BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC CẦN THƠ**

KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BỘ MÔN KHOA HỌC MÁY TÍNH

------🙢⯎🙠*------*



**BÁO CÁO ĐỒ ÁN**

**TRÍ TUỆ NHÂN TẠO**

**GIẢI THUẬT DI TRUYỀN  
CHO BÀI TOÁN SUDOKU**

**CÁN BỘ GIẢNG DẠY NHÓM SV THỰC HIỆN:**

**ThS. Võ Tri Thức Lã Thái Hòa; MSSV: B2113309**

**Ngô Trọng Phúc; MSSV: B2106805**

Ngành: Khoa Học Máy Tính– Khóa: 47

**Tháng 11/2023**

MỤC LỤC

[MỤC LỤC i](#_Toc151367990)

[DANH SÁCH HÌNH iii](#_Toc151367991)

[**CHƯƠNG I 1**](#_Toc151367992)

[**GIỚI THIỆU BÀI TOÁN SUDOKU VÀ THUẬT TOÁN DI TRUYỀN 1**](#_Toc151367993)

[1.1. Giới thiệu bài toán SUDOKU 1](#_Toc151367994)

[1.1.1. Định nghĩa bài toán 1](#_Toc151367995)

[1.1.2. Luật chơi 1](#_Toc151367996)

[1.1.3. Các mức độ khó 1](#_Toc151367997)

[1.2. Giới thiệu giải thuật di truyền 2](#_Toc151367998)

[1.2.1. Ý tưởng chính của giải thuật 2](#_Toc151367999)

[1.2.2. Các khái niệm cơ bản 2](#_Toc151368000)

[1.2.3. Các bước của giải thuật di truyền 2](#_Toc151368001)

[1.2.4. Ưu điểm và nhược điểm 3](#_Toc151368002)

[**CHƯƠNG II 4**](#_Toc151368003)

[**MÔ TẢ GIẢI THUẬT DI TRUYỀN CHO BÀI TOÁN SUDOKU 4**](#_Toc151368004)

[2.1. Tổng quát 4](#_Toc151368005)

[2.1.1. Các bước giải thuật 4](#_Toc151368006)

[2.1.2. Thư viện và các hằng được sử dụng 4](#_Toc151368007)

[2.1.3. Các cấu trúc được sử dụng 5](#_Toc151368008)

[2.1.3.1. Cấu trúc cá thể (Individual) 5](#_Toc151368009)

[2.1.3.2. Cấu trúc quần thể (Population) 5](#_Toc151368010)

[2.1.4. Các hàm bổ trợ 5](#_Toc151368011)

[2.1.4.1. Đọc dữ liệu bảng sudoku từ file 5](#_Toc151368012)

[2.1.4.2. Hàm in bảng sudoku ra màn hình 6](#_Toc151368013)

[2.2. Chi tiết các bước giải thuật 7](#_Toc151368014)

[2.2.1. Khởi tạo quần thể 7](#_Toc151368015)

[2.2.2. Đánh giá độ thích nghi 10](#_Toc151368016)

[2.2.3. Sắp xếp quần thể theo độ thích nghi giảm dần (fitness tăng dần) 12](#_Toc151368017)

[2.2.4. Chọn lọc cá thể 15](#_Toc151368018)

[2.2.5. Lai tạo 16](#_Toc151368019)

[2.2.6. Đột biến 22](#_Toc151368020)

[2.2.7. Di truyền quần thể 24](#_Toc151368021)

[2.2.8. Hàm main 26](#_Toc151368022)

[**CHƯƠNG III 28**](#_Toc151368023)

[**KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 28**](#_Toc151368024)

[3.1. Kết quả của giải thuật 28](#_Toc151368025)

[3.2. Kết luận 29](#_Toc151368026)

[3.3. Hướng phát triển 29](#_Toc151368027)

DANH SÁCH HÌNH

*Trang*

[**Hình 1. Thư viện và các hằng. 4**](#_Toc151208439)

[**Hình 2. Khai báo cấu trúc cá thể (Individual)). 5**](#_Toc151208440)

[**Hình 3. Khai báo cấu trúc quần thể (Population). 5**](#_Toc151208441)

[**Hình 4. Cài đặt hàm loadFile. 6**](#_Toc151208442)

[**Hình 5. Cài đặt hàm printGrid. 6**](#_Toc151208443)

[**Hình 6. Cài đặt hàm valueExistInBlock. 7**](#_Toc151208444)

[**Hình 7. Cài đặt hàm initializeIndividual. 8**](#_Toc151208445)

[**Hình 8. Cài đặt hàm initializePopulation. 9**](#_Toc151208446)

[**Hình 9. Cài đặt hàm evaluateFitness. 12**](#_Toc151208447)

[**Hình 10. Minh họa quá trình sắp xếp các cá thể trong quần thể. 13**](#_Toc151208448)

[**Hình 11. Cài đặt hàm sortPopulation. 14**](#_Toc151208449)

[**Hình 12. Minh họa quá trình chọn lọc. 15**](#_Toc151208450)

[**Hình 13. Cài đặt hàm selectFromPopulation. 16**](#_Toc151208451)

[**Hình 14. Minh họa quá trình lai tạo tạo ra cá thể mới. 17**](file:///C:\Study\HK%20I%202023\TTNT\Báo%20cáo\Báo%20cáo%20sudoku_updated.docx#_Toc151208452)

[**Hình 15. Cài đặt hàm containsValue. 18**](#_Toc151208453)

[**Hình 16. Cài đặt hàm \*generateRandomNumbers. 19**](#_Toc151208454)

[**Hình 17. Cài đặt hàm crossover. 21**](#_Toc151208455)

[**Hình 18. Minh họa quá trình chọn hoán đổi 2 ô trong block. 22**](file:///C:\Study\HK%20I%202023\TTNT\Báo%20cáo\Báo%20cáo%20sudoku_updated.docx#_Toc151208456)

[**Hình 19. Cài đặt hàm swapTwoCells. 23**](#_Toc151208457)

[**Hình 20. Cài đặt hàm mutate. 24**](#_Toc151208458)

[**Hình 21. Cài đặt hàm genetic. 26**](#_Toc151208459)

[**Hình 22. Cài đặt hàm main. 27**](#_Toc151208460)

[**Hình 23. Bảng SUDOKU đầu vào. 28**](file:///C:\Study\HK%20I%202023\TTNT\Báo%20cáo\Báo%20cáo%20sudoku_updated.docx#_Toc151208461)

[**Hình 24. Bảng SUDOKU sau khi khởi chạy thuật toán di truyền. 28**](file:///C:\Study\HK%20I%202023\TTNT\Báo%20cáo\Báo%20cáo%20sudoku_updated.docx#_Toc151208462)

CHƯƠNG I

GIỚI THIỆU BÀI TOÁN SUDOKU VÀ THUẬT TOÁN DI TRUYỀN

* 1. Giới thiệu bài toán SUDOKU
     1. Định nghĩa bài toán

**Sudoku** là một trò chơi giải đố ô chữ có nguồn gốc từ Nhật Bản. Trò chơi này được phát triển bởi nhà toán học người Mỹ Howard Garns và được xuất bản lần đầu tiên vào năm 1979. Sudoku nhanh chóng trở nên phổ biến trên toàn thế giới và được coi là một trong những trò chơi giải đố phổ biến nhất hiện nay.

