ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC

TIỂU LUẬN HỌC PHẦN
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Đề tài: Áp Dụng Thuật Toán Backtracking Giải Bài Toán Sudoku

Sinh viên thực hiện:
Đỗ Mạnh Hùng
Đoàn Minh Hiển
Nguyễn Đình Việt Anh
Lớp: K65A5 - Khoa học dữ liệu

HÀ NỘI - 2023

Mục lục

Lời nói đầu

1.1 Lý do chọn đề tài

Lý do chọn đề tài của nhóm là Áp Dụng Thuật Toán Backtracking Giải Bài Toán Sudoku là vì nhóm thấy đây là một bài toán thú vị và có nhiều ứng dụng trong thực tế. Sudoku là một trò chơi trí tuệ phổ biến trên thế giới, yêu cầu người chơi phải tư duy, phân tích, suy luận,... Để giải bài toán này, nhóm đã sử dụng thuật toán backtracking, một kỹ thuật quay lui trong lập trình, để thử các khả năng có thể và quay lại khi gặp bế tắc. Thuật toán này có thể giải được bất kỳ bài toán sudoku nào một cách hiệu quả và nhanh chóng.

1.2 Trình bày về đề tài

Giới thiệu: Nhóm sẽ giới thiệu về bài toán sudoku, thuật toán backtracking và mục tiêu của đề tài. Nội dung: Trình bày về cách áp dụng thuật toán backtracking để giải bài toán sudoku, bao gồm các bước thực hiện, các thuật ngữ và các ví dụ minh họa. Kết quả: Trình bày về kết quả đạt được khi áp dụng thuật toán backtracking, bao gồm độ phức tạp, độ chính xác, độ hiệu quả và độ khả thi của phương pháp. Đánh giá: Đánh giá ưu nhược điểm của phương pháp áp dụng thuật toán backtracking, so sánh với các phương pháp khác và đề xuất các hướng phát triển tiếp theo.

1.3 Vấn đề cần giải quyết

- Vấn đề cần giải quyết của việc áp dụng thuật toán backtracking giải bài toán sudoku là làm thế nào để tìm ra một lời giải hợp lệ cho một bảng sudoku cho trước, nếu có. Thuật toán backtracking là một kỹ thuật lập trình đệ quy, trong đó ta thử các khả năng có thể xảy ra cho một ô trống trên bảng sudoku, và kiểm tra xem khả năng đó có vi phạm các quy tắc của bài toán hay không. Nếu không vi phạm, ta tiếp tục thử các khả năng cho các ô trống tiếp theo, cho đến khi tìm ra một lời giải hoặc hết các ô trống. Nếu vi phạm, ta quay lui về ô trống trước đó và thử một khả năng khác, cho đến khi tìm ra một lời giải hoặc hết các khả năng.

- Thuật toán backtracking có ưu điểm là đơn giản và dễ hiểu, nhưng có nhược điểm là tốn nhiều thời gian và bộ nhớ, vì phải duyệt qua nhiều trường hợp không cần thiết.

Thông tin nhóm

2.1 Thành viên nhóm

Nhóm bao gồm 3 thành viên:

- 20002029 - Nguyễn Đình Việt Anh
- 20000394 - Đoàn Minh Hiển
- 20002053 - Đỗ Mạnh Hùng

2.2 Phân chia công việc

Để tạo lên môi trường làm việc chuyên nghiệp, công bằng, và tạo lên kết quả tốt nhất trong công việc thì nhóm đã chia sẻ công việc đều cho các thành viên, mỗi thành viên sẽ phụ trách một mảng chính của đề tài. Trong quá trình làm việc nếu gặp khó khăn khăn, cả nhóm sẽ thảo luận và đưa ra các phương án để giải quyết. Cụ thể:

- Đỗ Mạnh Hùng: Lên ý tưởng, viết chương trình tự động sinh một bảng sudoku để giải với kích thước mong muốn.
- Đoàn Minh Hiển: Áp dụng thuật toán Backtracking giải sudoku.
- Nguyễn Đình Việt Anh: Tìm hiểu kĩ thuật áp dụng tính song song để giải bài toán, triển khai sản phẩm.

Slide và báo cáo cả nhóm sẽ cũng làm và chỉnh sửa.

Cơ Sở Lí Thuyết

3.1 Trò chơi Sudoku

Sudoku là một trò chơi câu đố logic được chơi trên một lưới 9x9, 16x16, 25x25,... Mục tiêu của trò chơi là điền các số từ 1 đến 9 đối với 9x9, từ 1 đến 16 đối với 16 x16,... vào các ô vuông sao cho mỗi số chỉ xuất hiện một lần trong mỗi hàng, cột và lưới nhỏ.

