Le solitaire V2

# Présentation

Jeu de plateau à un joueur.

Le plateau est constitué d’un ensemble de cases carrées jointives par les bords.

Il existe principalement deux configurations :

|  |  |
| --- | --- |
| Le plateau classique (33 cases) | Le plateau français (37 cases) |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |

Au départ, chaque case est occupée par une pierre, sauf une case.

Dans les jeux classique et français, seule la case centrale est vide.

Les cases contigües sont celles qui partagent un même bord. Il n’y a pas de mouvement en diagonale.

Un mouvement consiste à choisir une pierre A et à la faire sauter par-dessus une pierre B contigüe pour autant que A atterrisse dans une case vide du plateau. Alors B est retiré. Cela revient à déterminer trois cases contiguës alignées, les deux premières contenant une pierre, la troisième à une des extrémités est vide, et d’inverser la présence des pierres dans ces trois cases.

Si on note ⬢ la présence d’une pierre, ⬡ une case sans pierre, les 4 mouvements de base sont :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ⬢⬢⬡ | → | ⬡⬡⬢ |  | ⬡⬢⬢ | → | ⬢⬡⬡ |  | ⬢ ⬢ ⬡ | → | ⬡ ⬡ ⬢ |  | ⬡ ⬢ ⬢ | → | ⬢ ⬡ ⬡ |

Chaque mouvement fait donc décroitre de exactement 1 unité le nombre de pierres sur le plateau.

L’objectif est d’arriver à ne conserver qu’une seule pierre sur le plateau.

La position de la dernière pierre n’est pas déterminante.

S’il y a N pierres sur le plateau initial, il faut exactement N − 1 mouvements pour atteindre une solution (si elle existe).

S’il y a exactement une pierre manquante sur le plateau à C cases, il faut C – 2 mouvements pour atteindre une solution (si elle existe).

Un mouvement est autorisé sur une situation donnée si les cases impliquées sont dans l’état voulu. Appliquer ce mouvement produit une nouvelle situation valide ayant une pierre de moins que la situation dont elle est issue.

# Objectif

Evidemment, l’intérêt du jeu est d’être résolu manuellement. Mais je ne suis pas doué. Et je me pose une question existentielle : y a-t-il une solution quelle que soit la pierre de départ retirée d’un plateau complet ?

Lançons-nous dans l’écriture d’un programme de recherche automatique de solutions.

L’approche « naïve » montre vite ses limites, la recherche en profondeur se heurte au retraitement de multiples situations déjà rencontrées, ou équivalentes, le temps de recherche interdit d’espérer répondre à la question.

Si on tente de mémoriser les situations rencontrées, la mémoire de l’ordinateur n’y suffit pas.

# Résultats

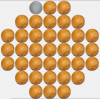
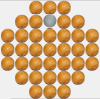
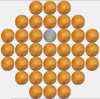
Au final, j’ai opté pour une représentation très compacte, et une stratégie qui m’ont permis de résoudre la question.

## Plateau Classique

* Quelle que soit la première pierre retirée, il existe toujours une suite de mouvements qui aboutit à une situation gagnante.

## Plateau Français

* Il n’y a de solution gagnante que pour trois situations initiales seulement (aux symétries près)

# Optimisations

## Symétries

Le programme tient compte des symétries du plateau (rotations, symétries axiales). Si une situation initiale a été résolue, alors toutes les situations initiales obtenues par symétrie sont réputées résolues, sans que le programme ne calcule lui-même de solution pour ces situations initiales symétriques.

## Convergence

Deux situations initiales non symétriques peuvent aboutir à deux situations intermédiaires égales ou symétriques l’une de l’autre. Si la première situation initiale a une solution, alors l’autre aura aussi une solution, seuls les premiers mouvements diffèreront, jusqu’à aboutir à la situation intermédiaire.

## Reprise

La recherche pouvant s’avérer très longue, il est possible de la suspendre, d’arrêter et de relancer le programme qui reprend là où il s’était arrêté.

## Dualité

On va exploiter une propriété du jeu très puissante, présentée ci-dessous.

🡺 Les rôles des cases occupées et vides peuvent être interchangés.

Supposons une situation S caractérisée par l’état occupé ou libre de chaque case du plateau.

La situation duale, notée , de S est celle où les pierres sont remplacées par des trous et vice-versa.

Soit un mouvement possible M décrit par ses trois cases (c1, c2, c3), où c1, c2 sont les cases occupées, et c3 la case vide, le résultat du mouvement étant que c1, c2 seront libres et c3 occupée.

Son mouvement dual, noté M-1 est un pseudo mouvement décrit par les mêmes trois cases (c1, c2, c3) où c1, c2 sont des cases vides, et c3 une case occupée, le résultat du mouvement étant que c1, c2 seront occupées et c3 vide.

Supposons qu’à partir d’une situation donnée S1 on lui applique un mouvement M autorisé pour aboutir à une situation S2.

Considérons alors la situation duale de S2 et appliquons-lui le pseudo mouvement dual M-1. On obtient alors la situation , duale de la situation S1.

Illustration :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | x | x | x |  |  |  |  |  | x | x | x |  |  | |  |  | x | x | x |  |  |  |  |  | x | x | x |  |  | | x | x | x | x | x | x | x |  | x | x | x | x | x | x | x | | x | x | x | x |  | x | x | ⇒ | x | x |  |  | x | x | x | | x | x | x | x | x | x | x |  | x | x | x | x | x | x | x | |  |  | x | x | x |  |  |  |  |  | x | x | x |  |  | |  |  | x | x | x |  |  |  |  |  | x | x | x |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  | x |  |  | ⇒ |  |  | x | x |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |

Supposons, pour le plateau à C cases, qu’à partir d’une situation initiale donnée SC-1 à C-1 pierres, on obtienne par M mouvements {m0, …, mM-1} une situation intermédiaire IC-1-M donnée à C-1-M pierres.

Partons alors de la situation duale de SC-1, appelons-la F1, elle est évidemment gagnante. Appliquons-lui dans le même ordre les mouvements duaux {m-10, …, m-1M-1} : on obtient une situation I’M+1 à M+1 pierres qui est la situation duale de IC-1-M.

On construit alors l’ensemble AM (aux symétries près) des situations possibles après M mouvements effectués à partir de l’ensemble des situations initiales possibles (aux symétries près). Ses situations ont toutes C-1-M pierres. L’ensemble des situations duales est noté BM. Il n’est même pas à construire, puisqu’il se déduit très aisément des situations de AM. Ses situations ont toutes M+1 pierres.

Une fois AM construit, il suffit que l’on cherche une suite de I=C-2-2M mouvements pour chaque situation S de AM qui mène (à une symétrie près) à une situation de BM, c’est-à-dire à une situation duale de AM. Arrivé à une telle situation duale, on sait qu’il existe une suite de mouvements qui mène certaines situations initiales à S (il nous faudra les retrouver, car on ne les stocke pas), puis d’appliquer les mouvements trouvés pour arriver à la situation duale trouvée. Puis on sait qu’il existe une suite de mouvements (qu’il nous faudra aussi retrouver) qui mène de cette duale à une situation gagnante.

Plus M est élevé, plus la suite intermédiaire à trouver est courte. Evidemment, on peut viser de pousser l’analyse jusqu’à obtenir A(C-2)/2.

Avec la représentation très compacte choisie, il a été tout à fait possible d’atteindre cet objectif.

## Description des situations

Le plateau est inscrit dans un rectangle minimal implicite, appelé rectangle encadrant, ou englobant. Ce rectangle est divisé en carrés dans certains desquels se situent les cases du plateau. Ces carrés sont numérotés à partir de 0, de gauche à droite, à partir du haut.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Les coordonnées et indices des cases du rectangle encadrant   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 |  |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | 0 | -3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | 0 | -3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 1 | -2 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |  | 1 | -2 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 2 | -1 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |  | 2 | -1 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | | 3 | 0 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |  | 3 | 0 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | | 4 | 1 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |  | 4 | 1 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | | 5 | 2 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |  | 5 | 2 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | | 6 | 3 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |  | 6 | 3 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | |

Pour diminuer la taille mémoire d’une situation, on va uniquement stocker sous forme de tableau de bits l’ensemble des cases susceptibles de contenir une pierre. Il y a 33 ou 37 cases, il faudra 33 ou 37 bits. On va donc utiliser un tableau de 5 octets fournissant donc 40 bits.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Les coordonnées et indices des cases du plateau   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 |  |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | 0 | -3 |  |  | 0 | 1 | 2 |  |  |  | 0 | -3 |  |  | 0 | 1 | 2 |  |  | | 1 | -2 |  |  | 3 | 4 | 5 |  |  |  | 1 | -2 |  | 3 | 4 | 5 | 6 | 7 |  | | 2 | -1 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  | 2 | -1 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | | 3 | 0 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |  | 3 | 0 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | | 4 | 1 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |  | 4 | 1 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | | 5 | 2 |  |  | 27 | 28 | 29 |  |  |  | 5 | 2 |  | 29 | 30 | 31 | 32 | 33 |  | | 6 | 3 |  |  | 30 | 31 | 32 |  |  |  | 6 | 3 |  |  | 34 | 35 | 36 |  |  | |

# Démarche

## Initialisation

Le programme initialise un espace de jeu où au plus 40 cases peuvent être occupées par des pierres.