Sudoku được chơi trên một lưới 9x9 gồm 81 ô. Mỗi ô chứa một số từ 1 đến 9. Mục tiêu của trò chơi là điền các số vào các ô còn trống sao cho mỗi hàng, mỗi cột và mỗi ô 3x3 đều chứa các số từ 1 đến 9, không trùng lặp.

* + 1. Luật chơi

Sudoku có luật chơi đơn giản như sau:

* Mỗi ô chứa một số từ 1 đến 9.
* Mục tiêu của trò chơi là điền các số vào các ô còn trống sao cho thoả mãn đồng thời 3 ràng buộc:
  + Các số trong cùng 1 hàng không được trùng nhau
  + Các số trong cùng 1 cột không được trùng nhau
  + Các số trong cùng 1 khối 3x3 không được trùng nhau
    1. Các mức độ khó

Sudoku có nhiều mức độ khó khác nhau. Mức độ khó của bài toán được xác định bằng sô lượng ô đã có khi trò chơi bắt đầu. Thông thường mức dễ sẽ có trước 20 ô, còn đối với mức khó là 10 ô.

* 1. Giới thiệu giải thuật di truyền
     1. Ý tưởng chính của giải thuật

Giải thuật di truyền (Genetic Algorithm, GA) là một thuật toán tìm kiếm dựa trên ý tưởng mô phỏng quá trình tiến hóa của các sinh vật. Giải thuật này bắt đầu với một quần thể các cá thể, sau đó các cá thể này sẽ được chọn lọc và lai tạo để tạo ra một quần thể mới. Quần thể mới này sẽ có một số lượng cá thể đột biến phụ thuộc vào tỉ lệ đột biến. Quá trình chọn lọc – lai tạo – đột biến sẽ được tiếp tục cho đến khi tìm thấy một giải pháp tối ưu cho bài toán.

* + 1. Các khái niệm cơ bản

1. Quần thể (population): quần thể là một tập hợp các cá thể. Mỗi cá thể là một giải pháp tiềm năng cho bài toán.
2. Độ thích nghi (fitness): độ thích nghi là một giá trị đo lường mức độ tốt của một cá thể. Quy ước cá thể có fitness thấp hơn sẽ được coi là tốt hơn.
3. Chọn lọc (selection): chọn lọc là quá trình chọn ra các cá thể phù hợp để tạo ra một quần thể mới.
4. Lai tạo (crossover): lai tạo là quá trình tạo ra các cá thể mới từ hai cá thể khác.
5. Đột biến (mutate): đột biến là quá trình thay đổi ngẫu nhiên một hoặc một số tính chất của một cá thể.
   * 1. Các bước của giải thuật di truyền

Giải thuật di truyền có thể được mô tả như sau:

1. Khởi tạo quần thể: quần thể ban đầu được tạo ra ngẫu nhiên, mỗi cá thể trong quần thể là một giải pháp tiềm năng cho bài toán.
2. Đánh giá độ thích nghi: fitness của mỗi cá thể trong quần thể được tính toán.
3. Chọn lọc: chọn lọc các cá thể phù hợp để tạo ra một quần thể mới.
4. Lai tạo: chọn ra các cá thể trong quần thể mới làm bố mẹ rồi lai tạo để tạo ra các cá thể mới.
5. Đột biến: một số cá thể trong quần thể mới có thể bị đột biến.
6. Tiếp tục các bước 2-5: Các bước 2-5 được lặp lại cho đến khi tìm thấy một giải pháp tối ưu cho bài toán.
   * 1. Ưu điểm và nhược điểm
7. Ưu điểm:

* Giải thuật di truyền có thể tìm kiếm giải pháp cho các bài toán khó mà các thuật toán khác không thể tìm thấy.
* Giải thuật di truyền có thể tìm kiếm giải pháp cho các bài toán có không gian tìm kiếm rộng, có số lượng biến lớn.
* Giải thuật di truyền có thể tìm kiếm giải pháp cho các bài toán có hàm giá trị mục tiêu phức tạp.

1. Nhược điểm:

* Giải thuật di truyền có thể mất nhiều thời gian để tìm kiếm giải pháp.
* Giải thuật di truyền có thể không tìm thấy giải pháp tối ưu cho các bài toán khó.

CHƯƠNG II

MÔ TẢ GIẢI THUẬT DI TRUYỀN CHO BÀI TOÁN SUDOKU

* 1. Tổng quát
     1. Các bước giải thuật

Giải thuật di truyền cho bài toán sudoku gồm các bước:

**B1:** Khởi tạo quần thể.

**B2:** Đánh giá độ thích nghi.

**B3:** Sắp xếp quần thể.

**B4:** Chọn lọc.

**B5:** Lai tạo.

**B6:** Đột biến.

**B7:** Lặp lại bước 2 đến bước 6 cho đến khi tìm thấy kết quả hoặc số lần lặp đã đạt đến một giới hạn nào đó cho trước.

* + 1. Thư viện và các hằng được sử dụng

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Thư viện và các hằng.

* + 1. Các cấu trúc được sử dụng

Cấu trúc cá thể (Individual) gồm 3 trường:

* **grid[SIZE][SIZE]**: ma trận 9 x 9 lưu bảng sudoku.
* **fixedValueCells[SIZE][SIZE]**: ma trận 9 x 9 lưu vị trí các ô được cho trước giá trị.
* **fitness**: độ thích nghi (fitness càng thấp độ thích nghi càng cao).

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Hình . Khai báo cấu trúc cá thể (Individual)).

* + - 1. Cấu trúc quần thể (Population)

Là **1 mảng** chứa các **cá thể (Individual).**

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Hình . Khai báo cấu trúc quần thể (Population).

* + 1. Các hàm bổ trợ
       1. Đọc dữ liệu bảng sudoku từ file



Hình . Cài đặt hàm loadFile.