Trò chơi được phát minh ở Mỹ vào cuối những năm 1970, nhưng chỉ trở nên phổ biến trên toàn thế giới vào đầu những năm 2000. Sudoku đã được xuất bản trên nhiều phương tiện khác nhau, bao gồm sách, tạp chí, báo, và các ứng dụng điện thoại di động.

Có nhiều cấp độ khó khác nhau của Sudoku, từ dễ dàng đến cực khó. Các câu đố Sudoku dễ dàng thường có nhiều ô vuông đã được điền sẵn, trong khi các câu đố Sudoku khó hơn có thể chỉ có một vài ô vuông đã được điền sẵn.

Để giải bài toán sudoku, người chơi phải áp dụng các chiến lược khác nhau, như loại trừ, nhóm, đoán và kiểm tra. Ví dụ, nếu một số đã xuất hiện trong một hàng, cột hoặc lưới nhỏ, thì số đó không thể xuất hiện ở bất kỳ ô vuông nào khác trong hàng, cột hoặc lưới nhỏ đó. Tuy nhiên, không phải tất cả các bài toán sudoku đều có thể được giải bằng cách sử dụng các chiến lược này. Một số bài toán sudoku có thể có nhiều hơn một lời giải hoặc không có lời giải nào. Do đó, cần có một phương pháp tổng quát hơn để giải bài toán sudoku, đó là áp dụng thuật toán backtracking.

3.2 Thuật Toán Backtracking

3.2.1 Tổng quan thuật toán

Backtracking là một cách tiếp cận có hệ thống được sử dụng để tìm giải pháp tối ưu cho một vấn đề nhất định bằng cách duyệt qua không gian tìm kiếm bằng phương pháp thử và sai. Nó liên quan đến việc xây dựng dần dần một giải pháp và sau đó loại bỏ nó nếu nó được xác định là không hợp lệ, do đó "quay lại" để khám phá các khả năng khác.

Ý tưởng chính là xây dựng từng bước các giải pháp tiềm năng, đánh giá chúng dựa trên một tập hợp các ràng buộc. Nếu một giải pháp một

phần không đáp ứng được các ràng buộc, chúng tôi quay lại điểm quyết định trước đó và thử các tùy chọn khác, tiếp tục cho đến khi tìm thấy giải pháp hợp lệ hoặc tất cả các khả năng đã hết.

3.2.2 Áp dụng Backtracking vào giải bài toán Sudoku

Chúng ta bắt đầu với một bảng trống và thử các vị trí chữ số khác nhau một cách có hệ thống cho đến khi tìm được giải pháp hợp lệ.

Các bước thực hiện:

- Bước 1: Tìm ô trống trên lưới Sudoku
- Bước 2: Thử các số từ 1 đến 9 vào ô trống.
- Bước 3: Nếu số vi phạm bất kỳ quy tắc Sudoku nào, hãy quay lại và thử chữ số tiếp theo.
- Bước 4: Nếu số không xung đột với bất kỳ chữ số hiện có nào, hãy chuyển sang ô trống tiếp theo và lặp lại các bước 2-4.
- Bước 5: Lặp đi lặp lại đến khi không còn ô trống nào nữa, khi đó giải pháp đã hoàn thành.

3.3 Xử lý đa luồng

3.3.1 Khái niệm về Thread và Multi-Thread

Thread (luồng) về cơ bản là một tiến trình con (sub-process). Một đơn vị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt. Trong Java, các luồng được quản lý bởi máy ảo Java (JVM)

Multi-Threas (đa luồng) là một tiếng trình thực hiện nhiều luồng đồng thời. Một ứng dụng Java ngoài luồng chính có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn.