* Le mode classique, traitant un plateau de 33 cases
* Le mode français, traitant un plateau de 37 cases

Cette initialisation est décrite par une chaine de caractères qui lui est transmise, situant les cases dans un rectangle. Seule la présence du caractère ‘x’ est prise en compte

* Pour le mode classique :

xxx

xxx

xxxxxxx

xxxxxxx

xxxxxxx

xxx

xxx

* Pour le mode français

xxx

xxxxx

xxxxxxx

xxxxxxx

xxxxxxx

xxxxx

xxx

Le programme calcule alors les symétries applicables au plateau.

Le programme calcule les positions initiales possibles, à une symétrie près, où exactement une pierre manque.

Le nombre de ces positions ne doit pas dépasser 8, ce qui est le cas pour les plateaux classique et français, et permet de limiter la taille disque et mémoire nécessaire.

Le programme stocke sur disque ces positions initiales.

Il est prêt pour passer à la recherche en largeur.

## Recherche en largeur

Supposons que le nombre de cases pouvant accueillir une pierre soit C.

Le programme part d’un ensemble de situations stocké sur disque représentant toutes les situations possibles de exactement P pierres, ⌈ C/2 ⌉ < P < C, à une symétrie près, pouvant être atteintes à partir des situations initiales.

Il construit alors et stocke sur disque toutes les situations possibles de exactement P – 1 pierres, à une symétrie près, pouvant être atteintes à partir des situations à P pierres précédentes.

Une fois ce calcul effectué, l’espace disque des situations à P pierres est conservé. Ces résultats permettront de retrouver plus rapidement les chemins nécessaires pour établir une solution complète à partir d’une solution partielle.

## Suspension recherche en largeur

Lors d’une recherche en largeur, le programme surveille un drapeau qui lui demande de s’interrompre lorsqu’il est levé. Le programme enregistre donc la situation qu’il était en train d’évaluer et arrête son traitement.

## Reprise recherche en largeur

Lorsque le programme de recherche en largeur démarre, il vérifie d’abord si un enregistrement de situation est présent. S’il est présent, le programme reprend son traitement à partir de celle-ci plutôt que de démarrer avec la première situation, et efface cet enregistrement.

## Début recherche en profondeur

Le choix de passer de la recherche incrémentale en largeur à une recherche en profondeur dépendra de la place disponible sur disque à la fin de chaque étape de la recherche en largeur.

Supposons que le nombre de cases soit C.

Supposons que l’ensemble des situation possibles à P pierres ait été établi.

Le programme recherche en profondeur pour la première des situations à P pierres, disons P1, les situations possibles à P cases vides. Si l’une d’entre elles est (à une symétrie près) la duale d’une des situations à P pierres, alors toutes les situations initiales pouvant amener à P1 ont une solution (qu’il reste à calculer pour chacune d’elles).

## Suite recherche en profondeur

Pour chaque situation Pi de l’ensemble des situations à P pierres, on ne traite Pi que si celle-ci est accessible depuis au moins une situation initiale n’ayant pas encore de solution.

En cas de découverte d’une nouvelle situation Pi résolvant au moins une situation initiale, si le programme établit que toutes les situations initiales ont été résolues, il stoppe la recherche.

## Suspension recherche en profondeur

Lors d’une recherche en profondeur, le programme surveille un drapeau qui lui demande de s’interrompre lorsqu’il est levé. Le programme enregistre donc la situation qu’il était en train d’évaluer et arrête son traitement.

## Reprise recherche en profondeur

Lorsque le programme de recherche en profondeur démarre, il vérifie d’abord si un enregistrement de situation est présent. S’il est présent, le programme reprend son traitement à partir de celle-ci plutôt que de démarrer avec la première situation, et efface cet enregistrement.

## Consolidation

Ici, on parle systématiquement d’une situation comme classe de ses situations équivalentes à une symétrie près.

Le plateau contient C cases.

L’initialisation calcule les situations initiales (donc à C−1 pierres) et leur affecte à chacune un flag différent 00000001, 00000010, …, 10000000.

L’ensemble de ces situations initiales est noté EC−1.

La première recherche en largeur calcule toutes les situations obtenues à partir de ces situations initiales en effectuant un mouvement exactement.

Chacune d’elles est associée à un flag qui est le OU logique des situations initiales pouvant mener à elle.

L’ensemble de ces situations obtenues après un mouvement est noté EC−2.

Et puis les recherches en largeur successives établissent successivement l’ensemble des situations obtenues après 2, 3, …, m mouvements.

Chacune d’elles est toujours associée à un flag qui est le OU logique des situations initiales pouvant mener à elle.

L’ensemble de ces situations obtenues après m mouvements est noté EC−1−m.

Faute d’espace disque suffisant, on interrompt cette recherche en largeur, aux résultats persistants, pour entreprendre une recherche en profondeur.

Dans cette recherche en profondeur, on se saisit successivement de chaque situation SC−1−m de EC−1−m pour entreprendre une recherche arborescente de ses situations successives. Et on leur associe toujours ce flag d’association aux situations initiales. Imaginons qu’on ait trouvé une solution pour une certaine liste de situations initiales, et que la situation SC−1−m de EC−1−m envisagée ne soit accessible qu’à partir de situations initiales toutes comprises dans cette liste. Continuer à rechercher une solution pour cette situation ne nous apportera aucune autre information, qu’elle soit résoluble ou non, elle ne permettra pas de résoudre de nouvelles situations initiales. Par contre, si SC−1−m est accessible par au moins une situation initiale non encore résolue, il en sera de même de toutes ses situations accessibles. Il est inutile de faire la vérification pour ces nouvelles situations.

Les situations ainsi calculées sont conservées en mémoire afin d’interrompre une recherche en profondeur si elle aboutit à une situation déjà rencontrée (à condition que l’obtention de cette nouvelle situation n’implique pas de nouvelles situations initiales non résolues). Cette mémoire est libérée en cas de condition de mémoire basse, ce qui a pour conséquence de recalculer éventuellement des situations déjà rencontrées mais effacées depuis.

On calcule en profondeur les situations issues de la situation SC−1−m jusqu’à établir une situation Sm+1 à m+1 pierres, donc à C – 1 – m cases vides. Si la situation duale , à C – 1 − m pierres, de Sm+1 est dans EC−1−m, alors on sait (puisqu’il existe une suite de mouvements amenant d’une situation initiale à ) qu’il existe une suite de mouvements amenant de Sm+1 à la duale de cette situation initiale. Les situations initiales associées à sont résolubles.

L’objectif est d’établir pour chaque situation initiale (non encore résolue) associée à SC – 1 − m au moins une suite de mouvements aboutissant à une situation finale.

Dans la recherche en profondeur, on maintient une liste de mouvements, de longueur C – 2 – 2m, peu gourmande en mémoire, qui a permis de passer de SC−1−m à Sm+1.

Il nous reste à établir les listes de mouvements suivantes :

* Celle(s) permettant de passer d’une des situations initiales non encore résolues à SC−1−m
* Une liste permettant de passer de Sm+1 à une situation finale. On n’a besoin que d’une seule solution, car on ne prend pas en considération l’emplacement de la pierre restante.

Ces recherches se font en différé, toujours pour des questions d’occupation mémoire, afin de ne pas influer sur la recherche en profondeur.

Rechercher une solution à partir de Sm+1 se fait par une recherche en profondeur. Mais on va exploiter les résultats de la recherche en largeur pour optimiser celle-ci : supposons qu’on obtienne une situation non finale Sk mais dont le dual, , à C-k pierres, n’est pas présent dans EC−k. Alors on peut interrompre la recherche sur ce Sk, sachant qu’on ne pourra trouver de solution finale à partir de lui, puisqu’il n’existe aucune situation initiale menant à son dual.

Rechercher un chemin qui mène de chaque situation initiale non résolue associée à SC – 1 − m est un peu plus compliqué.

On part de SC – 1 − m et on reconstitue les situations pouvant aboutir à SC – 1 − m en appliquant des mouvements « inverses » aux mouvements autorisés. On effectue encore une recherche arborescente. On utilise encore les résultats de la recherche en largeur, mais d’une manière légèrement différente. Supposons qu’on accède à une situation Sk qui n’est pas présente dans Ek, alors on peut abandonner la recherche depuis Sk. C’est l’équivalent de ce qui est fait dans la recherche de chemin vers une situation finale. Mais supposons que Sk soit présent, seulement la liste de ses situations initiales associées peut être différente de celle de SC – 1 – m; si cette liste n’est constituée que de situations initiales résolues, là encore, il devient inutile de rechercher les chemins « inverses » y menant.

