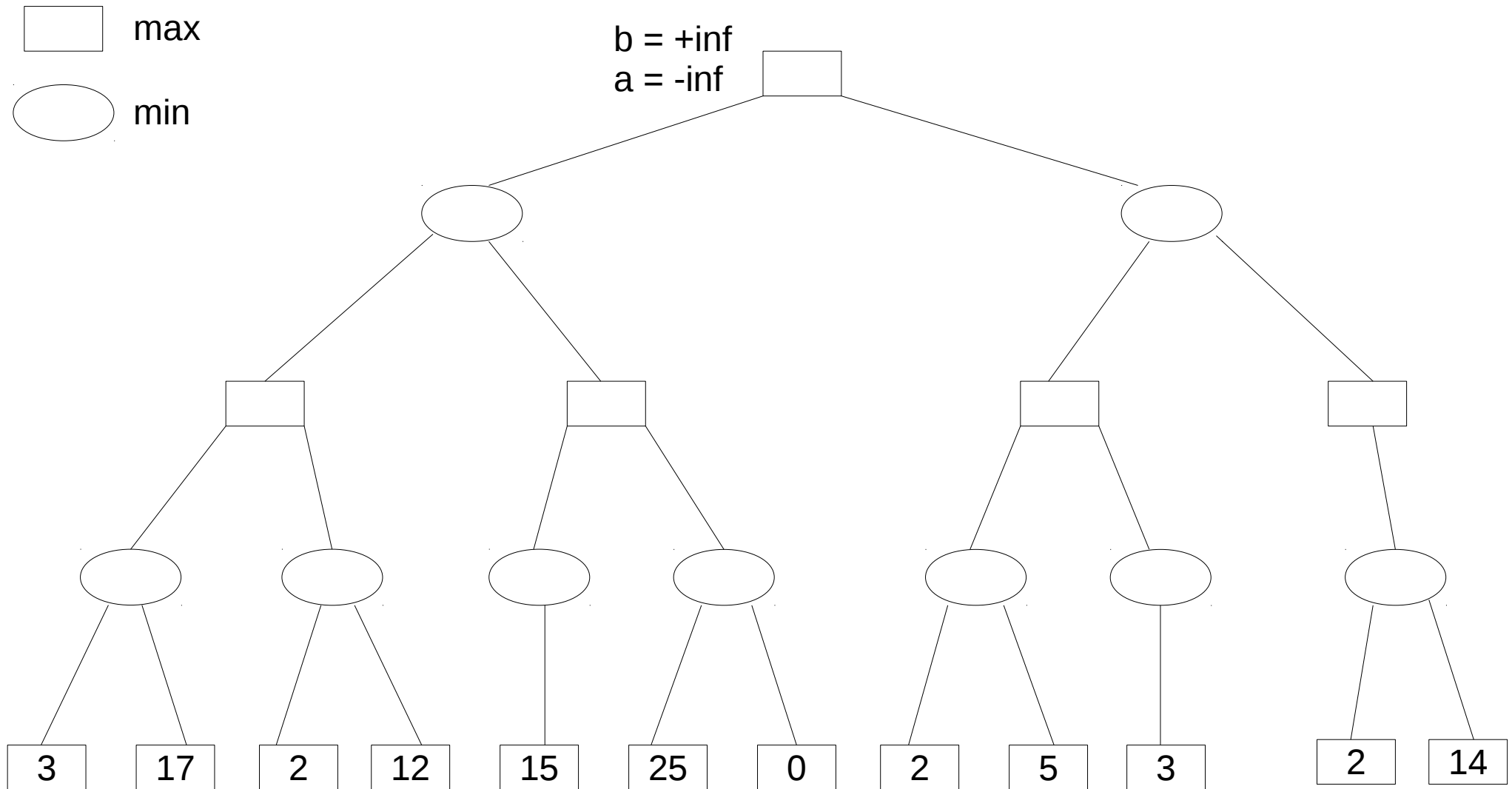
Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

Tiré de: <http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>



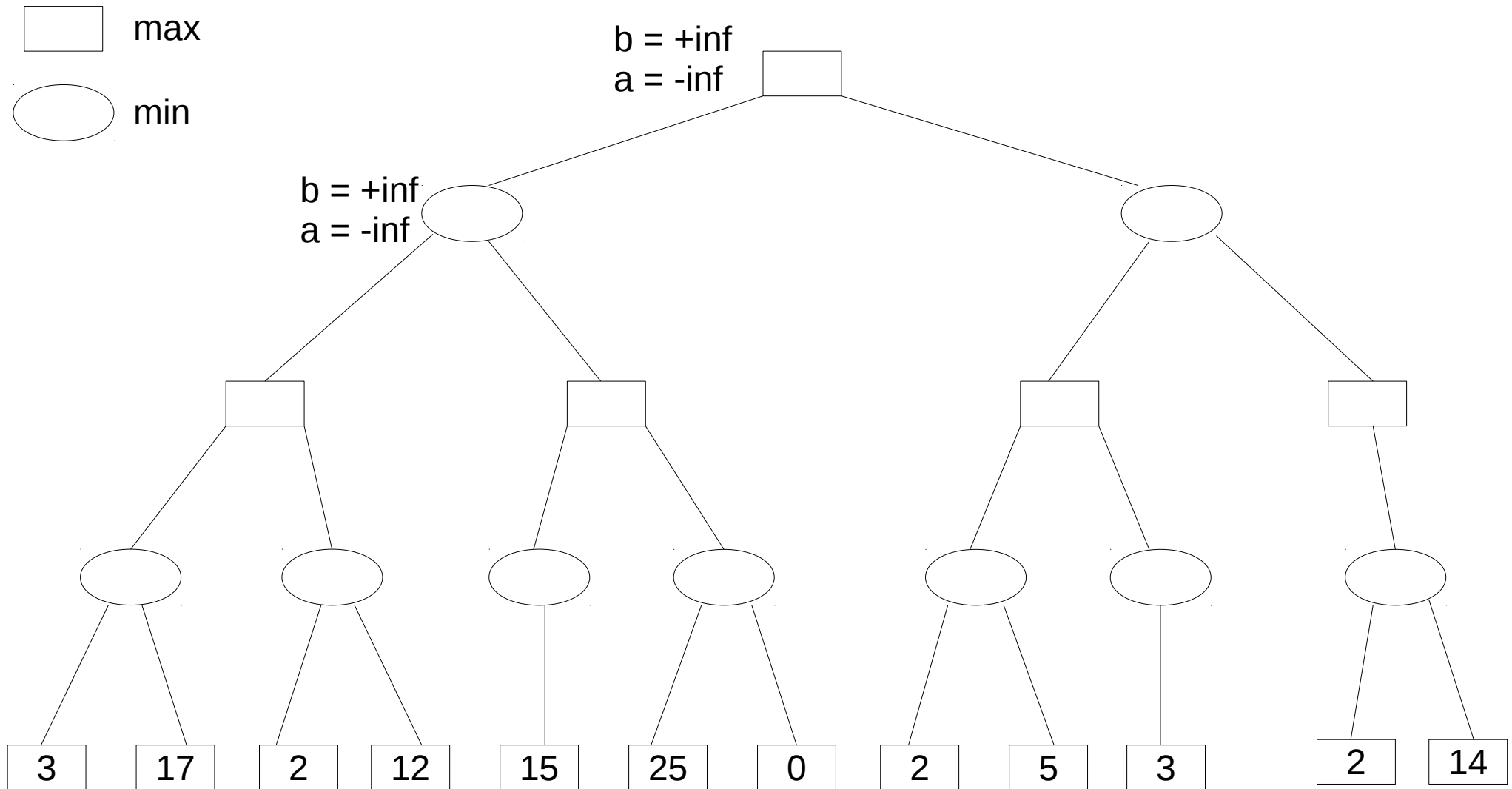
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



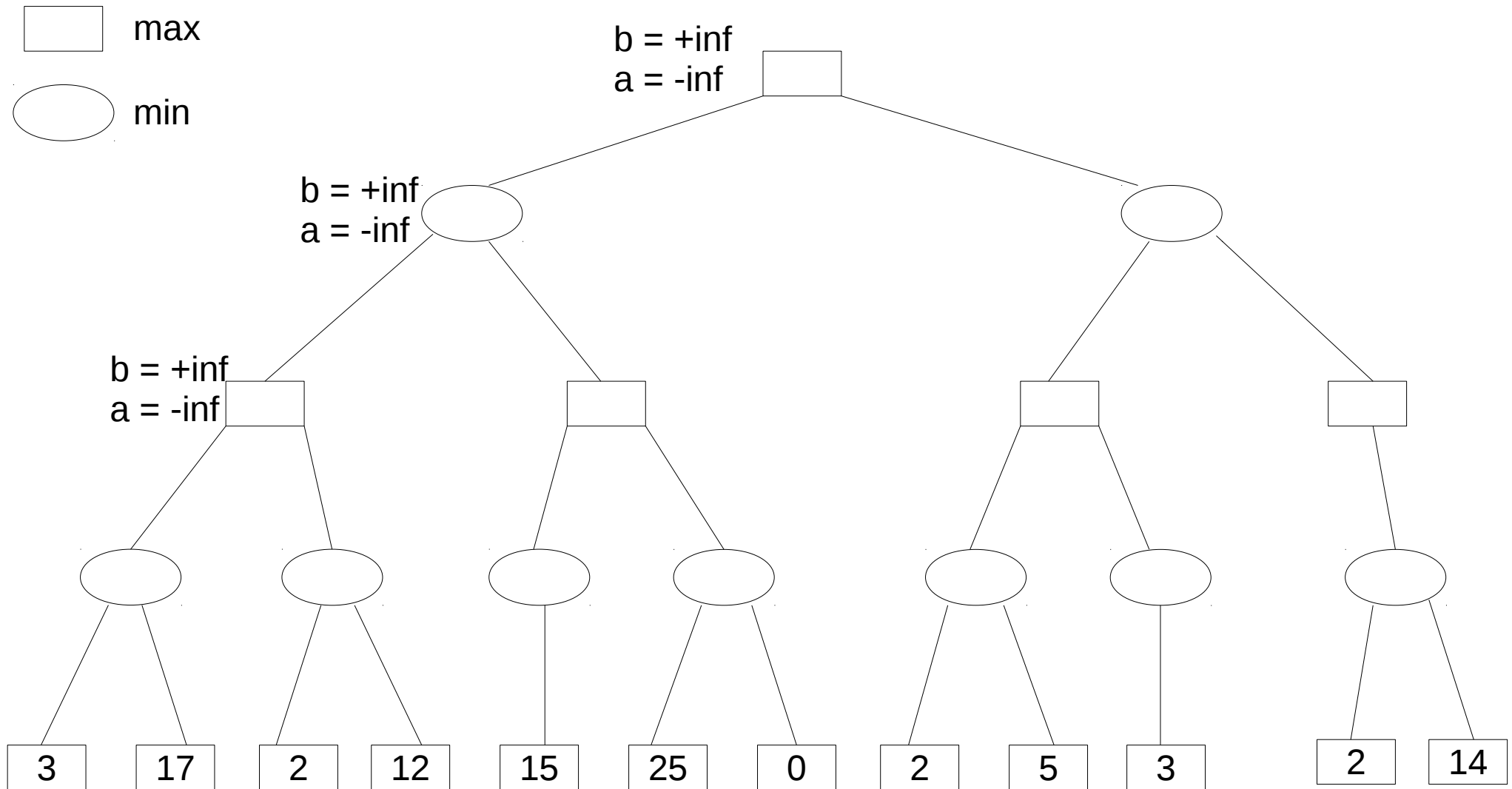
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



