**IA01**

A14

**TP2 : Jeu des allumettes**

**Alexis Schad - Matthieu Zins**

# Sommaire

[Sommaire 2](#_Toc403657573)

[1. Liste des états possibles pour une partie de 5 allumettes 3](#_Toc403657574)

[2. Liste des opérateurs en fonction de l'état courant 3](#_Toc403657575)

[3. Etat initial et états finaux 3](#_Toc403657576)

[4. Arbre de recherche 3](#_Toc403657577)

[5. Résolution du problème 3](#_Toc403657578)

[A. Représentation 3](#_Toc403657579)

[B. Fonctions de service 4](#_Toc403657580)

[C. Parcours en profondeur 5](#_Toc403657581)

[D. Parcours en largeur d'abord 7](#_Toc403657582)

# Liste des états possibles pour une partie de 5 allumettes

On représente un état par deux variables, une qui représente le joueur qui a la main (1 ou 2) et une autre qui représente le nombre d'allumettes restantes. Les états possibles sont donc :

(1, 5) (1, 4) (1, 3) (1, 2) (1, 1) (1, 0)

(2, 5) (2, 4) (2, 3) (2, 2) (2, 1) (2, 0)

# Liste des opérateurs en fonction de l'état courant

On représente un opérateur par deux variables. La première représente le joueur qui joue et la seconde représente le nombre d'allumettes qu'il retire.

|  |  |
| --- | --- |
| Etat courant | Opérateurs possibles |
| (1, 5) ou (1, 4) ou (1, 3) | (1, 1) ou (1, 2) ou (1, 3) |
| (2, 5) ou (2, 4) ou (2, 3) | (2, 1) ou (2, 2) ou (2, 3) |
| (1, 2) | (1, 1) ou (1, 2) |
| (2, 2) | (2, 1) ou (2, 2) |
| (1, 1) | (1, 1) |
| (2, 1) | (2, 1) |
| (1, 0) ou (2, 0) | aucun |

# Etat initial et états finaux

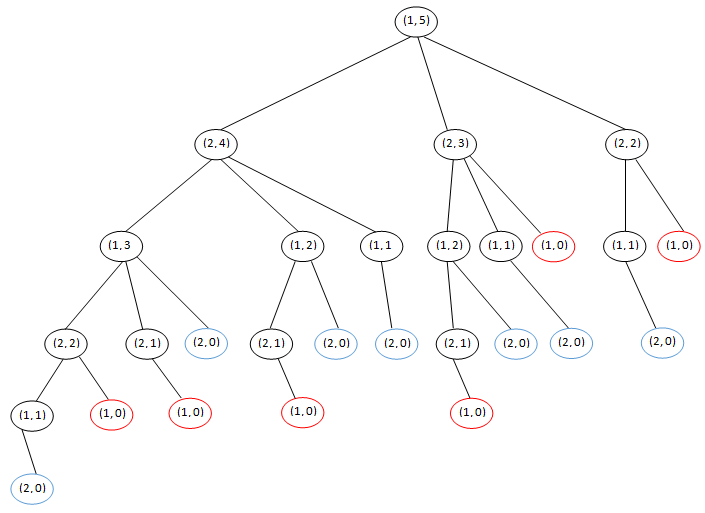
L'état initial est **(1, 5)**.

Les états finaux sont **(1, 0)** et **(2, 0)** en fonction du joueur qui doit gagner.

# Arbre de recherche

En rouge : les états finaux qui correspondent à une victoire du joueur 2

En bleu : ceux pour une victoire du joueur 1.



# Résolution du problème

## Représentation

Nous représenterons un état sous la forme d'une liste :

**( joueur\_qui\_a\_la\_main allumettes\_restantes)**

Exemple :

(setq etat\_intitial '(1 5))

Les opérateurs seront également représentés par des listes :

**(joueur allumettes\_retirées)**

Nous avons enregistré l'ensemble des coups possibles dans une liste (liste\_coups).

(setq liste\_coups '((1 1)

(1 2)

(1 3)

(2 1)

(2 2)

(2 3))

)

## Fonctions de service

**coups\_possibles** : Cette fonction renvoie la liste des coups possibles quand on est dans un certain état. Pour cela, on parcourt la liste de tous les coups et lorsqu'un coup est compatible on l'ajoute à "liste". Un coup est compatible à un état quand le joueur qui a la main et celui qui fait le coup sont les même et que le nombre d'allumettes retirées est inférieur au nombre d'allumettes restantes.