On remarque que la description d’un mouvement « inverse » est identique à celle du mouvement « normal ». Seule son application diffère, en remplaçant les pierres par des cases vides, et inversement.

Maintenant qu’on a trois listes de mouvements, il suffit de les assembler judicieusement pour obtenir une suite menant d’une situation initiale à une situation finale, c’est-à-dire une suite de mouvements qui résoud cette situation initiale.

# Situations initiales

Voici la liste des situations initiales du plateau français, avec en brun le trou et en brun clair la position du trou dans l’une des situations initiales symétriques.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Les situations initiales, une seule pierre retirée (cases brunes).  Avec mise en évidence des symétries (cases brun clair).   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Situation 1 | | | | | | |  | Situation 2 | | | | | | |  | Situation 3 | | | | | | |  | Situation 4 | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Situation 5 | | | | | | |  | Situation 6 | | | | | | |  | Situation 7 | | | | | | |  | Situation 8 | | | | | | | |

On peut remarquer que les situations 2 et 8 autorisent chacune un seul mouvement, aux symétries près pour la situation 8, et que ce mouvement aboutit à la même situation. Donc les situations 2 et 8 développent dès le premier mouvement exactement les mêmes situations dérivées.

Les situations gagnantes sont les duales de ces situations. Sur le plateau Français il y en a donc aussi 8.

Sur le plateau classique, il y a 7 situations initiales, seule la situation 3 du plateau français est absente.

# Volume

Calcul grossier du nombre de situations différentes :

Sur un plateau de N cases, chaque case peut contenir ou non une pierre. Le nombre théorique de situations différentes est donc

Pour 33 cases, cela représente environ 9 milliards de cas.

Pour 37 cases, cela représente environ 140 milliards de cas.

On peut réduire ces cas d’un facteur inférieur à 8 en considérant les symétries possibles.

Le nombre de cas dans les premières étapes est lui aussi réduit. Mais on peut penser que plus de mouvements sont effectués, et plus de situations potentielles sont atteignables.

# Les symétries

Le plateau est composé de cases données par leur indice dans un rectangle englobant dont on connait la largeur L et la hauteur H. Pour un indice de case i donné, on lui calcule des coordonnées cartésiennes x=i%L, y = I/L ;

Les coordonnées de ces cases sont situées dans deux intervalles identifiés : [xMin, xMax], [yMin, yMax].

Ici, on est assuré que xMin = yMin = 0, xMax = L-1, yMax = H-1.

Une « symétrie » est une rotation ou une symétrie axiale qui conserve globalement toutes les cases du plateau. On envisage les 3 rotations de 90°, 180°, 270°, dont le centre reste à déterminer, et les symétries axiales selon des axes parallèles aux axes du repère ou à ses deux diagonales principales.

L’objectif est de dresser pour chaque symétrie trouvée un tableau T de taille L\*H, tel que T[i] fournit l’indice de la case image de la case d’indice i par la symétrie.

Ainsi, lorsqu’on voudra les symétries d’une situation donnée, situation elle-même inscrite dans le rectangle englobant, et fournissant la liste de ses cases occupées, il sera aisé d’exploiter ces tableaux.

Pour que le plateau soit conservé, il est nécessaire, mais pas suffisant, que son rectangle englobant le soit. Cela est le cas pour la rotation 180° et les deux symétries principales. Cela n’est le cas pour les autres rotations et symétries que si le rectangle est un carré.

Le centre du rectangle englobant joue un rôle essentiel dans la recherche de symétries. Mais dans le cas général, ses coordonnées peuvent ne pas être entières. Soient ses coordonnées. Si le rectangle est un carré , les coordonnées de son centre sont soit toutes deux entières, soit toutes deux demi-entières. Leur somme ou leur différence est donc entière.

Si

Rotation 90° autour du centre du carré englobant

Rotation 180° autour du centre du rectangle englobant

Rotation 270° autour du centre du carré englobant

Symétrie du rectangle selon un axe vertical

Symétrie du rectangle selon un axe horizontal

Symétrie du carré selon une parallèle à la 1ère diagonale

Symétrie du carré selon une parallèle à la 2ième diagonale

Résumé des 8 symétries possibles :

Identité

Rotation de 90°

Rotation de 180°

Rotation de 270°

Symétrie selon un axe vertical

Symétrie selon un axe horizontal

Symétrie selon un axe parallèle à la 1ère diagonale

Symétrie selon un axe parallèle à la 2ième diagonale

# Compacité ?

On part de l’hypothèse qu’il existe une solution et on voudrait orienter la recherche sur les cas qui semblent les plus prometteurs pour limiter les temps de recherche.

Comment améliorer la recherche ?

Il pourrait sembler que lorsqu’une situation présente des trous énormes entre pierres, il peut paraitre difficile d’aller chercher les pierres éloignées pour les éliminer.

Par exemple, la situation suivante (à 15 pierres) n’a pas de solution.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | -3 |  |  |  |  |  |  |  | | -2 |  | x |  |  |  |  |  | | -1 |  |  | x |  | x |  |  | | 0 |  |  |  |  |  |  |  | | 1 | x | x | x | x |  |  | x | | 2 |  | x | x | x | x | x |  | | 3 |  |  | x |  | x |  |  | |

On essaie alors de calculer une mesure de cette dispersion, pour avantager les situations les moins dispersées.

N : Nombre de points, indicés de 0 à N-1 ;

(xi,yi) : coordonnées du point d’indice i ;

G : centre de gravité des points, G a pour coordonnées (xG, yG)

Vi : variance du point d’indice i par rapport à G

En appliquant ce calcul à l’ensemble des situations à 14 pierres, on obtient l’ensemble des variances des situations finales, dont on peut calculer la moyenne et la variance : environ 9,30

Et si on calcule la variance de la situation perdante ci-dessus (qui possède 15 pierres, on obtient 4,60

Cette démarche n’est donc pas pertinente. Parmi toutes les 75 432 581 situations finales (et donc pouvant aboutir à une situation gagnante), la suivante a une variance de 7,92 :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | -3 |  |  | x | x | x |  |  | | -2 |  |  | x |  |  |  |  | | -1 |  |  |  |  |  |  |  | | 0 | x |  |  |  |  | x | x | | 1 | x | x |  |  |  | x | x | | 2 |  | x |  |  |  | x |  | | 3 |  |  |  | x |  |  |  | |

Et pourtant elle a une solution …

La moyenne des variances des SF s’établit à 1,72. Sachant que toutes ces variances sont positives, ça semble vouloir dire que ces situations sont en général très compactes. Mais la variance des variances atteint 9,30.

# Occupation mémoire

Le stock de situations déjà rencontrées occupe rapidement la mémoire vive.

Dans les premières versions du programme, nous avons eu des OutOfMemory exceptions. Ces exceptions sont déclenchées lorsqu’un objet requiert trop de mémoire.

Pour augmenter la taille des objets manipulables en mémoire, plusieurs réglages :

* Compiler pour Any CPU ou 64 bits,
* Exécuter sur une machine 64 bits
* Dans App.Config de l’application (l’exécutable), placer les lignes suivantes :

<configuration>

\*\*\*

<runtime>

<gcAllowVeryLargeObjects enabled="true" />

</runtime>

</configuration>

* Décocher l’option « Préférer 32 bits »

Attention : vérifier ces réglages en mode Debug comme en mode Release.

Tout cela permet effectivement d’augmenter la quantité de ressources mémoire accaparée par le programme, mais un autre problème survient alors : le swap des pages mémoire.

J’ai alors trouvé sur Internet des instructions permettant d’être averti lorsque la mémoire RAM commence à manquer. Une classe SurveillanceMemoire met en œuvre ce mécanisme, et lorsque le moment arrive, la classe déclenche un événement, et le programme abonné peut libérer de la mémoire vive qu’il a constituée pour éviter des recherches redondantes. Il va alors reprendre des recherches qui auraient été inutiles sans cette limitation.

## Dans la recherche en largeur

Dans cette recherche, on utilise principalement deux B-Tree persistants :

* Celui de l’avant dernière couche dont on lit les situations les unes après les autres
* Celui de la dernière couche qu’on alimente avec les situations nouvellement trouvées dérivées de celles de la couche précédente

Avant de libérer de la mémoire, il faut d’abord sauvegarder sur disque les éléments de la couche en cours de constitution : on ne les sauve pas à chaque fois qu’on en trouve une, mais dans le processus normal à la fin de la scrutation.

Ensuite, pour libérer de la mémoire d’un B-Tree persistant, on va effacer les nœuds des feuilles, et dans le niveau précédent, libérer et mettre à null les pointeurs sur ces nœuds.