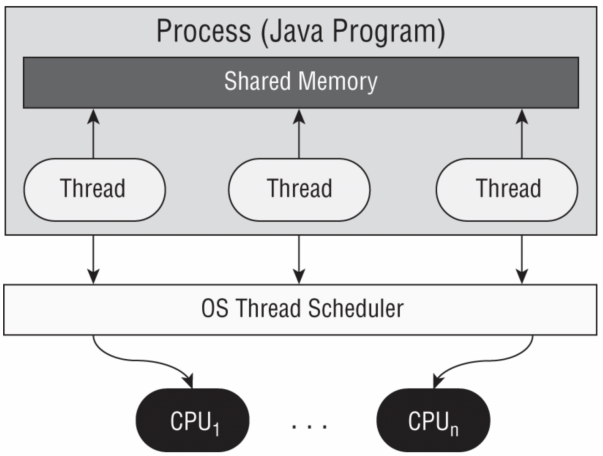
Multitasking là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành. Hệ điều hành quản lý việc này và sắp xếp phù hợp cho các chương trình trong đó. Ví dụ, trên hệ điều hành Windows chúng ta có thể làm việc đồng thời với các chương trình khác

nhau: Microsoft Word, Excel, ... Việc sử dụng đa nhiệm để tận dụng tính căng của CPU.
Đa nhiệm có thể đạt được bằng cách:
1. Đa nhiệm dựa trên đơn tiến trình (Process) - Đa tiến trình (Multiprocessing).

- Mỗi tiến trình có địa chỉ riêng trong bộ nhớ, tức là mỗi tiến trình phân bổ vùng nhớ riêng biệt
- Tiến thình là nặng
- Sự giao tiếp giữa các tiến trình có chi phí cao
- Chuyển đổi từ tiến trình này sang tiếng trình khác đòi hỏi thời gian để đăng ký việc lưu và tải các bản đồ bộ nhớ, các danh sách cập nhật, ...

2. Đa nhiệm dựa trên luồng (Thread) - Đa luồng (MultitThreading)

- Các luồng chia sẻ không gian địa nhớ ô nhớ giống nhau
- Luồng là nhẹ
- Sự giao tiếp giữa các luồng có chi phí thấp



Hình 1: Enter Caption

3.3.2 Ưu điểm và nhược điểm của lập trình đa luồng

Ưu điểm của đa luồng:

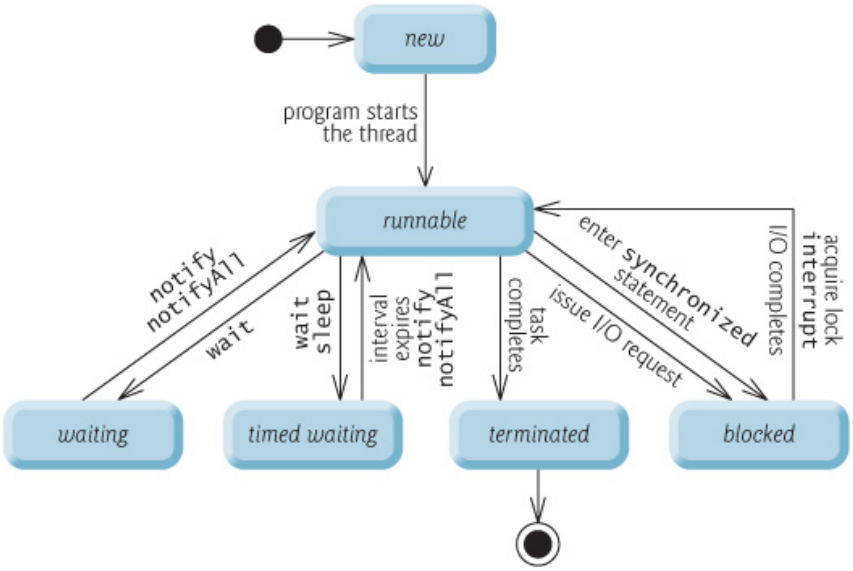
- Nó không chặn người sử dụng vì các luông là độc lập và có thể thực hiện nhiều công việc một lúc
- Mỗi luồng có theere dùng chung và chia sẻ tài nguyên trong quá trình chạy, nhưng vẫn có thể thực hiện một cách hoạt động.
- Luồng là độc lập vì vậy nó không ảnh hưởng đến luồng khác nếu ngoại lệ xảy ra trong một luồng duy nhất
- Có thể thực hiện nhiều hoạt động với nhau để tiết kiệm thời gian.

Nhược điểm của đa luồng:

- Càng nhiều luồng xử lý càng phức tạp
- Xử lý vấn đề tranh chấp bộ nhớ đông bộ dữ liệu phức tạp
- Cần phát hiện tránh các luoofg chết (dead lock), luồng chạy mà không làm gì trong ứng dụng.

3.3.3 Vòng đời của một Thread trong Java

Vòng đời của thread trong java được kiểm soát bởi JVM. Java định nghĩa các trạng hái của luồng trong các thuộc tính static của lớp Thread.State.