* + - 1. Hàm in bảng sudoku ra màn hình

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

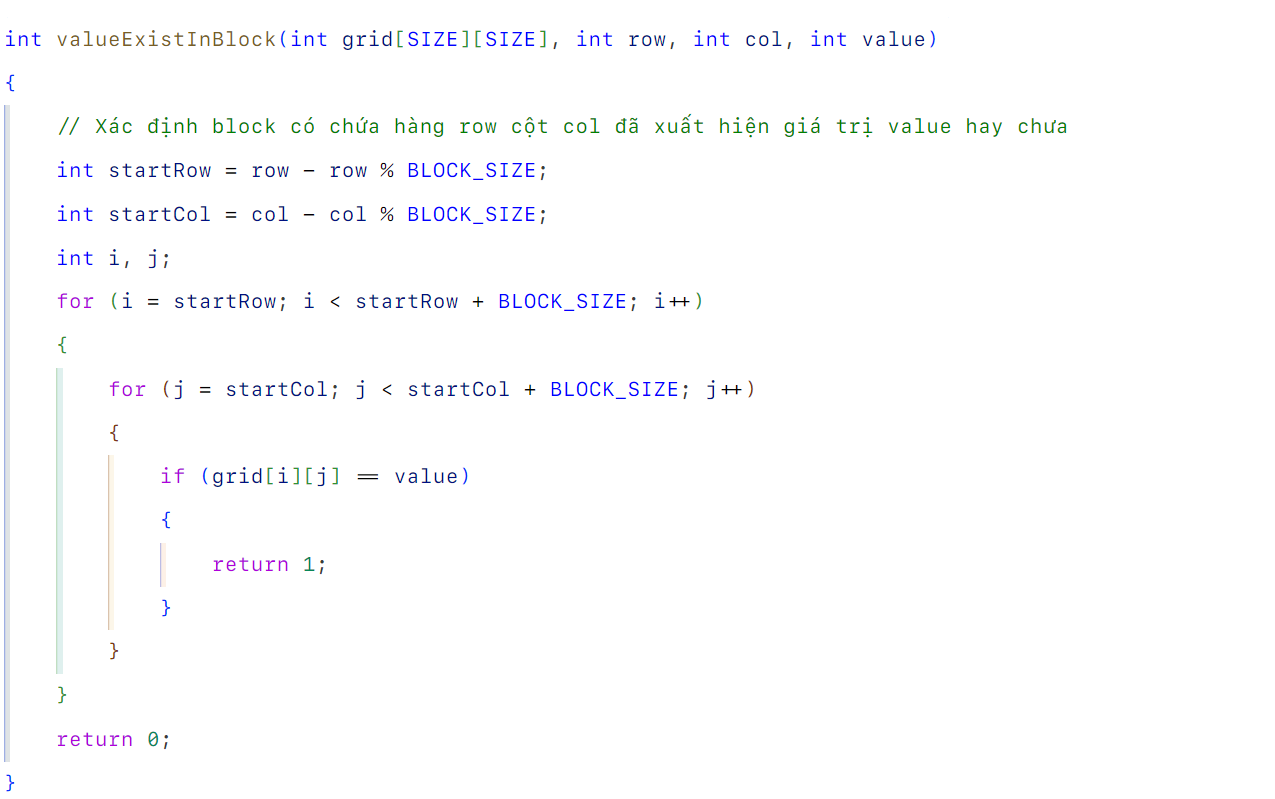
Mô tả được tạo tự động

Hình . Cài đặt hàm printGrid.

* 1. Chi tiết các bước giải thuật
     1. Khởi tạo quần thể

Trong bước này, quần thể ban đầu được tạo ra ngẫu nhiên, mỗi cá thể trong quần thể là một bảng Sudoku đã được điền một phần. Các ô trống sẽ được gán giá trị ngẫu nhiên từ 1 đến 9 sao cho trong mỗi block 3x3 không có giá trị bị lặp lại (thoả mãn ràng buộc thứ 3 đã được nói đến ở phần luật chơi).

Có một số cách để khởi tạo quần thể. Một cách đơn giản là sử dụng hàm **rand()** của thư viện tiêu chuẩn C để gán ngẫu nhiên giá trị cho các ô trống (Một cách khác là sử dụng thuật toán **backtracking** để điền các ô trống một cách hợp lệ).



Hình . Cài đặt hàm valueExistInBlock.

Hàm **valueExistInBlock()** dùng để kiểm tra giá trị (Value) đã xuất hiện trong block hay chưa.

**B1**: xác định block chứa hàng row, cột col.

int startRow = row - row % BLOCK\_SIZE;

int startCol = col - col % BLOCK\_SIZE;

startRow, startCol lần lượt là hàng đầu tiên, cột đầu tiên của block chứa row, col.

VD: row = 2, col = 3, thì startRow tương ứng bằng 0, startCol tương ứng bằng 3. Do hàng 2 cột 3 trong bảng sudoku tương ứng với block số 1, hàng đầu tiên của block số 1 là hàng 0, cột đầu tiên của block số 1 là cột 3.

**B2**: Duyệt qua các ô trong block tương ứng và so sánh giá trị value với giá trị của ô, nếu có bất cứ ô nào mang giá trị value thì hàm trả về 1 (true) còn nếu như hàm duyệt hết block mà không có ô nào mang giá trị value thì hàm trả về 0 (false).

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự độngẢnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm initializeIndividual.

Hàm **initializeIndividual()** sẽ khởi tạo ngẫu nhiên cá thể sao cho mỗi block 3x3 không có giá trị nào bị lặp lại (thoả mãn ràng buộc thứ 3).

**B1**: gán giá trị inputGrid vào cá thể.

**B2**: điền các ô trống bằng các giá trị ngẫu nhiên từ 1 - 9.

Các giá trị điền vào ô trông được kiểm tra bằng hàm **valueExistInBlock** để kiểm tra xem giá trị này có bị lặp lại trong block chứa ô đó hay không. Nếu có thì hàm sẽ tiếp tục chạy lại để sinh ra một giá trị ngẫu nhiên khác.

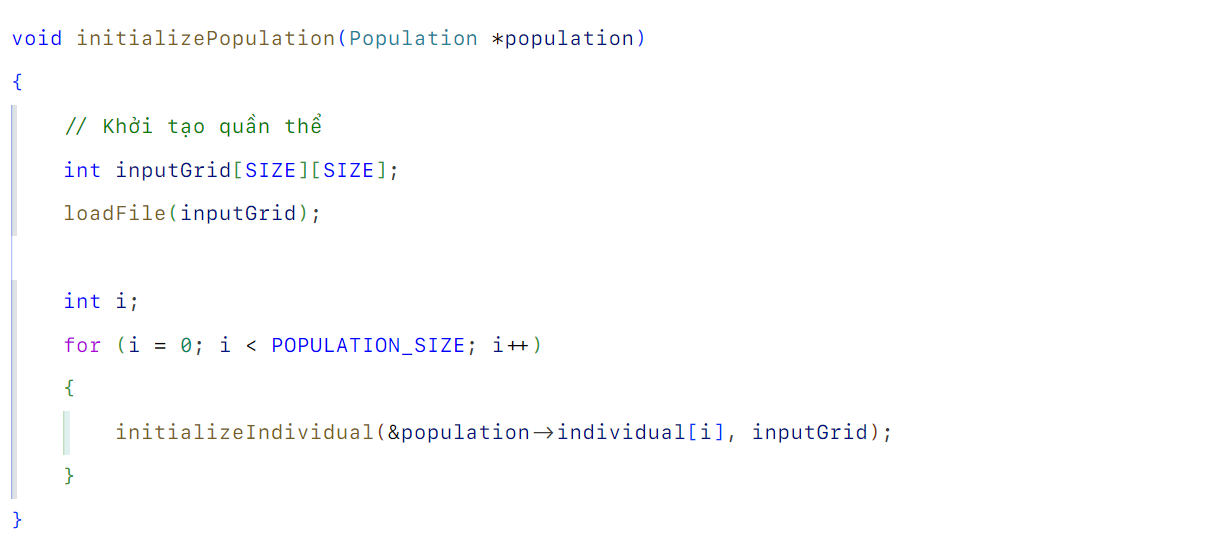
VD: giá trị cần điền vào ô (0, 0) là 5.