On peut envisager de libérer ainsi plusieurs couches de nœuds, on va déjà se contenter de cette couche des feuilles, de loin la plus gourmande en place mémoire.

Les nœuds parent des nœuds éliminés conservent les offsets de leurs enfants, et le lien ayant été mis à null, lors des traitements suivants ces enfants seront rechargés en mémoire si nécessaire.

## Dans la recherche en profondeur

Ici, on scrute les éléments de la dernière couche de la recherche en largeur, stockés dans un B-Tree persistant. On peut aussi libérer les feuilles de cette couche comme précédemment.

Et on utilise une suite de B-Tree volatiles qui optimisent la recherche en évitant de poursuivre une recherche si on rencontre une situation déjà traitée précédemment. Mais on renouvelle cette suite de B-Tree volatiles à chaque nouvelle situation scrutée (voir [Recherche en profondeur](#_Recherche_en_profondeur) ).

# Implantation

On a vu qu’une situation sera représentée en mémoire par 5 octets, chacun de ses bits étant à 1 si une pierre est présente dans la case représentée par ce bit, à 0 sinon.

Pour rendre la comparaison de deux situations la plus naturelle possible, on parcourt les cases de gauche à droite depuis le haut vers le bas. Les cases sont regroupées en groupes de 8 cases, chaque groupe étant codé sur un octet. Dans chaque groupe les cases sont mappées aux bits depuis ceux de poids fort vers ceux de poids faible.

Les 33 cases du plateau classique, les 37 cases du plateau français sont réparties dans des octets représentés par une couleur de plus en plus prononcée dans l’image ci-dessous :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 0 |  |  | 0 | 1 | 2 |  |  |  | 0 |  |  | 0 | 1 | 2 |  |  | | 1 |  |  | 3 | 4 | 5 |  |  |  | 1 |  | 3 | 4 | 5 | 6 | 7 |  | | 2 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  | 2 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | | 3 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |  | 3 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | | 4 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |  | 4 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | | 5 |  |  | 27 | 28 | 29 |  |  |  | 5 |  | 29 | 30 | 31 | 32 | 33 |  | | 6 |  |  | 30 | 31 | 32 |  |  |  | 6 |  |  | 34 | 35 | 36 |  |  | |

Les 8 premières cases, de 0 à 7 sont codées sur les 8 bits du premier octet.

La case d’indice 0 est codée sur le bit de poids fort.

Une situation est facilement comparable à une autre, en comparant les valeurs des 5 octets qui la représentent (ordre total).

On peut donc facilement choisir un représentant parmi l’ensemble des situations équivalentes par symétrie : on choisit la plus petite selon le critère de comparaison ci-dessus.

Après ces 5 octets, on ajoute un octet dont les bits représentent les situations initiales pouvant mener à la situation décrite.

Remarque : Cette comparaison fournit un résultat « naturel » : on souhaite que soit définie comme « plus petite » la situation dans laquelle on rencontre un trou alors que l’autre a une pierre à la place, quand on parcourt les cases de gauche à droite depuis le haut vers le bas.

Dans le plateau classique ci-dessus, si la case 0 est vide, le premier octet vaut 0111 1111 = 0x7F = 127

Si on compare avec une situation où toutes les cases sont remplies, le 1er octet vaut 1111 1111 = 0xFF = 255

Et si la case 7 est vide, alors ce premier octet a pour valeur 1111 1110 = 0xFE = 254

Ainsi cette situation où la case 0 est vide apparait bien « plus petite » quand on compare les 5 octets successivement.

# B-Tree

## Présentation

On a besoin de vérifier qu’une situation obtenue par un mouvement n’a pas déjà été rencontrée (à une symétrie près). L’utilisation de HashCode, Hashsets, a un moment fait illusion, mais la mémoire n’a pas suffi.

On stocke donc les situations dans une structure arborescente de type [B-Tree (wikipedia)](https://en.wikipedia.org/wiki/B-tree). Les situations obtenues à partir des situations initiales sont aussi stockées dans une arborescence de ce type, persistante pour permettre les suspensions, reprises de traitement.

Dans ce contexte, on appelle élément (abréviation elt) une situation stockée dans le B-Tree.

Soit Ordre l’ordre du B-Tree : chaque nœud peut contenir au maximum Ordre – 1 éléments et s’il n’est pas une feuille, jusqu’à Ordre nœuds enfants.

Nombre maximal de nœuds à une profondeur p donnée :

, , ce qui donne

Nombre maximal de nœuds dans l’ensemble de l’arbre pour une profondeur p donnée :

Nombre maximal d’éléments à une profondeur p donnée :

Nombre maximal d’éléments dans l’ensemble de l’arbre pour une profondeur p donnée :

Une situation de plateau classique ou français est décrite par 5 octets, plus 1 octet qui renseigne sur les situations initiales susceptibles d’y mener.

Un nœud contient un tableau de taille fixe de (Ordre – 1) \* 6 octets.

Un nœud ne contient pas nécessairement Ordre – 1 éléments. Les éléments sont triés et placés en début du nœud. Un octet supplémentaire est ajouté en entête du tableau des Ordre - 1 buffers, il contient le nombre d’éléments actuellement stockés dans le nœud.

## Stockage

Je stocke les nœuds d’une même profondeur dans un même fichier.

Soit 0.dat le fichier constitué initialement. Il contiendra les feuilles du B-Tree. Il contient initialement un seul nœud, la racine, vide. Quand cette racine est saturée, l’ajout suivant la scinde en deux, et remonte un élément stocké dans un nouveau nœud, la nouvelle racine, dans un nouveau fichier 1.dat.

Entre-temps, 0.dat contient maintenant deux nœuds. Sa taille utile a doublé.

Le B-Tree est donc stocké dans une suite de fichiers <n>.dat, avec <n> variant de 0 (ses feuilles) à p (sa profondeur). Pour éviter de devoir renommer les fichiers, c’est <p>.dat qui contient la racine.

Le fichier 0.dat contient à tout moment uniquement des feuilles.

Dans ce fichier 0.dat, chaque nœud est représenté par un tableau de 1 + (Ordre – 1) \* 6 octets.

Les autres fichiers <n>.dat contiennent des nœuds qui ont tous des descendants dans le fichier <n – 1>.dat

Dans ces fichiers, chaque nœud est représenté par deux tableaux :

Le premier est de 1 + (Ordre – 1) \* 6 octets, il représente les éléments de ce nœud.

Le second est de Ordre \* 4 octets. Chaque 4-uplet est un entier non signé sur 32 bits, Uint32. Cette valeur est l’index d’un nœud du fichier <n-1>.dat. La position de ce descendant est donc égale au numéro de son index multiplié par la taille du nœud dans <n-1>.dat.

Donc suivant que n=1 (ce fichier pointe sur des éléments du fichier 0.dat ne contenant que des feuilles) ou n>1, le calcul de cet offset sera différent.

Leur chargement en mémoire suivra les mêmes principes, à ceci près :

Un Uint32 rappelle l’index du nœud dans son fichier.

Un booléen est levé à true si le nœud est modifié en mémoire, à false à sa lecture et son enregistrement.

Tous les nœuds contiennent le tableau de 1 + (Ordre – 1) \* 6 octets des situations du nœud et de leur nombre.

Une feuille ne contient pas d’indications sur des descendants qu’elle ne possède pas.

Les autres nœuds contiennent le tableau des index des descendants.

Ils contiennent aussi un tableau de pointeurs sur les versions en mémoire de ces descendants.

Si le programme a besoin d’un descendant et que son pointeur est null, alors il le lit à partir du fichier correspondant.

Les nœuds modifiés sont sauvegardés s’ils doivent être libérés

* A la fin d’une recherche en largeur.
* Si trop de mémoire vive est utilisée, et qu’il faut la réduire afin de pouvoir continuer la recherche.
  + Dans ce dernier cas, seuls les nœuds modifiés au-delà d’une certaine profondeur sont sauvés, et seuls les nœuds chargés à cette profondeur et aux suivantes sont supprimés de la mémoire vive. Ceux des niveaux inférieurs sont conservés en mémoire.

Les méthodes de lecture et d’écriture des données du fichier sont puissantes.

* Le fichier est représenté par un objet de type FileStream.
* On peut connaitre la taille actuelle du fichier par sa propriété Length qui renvoie un entier de type Long.
* On peut régler sa taille actuelle, en plus comme en moins, par sa méthode SetLength().

Mais on peut faire encore mieux :

* On peut aussi demander à lire un tableau d’octet au-delà de sa taille actuelle, simplement, la méthode Read(byte[] buffer, int offset, int count) renvoie un nombre d’octets lus égal à 0.
* On peut aussi demander à écrire un tel tableau au-delà de sa taille actuelle, (Write(byte[] buffer, int offset, int count)) et pas seulement juste après sa fin, le système réserve alors (et semble-t-il initialise à 0) les octets qui séparent la fin actuelle du fichier de l’emplacement où on veut écrire ce buffer.
* On se positionne à l’emplacement voulu pour la lecture ou l’écriture par la méthode Seek().