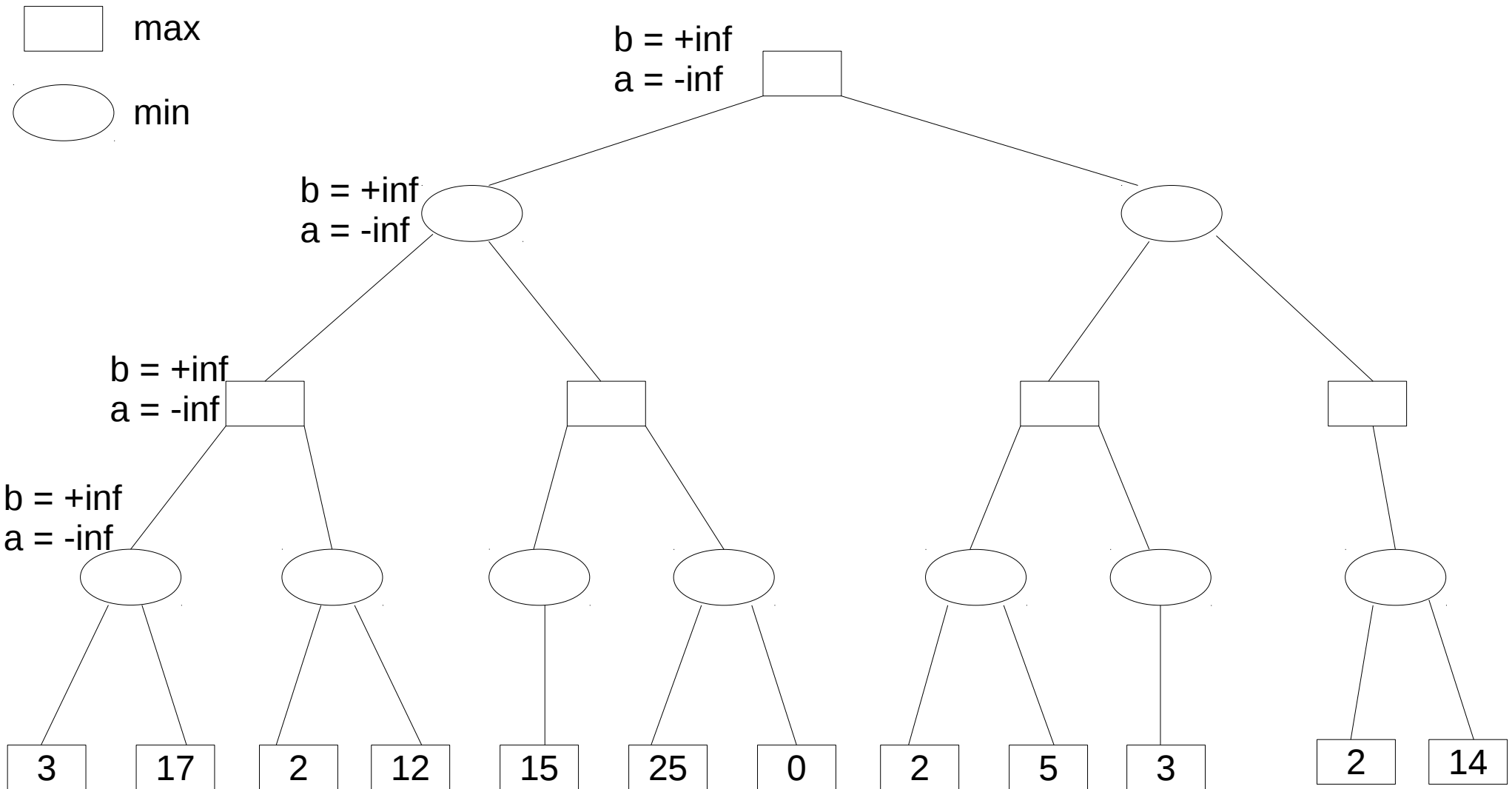
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



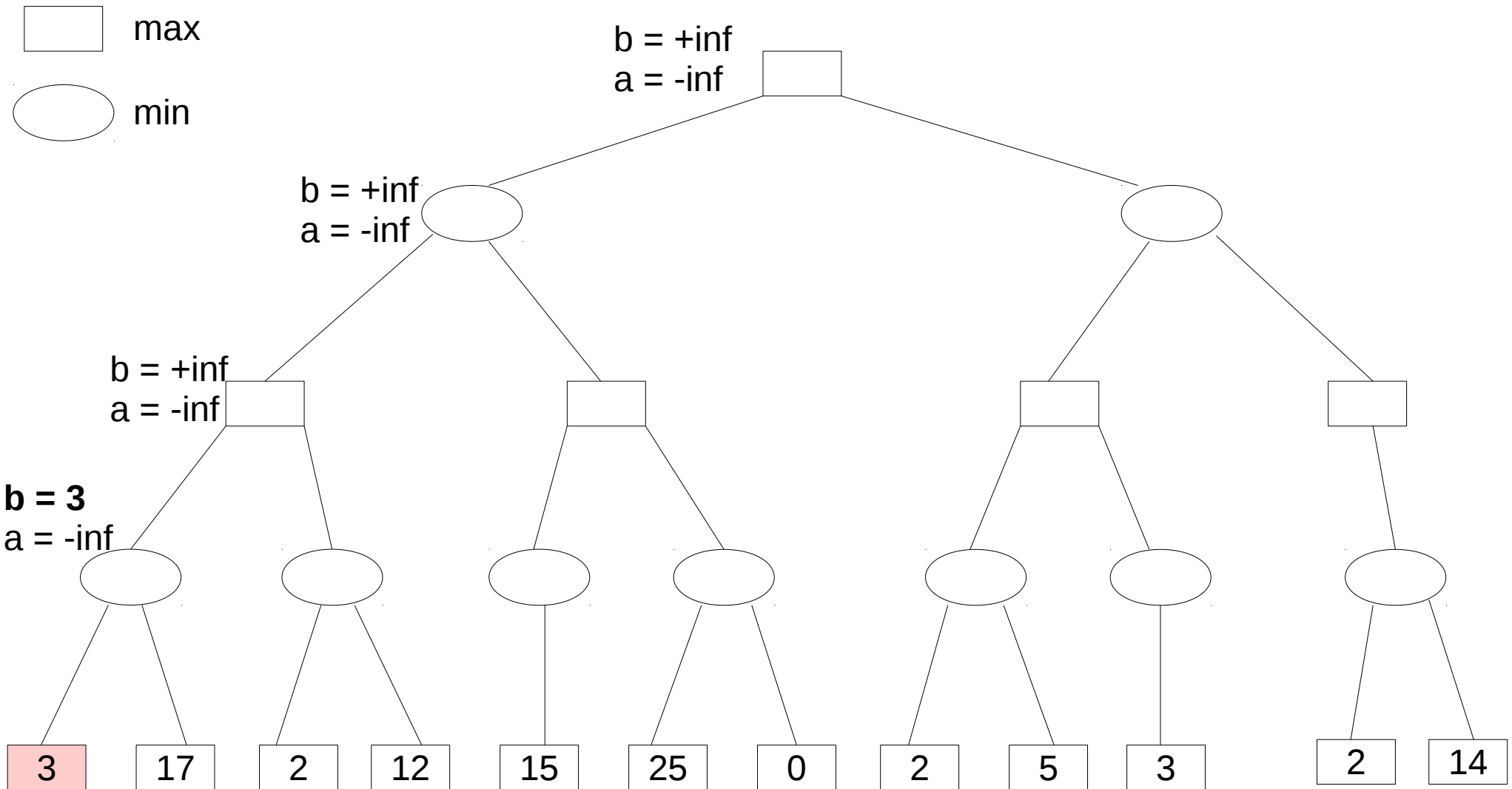
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