Hình 2: Enter Caption

Việc tạo và xử lý các luồng ta sử dụng gói ExecutorService để tạo luồng, phân luồng và quản lý nó.

Triển khai

4.1 Khởi tạo bảng sudoku

Để đơn giản hình dung các bước thực hiện ta sẽ lấy minh họa ví dụ với bảng Sudoku kích thước 9x9.

Các thước thực hiện:

1. Xác định một bảng sudoku bất kì hoàn chỉnh đã được giải.

   Ở đây ta sẽ sinh một bảng sudoku cơ bản nhất như sau:

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	1	5	6	4	8	9	7
5	6	4	8	9	7	2	3	1
8	9	7	2	3	1	5	6	4
3	1	2	6	4	5	9	7	8
6	4	5	9	7	8	3	1	2
9	7	8	3	1	2	6	4	5

Hình 3: Bảng sudoku cơ bản

Ví dụ bảng sudoku trên 9x9 ta sẽ có tổng cộng 3 block theo hàng và 3 bclock theo cột, vì trong mỗi block đã thỏa mãn điều kiện của sudoku nên việc đảo các hàng, cột trong mỗi block hay đảo các block với nhau đều sẽ không ảnh hưởng đến việc sai kết quả.

Hình 4: Code sinh ra bảng sudoku cơ bản

2. Đảo các hàng trong mỗi block với nhau:

   Với việc trong một block đã thỏa mãn điều kiện của sudoku thì việc đảo các hàng trong một block của bảng sudoku ban đầu hoàn toàn không sợ việc bị sai điều kiện.

Hình 5: Code thực hiện đảo hàng trong mỗi block hàng

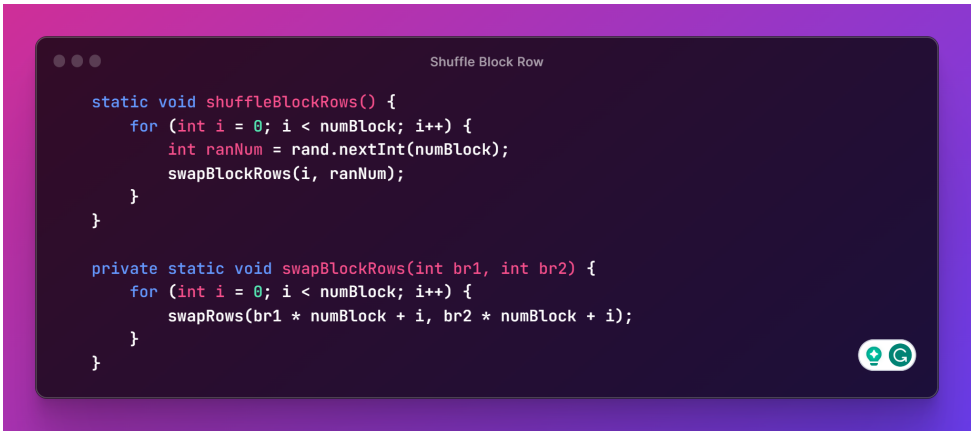
3. Đảo các cột trong mỗi block với nhau:

   Tương tự với việc đảo hàng trong block thì việc đảo các cột trong một block cũng vẫn hoàn toàn đưa ra một sudoku mới thỏa mãn.

Hình 6: Code thực hiện đảo hàng trong mỗi block cột

4. Đảo các block theo hàng:

   Nghĩa là thay vì đảo các hàng, cột trong block cho nhau như bước trước thì ta thực hiện đảo cảm cụm block cho nhau theo hàng khi đó tương ứng với việc ta có một ma trận 3 x 9 và đảo 3 hàng cho nhau, thứ tự 9 cột giữ nguyên, 3 hàng ở đây cụ thể là 3 block theo hàng.



Hình 7: Code thực hiện đảo các block hàng

5. Đảo các block theo cột:

   Tương tự như trên thì việc làm này tương ứng với việc ta có một ma trận 9 x 3 và đảo 3 cột cho nhau, thứ tự 9 hàng giữ nguyên, 3 cột ở đây cụ thể là 3 block theo hàng.