Les nœuds d’un fichier donné sont stockés de manière à calculer facilement d’indice d’un nœud donné descendant d’un nœud parent.

L’indice du seul nœud de <p>.dat, où <p> est égal à la profondeur du B-Tree, est évidemment 0.

Dans le fichier <i>.dat, qui regroupe les nœuds de la profondeur p-i, le nœud d’indice k a ses Ordre – 1 descendants dans le fichier <i-1>.dat, à partir de l’indice (Ordre – 1) \* k

## Recherche et Insertion

Le B-Tree dont j’ai besoin doit permettre la lecture et l’insertion de nœuds. Je néglige l’opération de suppression, qui ne m’est pas utile ici. L’algorithme de recherche se déduit aisément de la première partie de l’algorithme d’insertion.

### Algorithme général d’insertion

eltAInserer est l’élément à insérer

pile est un tableau [0..profondeur] ' Utile pour l’insertion uniquement

noeudActuel = racine

pour idxProfondeur=profondeur ; idxProfondeur ≥ 0 ; idxProfondeur--

' Renvoie true si eltAInserer est présent dans noeudActuel, false sinon

' idxPosition = -1 si eltAInserer < noeudActuel[0]

' sinon tel que noeudActuel[idxPosition] est le plus grand élément ≤ eltAInserer

si noeudActuel.RechercheDichotomique(eltAInserer, out idxPosition)

return LaPositionEstDejaPresente

fin si

pile[idxProfondeur] = (noeudActuel, idxPosition)

fin pour

' Ici, l’eltAInserer est absent de l’arbre,

' et pile contient les infos nécessaires pour son insertion

nouveauNoeud = null ' utile pour les splits de nœuds

pour idxProfondeur=0 ; idxProfondeur ≤ profondeur ; idxProfondeur++

(noeudActuel, idxPosition) = pile[idxProfondeur]

si noeudActuel non saturé

noeudActuel.Inserer(eltAInserer, nouveauNoeud, idxPosition+1)

return LelementAEteInsere

fin si

Split(noeudActuel, ref eltAInserer, idxPosition+1, ref nouveauNoeud)

fin pour

' Ici, la racine a été scindée, on sauve une nouvelle racine

augmenter la taille de pile ' pour la prochaine insertion

profondeur++

racine = nouveauNoeud

sauver racine dans un nouveau fichier <profondeur>.dat

### Split de nœud

L’opération de Split est la plus complexe.

La difficulté tient à ce qu’on doit opérer la scission avant l’insertion pour ne pas provoquer de débordement dans les tableaux. L’insertion se fait après la scission, soit dans le nœud d’origine, maintenant dépeuplé, soit dans le nouveau nœud ayant acquis environ la moitié des éléments (et enfants) du nœud d’origine.

Dans ce mécanisme, il faut rattacher le nouveau nœud à l’arbre. On utilise l’élément qui est juste inférieur aux éléments transférés dans le nouveau nœud, qu’on va insérer dans le nœud parent, sachant que son enfant supérieur est ce nouveau nœud.

Cet élément qui remonte est soit un de ceux présents dans l’ancien nœud avant sa scission, soit celui qui devait y être inséré.

Un petit raffinement tient au fait que si l’insertion doit se faire dans le nouveau nœud, pour éviter un double déplacement de certains des éléments (et enfants) plus grands que l’élément à insérer, on déplace les éléments de l’ancien nœud dans le nouveau en deux phases, pour laisser un espace qui accueille le nouvel élément.

Paramètres généraux :

Ordre, nommé O dans les schémas, est l’ordre du B-Tree.

Ordre – 1 est le nombre d’éléments, indicés de 0 à Ordre – 2

Si le nœud n’est pas une feuille, Ordre est le nombre de liens vers les enfants, indicés de 0 à Ordre – 1

Paramètres de la méthode :

noeudActuel est le nœud à scinder.

eltAInserer est l’élément à insérer.

noeudSuperieur est un nœud associé à eltAInserer (ou est null si noeudActuel est une feuille).

Il contient des éléments supérieurs à eltAInserer, mais inférieurs à l’éventuel élément de noeudActuel devant lequel doit s’insérer eltAInserer.

idxInsertion est l’indice théorique où insérer eltAInserer dans noeudActuel s’il n’était pas saturé.

Au sortir de la procédure, eltAInserer contient une copie d’un élément qui va devoir prendre place dans le parent (ou participer à la création une nouvelle racine)

Au sortir de la procédure, noeudSuperieur représente le nœud nouvellement créé.

On calcule un index appelé idxPivot (on tient compte de l’éventuelle parité de Ordre)

(Division euclidienne)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parité Ordre | Nb elt max | idxPivot | Nb elt avant idxPivot | Nb elt après idxPivot |
| O=2p | 2p-1 | p-1 | p-1 (indicés de 0 à p-2) | p-1 (indicés de p à 2p-2) |
| O=2p+1 | 2p | p-1 | p-1 (indicés de 0 à p-2) | p (indicés de p à 2p-1) |

Exemples :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ordre | p | nb elt maximum | Idx pivot | nb elt avant pivot | nb elt après pivot |
|  |  | Ordre-1 | Ordre/2-1 |  |  |
| 8 | 4 | 7 | 3 | 3 | 3 |
| 9 | 4 | 8 | 3 | 3 | 4 |
| 16 | 8 | 15 | 7 | 7 | 7 |
| 17 | 8 | 16 | 7 | 7 | 8 |

Situation initiale :

Ordre=8, idxPivot = 3 Ordre=9, idxPivot=3

0

1

2

3

4

5

6

e

0

1

2

3

4

5

6

7

e

Situation finale, idxInsertion ≤ idxPivot, ici idxInsertion=1 :

0

e

1

4

5

6

3

2

0

e

1

4

5

6

3

2

7

Situation finale, idxInsertion = idxPivot+1, ici idxInsertion=4 :

0

1

2

4

5

6

e

3

0

1

2

4

5

6

e

3

7

Situation finale, idxInsertion > idxPivot+1, ici idxInsertion=6 :

0

1

2

5

e

6

4

3

0

1

2

5

e

6

4

3

7

nouveauNoeud = allocation dans le niveau de noeudActuel d’un nouveau nœud vide.

Si idxInsertion ≤ idxPivot

eltRemonte = noeudActuel[idxPivot] ' copie de l’élément qui va remonter dans le parent

Copie de noeudActuel[idxPivot + 1 .. Ordre – 2] 🡪 nouveauNoeud[0 .. Ordre – 3 – idxPivot] ' nb = Ordre – 2 – idxPivot

Si non noeudActuel.Feuille, copie de noeudActuel.Enfants[idxPivot + 1 .. Ordre – 1] 🡪 nouveauNoeud.Enfants[0 .. Ordre – 2 – idxPivot] ' nb = Ordre – 1 – idxPivot

Déplacement de 1 case à droite de noeudActuel[idxInsertion .. idxPivot – 1] ' nb = idxPivot – idxInsertion

Si non noeudActuel.Feuille, déplacement de 1 case à droite de noeudActuel.Enfants[idxInsertion + 1 .. idxPivot]

noeudActuel[idxInsertion] = eltAInserer

si non noeudActuel.Feuille, noeudActuel.Enfants[idxInsertion + 1] = noeudSuperieur

sinon si idxInsertion == idxPivot+1

eltRemonte = eltAInserer

copie de noeudActuel[idxPivot + 1 .. Ordre – 2] 🡪 nouveauNoeud[0 .. Ordre – 3 – idxPivot] ' nb = Ordre – 2 – idxPivot

si non noeudActuel.Feuille

nouveauNoeud[0] = noeudSuperieur

copie de noeudActuel.Enfants[idxPivot + 2 .. Ordre – 1] 🡪 nouveauNoeud.Enfants[1 .. Ordre – 2 – idxPivot]

sinon

eltRemonte = noeudActuel[idxPivot + 1]

copie de noeudActuel[idxPivot + 2 .. idxInsertion – 1] 🡪 nouveauNoeud[0 .. idxInsertion – 3 – idxPivot] ' nb = idxInsertion – 2 – idxPivot

si non noeudActuel.Feuille, copie de noeudActuel.Enfants[idxPivot + 2 .. idxInsertion] 🡪 nouveauNoeud[0 .. idxInsertion – 2 – idxPivot]

nouveauNoeud[idxInsertion – 2 – idxPivot] = eltAInserer

si non noeudActuel.Feuille, nouveauNoeud.Enfants[idxInsertion – 1 – idxPivot] = noeudSuperieur

copie de noeudActuel[idxInsertion .. Ordre − 2] 🡪 nouveauNoeud[idxInsertion – 1 − idxPivot .. Ordre – 3 – idxPivot] ' nb = Ordre – 1 – idxInsertion

si non noeudActuel.Feuille, copie de noeudActuel.Enfants[idxInsertion + 1 .. Ordre − 1] 🡪 nouveauNoeud[idxInsertion – idxPivot .. Ordre – 2 − idxPivot]

noeudActuel.NbElt = idxPivot + 1

nouveauNoeud.NbElt = Ordre – 2 – idxPivot

eltAInserer = eltRemonte

noeudSuperieur = nouveauNoeud

# Organisation des données sur disque

A l’initialisation, un nom de répertoire inexistant ou entièrement vide est fourni. Le répertoire est créé au besoin.