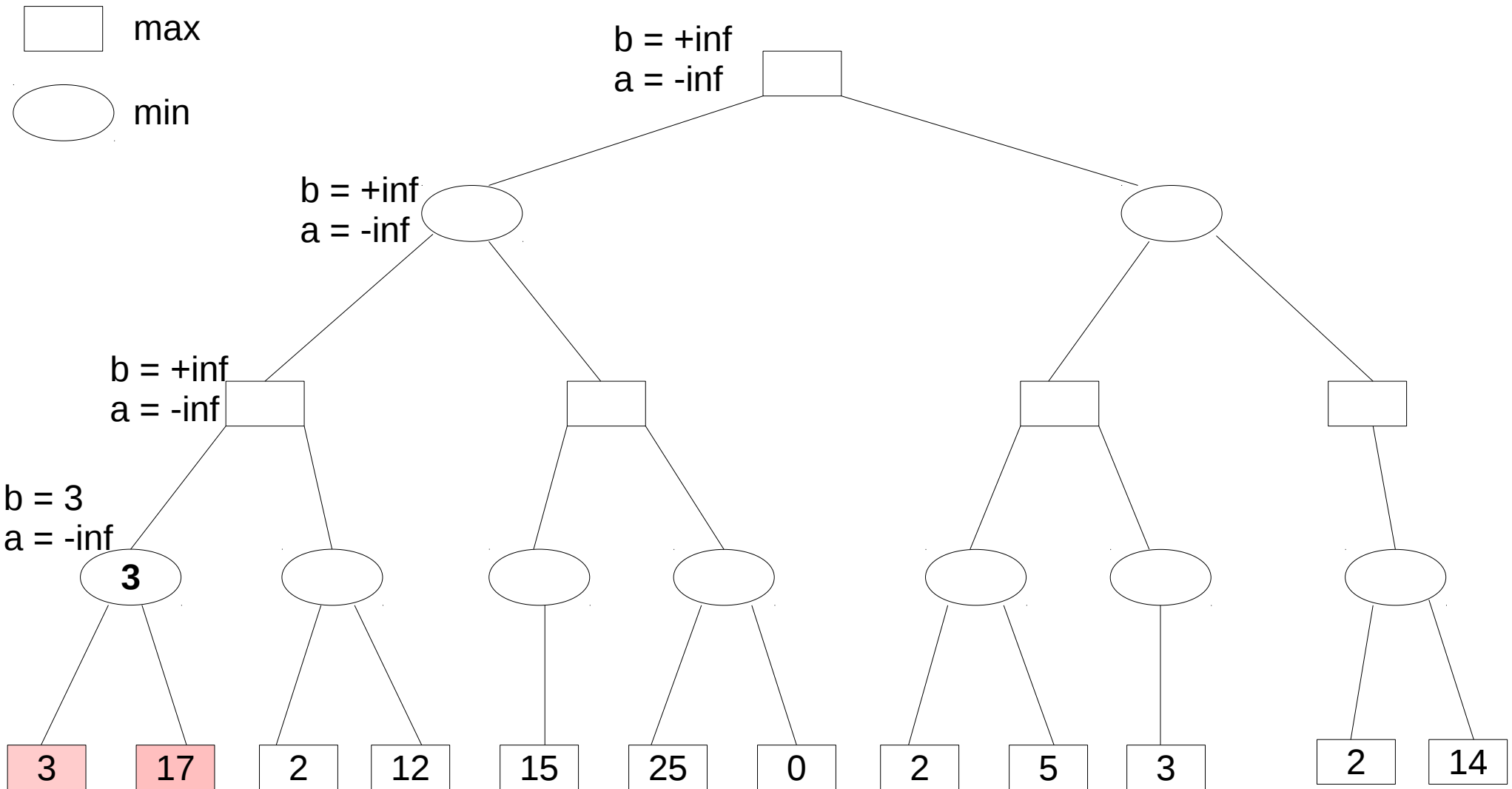
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



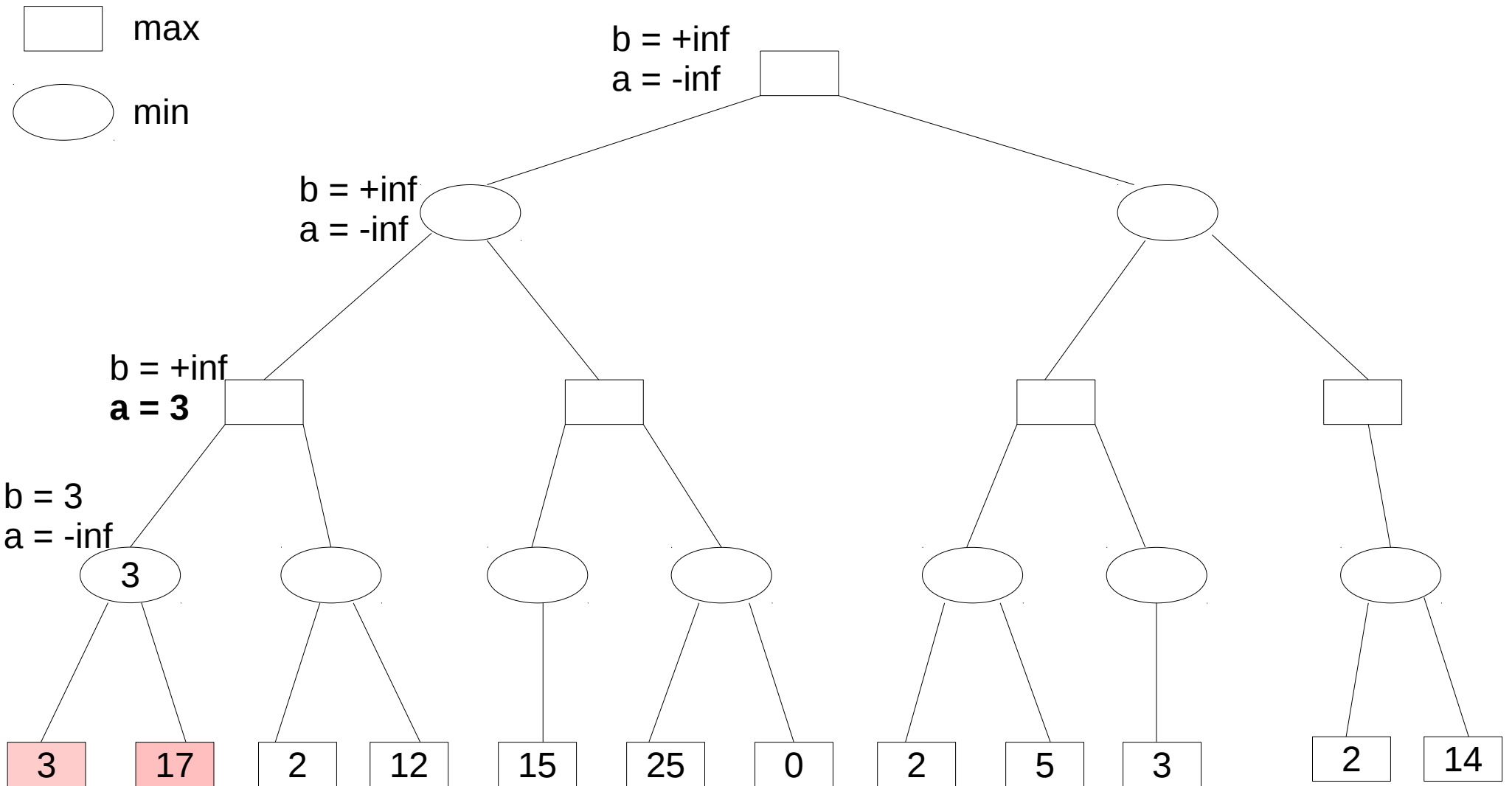
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



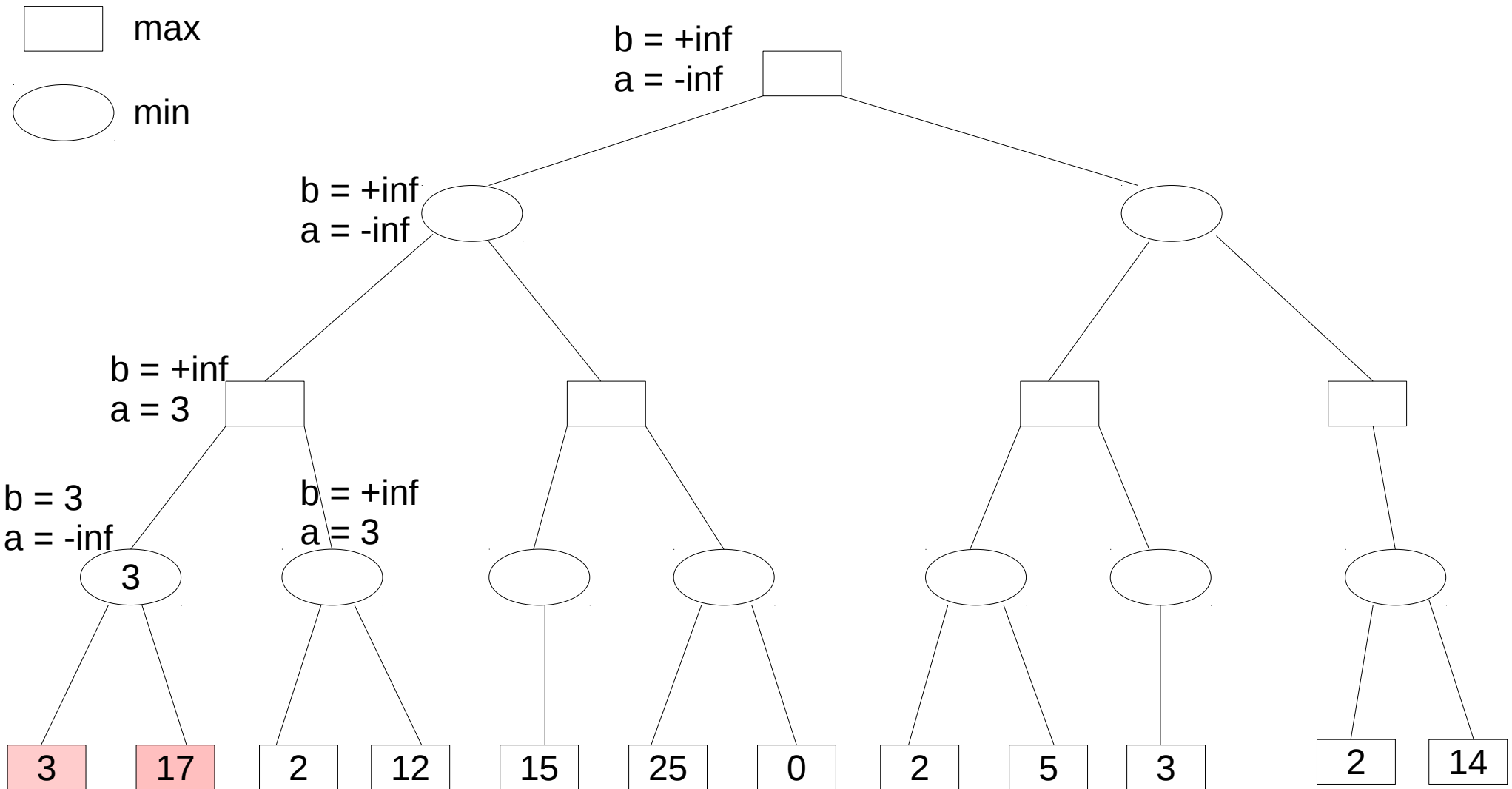
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