(defun coups\_possibles (etat liste\_coups)

(let (liste)

(dolist (x liste\_coups liste)

(when (and (equal (car x) (car etat)) (>= (cadr etat) (cadr x)))

(setq liste (append liste (list x)))

)

)

)

)

Exemple : On cherche les coups possibles quand le joueur 1 a la main et qu'il reste 5 allumettes.

>(coups\_possibles '(1 5) liste\_coups)

((1 1) (1 2) (1 3))

**suivant** : Cette fonction renvoie l'état suivant en fonction de l'action effectuée. Si le joueur 1 joue, le joueur 2 aura la main dans l'état suivant et le nombre d'allumettes restantes sera mis à jour.

(defun suivant (etat choix)

(let (suiv joueur)

(if (= (car etat) 1) (setq joueur 2) (setq joueur 1))

(setq suiv (list joueur (- (cadr etat) (cadr choix))))

)

)

Exemple : Il reste 5 allumettes, le joueur 1 a la main il en enlève 2 et on veut calculer l'état suivant.

>(suivant '(1 5) '(1 2))

(2 3)

**calc\_nb\_coups** : Cette fonction servira dans l'affichage final pour calculer le nombre de coups effectués par le joueur. Elle prend en paramètre une liste d'actions qui correspondent au déroulement d'une partie et le numéro du joueur. On parcourt les actions et on incrémente une variable lorsque le bon joueur joue.

(defun calc\_nb\_coups (actions joueur)

(let ((n 0))

(dolist (x actions n)

(when (equal joueur (car x))

(setq n (+ 1 n))

)

)

)

)

Exemple : On veut calculer le nombre de fois que le joueur 1 a joué lors de cette partie

>(calc\_nb\_coups '((1 1) (2 1) (1 1) (2 1) (1 1)) 1)

3

**afficher** : Cette fonction prend en paramètre un chemin (une liste d'états) qui correspond aux états successifs du déroulement d'une partie et des actions (une liste d'opérateur) qui correspondent aux coups joués lors d'une partie. On utilise mapcar pour afficher le déroulement d'une partie. A la fin, on affiche le joueur qui gagne. Celui-ci correspond au dernier joueur qui a joué, donc au car du dernier coup que l'on obtient avec (car (last actions)).

(defun afficher (chemin actions)

(mapcar #'(lambda (c a)

(format t "Le joueur ~S a la main. Il reste ~S allumette(s). ~&" (car c) (cadr c))

(format t "Le joueur ~S enlève ~S allumette(s). ~&" (car a) (cadr a))

)

chemin actions)

(format t "Le joueur ~S gagne !!~%~%" (caar (last actions)))

)

## Parcours en profondeur

**explore\_profondeur** : Cette fonction prend en paramètre l'état, la liste des coups, le joueur que l'on veux faire gagner et deux paramètres qui serviront à enregistrer les chemins et les actions.

Pour ce parcours, il faut rappeler la fonction récursivement. Lors d'un appel, s'il ne reste plus d'allumettes et que le joueur "vainqueur" a la main, on renvoie nil puisqu'il aura perdu. Sinon, on enregistre l'état dans le chemin et on teste si le joueur "vainqueur" a gagné, c'est-à-dire s'il ne reste plus d'allumettes et que l'adversaire a la main. Si c'est le cas, on pourrait afficher le résultat directement en utilisant la fonction afficher, mais nous avons plutôt choisi de renvoyer, à la fin la fonction, une liste de toutes les solutions possibles et que l'affichage se fasse dans une autre fonction pour pouvoir compter les solutions et trouver une solution optimale. Il faut donc renvoyer la solution trouvée. Une solution est une liste composée de la liste des états successifs (chemin) et de la liste des actions (actions). On renvoie (list (list chemin actions)) parce que l'on utilisera append pour l'ajouter aux autre solutions.

Sinon, on parcourt les coups possibles dans notre état. Pour chaque coup on rappelle la fonction avec en paramètre l'état suivant, la liste de tous les coups, le vainqueur, le chemin et les actions auxquelles on aura ajouté le coup considéré.

Lorsque la fonction renvoie autre chose que nil, on ajoute la ou les solutions trouvées (temp) à res. Une fois que tous les coups possibles ont été parcourus, res contiendra l'ensemble des solutions trouvées à partir de l'état où l'on se trouve et sera renvoyé.