Les résultats des recherches en largeur sont sauvés dans des sous-répertoires qui portent un n° incrémental à partir de 0. Celui d’indice 0 contient les situations initiales. A chaque recherche en largeur, un sous-répertoire est créé pour contenir les résultats de la recherche.

Dans chacun des sous-répertoires, le programme stocke les fichiers <n>.dat qui contiennent les situations obtenues, à une symétrie près, à partir des situations initiales.

A la racine du répertoire le programme stocke un fichier pilote qui contient :

* Une description du plateau
* Un flag indiquant si la recherche est totalement terminée ou non
* L’éventuel état où il a été interrompu
  + Phase 1 ou phase 2
  + Description de la situation à partir de laquelle reprendre la recherche
* La liste des solutions, et pour chacune d’elles
  + La description de sa situation résolue S
  + La liste des mouvements permettant de passer de S à F
    - Si S n’est pas une situation initiale, alors la solution est partielle et doit être consolidée

Les symétries et opérations de manipulation ne sont pas stockées et sont recalculées à l’initialisation.

Exemple du contenu du fichier pilote

<?xml version="1.0"?>

<LeSolitaire>

<!-- liste des coordonnées des cases du plateau -->

<Plateau>

<Case x="2" y="0"/>

<Case x="3" y="0"/>

<!-- suite des autres coordonnées des cases du plateau -->

</Plateau>

<Etat complet="false" reprise="|largeur|profondeur" situation="[les 6 octets sous forme hexa]"/>

<!-- liste des solutions (complètes ou partielles) -->

<Solution situation="[les 6 octets sous forme hexa]">

<!-- liste des mouvements -->

<!— les indices sont ceux des mouvements dans la liste des mouvements possibles

calculés par le programme à partir de la description du plateau -->

<!— ils sont stockés dans l’ordre de leur application -->

<Mvt idx="23"/>

<Mvt idx="12"/>

</Solution>

<!-- suite des autres solutions (complètes ou partielles) -->

</LeSolitaire>

Répertoire racine

**Pilote.xml**

 **0** Répertoire contenant les situations initiales

 **0.dat** Fichier contenant les situations initiales

 **1** Répertoire contenant les situations possibles

par un mouvement à partir des situations initiales

 **0.dat** Fichier contenant ces situations, ce sont les feuilles du B-Tree

…

 **<p>.dat** Fichier contenant le nœud racine du B-Tree

 **2** Répertoire contenant les situations possibles

par deux mouvements à partir des situations initiales

…

# Calcul du fichier

## Initialisation

Algorithme de constitution du B-Tree des situations initiales (profondeur 0)

Soit B le B-Tree initialement vide.

Rappel : chaque position calculée (clé) est un tableau de 6 octets, les bits des 5 premiers décrivant l’état de chaque case du plateau (contient ou non une pierre), le dernier octet est un tableau de 8 flags, chaque flag désignant une des positions initiales depuis lesquelles on peut accéder à la position considérée.

On appelle cet octet F.

On gère un masque M qui vaut initialement 1 et qui est décalé de 1 bit après enregistrement de chaque nouvelle position initiale dans B

M = 1

P : byte[6]

Pour chaque case C de P

P[0..4] = 0xFF

P[C] = 0

Calculer la position symétrique minimale P’ de P

Si P’ n’est pas déjà dans le B-Tree

P’[F] = M

Enregistrer P’ dans le B-Tree

M << 1

Fin Si

Fin Pour

## Croissance

Algorithme de constitution du fichier des situations obtenues à une profondeur p depuis les situations obtenues à la profondeur p-1.

Soit Bp-1 le B-Tree des situations obtenues de profondeur p-1, Bp celui à constituer, initialement vide.

Pour chaque position P dans Bp-1

Pour chaque mouvement potentiel M possible qu’il est possible d’appliquer à P

Créer P’ copie de P

Appliquer M sur P’

Calculer la position symétrique minimale P’’ de P’

Si P’’ est déjà dans Bp, soit P’’’ la valeur de P’’ dans Bp

Maj P’’’ [F] |= P[F]

Sinon

Maj P’’[F] = P[F]

Enregistrer P’’ dans Bp

Fin Si

Fin Si

Fin Pour

Pour optimiser les accès au B-Tree, il est souhaitable de pouvoir effectuer en un seul accès

* La recherche d’une position
* Son insertion si la recherche ne l’a pas trouvé
* La mise à jour éventuelle de son flag F au besoin.

Renvoie 0 si P existe déjà et que sa copie P’ n’a pas été modifiée

Renvoie 1 si P existe déjà mais que sa copie P’ a été modifiée

Renvoie 2 si P n’existait pas déjà et qu’elle a été insérée

int InsertOrUpdate(P)

Si ∃ P’ == P

Si Maj(P’,P)

Sauve P’

Return 1

Return 0

Insert P

Sauve P

Return 2

B-Tree doit disposer de deux fonctions

* Comparaison de P avec toute position P’ déjà enregistrée
  + Renvoie un entier <0 si P < P’, 0 si P == P’, >0 si P > P’
* Maj de P’ grâce à P
  + Renvoie true si P’ a été modifiée par cette fonction de mise à jour, false sinon

# Collecte de statistiques

## B-Tree

Objectif : parcourir un B-Tree pour compter diverses grandeurs

* Profondeur
* Nombre de nœuds par profondeur
* Nombre d’éléments par profondeur

Ces données permettent ensuite de calculer :

* Nombre total de nœuds
* Nombre total d’éléments
* Pourcentage de remplissage moyen de chaque nœud, par profondeur, global

## Stockage

Objectif : fournir des renseignements sur la répartition des situations en fonction des situations initiales

* Sortir les statistiques sur le dernier niveau de recherche en largeur disponible
* Nombre de situations par situation initiale
* Liste des valeurs différentes des flags et nombre de situations avec ce flag

Le flag d’une situation est un byte, donc de 8 bits.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Ce flag n’est jamais égal à 0, les situations initiales ont un flag non nul et les situations qui en découlent ont un flag qui est l’union des flags des situations initiales d’où on peut atteindre celle-ci.

Il y a au plus 8 situations initiales dont les flags sont 2p, 0 ≤ p < 8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00000001 | 00000010 | 00000100 | 00001000 | 00010000 | 00100000 | 01000000 | 10000000 |
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |

Le flag d’une situation quelconque est donc une combinaison de ces flags, sa valeur la plus grande est donnée par

|  |
| --- |
| 11111111 |
| 255 |

On distribue donc ces situations dans un tableau de 256 longs, les cases sont indicées de 0 à 255.

Chaque situation incrémente le compteur de la case égale à son flag. Donc la case d’indice 0 reste à 0.

On gère aussi un tableau de 8 long. Chaque situation incrémente la ou les cases correspondant à chaque bit levé de son flag.

On génère le compte-rendu sous la forme de deux tableaux

Répartition suivant la situation initiale :

|  |  |
| --- | --- |
| **Flag** | **Nombre** |
| aa | 321564 |

Répartition suivant la valeur du flag :

|  |  |
| --- | --- |
| **Flag** | **Nombre** |
| aa | 321564 |

La somme des Nombres du second tableau est égale au nombre total de situations dans le B-Tree.

Celle du premier est supérieure, chaque situation étant comptée autant de fois qu’elle est accessible par une situation initiale différente.

Seules les lignes dont le nombre est ≠ 0 sont fournies.

# Vérification du stock

On ajoute une tâche de vérification du stock constitué.

Profondeur est le niveau de la dernière couche de données calculé.

La vérification suppose que tous les niveaux sont constitués (pas de reprise en cours)

* Vérification que les éléments du niveau n, 0 ≤ n ≤ Profondeur, sont rangés.
* Vérification que chaque élément est normalisé.
* Pour chacun des éléments du niveau n, 0 ≤ n < Profondeur, vérification que les images normalisées sont présentes dans le niveau n+1

Si cette vérification est suspendue, elle n’enregistre pas son état actuel.