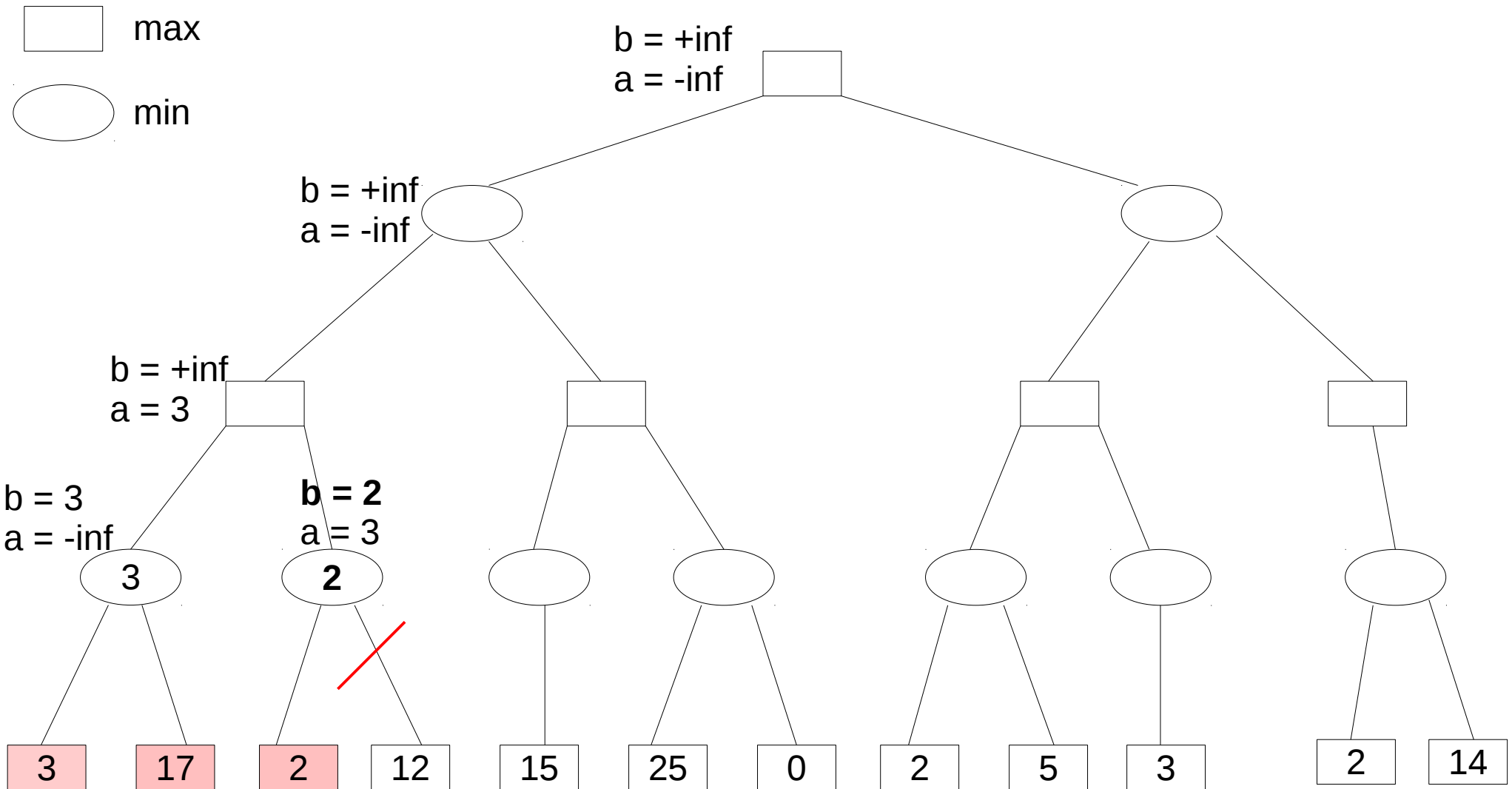
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



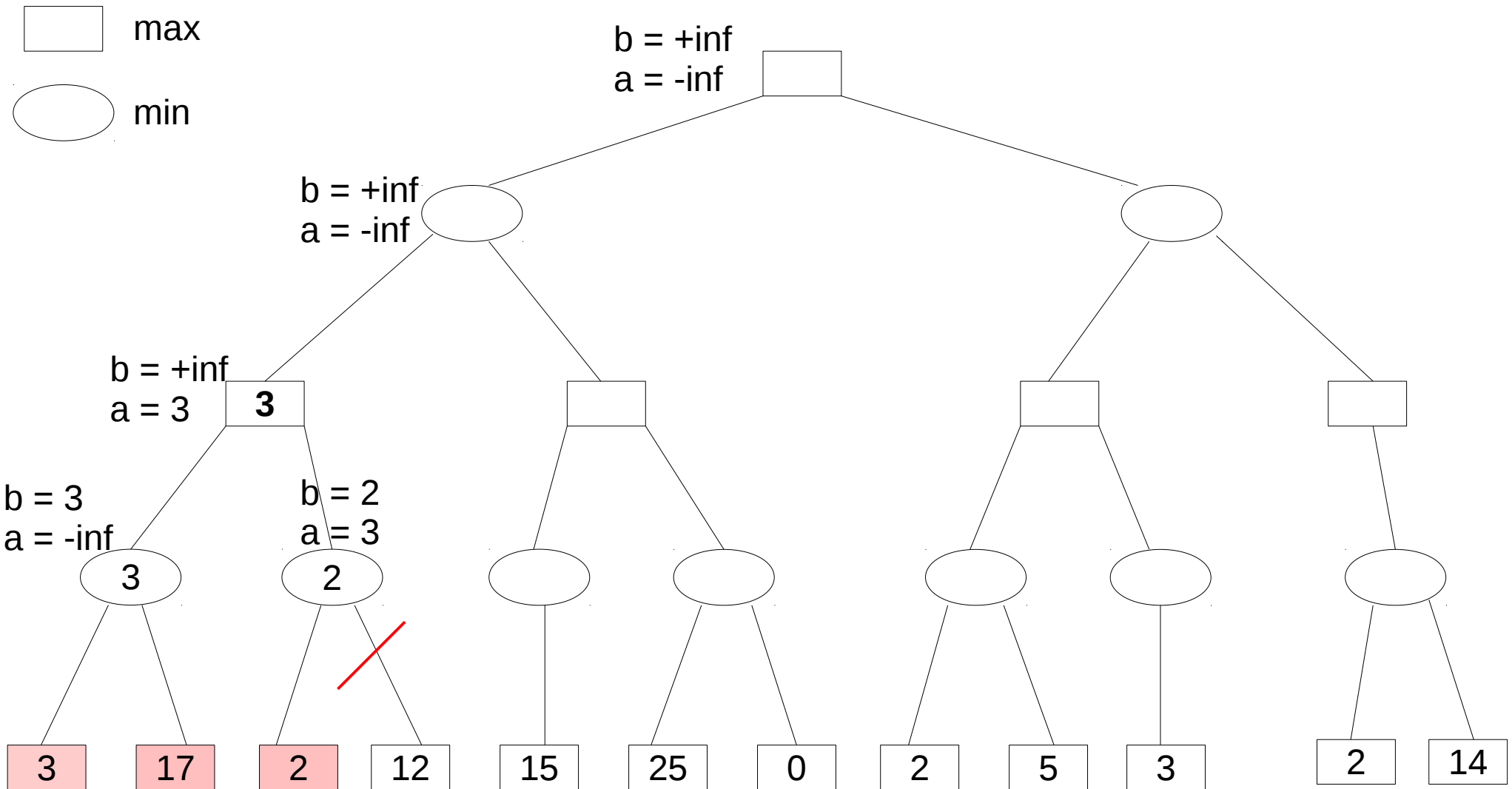
Exemple de Min-Max avec élagage α/β

Tiré de: <http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>



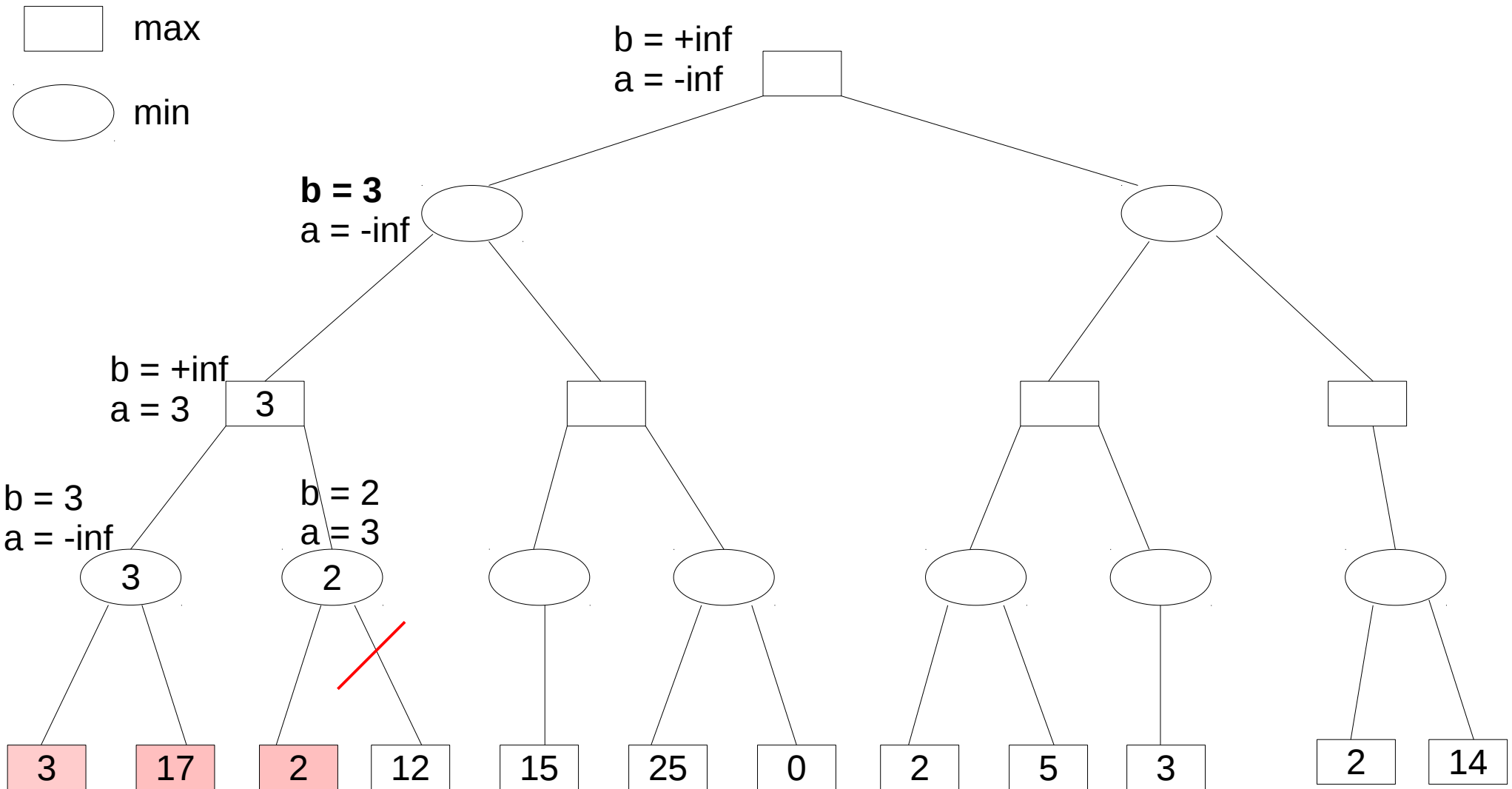
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