Hàm **initializeIndividual()** sẽ kiểm tra xem giá trị 5 có bị lặp lại trong block của ô (0, 0) hay không. Nếu giá trị 5 không bị lặp lại, thì hàm sẽ điền giá trị 5 vào ô (0, 0).

Nếu giá trị 5 bị lặp lại, thì hàm sẽ tiếp tục sinh một giá trị ngẫu nhiên khác. Ví dụ, hàm có thể sinh giá trị 6.

Kết luận:

Hàm **initializeIndividual()** là một hàm quan trọng trong giải thuật di truyền. Hàm này đảm bảo rằng mỗi cá thể trong quần thể là một giải pháp hợp lệ cho bài toán Sudoku.



Hình . Cài đặt hàm initializePopulation.

Hàm **initializePopulation()** dùng để khởi tạo 1 quần thể .

**B1**: Tải ma trận inputGrid (input đầu vào của bài toán) từ file input.txt.

Trong bước này, hàm **loadFile()** sẽ tải inputGrid từ file input.txt. Định dạng của file input.txt là mảng hai chiều các số nguyên, trong đó mỗi số nguyên đại diện cho giá trị của ô trong câu đố Sudoku, số 0 biểu thị rằng ô đó là ô trống. **Hàm loadFile()** sẽ lưu lưới này vào mảng inputGrid.

**B2**: Chạy vòng lặp.

Chạy vòng lặp với số bước bằng với kích thước quần thể được xác định trước.

VD: nếu kích thước của quần thể là 500, thì hàm **initializePopulation()** sẽ chạy vòng lặp có 500 bước.

**B3:** Khởi tạo từng cá thể.

Ở mỗi bước lặp, một cá thể sẽ được khởi tạo bằng hàm **initializeIndividual().**

Hàm **initializeIndividual()** dùng giá trị inputGrid để khởi tạo cá thể.

* + 1. Đánh giá độ thích nghi

Định nghĩa: độ thích nghi được tính bằng tổng số các xung đột trong hàng và cột của bảng sudoku. Cứ mỗi cặp số lặp lại trong bảng sudoku được tính là 1 xung đột. Cá thể nào có độ thích nghi bằng 0 chính là đáp án của bài toán.

Hàm **evaluateFitness()** dùng để tính độ thích nghi của cá thể.

**B1:** Khởi tạo biến conflicts = 0. Biến conflicts dùng để lưu trữ số xung đột của cá thể.

**B2:** Khởi tạo mảng count[SIZE+1]. Mảng count[] được khởi tạo với tất cả các phần tử đều bằng 0, được sử dụng để đếm số lần xuất hiện của 1 số trong 1 hàng hay 1 cột.

**B3:** kiểm tra số xung đột trong hàng.

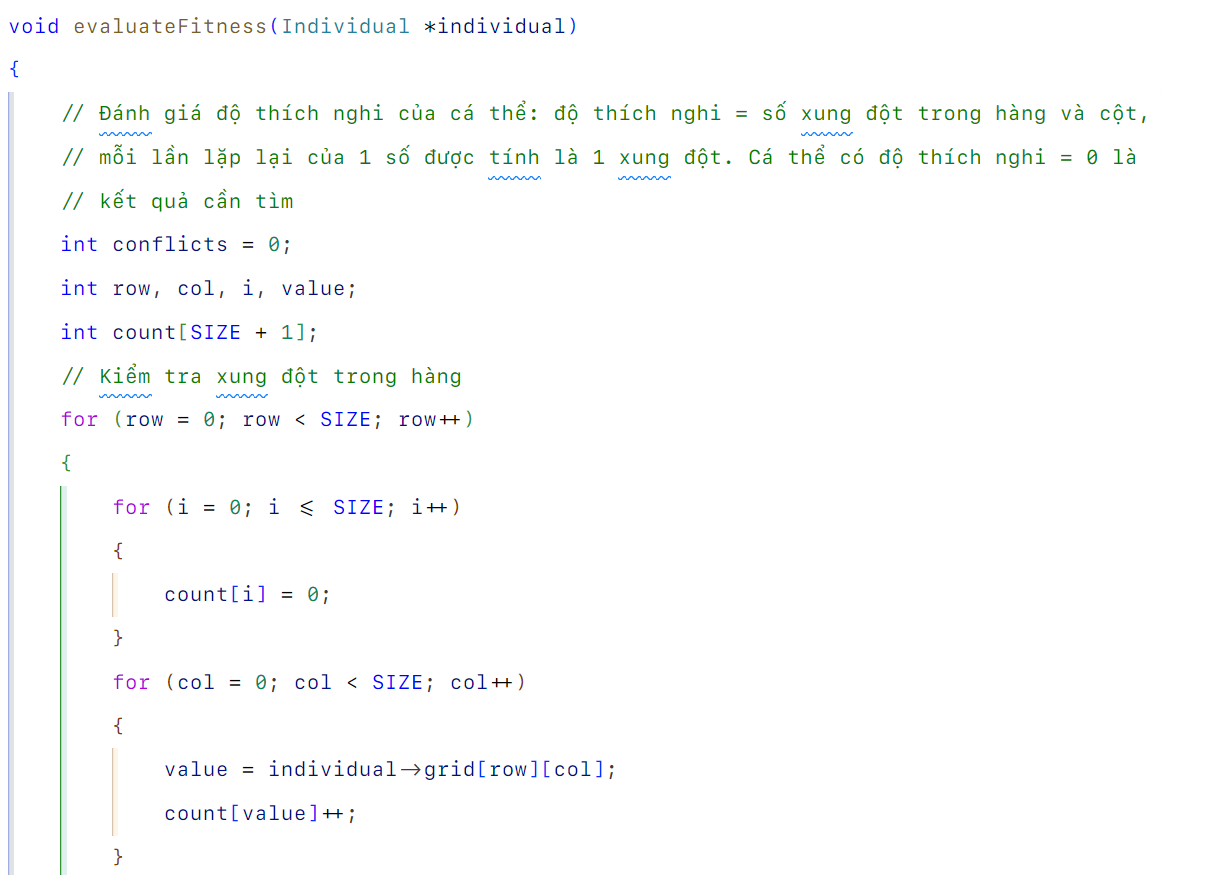
Trong bước này, hàm sẽ kiểm tra xung đột trong hàng. Hàm sẽ duyệt qua các hàng trong lưới. Trong mỗi hàng, hàm sẽ duyệt qua các ô trong hàng đó.