   Sau khi hoàn tất xáo trộn kĩ càng ra được bảng Sudoku sau đây:

Hình 8: Code thực hiện đảo các block hàng

9	7	8	4	6	5	3	1	2
3	1	2	7	9	8	6	4	5
6	4	5	1	3	2	9	7	8
7	8	9	5	4	6	1	2	3
1	6	4	8	9	7	2	3	1
4	5	6	2	1	3	7	8	9
5	6	4	3	2	1	8	9	7
2	3	1	9	8	7	5	6	4
8	9	7	6	5	4	2	3	1

Hình 9: Bảng sudoku sau khi được tráo qua các bước

6. Ẩn đi các ô:

   Bây giờ ta đã có được một bảng sudoku hợp lệ với các vị trí được tráo kĩ và ngẫu nhiên. Sau đó ta sẽ thực hiện ẩn đi các vị trí ngẫu nhiên trên bảng sudoku với quy tắc như sau: Game tạo ra sẽ có 3 mức "Nhập môn", "Cơ bản", "Cao thủ" số lượng ô cần phải điền tương ứng lần lượt là 45%, 60%, 75%. từ đó ta sẽ thực hiện ẩn đi số lượng ô từ bảng sudoku tạo phía trên tương ứng mới cấp độ người chơi chọn. Việc ẩn đi với mỗi bảng sudoku tạo ra là ngẫu nhiên cho các vị trí.

   Đây là kết quả sau khi ẩn đi với mức cao thủ (75% số lượng ô):

9	7	8	0	6	0	0	0	2
0	1	2	0	0	0	0	0	0
0	0	5	0	3	2	9	7	8
0	0	9	0	0	6	1	2	3
1	0	0	8	0	7	0	0	1
4	0	0	0	0	3	7	8	0
5	0	0	0	0	1	0	9	0
2	3	0	0	8	0	0	6	4
8	0	0	6	5	4	2	3	1

Hình 10: Bảng sudoku sau khi ẩn 75% số lượng ô

Hide Cell

```
static void hideCell(double rate) {
    boardHide = new int[n][n];
    // Coppy board to boardHine
    for (int k = 0; k < n; k++) {
        for (int l = 0; l < n; l++) {
            if (board[k][l] != 0) {
                boardHide[k][l] = board[k][l];
            }
        }
    }

    // Hide Cell
    for (int i = 0; i < n * n * rate; i++) {
        int row = rand.nextInt(n);
        int col = rand.nextInt(n);
        boardHide[row][col] = 0;
    }
}
```

Hình 11: Code thực hiện ẩn theo tỉ lệ "rate" tương ứng

Từ kết quả trên sẽ được đưa qua thuật toán Backtracking để điền lại toàn bộ các ô đã bị ẩn.

4.2 Áp Dụng Thuật Toán Backtracking giải Sudoku

Để áp dụng thuật toán Backtracking, một số hàm cơ bản ta cần khởi tạo để giải bài toán Sudoku:

1. Kiểm tra điều kiện

```java
private static boolean isNumberInRow(int[][] board, int number, int row) {
        for (int i = 0; i < GRID_SIZE; i++) {
                if (board[row][i] == number) {
                        return true;
                }
        }
        return false;
}
```

Hình 12: Kiểm tra điều kiện của hàng

```java
private static boolean isNumberInColumn(int[][] board, int number, int col) {
        for (int i = 0; i < GRID_SIZE; i++) {
                if (board[i][col] == number) {
                        return true;
                }
        }
        return false;
}
```

Hình 13: Kiểm tra điều kiện của cột

```java
private static boolean isNumberInBox(int[][] board, int number, int row, int col) {
        int localBoxRow = row - row % 3;
        int localBoxCol = col - col % 3;
        for (int i = localBoxRow; i < localBoxRow + 3; i++) {
                for (int j = localBoxCol; j < localBoxCol + 3; j++) {
                        if (board[i][j] == number) {
                                return true;
                        }
                }
        }
        return false;
}
```

Hình 14: Kiểm tra điều kiện của ô

2. Tính hợp lệ của ô đã chỉ định

```
private static boolean isValidPlacement(int[][] board, int number, int row, int col) {
        return !isNumberInRow(board, number, row) && !isNumberInColumn(board, number, col)
                        && !isNumberInBox(board, number, row, col);
}
```

Hình 15: Tính hợp lệ của ô vị trí đã chỉ định

3. Hàm giải

```
private static boolean solveBoard(int[][] board) {
        for (int row = 0; row < GRID_SIZE; row++) {
                for (int col = 0; col < GRID_SIZE; col++) {
                        if (board[row][col] == 0) {
                                for (int numberToTry = 1; numberToTry <= GRID_SIZE; numberToTry++) {
                                        if (isValidPlacement(board, numberToTry, row, col)) {
                                                board[row][col] = numberToTry;
                                                if (solveBoard(board)) {
                                                        return true;
                                                } else {
                                                        board[row][col] = 0;
                                                }
                                        }
                                }
                                return false;
                        }
                }
        }
        return true;
}
```