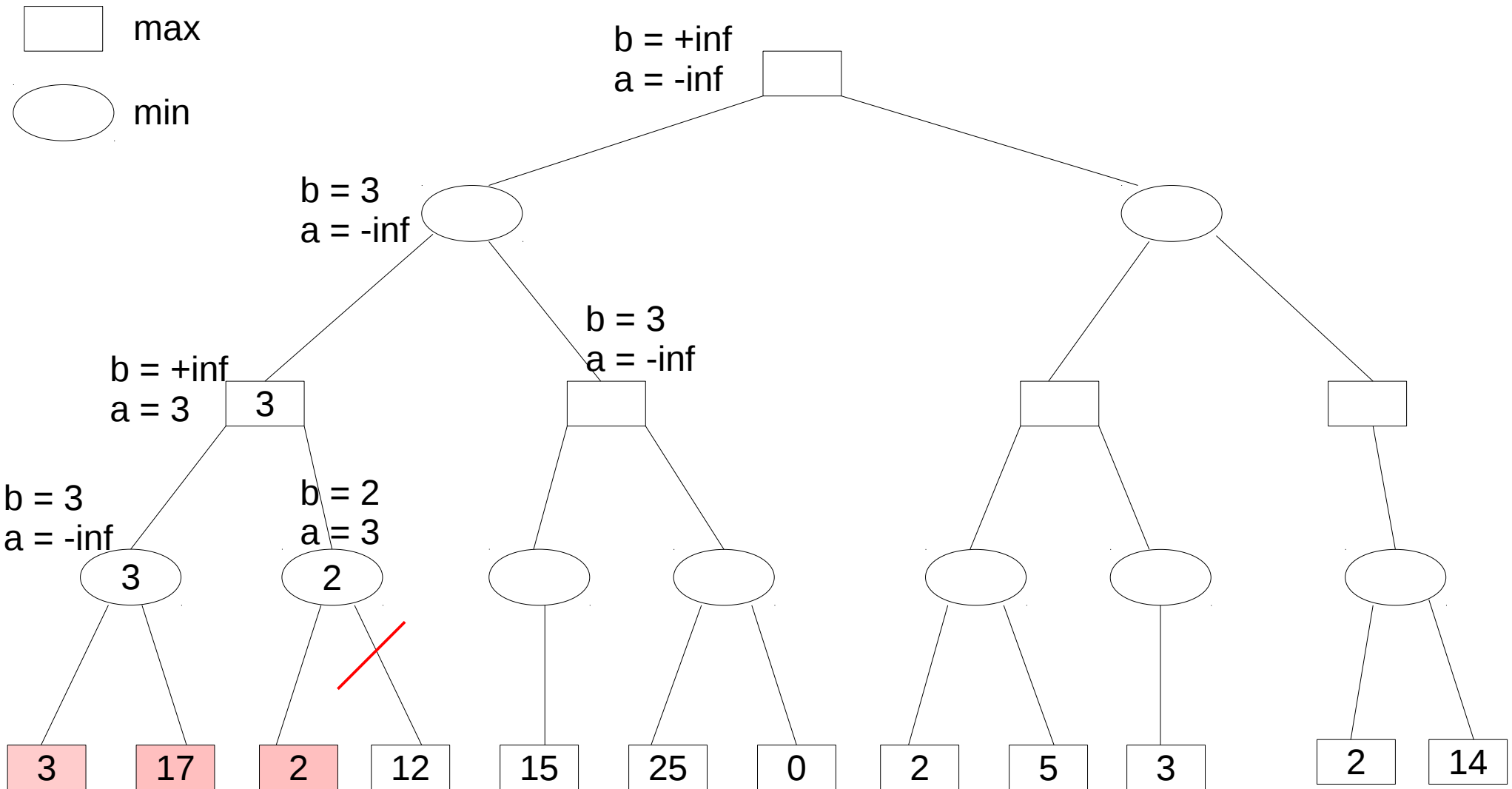
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



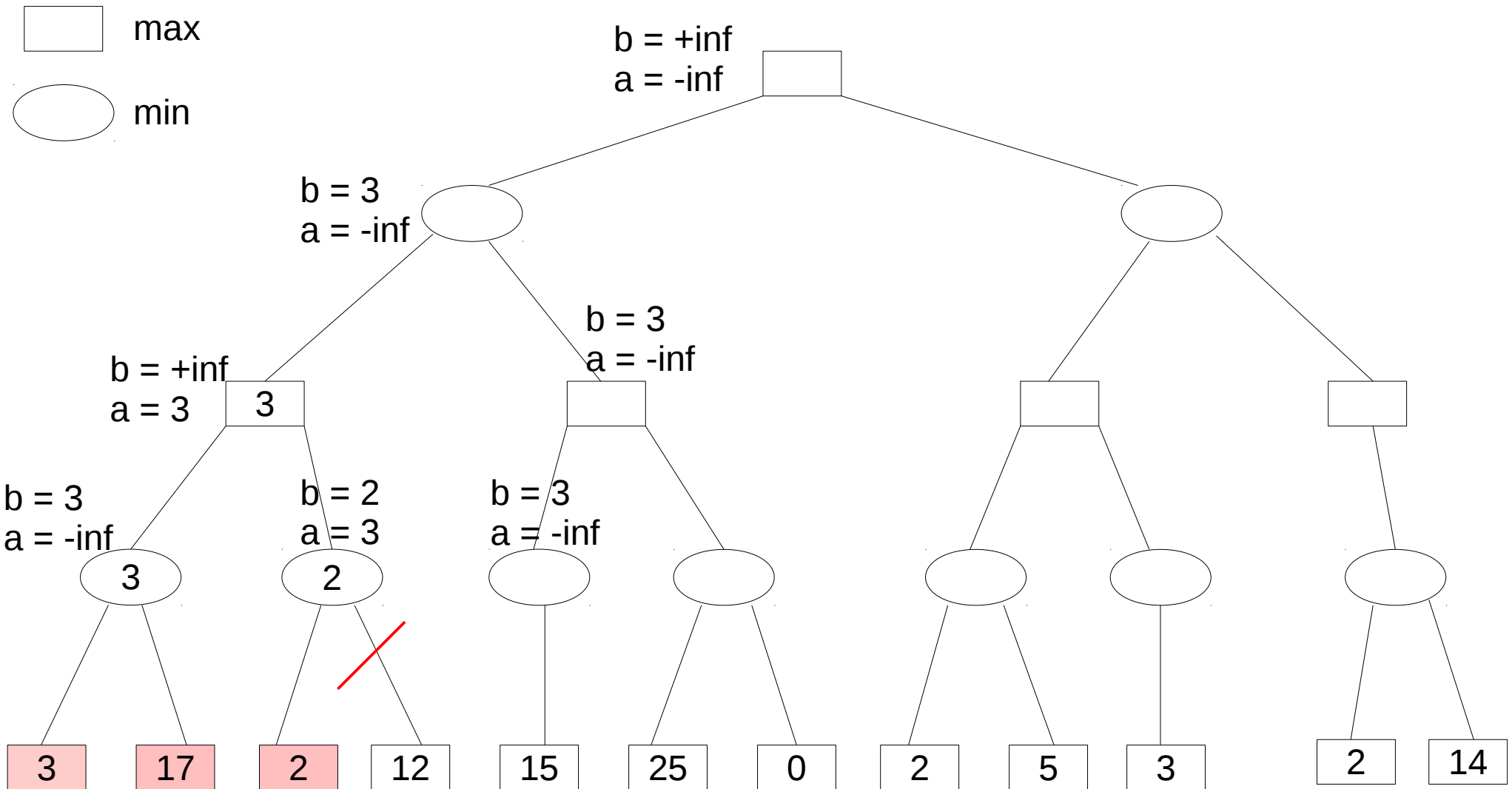
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



