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).

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 du PC n’y suffit pas.

# Optimisations

## Symétries

Le programme tient compte des symétries du plateau. 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 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 rêver de pousser l’analyse jusqu’à obtenir A(C-2)/2. La capacité de nos disques est hélas limitée.

## 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.

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 libéré.] 🡸 A voir. Il pourrait être utile de le conserver pour établir plus efficacement les listes de mouvements d’une solution.

## 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 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.

## 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

On se place au moment où la recherche en profondeur a trouvé une situation à P pierres SP, accessible depuis la liste des situations initiales L (à une symétrie près) dont une au moins n’a pas encore de solution et découvert une situation à TP cases vides duale d’une situation à P pierres S’P présente dans l’ensemble des situations accessibles. Il a aussi connaissance d’une liste de mouvements autorisés L2 qui permet de passer de SP à TP

Le programme traite alors SP et TP de la manière suivante :

Le programme part de TP et recherche en profondeur avec mémorisation la première situation finale F possible en appliquant les mouvements autorisés. Il établit donc une liste de mouvements autorisés L3 permettant de passer de TP à F.

Il part de SP et « remonte » aux situations initiales en appliquant une recherche en profondeur avec mémorisation et en utilisant les mouvements duaux autorisés. Il établit donc une liste de mouvements duaux permettant de passer de SP à une situation initiale I n’ayant pas encore de solution.

En prenant ces mouvements dans l’ordre inverse, il est maintenant possible de passer de I à SP. Appelons les L.

La liste de mouvement obtenue en concaténant L1, L2, L3 permet d’établir une liste L de mouvements résolvant I.

Remarque : Ces recherches peuvent être trop gourmandes en mémoire. Si on a conservé les résultats des recherches successives en largeur, on peut déterminer s’il convient de poursuivre une recherche en testant la présence d’une situation dans ces données conservées. On ralentit cette recherche mais on n’occupe pas d’espace mémoire important.

# 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.

# 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, le programme libère 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.

# 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 situations initiales résolues, et pour chacune d’elles
  + La description de cette situation initiale I
  + Un flag indiquant si la liste complète des mouvements a été établie
  + Si elle a été établie, la liste de ces mouvements,
  + Sinon
  + La situation S1 obtenue à partir de laquelle a été obtenue la situation duale résoluble
  + La situation résoluble S2
  + La liste des mouvements permettant de passer de S1 à S2

Les symétries et opérations de manipulation ne sont pas stockées et sont recalculées au début de chaque traitement.

Exemple du contenu du fichier pilote

<?xml version="1.0"?>

<LeSolitaire>

<Plateau>

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

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

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

</Plateau>

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

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

<!-- c1, c2 indices des cases avec pierre, c3 indice de la case vide -->

<Mvt i="0" c1="2" c2="1" c3="0"/>

<Mvt i="1" c1="2" c2="1" c3="0"/>

<!-- liste des mouvements -->

</Solution>

<!-- liste des autres solutions -->

<!-- Les solutions partielles joignent une situation accessible depuis une ou plusieurs situations initiales à une situation gagnante -->

<SolutionPartielle situationIntermediaire="[les 6 octets sous forme hexa]">

<Mvt i="0" c1="2" c2="1" c3="0"/>

<Mvt i="1" c1="2" c2="1" c3="0"/>

<!-- liste des mouvements -->

</SolutionPartielle>

</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

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

# 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.

Lorsqu’elle détecte qu’elle a renvoyé le dernier (le plus grand) élément, elle quitte en effectuant un yield break, 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