(defun explore\_profondeur (etat liste\_coups vainqueur chemin actions)

(let ((res) (temp))

(if (and (equal (car etat) vainqueur) (= (cadr etat) 0))

nil

(progn

(setq chemin (append chemin (list etat)))

(if (and (not (equal (car etat) vainqueur)) (= (cadr etat) 0))

(list (list chemin actions)) ;; on pourrait aussi afficher directement les solutions ici

(dolist (x (coups\_possibles etat liste\_coups) res)

(setq temp (explore\_profondeur (suivant etat x) liste\_coups vainqueur chemin (append actions (list x))))

(when temp

(setq res (append res temp)

)

)

)

)

)

)

)

)

Exemple : On obtient une liste de solutions pour un partie à 5 allumettes avec le joueur 1 qui commence. Caque solution comporte une première liste des états successifs et une seconde des coups qui ont été joués.

>(explore\_profondeur '(1 5) liste\_coups 1 () ())

((((1 5) (2 4) (1 3) (2 2) (1 1) (2 0)) ((1 1) (2 1) (1 1) (2 1) (1 1)))

(((1 5) (2 4) (1 3) (2 0)) ((1 1) (2 1) (1 3)))

(((1 5) (2 4) (1 2) (2 0)) ((1 1) (2 2) (1 2)))

(((1 5) (2 4) (1 1) (2 0)) ((1 1) (2 3) (1 1)))

(((1 5) (2 3) (1 2) (2 0)) ((1 2) (2 1) (1 2)))

(((1 5) (2 3) (1 1) (2 0)) ((1 2) (2 2) (1 1)))

(((1 5) (2 2) (1 1) (2 0)) ((1 3) (2 1) (1 1))))

**afficher\_solutions\_profondeur** : Cette fonction va utiliser explore\_profondeur et gérer l'affichage des solutions. Elle prend l'état initial, la liste de tous les coups et le numéro du joueur que l'on veut faire gagner en paramètres.

Elle va ensuite appeler explore\_profondeur avec les mêmes paramètres plus deux listes vides et stocker le résultat dans sol. Puis pour chaque solution on l'affiche grâce à notre fonction de service. Les paramètres passés à afficher sont le chemin (car x) et les actions (cadr x). On affiche également le numéro de la solution grâce à "i" que l'on incrémente à chaque fois. Cela permettra de savoir combien de déroulements différentes permettent de faire gagner le joueur. On affiche également le nombre de coups joués par le joueur que l'on calcule grâce à la fonction calc\_nb\_coups. Ensuite on cherche une solution optimale en parcourant les solutions et cherchant celle dont la taille de la liste des actions est la minimale. On l'affiche à la fin.

(defun afficher\_solutions\_profondeur (etat liste\_coups vainqueur)

(let ((i 1)(min)(optimal)(sol))

(setq sol (explore\_profondeur etat liste\_coups vainqueur () ()))

(dolist (x sol)

(format t "~%Solution ~S en ~S coups : ~&" i (calc\_nb\_coups (cadr x) vainqueur))

(afficher (car x) (cadr x))

(setq i (+ i 1))

)

(setq min (length (cadar sol)))

(setq optimal (car sol))

(dolist (x sol)

(when (< (length (cadr x)) min)

(setq min (length (cadr x)))

(setq optimal x)

)

)

(format t "~%~%Solution optimale : en ~S coups : ~&" (calc\_nb\_coups (cadr optimal) vainqueur))

(afficher (car optimal) (cadr optimal))

)

)

Exemple : On veut afficher les solutions pour faire gagner le joueur 1 dans une partie à 5 allumettes où celui-ci commence.

> (afficher\_solutions\_profondeur '(1 5) liste\_coups 1)

Solution 1 en 3 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 3 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 2 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 2 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 3 allumette(s).

Le joueur 1 enlève 3 allumette(s).

Le joueur 1 gagne !!

Solution 3 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 2 allumette(s).

Le joueur 1 a la main. Il reste 2 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 1 gagne !!

Solution 4 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 3 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 5 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 2 a la main. Il reste 3 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 2 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 1 gagne !!

Solution 6 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 2 a la main. Il reste 3 allumette(s).

Le joueur 2 enlève 2 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 7 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 3 allumette(s).

Le joueur 2 a la main. Il reste 2 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution optimale : en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 3 allumette(s).

Le joueur 1 enlève 3 allumette(s).

Le joueur 1 gagne !!

## Parcours en largeur d'abord

**explore\_largeur** : Cette fonction cherche les solutions et réalise également l'affichage. Pour parcourir l'arbre en largeur d'abord, la fonction n'est pas récursive mais il faut utiliser un file. On utilisera une liste pour la réaliser. Dans cette liste on enfile à la fin grâce à append et on défile le premier élément grâce à pop.