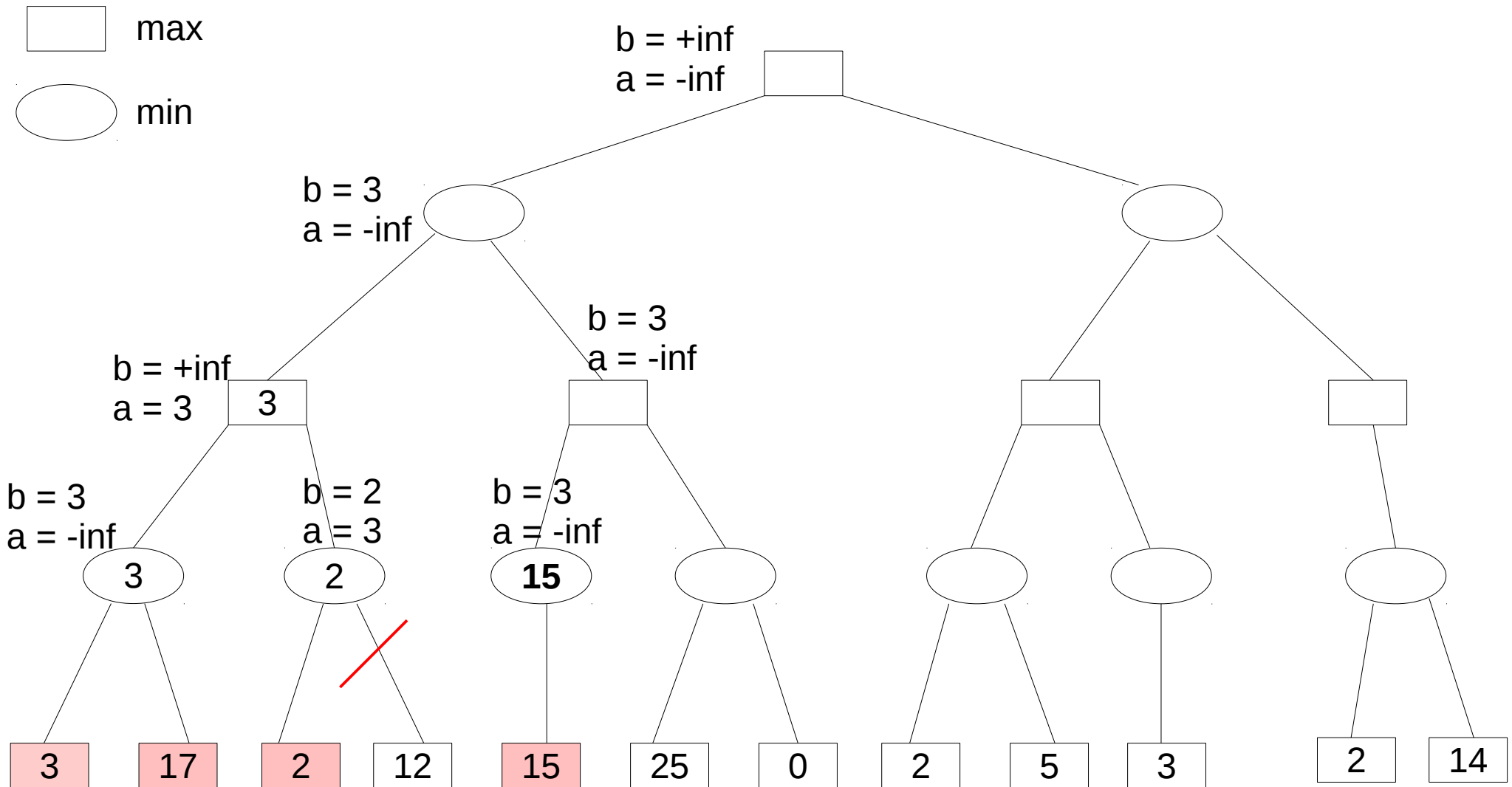
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

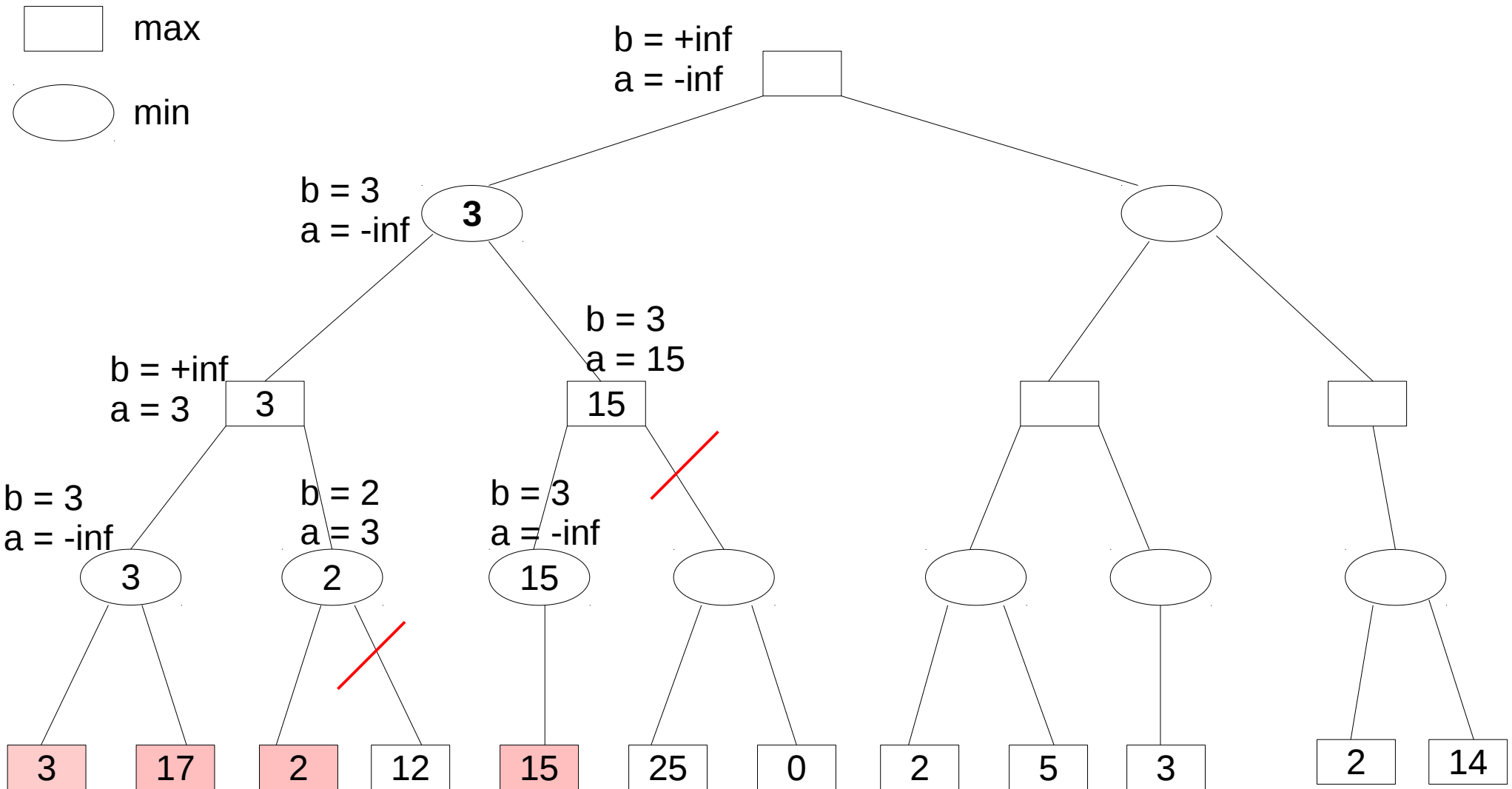
max

min



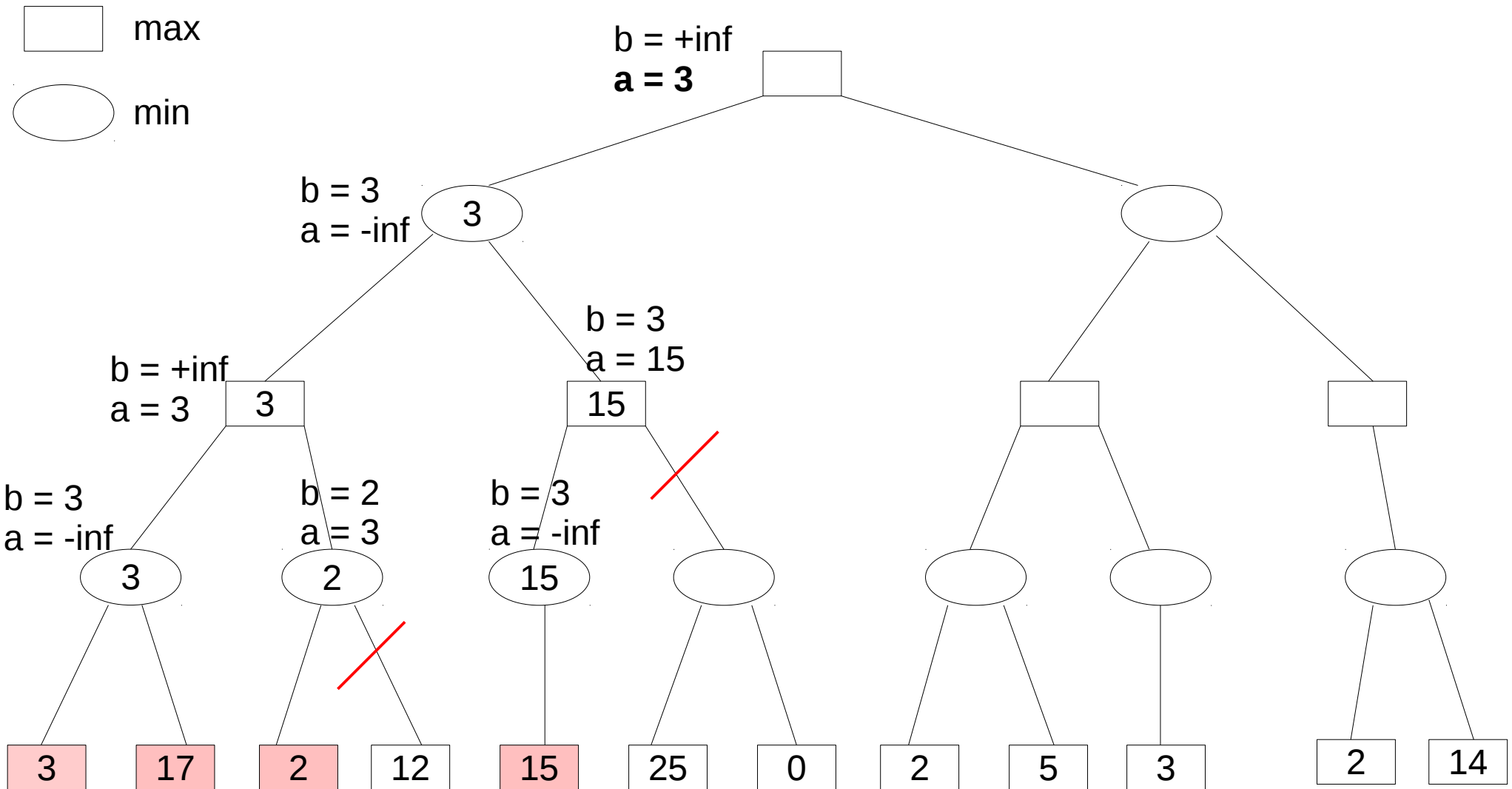
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

max

min



Tiré de: <http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>

A diagram illustrating the components of a game tree. It consists of two parts: a rectangle labeled "max" and an oval labeled "min".



Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

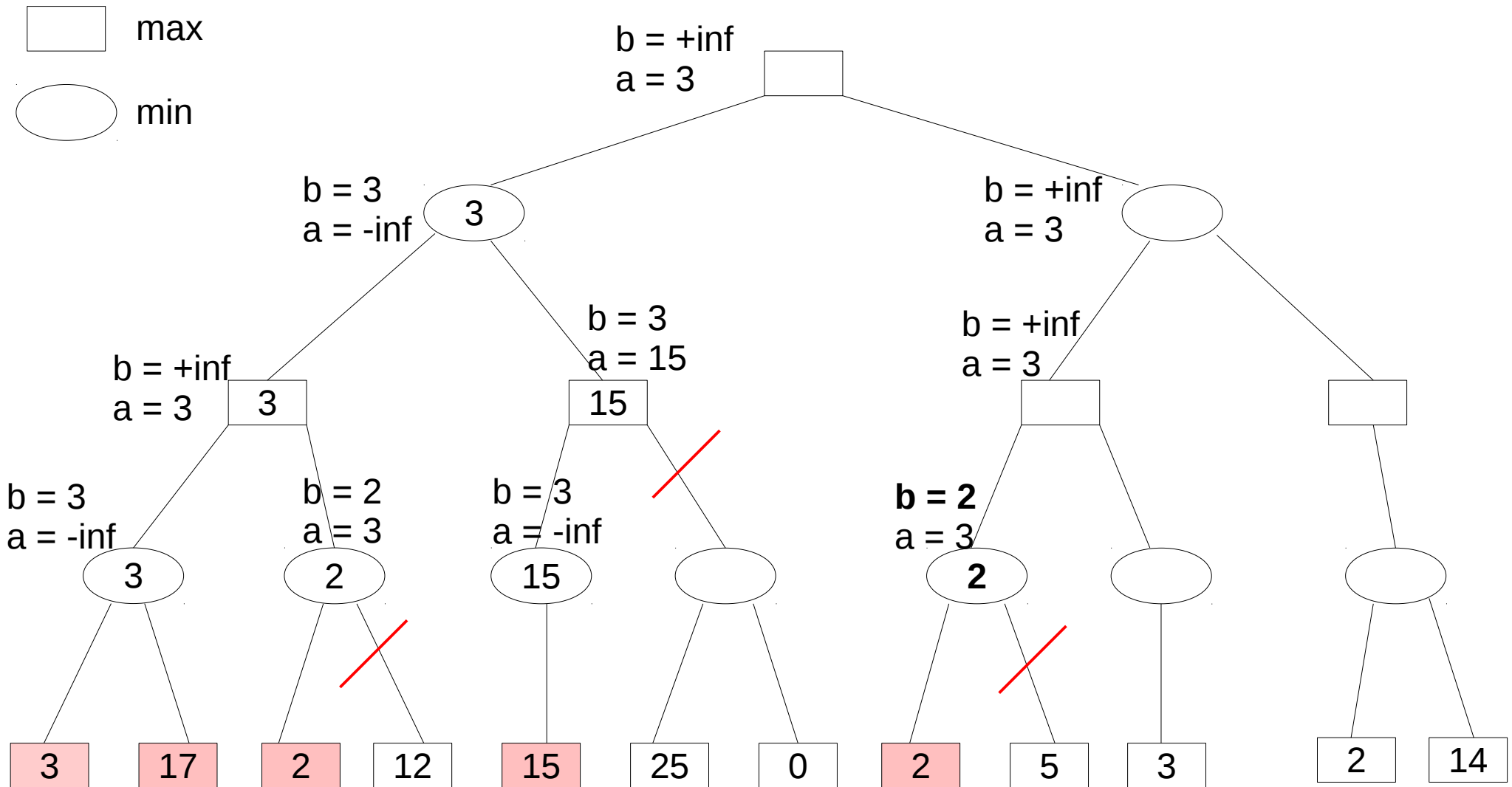
max

min



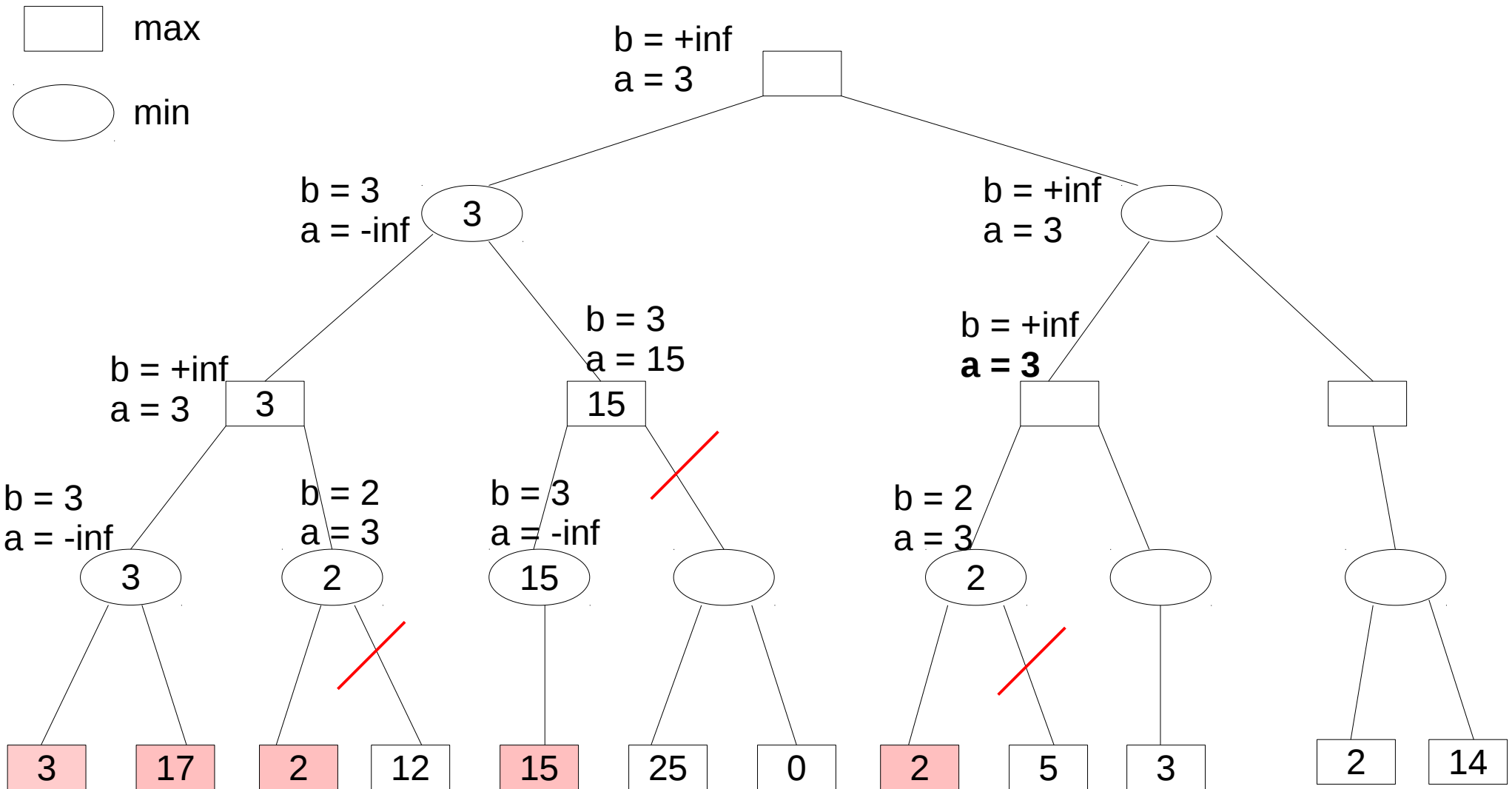
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



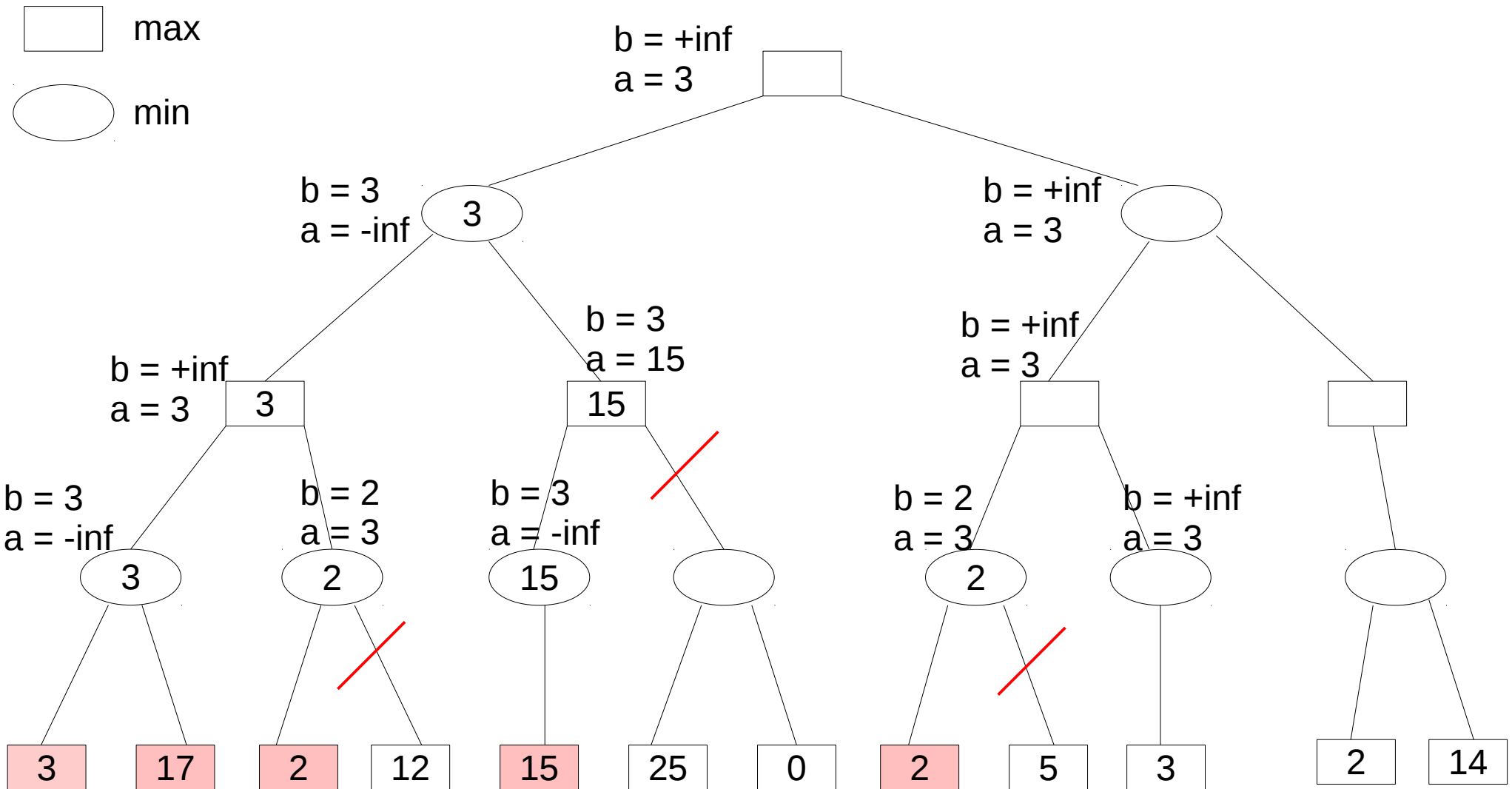
Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>



Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>>

max

min



Tiré de: <http://web.cs.ucla.edu/~rosen/161/notes/alpha.html>

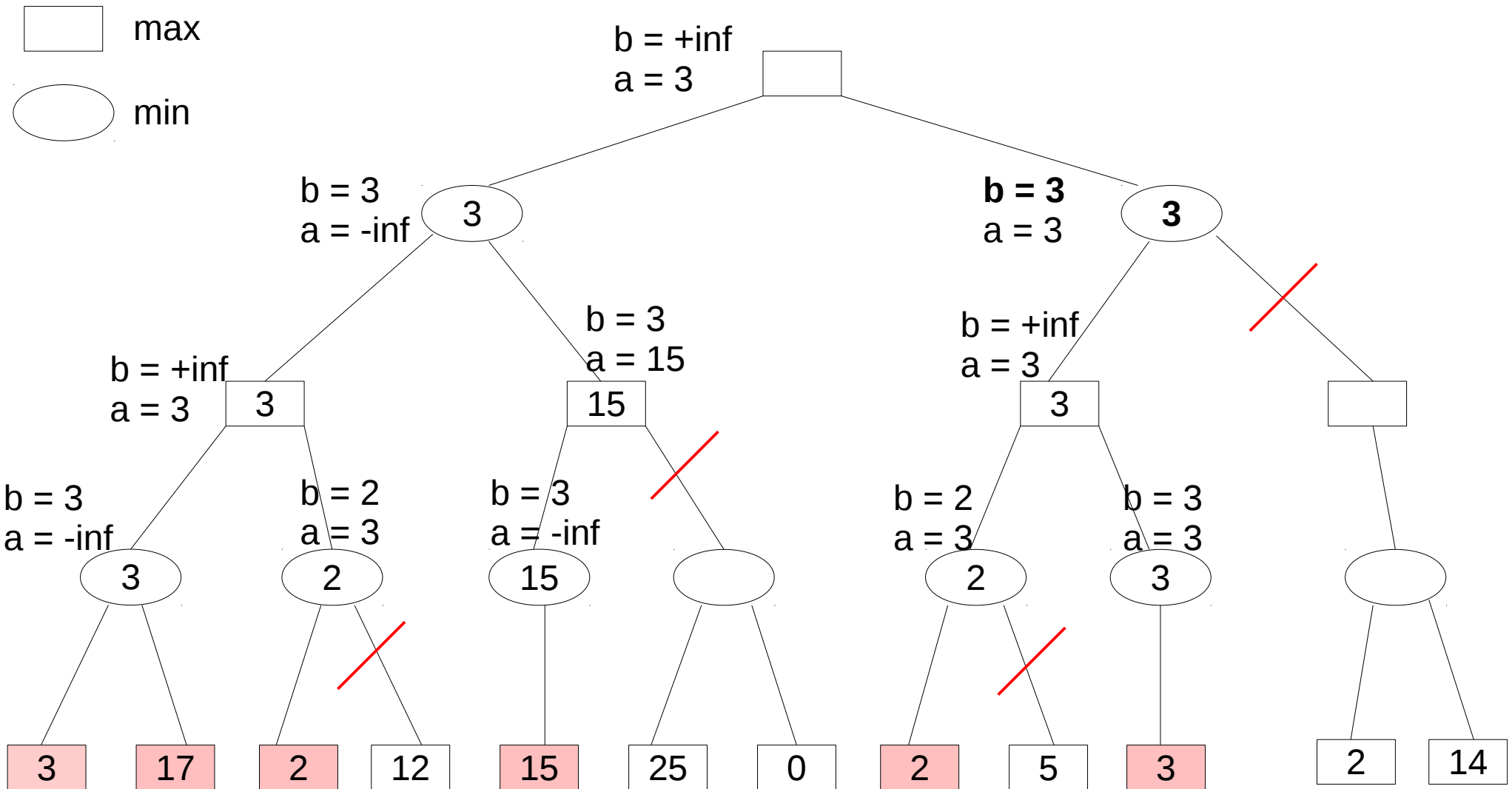
max

min



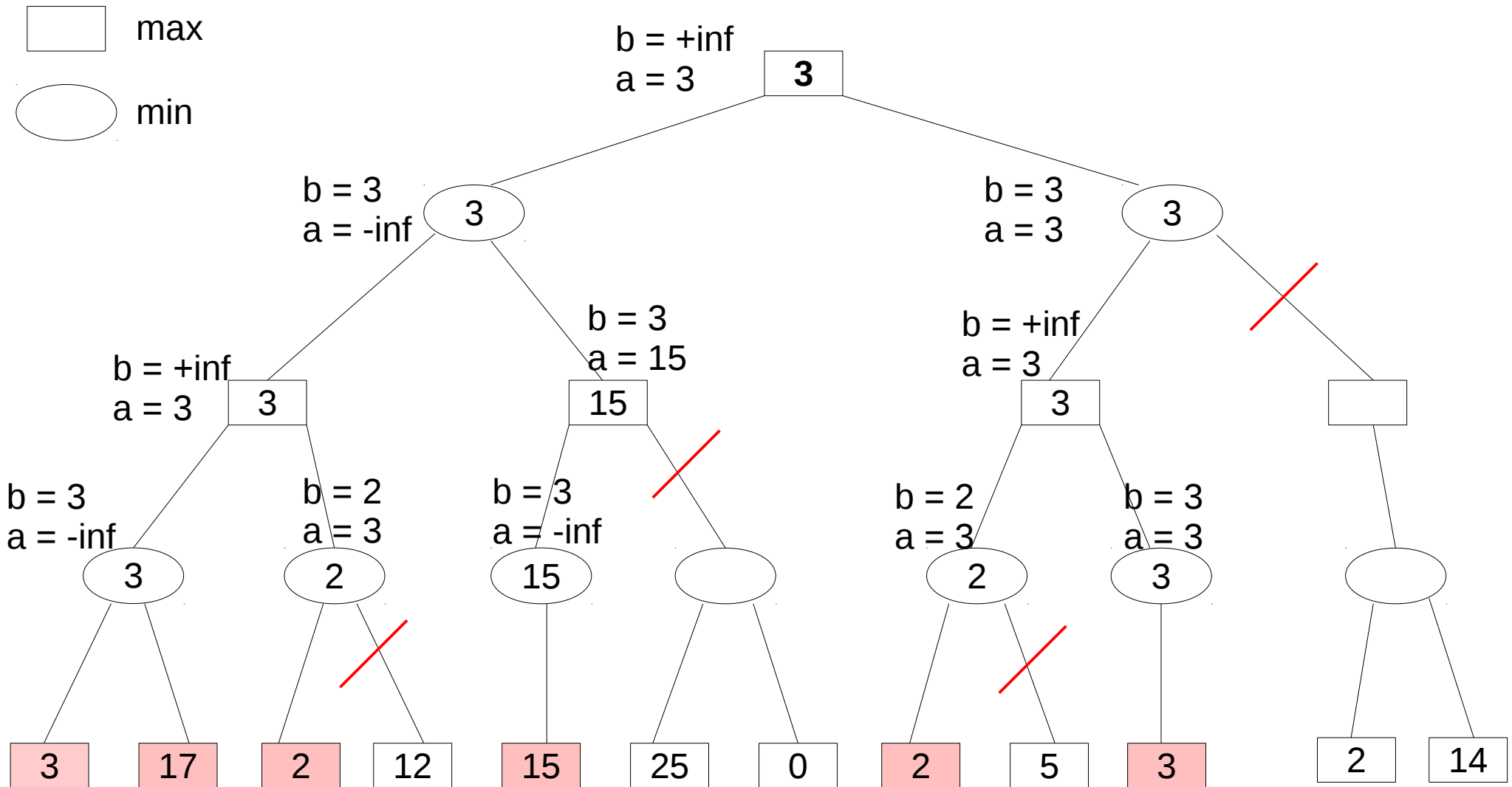
Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Exemple de Min-Max avec élagage α/β

Tiré de: <<http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html>>



Application au jeu d'échecs

Représentation d'une configuration

structure de données représentant une configuration dans le jeu :

La disposition des pièces dans l'échiquier et certaines informations utiles ...

Génération de successeurs

fonction pour générer tous les coups possibles d'un joueur donné ('min' ou 'max'), à partir d'une configuration donnée.

Pour chaque coup, une nouvelle configuration est générée

Evaluation d'une configuration terminale

fonction Coût(J) retournant :

-100 en cas de victoire pour 'min',

+100 en cas de victoire pour 'max' et,

0 en cas de match nul.

Evaluation d'une configuration non terminale

fonction Estimation(J) retournant une valeur dans $]-100, +100[$ et caractérisant la qualité estimée de la configuration J

Perspectives

- a) Compléter et éventuellement corriger la prise en compte des règles du jeu**
- b) Incorporer les bases de données pour les bonnes ouvertures**
- c) Tests de fonctions d'estimation plus évoluées**
- d) Développer ou utiliser des interfaces GUI**
- e) Prendre en considération les limites imposées au temps de réponses**
- f) Evaluations Parallèles**
- g) Introduction d'heuristiques dans le choix des alternatives à considérer à chaque niveau**
- h) Elaborer des stratégies guidées par des sous-buts intermédiaires**