Đối với mỗi ô trong hàng, hàm sẽ lưu trữ giá trị của ô đó trong biến value. Sau đó, hàm sẽ tăng giá trị của phần tử count[value] lên 1.

Sau khi duyệt qua tất cả các ô trong hàng, hàm sẽ kiểm tra xem có giá trị nào xuất hiện nhiều hơn 1 lần hay không. Nếu có, thì hàm sẽ cộng dồn số lần lặp lại của giá trị đó (count[i] – 1) vào biến conflicts.

VD: Xét một hàng có 2 ô số 2, 4 ô số 3, khi đó count[2] = 2, count[3] = 4. Số 2 lặp lại 1 lần, số 3 lặp lại 3 lần nên conflicts = count[2] – 1 + count[3] – 1 = 1 + 3 = 4.

**B4**: Kiểm tra xung đột trong cột. Giống bước 3 nhưng thay hàng bằng cột.



Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, số

Mô tả được tạo tự động

Ảnh có chứa văn bản, Phông chữ, hàng, ảnh chụp màn hình

Mô tả được tạo tự động

Hình . Cài đặt hàm evaluateFitness.

* + 1. Sắp xếp quần thể theo độ thích nghi giảm dần (fitness tăng dần)

Dùng **Selection Sort** (thuật toán sắp xếp chọn) để sắp xếp quần thể. Thuật toán sắp xếp chọn hoạt động bằng cách lặp đi lặp lại việc tìm phần tử nhỏ nhất trong phần chưa được sắp xếp của mảng và hoán đổi nó với phần tử ở chỉ mục hiện tại. Kết quả thu được là quần thể gồm các cá thể được sắp xếp theo thứ tự giảm dần độ thích nghi.

Ảnh có chứa văn bản, ảnh chụp màn hình, Hình chữ nhật, số

Mô tả được tạo tự động

Hình . Minh họa quá trình sắp xếp các cá thể trong quần thể.

**B1**: Lặp qua các cá thể trong quần thể.

Trong bước này, vòng lặp ngoài sẽ lặp qua các cá thể trong quần thể.

Ví dụ, nếu kích thước của quần thể là 100, thì vòng lặp ngoài sẽ lặp qua 100 cá thể.

**B2**: Lặp qua các cá thể trong phần không sắp xếp của quần thể.

Trong bước này, vòng lặp trong sẽ lặp qua các cá thể trong phần không sắp xếp của quần thể. Phần không sắp xếp của quần thể là phần của quần thể bắt đầu từ cá thể ở chỉ mục i và kết thúc ở cá thể cuối cùng.

Ví dụ, nếu quần thể hiện tại có 5 cá thể, thì vòng lặp trong sẽ lặp qua các cá thể từ cá thể thứ 2 đến cá thể thứ 5.

**B3**: Tìm phần tử nhỏ nhất trong phần không sắp xếp của quần thể.

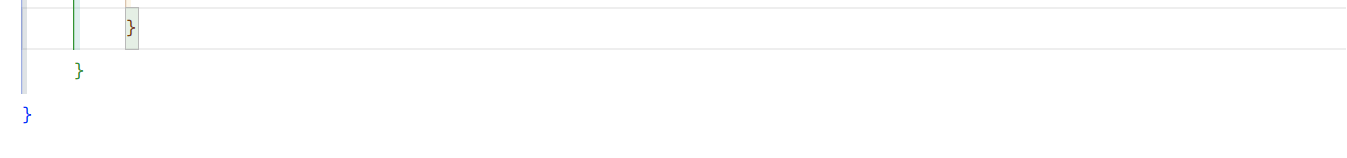
Trong bước này, biến **minIndex** sẽ được cập nhật để chỉ ra vị trí của phần tử nhỏ nhất trong phần không sắp xếp của quần thể.

Ví dụ, giả sử cá thể thứ 2 có độ thích nghi thấp hơn cá thể thứ 1. Khi đó, biến minIndex sẽ được cập nhật để chỉ ra vị trí của cá thể thứ 2.

**B4**: Hoán đổi hai cá thể nếu cần.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự độngTrong bước này, hai cá thể sẽ được hoán đổi nếu cá thể ở vị trí i có độ thích nghi cao hơn cá thể ở vị trí **minIndex**.



Hình . Cài đặt hàm sortPopulation.

* + 1. Chọn lọc cá thể

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ, Phông chữ

Mô tả được tạo tự độngChọn một phần cá thể ở đầu quần thể vừa được sắp xếp (đây là các cá thể "tốt"), số cá thể còn lại được chọn ngẫu nhiên để tránh thuyết ưu sinh (nghĩa là chỉ chọn các cá thể tốt mà loại bỏ hoàn toàn cá thể không tốt).

Hình . Minh họa quá trình chọn lọc.

**B1**: Chọn **SELECTION\_RATE%** cá thể tốt nhất.

Ví dụ, nếu kích thước của quần thể là 100 và **SELECTION\_RATE** là 0.3 thì vòng lặp for đầu tiên gồm 30 bước lặp, mỗi bước lặp thứ i sẽ chọn ra cá thể thứ i trong quần thể.