# Parcourt de l’arbre

On utilise un itérateur implicite, exploitant le mot clé yield du langage C#. Ce mot clé met en place une classe masquée qui fonctionne comme un itérateur.

La méthode IEnumerable<Element> EnumereElements() permet le parcourt de l’arbre du plus petit au plus grand élément.

Elle peut être utilisée ainsi :

Foreach(var element in EnumereElements()) { … }

Elle fonctionne ainsi : à son premier appel, le code se déroule normalement jusqu’à rencontrer une instruction yield return element.

A ce moment, elle s’interrompt mais à son prochain appel reprendra le traitement juste après ce yield return, avec ses variables locales inchangées.

Il « suffit » d’exécuter ce yield return dans une boucle pour que la reprise de traitement s’exécute dans cette boucle, responsable de la délivrance de l’élément suivant celui qui a été fourni précédemment.

Lorsqu’elle détecte qu’elle a renvoyé le dernier (le plus grand) élément, elle quitte la boucle, ce qui indique que l’énumération est terminée.

Elle gère une pile de couples (noeud, idxElt) de taille fixe [0 .. Profondeur] et d’un entier idxProfondeur. Sa première tâche est d’initialiser ces variables « d’état » :

Pile<(noeud, idxElt)> est une liste permettant la récursion sans fonction récursive

Elle a une taille 1 + Profondeur

Initialiser Pile avec la liste des noeuds de gauche (idxElt==0) depuis la racine jusqu’à la feuille

for(noeud = Racine ;  ; )

Pile.Add((noeud, 0))

If noeud.isFeuille, break

noeud = noeud.Enfants[0]

Next

idxProfondeur est un entier qui pointe dans Pile et débute à la feuille contenant le + petit élément

idxProfondeur = Profondeur

Pile[0] contient la racine , Pile[Profondeur] contient un nœud feuille.

Puis elle rentre dans une boucle « infinie » for( ;; ) dans laquelle elle prépare l’élément à renvoyer et règle ses variables dans l’optique du prochain appel

For( ; il y a encore au moins un élément à renvoyer; )

l’élément courant repéré par Pile et idxProfondeur

elementActuel = Pile[idxProfondeur].noeud.Elements[Pile[idxProfondeur].idxElt]

Régler Pile et idxProfondeur en vue du prochain appel, voir ci-dessous

yield return elementActuel

next

Toute la difficulté est de faire ce réglage. Il faut pointer sur le plus petit élément supérieur à celui qu’on renvoie ici.

if idxProfondeur == Profondeur ' Pile[Profondeur].noeud est une feuille

if Pile[idxProfondeur].idxElt < Pile[idxProfondeur].noeud.nbElt – 1

Pile[idxProfondeur].idxElt++

Else

' Cette feuille est épuisée, il faut trouver un noeud dans Pile qui puisse être utilisé au prochain appel

do

if --idxProfondeur < 0 then yield break ' Fin de l’itération

if Pile[idxProfondeur].idxElt < Pile[idxProfondeur].noeud.NbElt

break

end if

loop

End if

Else

Quand on reviendra à ce niveau, il faudra traiter l’élément suivant de ce nœud

Cet idxElt est aussi l’index de l’enfant qui contient les éléments qui suivent celui en cours de traitement

idxElt = ++Pile[idxProfondeur].idxElt

On va préparer la Pile pour pointer le + petit élément > à l’actuel

for( ; idxProfondeur < Profondeur ; idxProfondeur++)

Pile[idxProfondeur + 1].noeud = Pile[idxProfondeur].noeud.Enfants[idxElt]

A la 1ère passe, on a pris un enfant qui n’était pas à l’index 0, mais ensuite, c’est cet enfant de gauche qu’on prend

Et on prend aussi toujours le 1er élément de ces enfants

Pile[idxProfondeur].idxElt = idxElt = 0

next

A la sortie de la boucle, idxProfondeur == Profondeur : on pointe une feuille de l'arbre

end if

## Variante d’initialisation

Il nous sera utile de débuter la lecture de l’arbre à partir d’un certain élément seulement (cas de reprise de traitement).

On pourrait se contenter de parcourir tout l’arbre et de négliger ceux des éléments qui précèdent celui par qui on souhaite débuter le traitement, mais ce ne serait pas très efficace.

La solution consiste à modifier l’initialisation faite avant d’entrer dans la boucle infinie pour placer la pile et le pointeur dans l’état dans lequel ils seraient dans le parcours complet, juste avant l’appel qui délivrera l’élément souhaité.

Elt est l’élément à partir duquel on souhaite débuter l’énumération

Son absence de l’arbre est considéré comme une erreur

Pile<(noeud, idxElt)> est une liste permettant la récursion sans fonction récursive

idxProfondeur = -1 est un entier qui pointe dans Pile et débute à la feuille contenant le + petit élément

for(noeud = Racine ;  ; )

si RechercheDichotomique(noeud, Elt, out idx)

Pile.Add((noeud, idx))

idxProfondeur = Pile.Count − 1

break

Fin si

If noeud.isFeuille, throw EXCEPTION

Pile.Add((noeud, idx+1))

noeud = noeud.Enfants[idx+1]

next

While Pile.Count ≤ Profondeur

Pile.Add((null,0)) Le contenu du reste de la pile sera recalculé lorsque sera délivré Elt

Wend

# Recherche en profondeur

Le plateau comporte C cases.

Les situations initiales contiennent C – 1 pierres.

La recherche en largeur a exécuté N mouvements à partir des situations initiales

Il a établi l’ensemble des situations atteiganbles à P pierres.

P = C – 1 – N

Le répertoire de stockage contient

* Le répertoire 0 des situations initiales
* Et un répertoire par mouvement effectué dans la recherche en largeur.

Ces répertoires sont donc au nombre de N+1, numérotés de 0 à N.

Il faut encore M mouvements pour établir à partir des situations à P pierres des situations à C − P pierres, duales des situations à P pierres.

C – 1 – N – M = C – P

M = P – 1 − N = C – 2(1 + N)

On utilise une pile PILE à M − 1 cellules, de 0 à M – 2, chaque cellule stocke :

* Un indice de mouvement
* Un B-Tree volatile
* Un enumérateur des situations autorisées issues de la situation du niveau supérieur

Remarque à propos de ces B-Tree volatiles :

On part d’une situation sit0 issue de la dernière couche persistante de recherche en largeur. Cette situation sit0 est associée à un flag flg0 des situations initiales pouvant mener à sit0. On effectue sur sit0 une recherche en profondeur pour tenter de résoudre sit0. Mais on stocke à chacun des niveaux de cette recherche en profondeur les situations rencontrées. Ainsi, lors des retours en arrière, on peut couper dans l’arbre des possibles si on rencontre une situation déjà rencontrée et donc déjà testée.

Si sit0 est résoluble, alors ses situations initiales associées, flg0, le sont aussi. On les ajoute à une liste RES de situations initiales résolues.

Mais est-il utile de conserver cette arborescence de situations déjà rencontrées lors du traitement de sit0 quand on traitera une autre situation sit1 issue de la dernière couche persistante de recherche en largeur ? Si on s’intéresse à sit1, c’est qu’il est associé à un flag flg1 de situations initiales qui est un sur-ensemble **strict** de RES, donc de flg0. Supposons qu’on conserve l’arborescence des situations rencontrées dans le traitement de sit0, et que dans la recherche sur sit1, on rencontre l’une d’elles, disons sit00. On n’a pas enregistré si sit00 mène ou non à une solution. Si sit0 a été résolu, il se peut que ce soit sans passer par sit00. Cette rencontre dans le cadre de la recherche sur sit1 ne nous est pas utile. Pire, conserver cette arborescence sur sit0 peut amener à saturer la mémoire sans fournir de service.

Donc on libère les B-Tree volatiles associés à chaque sit0 issu de la dernière couche persistante de recherche en largeur quand on passe au sit1 suivant.

Pour chaque situation sit0 à P pierres

Si elle référence au moins une situation initiale non résolue

Pour chaque mouvement mvt autorisé sur sit0

sit1 = situation obtenue en appliquant mvt à sit0

Traiter sit1, 0 ' récursif, 0 est la profondeur prf de recherche

Fin Pour

Fin Si

Fin Pour

Traitement de sit1, à la profondeur prf

(description récursive, même si on implémente une procédure non récursive)

Si prf == M – 1

Tester le dual de sit1

Sinon

Si sit1 est dans le B-Tree de PILE[prf], ignorer sit1

Voir (\*)

Insert ou maj de sit1 dans le B-Tree volatile de PILE[prf]

Enregistrement de mvt dans PILE[prf]

Pour chaque mouvement mvt1 autorisé sur sit1

Traiter sit1, mvt1 à la profondeur de recherche prf+1

Fin Pour

Fin Si

(\*) On pourrait aussi tester si la sit1 est l’une des situations des solutions déjà déterminées. Si ses situations initiales sont un sur-ensemble, les nouvelles situations initiales ont une solution qui passe par sit1 et se termine de la manière dont cela a été établi lorsqu’on a rencontré sit1 la fois précédente.