On utilisera trois files, la première (file) pour enregistrer les états atteints, la deuxième (file\_chemins) pour les chemins parcourus et la troisième (file\_actions) pour les coups joués.

Au début, on initialise file avec une liste contenant l'état initial, et file\_chemins avec une liste contenant une liste de l'état initial. Ensuite on boucle jusqu'à ce que la file principale soit vide. A chaque itération on défile les files et on enregistre les éléments défilés dans temp, temp\_chemins et temp\_actions.

Lorsque l'élément défilé (temp) est un état solution (c'est-à-dire si l'adversaire a la main et qu'il ne reste plus d'allumettes) on affiche la solution. Pour l'affichage on réutilise notre fonction de service. Avec ce parcours la première solution trouvée est forcément une solution optimale puisque la recherche se fait par niveau.

Si ce n'est pas un état solution, on enfile tous ses états successeurs dans la file. Pour cela, on parcourt chaque coups possibles, on calcule l'état suivant et on l'enfile. On enfile en parallèle le chemin et la liste des actions qui mènent à cet état. Pour cela on ajoute l'état suivant (x) à l'ancien chemin (temp\_chemins) et on ajoute le coup joué (a) à l'ancienne liste des actions (temp\_actions) et on enfile ces deux nouvelles listes. Avec cet enfilage en parallèle, chaque état de la file a son chemin et la liste des actions dans les files file\_chemins et file\_actions au même indice.

Exemple de la gestion des trois files :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Etat initial | Dans la boucle | Après une itération |
| file | ((1 5)) | On défile (1 5) qui n'est pas un état final, donc on enfile tous ses successeurs. | ((2 4) (2 3)(2 2)) |
| file\_chemins | (((1 5))) | A chaque successeur on enfile le chemin qui a permis de l'atteindre | (((1 5) (2 4))  ((1 5) (2 3))  ((1 5) (2 2))) |
| file\_actions | () | A chaque successeur on enfile les listes des actions qui ont permis de l'atteindre | (((1 1)) ((1 2)  ((1 3))) |

(defun explore\_largeur (etat liste\_coups vainqueur)

(let ((temp)(file)(file\_chemins)(temp\_chemins)(file\_actions)(temp\_actions)(i 1)(x))

(setq file (list etat))

(setq file\_chemins (list (list etat)))

(while file

(setq temp (pop file))

(setq temp\_chemins (pop file\_chemins))

(setq temp\_actions (pop file\_actions))

(if (and (not (equal vainqueur (car temp))) (= 0 (cadr temp)))

(progn

(if (= i 1)

(format t "~%Solution optimale en ~S coups : ~&" (calc\_nb\_coups temp\_actions vainqueur))

(format t "~%Solution ~S en ~S coups : ~&" i (calc\_nb\_coups temp\_actions vainqueur))

)

(afficher temp\_chemins temp\_actions)

(setq i (+ 1 i))

)

(dolist (a (coups\_possibles temp liste\_coups))

(setq x (suivant temp a))

(setq file (append file (list x)))

(setq file\_chemins (append file\_chemins (list (append temp\_chemins (list x))))))

(setq file\_actions (append file\_actions (list (append temp\_actions (list a)))))

)

)

)

)

Exemple : On veut afficher les solutions pour faire gagner le joueur 1 dans une partie à 5 allumettes où celui-ci commence.

>(explore\_largeur '(1 5) liste\_coups 1)

Solution 1 (optimale) en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 3 allumette(s).

Le joueur 1 enlève 3 allumette(s).

Le joueur 1 gagne !!

Solution 2 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 2 allumette(s).

Le joueur 1 a la main. Il reste 2 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 1 gagne !!

Solution 3 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 3 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 4 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 2 a la main. Il reste 3 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 2 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 1 gagne !!

Solution 5 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 2 allumette(s).

Le joueur 2 a la main. Il reste 3 allumette(s).

Le joueur 2 enlève 2 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 6 en 2 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 3 allumette(s).

Le joueur 2 a la main. Il reste 2 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!

Solution 7 en 3 coups :

Le joueur 1 a la main. Il reste 5 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 4 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 3 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 2 a la main. Il reste 2 allumette(s).

Le joueur 2 enlève 1 allumette(s).

Le joueur 1 a la main. Il reste 1 allumette(s).

Le joueur 1 enlève 1 allumette(s).

Le joueur 1 gagne !!