**B2**: Copy các cá thể tốt nhất vừa chọn được sang quần thể mới.

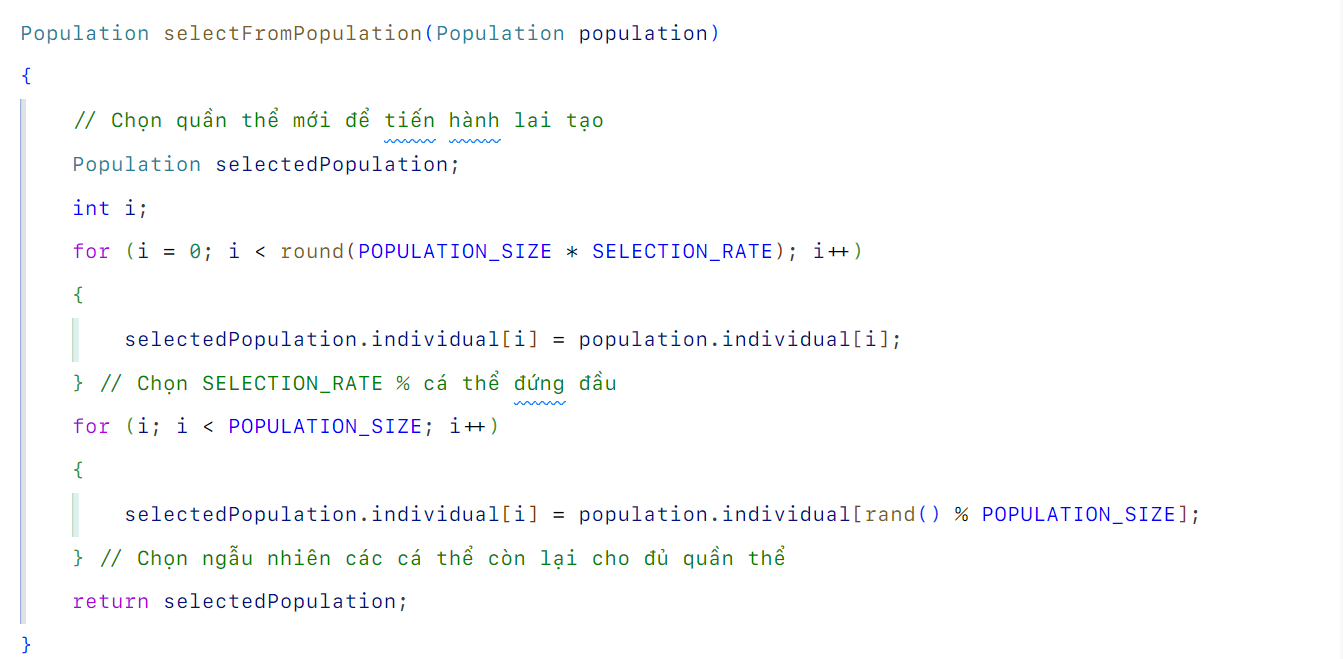
**B3**: Chọn ngẫu nhiên các cá thể còn lại.

Trong bước này, số bước lặp của vòng lặp for = tổng số cá thể của quần thể - số bước lặp của vòng for ở bước 1. Ở mỗi bước lặp, chọn ra **ngẫu nhiên** 1 cá thể trong toàn bộ quần thể đã được sắp xếp

Ví dụ, nếu kích thước của quần thể là 100 và **SELECTION\_RATE** là 0.3, thì vòng lặp for thứ hai gồm 100 – 30 = 70 bước lặp. Ở mỗi bước lặp, một cá thể ngẫu nhiên nào đó sẽ được chọn.

**B4**: Copy các cá thể ngẫu nhiên vừa được chọn sang quần thể mới.

Kết quả thu được là một quần thể mới gồm **SELECTION\_RATE** số cá thể tốt nhất từ quần thể cũ, số còn lại có thể có cả những cá thể xấu.



Hình . Cài đặt hàm selectFromPopulation.

* + 1. Lai tạo

Chọn ngẫu nhiên 2 cá thể từ quần thể đã được chọn lọc để làm bố mẹ

Cách thức lai tạo:

- Chọn ngẫu nhiên n khối 3 x 3 (n từ 1 – 8) để lấy từ cá thể mẹ, 9 - n khối còn lại lấy từ cá thể bố.

- Ghép các khối đã chọn từ 2 cá thể bố mẹ lại thành cá thể con

*VD: Chọn ngẫu nhiên các khối 0, 2, 4, 5 lấy từ mẹ, các khối 1, 3, 6, 7, 8 lấy từ bố*

Ảnh có chứa Hình chữ nhật, hình vuông, ảnh chụp màn hình, văn bản

Mô tả được tạo tự độngẢnh có chứa hình vuông, Hình chữ nhật, ảnh chụp màn hình, văn bản

Mô tả được tạo tự động

Hình . Minh họa quá trình lai tạo tạo ra cá thể mới.

Hàm **containsValue()** dùng để kiểm tra xem giá trị đã có ở trong mảng hay chưa.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm containsValue.

**Hàm** \*generateRandomNumbers() **dùng để** tạo ra một mảng các số ngẫu nhiên trong khoảng từ min đến **max.**

Hàm hoạt động như sau:

1. Khởi tạo một mảng có kích thước n.
2. Khởi tạo biến count để đếm số lượng số đã tạo.
3. Lặp lại cho đến khi count bằng n:
4. Tạo một số ngẫu nhiên trong khoảng từ min đến max.
5. Kiểm tra xem số vừa tạo đã tồn tại trong mảng chưa.
6. Nếu số vừa tạo chưa tồn tại thì thêm vào mảng.
7. Trả về mảng các số ngẫu nhiên.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm \*generateRandomNumbers.

Hàm **crossover()** là một hàm quan trọng trong thuật toán di truyền. Hàm này giúp tạo ra các cá thể con mới với các đặc điểm kết hợp từ các cá thể bố và mẹ.Hàm nhận 3 cá thể Individual làm tham số: **father**, **mother**, và **child**. Sau đó, nó thực hiện các bước sau:

1. Dùng hàm **rand()** để tạo ra một số ngẫu nhiên từ 1 đến SIZE – 1 rồi gán vào biến **numOfBlocks**. Số này chính là số khối mà ta sẽ lấy từ cá thể mẹ. Trong bài này, SIZE được đặt thành 9, vì vậy **numOfBlocks** có thể nhận giá trị từ 1 - 8.
2. Chọn ra ngẫu nhiên một mảng **randomBlock** chứa chỉ số của các khối, mảng này có **numOfBlocks** phần tử. Điều này được thực hiện bằng cách gọi hàm **generateRandomNumbers()** để tạo ra một danh sách **numOfBlocks** số ngẫu nhiên từ 0 đến SIZE – 2 (do các khối được đánh chỉ số từ 0). Mỗi một phần tử trong mảng chính là chỉ số của khối mà ta sẽ lấy từ cá thể mẹ.
3. Lặp qua từng khối. Đối với mỗi khối, các bước sau được thực hiện:
4. Xác định hàng và cột bắt đầu của khối đang xét bằng công thức sau:
   * hàng bắt đầu = chỉ số khối / **BLOCK\_SIZE** \* **BLOCK\_SIZE**
   * cột bắt đầu = chỉ số khối % **BLOCK\_SIZE** \* **BLOCK\_SIZE**
5. Lặp qua từng ô trong khối. Đối với mỗi ô, các bước sau được thực hiện:
6. Kiểm tra xem ô có giá trị cố định hay không. Nếu có thì sao chép giá trị của ô đó vào ô tương ứng trong cá thể con.
7. Kiểm tra xem khối đang xét có phải là một trong các khối được chọn hay không. Nếu có, giá trị của ô tương ứng từ mẹ được sao chép sang con.
8. Nếu khối đang xét không phải là một trong các khối được chọn thì giá trị của ô tương ứng từ bố được sao chép sang con.
9. Giải phóng bộ nhớ được cấp phát cho mảng **randomBlocks**.