Test du dual d’une situation sit à C-P pierres

Calcul de tis, dual de sit

Si tis n’est pas dans le B-Tree persistant des situations à P pierres, ignorer sit

Enregistrer la situation sit0, la suite de mouvements inscrits dans PILE, le dernier mvt qui a permis de calculer sit

Abandonner le travail sur sit0 pour passer à la situation suivante

Implémentation non récursive

idxP représente la profondeur de recherche, débutant à 0, allant jusqu’à M – 2

Pile simule la stack autrement nécessaire à la récursivité.

Initialement, idxP = 0, Pile[0] contient l’énumérateur E de situations propre à cette profondeur.

L’énumérateur E dispose des infos propres à son état interne, idxMvt et SituationCourante, et aussi le stock volatile des situations déjà rencontrées à ce niveau.

Le premier appel à E.Next() le positionne sur la première situation de l’énumération.

A l’intérieur de la boucle, on a la logique propre au traitement, et celle qui détermine si on doit augmenter ou diminuer idxProfondeur, ce qui simule l’appel récursif ou au contraire la remontée.

for( ; ; )

if idxP < 0 then break

if not Pile[idxP].E.Next() then

idxP--

continue

end if

sit = Pile[idxP].E.SituationCourante

if idxP == M – 2

if dual sit dans stock permanent

enregistre solution partielle

idxP = -1

end if

continue

end if

if sit ∈ Pile[idxP + 1].Stock

continue

end if

Pile[idxP].idxMvt = Pile[idxP].E.idxMvt

idxP++

Pile[idxP].E = new (sit)

next

# Consolidation

Le plateau comporte C cases. Les situations initiales possèdent C – 1 pierres.

La recherche en largeur a produit un ensemble de situations en appliquant L mouvements. Ces situations possèdent P = C – 1 – L pierres.

## Partie centrale

La recherche en profondeur sur une de ces situations à C – 1 – L pierres, disons sit0, effectue M = 2P – C mouvements, disons mi, avec 0 ≤ i < M pour obtenir une situation, disons sitM, à P – M = C – P pierres, c’est-à-dire P cases vides. Cette recherche en profondeur a associé M mouvements à sit0. Mais ces mouvements n’ont pas été appliqués aux images successives de sit0 mais à leurs situations normalisées. L’objectif est de retrouver les mouvements qu’il faut appliquer aux images de sit0 pour obtenir une situation sit'M équivalente à sitM.

L’initialisation établit un tableau à deux entrées T[sym][mvt] qui renvoie un mouvement mvt'.

Imaginons une situation sit, et sa situation normalisée sitN. Celle-ci a été obtenue grâce à une symétrie sym. Si on veut appliquer à sitN un mouvement mvt pour obtenir une situation sitN1, alors T[sym][mvt] fournit le mouvement, disons mvt', qu’il faut appliquer à sit pour obtenir une situation sit1 équivalente à sitN1.

Le mouvement mvt' est celui dont les cases sont les images de celles de mvt par la réciproque de sym.

## Mouvemens finals

Repartons de sit0 et appliquons lui les M mouvements calculés précédemment. On obtient une situation sitM à P cases vides. Si la recherche en profondeur a retenu cette solution, c’est que sitM a pour dual une situation dualM à P pierres qui est présente (à une symétrie près) dans la dernière couche de la recherche en profondeur.

Si dualM est présente dans ce stock, c’est qu’il existe au moins une situation initiale sitIni (à une symétrie près) et une suite de L mouvements qui, appliqués à sitIni mène à dualM.

Si on applique à sitM dans l’ordre inverse les mouvements duaux de ceux de cette suite, on aboutira à la situation duale de sitIni, à une symétrie près, c’est-à-dire à une situation gagnante.

Partons de dualM qui possède P pierres et appliquons les mouvements duals autorisés sur dualM. Chaque situation obtenue contient P+1 pierres et est testée : sa normalisée est-elle dans la couche de stock de la recherche en largeur associée aux situations à P+1 pierres ? Si oui, on conserve la situation pour lui appliquer à nouveau le processus. Si non, on remonte dans la pile de recherche pour trouver un autre mouvement à appliquer. On finit après avoir ainsi empilé L mouvements duaux à une situation initiale.

Il faut remarquer qu’un mouvement dual a exactement la même description que le mouvement normal, simplement là où dans le mouvement normal on enlève une pierre (les deux premières cases du mouvement), dans le mouvement dual on place deux pierres, et inversement pour la troisième case.

Et cette liste est déjà inversée par rapport à une recherche de la situation initiale vers dualM.

## Mouvements initiaux

La situation sit0 qui est associée à la solution partielle possède P pierres et un flag f0 qui renseigne sur la liste des situations initiales qui peuvent mener à elle (à une symétrie près).

Certaines situations initiales ont déjà pu être résolues. Leur liste est synthétisée par un flag fI.

On va partir de sit0 et reconstruire partiellement l’arborescence de ses ascendants en appliquant des mouvements duaux autorisés sur elle et ses ascendants retrouvés pour retrouver la liste de mouvements menant à ces situations initiales (à une symétrie près).

Mais on exploite le stock constitué par la recherche en largeur, f0, fI et les flags des ascendants rencontrés pour limiter cette exploration.

Si la situation normalisée d’un ascendant n’est pas dans le stock, inutile de poursuivre la recherche sur cet ascendant.

Sinon, on consulte le flag de ses situations initiales associées.

Si ce flag fA n’adresse que des situations initiales déjà résolues (fA ⊂ fI), là aussi il est inutile de poursuire la recherche.

Une fois obtenu un ascendant valide (au sens vu ci-dessus) après P-1 mouvements duaux, on est en présence d’une situation initiale non résolue (à une symétrie près).

Alors la liste inverse des mouvements effectués permet de remonter à sit0 depuis cet ascendant.

En la complétant de la liste établie pour résoudre sit0, on obtient une liste de mouvemens qui résolvent cette situation initiale. On stocke cette nouvelle solution, et on met à jour fI pour y inclure cette situation initiale. Et si il arrive à cette occasion que f0 ⊂ fI, alors là aussi la recherche peut être interrompue.

L’algorithme ressemble à celui de la phase de recherche des mouvements finals. Simplement on ne s’arrête pas à la première solution obtenue, car une situation sit0 peut être un point de passage pour la résolution de plusieurs situations initiales. Et on exploite les flags associés pour restreindre la recherche.

## En conclusion

La recherche a pu établir une liste de solutions complète.

On supprime la solution partielle de la liste des solutions du fichier pilote.  
On ajoute cette liste de solutions complètes à cette liste de solutions du fichier pilote.

# IHM

Le programme est appelable par une IHM qui autorise les situations suivantes :

* Initialisation d’une recherche
  + Saisie d’un chemin/nom de fichier texte qui contient la description du plateau
  + Saisie d’un chemin de répertoire dans lequel seront stockées les données et résultats
  + 🡪 Le programme vérifie les entrants et initialise le répertoire
* Choix du répertoire de travail (établi par une précédente initialisation)
  + Inutile si l’initialisation vient de se faire, utile lors d’une reprise
  + 🡪 Le programme vérifie les entrants
* Lancement d’une recherche en largeur
  + 🡪 Le programme vérifie les entrants
  + 🡪 Il lance une tâche de fond qui reprend les situations obtenues après un maximum de mouvements et calcule les prochaines situations
  + 🡪 Cette tâche est à même de détecter qu’une interruption a eu lieu auparavant et de reprendre à partir de cette interruption
* Lancement d’une recherche en profondeur
  + 🡪 Le programme vérifie les entrants
  + 🡪 Il lance une tâche de fond qui reprend les situations obtenues après un maximum de mouvements et calcule les prochaines situations en profondeur à la recherche d’une solution
  + 🡪 Cette tâche est à même de détecter qu’une interruption a eu lieu auparavant et de reprendre à partir de cette interruption
* Consolidation des solutions partielles
  + 🡪 Le programme vérifie les entrants
  + 🡪 Il lance une tâche de fond qui reprend les solutions partielles obtenues pour leur trouver les situations initiales
  + 🡪 Cette tâche est à même de détecter qu’une interruption a eu lieu auparavant et de reprendre à partir de cette interruption
* Suspension de la tâche de recherche (en largeur ou en profondeur ou de consolidation)
  + Levée d’un flag de surveillance
  + 🡪 La tâche de recherche détecte la levée de ce flag
  + 🡪 Elle enregistre son état actuel et s’arrête
* Liste des situations initiales résolues
  + Choix de l’une d’elles
  + Affichage d’une animation qui montre la succession des mouvements pouvant résoudre cette situation