Hình 16: Áp dụng Backtracking vào giải Sudoku

4.3 Áp dụng xử lý đa luồng trong giải Sudoku

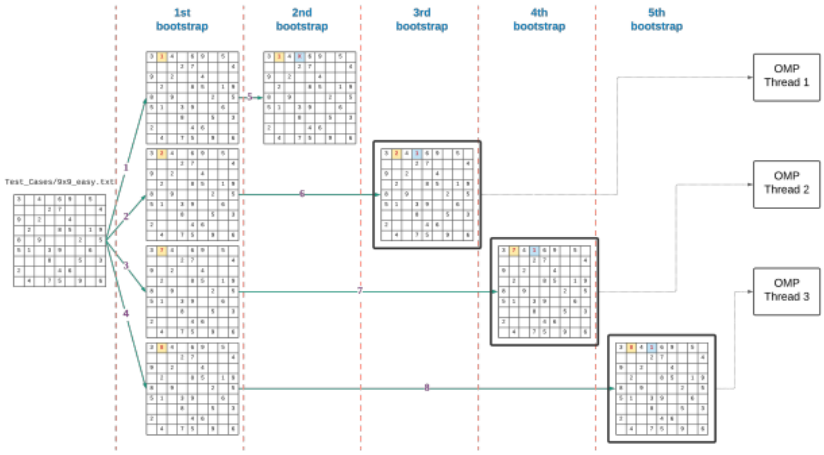
4.3.1 Thuật toán Brute-Force

Thuật toán Brute-Force là một thuật toán thô sơ, nó sẽ liệt kê tất cả các tổ hợp có thể của mỗi ô trống. Thuật toán này sẽ sinh ra một không gian mẫu vô cùng lớn. Việc giải tuần tự của thuật toán này sẽ tốn kém nhiều công sức và thời gian. Tuy nhiên, đây là thuật toán có thể ứng dụng xử lý đa luồng, bằng cách để mỗi luồng giải một tổ hợp trong đó. Điều này cho phé cải thiện hiệu suất của thuật toán Brute-Force, các bước thực hiện như sau:

1. Đẩy các bảng đầu vào lên ngăn xếp để khởi tạo danh sách các giải pháp trung gian. Tiếp theo đó, ta lấy ra từng bảng trong danh sách và điền vào các ô trống của mỗi bảng, và tiếp tục đưa vào danh sách.

2. Sau khi đã tiến hành đủ và loại bỏ các bảng không còn giải được tiếp, ta sẽ phân các bảng này vào từng thread để nó xử lý.

3. Mô hình hóa dưới đây đại diện cho quá trình này:

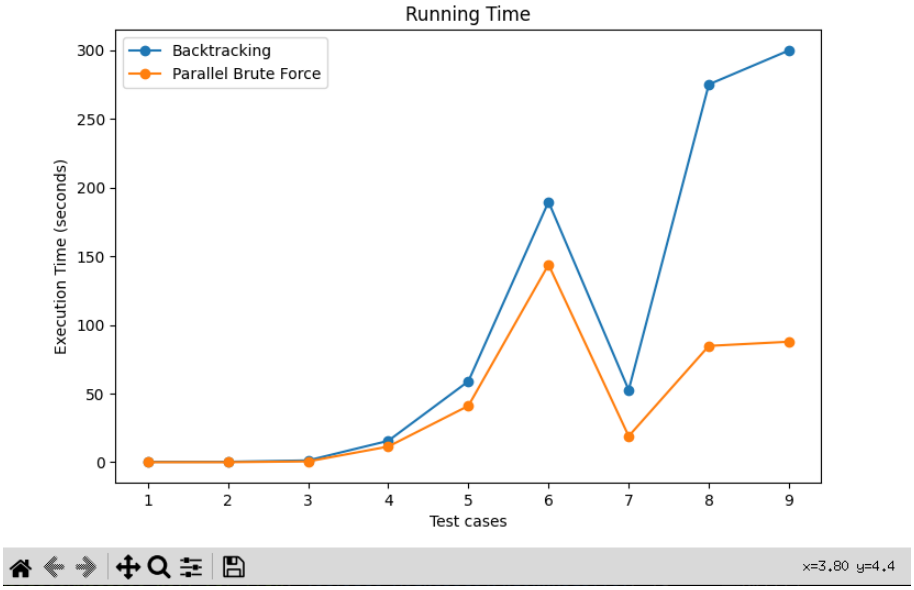


Hình 17: Lấy mẫu bằng phương pháp Brute-Force

Kết quả

5.1 So sánh thời gian chạy giữa 2 phương pháp

Sau khi tiến hành chạy thử nghiệm trên tập đánh giá, ta nhận được kết quả sau đây