Sau khi quá trình lai tạo hoàn tất, biến **child** gồm một phần các khối được lấy từ biến **mother**, phần còn lại được lấy từ biến **father**. Số cá thể con được tạo ra bằng với số cá thể của quần thể, tất cả các cá thể con này tạo thành một quần thể hoàn toàn mới.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm crossover.

* + 1. Đột biến

Đột biến (chọn tỉ lệ đột biến = 0.25):

Cách thức đột biến:

- Chọn ngẫu nhiên 1 khối 3 x 3

- Trong khối vừa chọn, chọn ngẫu nhiên 2 ô số không chứa giá trị cố định cho trước

- Hoán đổi vị trí 2 ô số này

Ảnh có chứa văn bản, hình vuông, ảnh chụp màn hình

Mô tả được tạo tự độngẢnh có chứa văn bản, hình vuông, ảnh chụp màn hình

Mô tả được tạo tự động*VD: Chọn ngẫu nhiên khối số 4, sau đó chọn ngẫu nhiên 2 ô số 1, 7 để hoán đổi (các ô tô vàng là ô có giá trị cố định cho trước)*

Hình . Minh họa quá trình chọn hoán đổi 2 ô trong block.

Hàm **swapTwoCells()** thực hiện hoán đổi giá trị của hai ô không chứa giá trị cố định trong lưới của một cá thể. Hàm có một tham số:

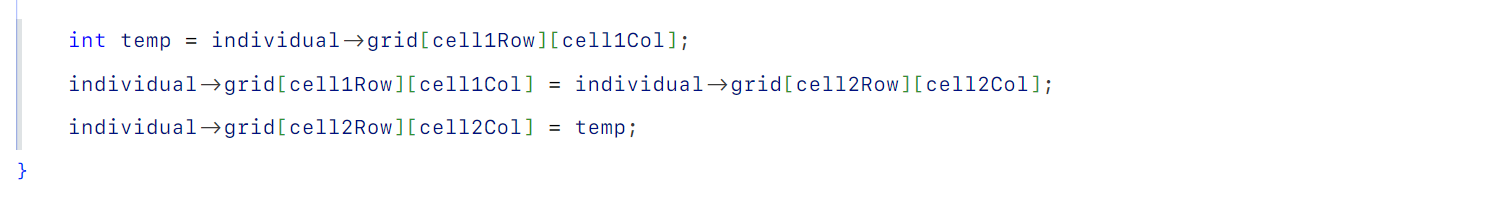
* **individual**: Cá thể mà lưới sẽ được sửa đổi

Hàm hoạt động như sau:

1. Chọn ngẫu nhiên một khối trong lưới. Biến **block** sẽ được gán giá trị bằng một số ngẫu nhiên trong khoảng từ 0 đến SIZE - 1.
2. Tính toán chỉ số hàng và cột bắt đầu của khối được chọn. Biến startRow và startCol sẽ được gán giá trị bằng chỉ số hàng và cột bắt đầu của khối được chọn.
3. Chạy vòng lặp để chọn ra ngẫu nhiên 2 ô trong khối được chọn. Nếu 2 ô được chọn có chứa giá trị cố định thì tiếp tục tìm cho đến khi cả 2 ô đều không chứa giá trị cố định.
4. Khi hai ô hợp lệ đã được chọn, hàm sẽ hoán đổi giá trị của chúng.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động



Hình . Cài đặt hàm swapTwoCells.

Hàm **mutate()** thực hiện đột biến đối với các cá thể trong quần thể. Hàm có một tham số:

* **population**: Quần thể

Hàm hoạt động như sau:

1. Lặp qua tất cả các cá thể trong quần thể.
2. Đối với từng cá thể, nếu tỉ lệ ngẫu nhiên nhỏ hơn **MUTATION\_RATE** thì tiến hành đột biến.
3. Đột biến được thực hiện bằng cách hoán đổi giá trị của hai ô ngẫu nhiên trong cá thể. Hàm sẽ gọi hàm **swapTwoCells()** để hoán đổi giá trị của hai ô ngẫu nhiên trong cá thể đó.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Hình . Cài đặt hàm mutate.

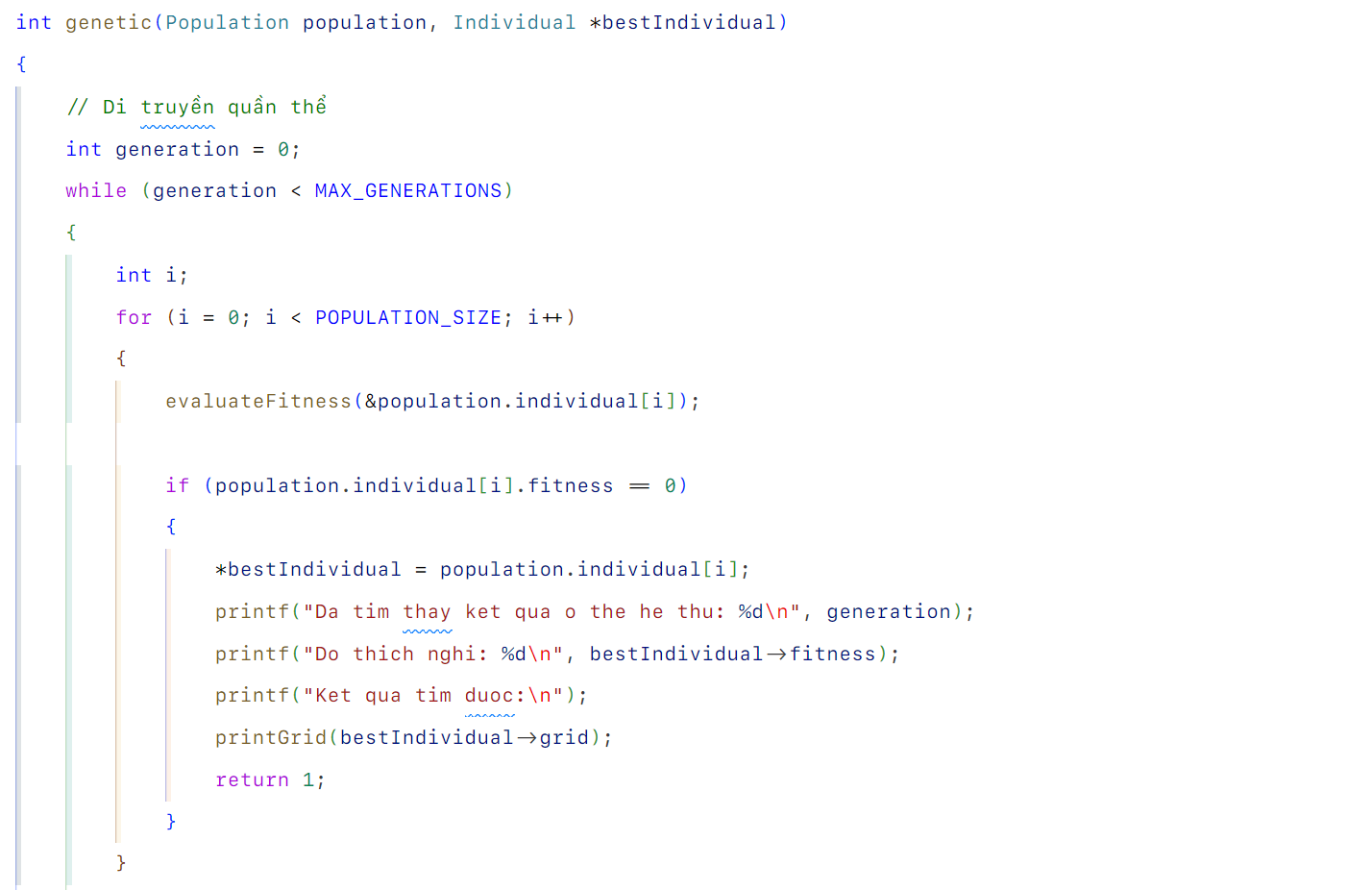
* + 1. Di truyền quần thể

Hàm **genetic()** thực hiện thuật toán di truyền để tìm cá thể tốt nhất trong quần thể. Hàm này thực hiện các bước cơ bản của thuật toán di truyền, bao gồm đánh giá độ thích nghi, chọn lọc, lai tạo và đột biến. Hàm có hai tham số:

* **population**: Quần thể ban đầu
* **bestIndividual**: Con trỏ tới cá thể tốt nhất được tìm thấy cho đến nay

Hàm hoạt động bằng cách lặp lại các bước sau cho đến khi số thế hệ tối đa được đạt tới (khởi đầu bộ đếm thế hệ = 0):

1. Đánh giá độ thích nghi của từng cá thể trong quần thể. Hàm **evaluateFitness()** sẽ được gọi để đánh giá độ thích nghi của từng cá thể trong quần thể.
2. Nếu bất kỳ cá thể nào có độ thích nghi bằng 0, thì đó là cá thể tốt nhất và hàm sẽ trả về.
3. Sắp xếp quần thể theo độ thích nghi. Hàm **sortPopulation()** sẽ được gọi để sắp xếp quần thể theo độ thích nghi.
4. Chọn lọc cá thể. Hàm **selectFromPopulation()** sẽ được gọi để chọn lọc các cá thể phù hợp.
5. Tạo một quần thể mới bằng cách thực hiện lai tạo và đột biến trên quần thể vừa được chọn lọc. Hàm **crossover()** sẽ được gọi để thực hiện lai giữa hai cá thể bố mẹ. Hàm **mutate()** sẽ được gọi để thực hiện đột biến trong quần thể mới.
6. Tăng bộ đếm thế hệ lên 1.



Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm genetic.

* + 1. Hàm main

Hàm main: cho chạy giải thuật tối đa 5 lần, sau mỗi lần chạy nếu không tìm được kết quả thì tự động khởi tạo lại 1 quần thể hoàn toàn mới và chạy lại từ đầu.

Hàm thực hiện các bước sau:

1. Khởi tạo bộ đếm **runTimes** để theo dõi số lần chương trình đã được chạy.
2. Khởi tạo một cờ **success** để chỉ ra liệu chương trình đã tìm thấy giải pháp hay chưa.
3. Dùng vòng lặp để chạy giải thuật cho đến khi số lần khởi động lại đạt tối đa hoặc chương trình đã tìm thấy giải pháp.
4. Bên trong vòng lặp, khởi tạo một quần thể các cá thể. Hàm **initializePopulation()**được gọi để tạo ra một quần thể mới các cá thể.
5. Gọi hàm **genetic()** để thực hiện thuật toán di truyền.
6. Nếu hàm **genetic()** tìm thấy giải pháp, đặt cờ **success** thành **true**.
7. Tăng bộ đếm **runTimes** lên 1.
8. Thoát khỏi vòng lặp.
9. Nếu cờ **success** vẫn là **false**, in một thông báo lên bảng điều khiển cho biết không tìm thấy giải pháp.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Hình . Cài đặt hàm main.

CHƯƠNG III

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

* 1. Kết quả của giải thuật

Ảnh có chứa văn bản, ảnh chụp màn hình, Hình chữ nhật, hình vuông

Mô tả được tạo tự độngVới giá trị đầu vào bài toán như sau:

Hình . Bảng SUDOKU đầu vào.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự độngSau khi khởi chạy giải thuật ta thu được kết quả:

Hình . Bảng SUDOKU sau khi khởi chạy thuật toán di truyền.

* 1. Kết luận

Giải thuật di truyền là một kỹ thuật hiệu quả để giải bài toán Sudoku. Giải thuật di truyền có thể giải được các bài toán Sudoku có độ khó cao một cách hiệu quả. Tuy nhiên, thời gian giải quyết của giải thuật di truyền phụ thuộc vào độ khó của bài toán. Giải thuật di truyền không bảo đảm luôn luôn cho ra được lời giải của bài toán. Các hàm xử lí trong giải thuật phần lớn dựa vào tính ngẫu nhiên nên sẽ có trường hợp cá thể con được tạo ra sau quá trình lai tạo và đột biến có độ thích nghi thấp hơn nhiều so với cá thể bố mẹ tạo ra nó.

* 1. Hướng phát triển

Có một số hướng phát triển để cải thiện hiệu quả của giải thuật di truyền để giải bài toán Sudoku, bao gồm:

* 1. Điều chỉnh các tham số của giải thuật. Các tham số của giải thuật di truyền, chẳng hạn như tỷ lệ chọn lọc, tỷ lệ đột biến, và tỷ lệ trao đổi chéo, có thể ảnh hưởng đến hiệu quả của giải thuật. Điều chỉnh các tham số này có thể giúp giải thuật tìm kiếm giải pháp nhanh hơn hoặc hiệu quả hơn.
  2. Có thể hạn chế việc tạo ra một cá thể mới quá xấu sau khi lai tạo hoặc đột biến bằng cách kiểm tra cá thể con được tạo ra, nếu độ thích nghi bị giảm quá 1 ngưỡng giới hạn nào đó thì tiến hành lai tạo lại hoặc đột biến lại.
  3. Có thể cải tiến hàm lai tạo để tạo ra nhiều cá thể con trong 1 lần lai tạo.
  4. Sử dụng học máy để học các quy tắc logic. Học máy có thể được sử dụng để học các quy tắc logic để loại bỏ các khả năng cho các ô còn trống.

Với những hướng phát triển này, giải thuật di truyền có thể trở thành một kỹ thuật hiệu quả hơn để giải bài toán Sudoku.