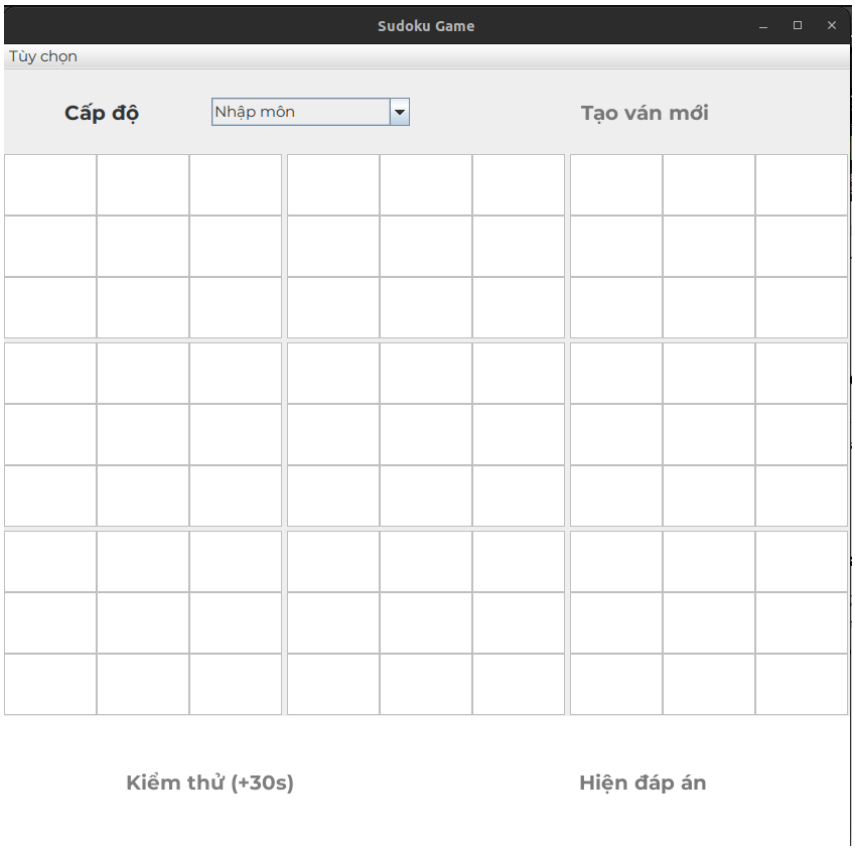
Hình 18: Enter Caption

Sau thực hiện đánh giá ta nhận thấy việc tứng dụng đa luồng trong xử lý bài toán cần quan tâm đến số lượng luồng ta sử dụng và độ phức tạp của bài toán. Với các bảng đơn giản thì Backtracking và đa luồng không chênh lệch quá nhiều, nhưng khi bảng trở nên phức tạp hơn ta thấy đa luồng tốt hơn hẳn tuy nhiên phải đánh đổi bài tài nguyên mà phương pháp này sử dụng.
Trong phạm vi nghiên cứu của đề tài, nhóm đa chứng minh được hiệu quả của việc ứng dụng đa luồng vào giải quyết bài toán Sudoku. Và nhóm nghiên cứu rút ra được những ý sau:

- Sử dụng đa luồng có hiệu quả trong các bài toán phức tạp, nhưng tốn dung lượng tính toán hơn
- Tầm quan trọng của thuật toán, vì có nhiều thuật toán được tối ưu và có các ràng buộc rõ ràng sẽ khiến số lựa chọn giảm đi, từ đó tăng tốc độ giải quyết.

5.2 Sản phẩm đạt được

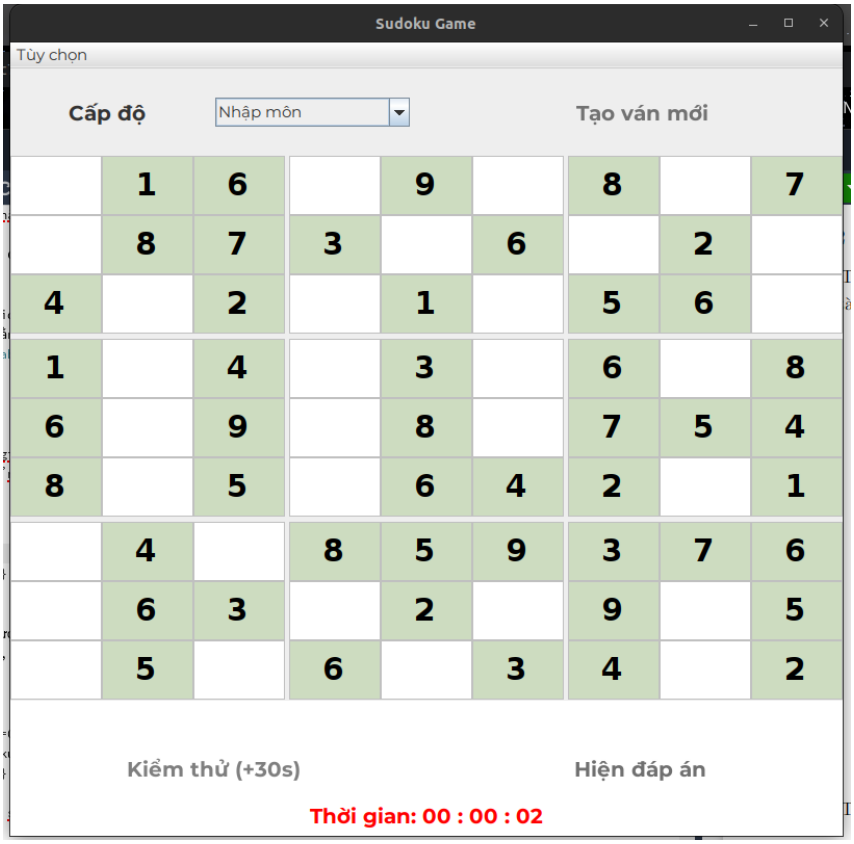
Trong quá trình nghiên cứu, nhóm có làm một sản phẩm liên quan đến đề tài là trò chơi Sudoku theo kích thước chuẩn là 9x9.



Hình 19: Trò chơi Sudoku

Trò chơi gồm các thông tin sau:

- Trò chơi bao gồm 3 mức độ chơi gôm dễ, trung bình, khó và có thời gian theo dõi.
- Trò chơi có chức năng gợi ý dùng để kiểm tra xem các ô điền đã đúng chưa. Nếu chưa đúng sẽ có cảnh báo các ô sai và cộng 30s vào thời gian chơi.
- Lựa chọn hiện đáp án nhằm để demo kết quả đạt được và yêu cầu mật khẩu.
- Khi vào game ta cần chọn "Tạo ván mới" để bắt đầu chơi, thời gian sẽ bắt đầu tính từ đây.
- Có các lựa chọn để tạm dừng, tiếp tục và thoát trong trò chơi.



Hình 20: Bắt đầu trò chơi

Lời cảm ơn

Chúng em xin bày tỏ lòng biết ơn chân thành và sâu sắc đến thầy Vũ Đức Minh và thầy Trần Bá Tuấn đã dày công truyền đạt kiến thức và tận tình hướng dẫn, giúp đỡ chúng em hoàn thành tốt bài tiểu luận. Trong thời gian làm việc với thầy, nhóm không ngừng tiếp thu thêm kiến thức bổ ích mà còn học tập được tinh thần làm việc, thái độ nghiên cứu khoa học nghiêm túc, hiệu quả, đây là những điều vô cùng cần thiết cho nhóm trong quá trình học tập và công tác sau này.

Mặc dù nhóm đã có nhiều cố gắng để thực hiện tiểu luận một cách hoàn chỉnh nhất. Song do kiến thức là vô hạn cùng thời gian hoàn thành còn hạn chế nên bài tiểu luận của chúng em không tránh khỏi những sai sót nhất định. Chúng em rất mong nhận được sự góp ý của thầy để chúng em có điều kiện hoàn thiện hơn sản phẩm của mình.

Chúng em xin chân thành cảm ơn!

Nguyễn Đình Việt Anh
Đoàn Minh Hiển
Đỗ Mạnh Hùng

Tài liệu tham khảo

[1] https://www.baeldung.com/java-executor-service-tutorial

[2] https://www.geeksforgeeks.org/sudoku-backtracking-7/

[3] https://www.javatpoint.com/java-swing

[4] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser - Data Structure and Algorithms in Java-Wiley (2014)