**MỤC LỤC**

[I. Giới thiệu đồ án 4](#_Toc531337537)

[**1.** **Lý do chọn đề tài** 4](#_Toc531337538)

[**2.** **Thuật toán quay lui** 4](#_Toc531337539)

[**3.** **Giới thiệu bài toán ứng dụng** 5](#_Toc531337540)

[II. Nội dung 7](#_Toc531337541)

[**1.** **Xây dựng giải thuật quay lui bằng phương pháp đệ quy.** 7](#_Toc531337542)

[**2.** **Giải quyết vấn đề: Khử đệ quy** 9](#_Toc531337543)

[**3.** **Giới thiệu về Stack-Ngăn xếp** 9](#_Toc531337544)

[**4.** **Áp dụng Stack để khử đệ quy giải quyết bài vấn đề** 9](#_Toc531337545)

[**5.** **Áp dụng Stack để viết hàm Undo và Redo** 12](#_Toc531337546)

[**6.** **Chức năng hàm viết nháp** 12](#_Toc531337547)

[**7.** **Các khó khăn khi gặp phải** 16](#_Toc531337548)

[**8.** **Bảng phân công** 16](#_Toc531337549)

[III. Kết luận 16](#_Toc531337550)

[**1.** **Ưu điểm** 16](#_Toc531337551)

[**2.** **Nhược điểm** 16](#_Toc531337552)

[**3.** **Hướng phát triển** 16](#_Toc531337553)

**DANH MỤC**

[Figure 1. Sơ đồ thuật toán quay lui 3](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337402)

[Figure 2. Ma trận Sudoku chưa được giải 5](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337403)

[Figure 3. Ma trận Sudoku đã được giải 6](#_Toc531337404)

[2.3.2 Thuật toán giải với đệ quy 7](#_Toc531337405)

[2.4.1. Thuật toán giải với Stack 1 10](#_Toc531337406)

[2.4.2. Thuật toán giải Stack 2 10](#_Toc531337407)

[2.5.1. Hàm viết nháp 1 12](#_Toc531337408)

[Figure 4. Hàm viết nháp 2 13](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337409)

[Figure 5. Hàm viết nháp 4 14](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337410)

[Figure 6. Hàm viết nháp 5 14](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337411)

[Figure 7. Hàm viết nháp 6 14](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337412)

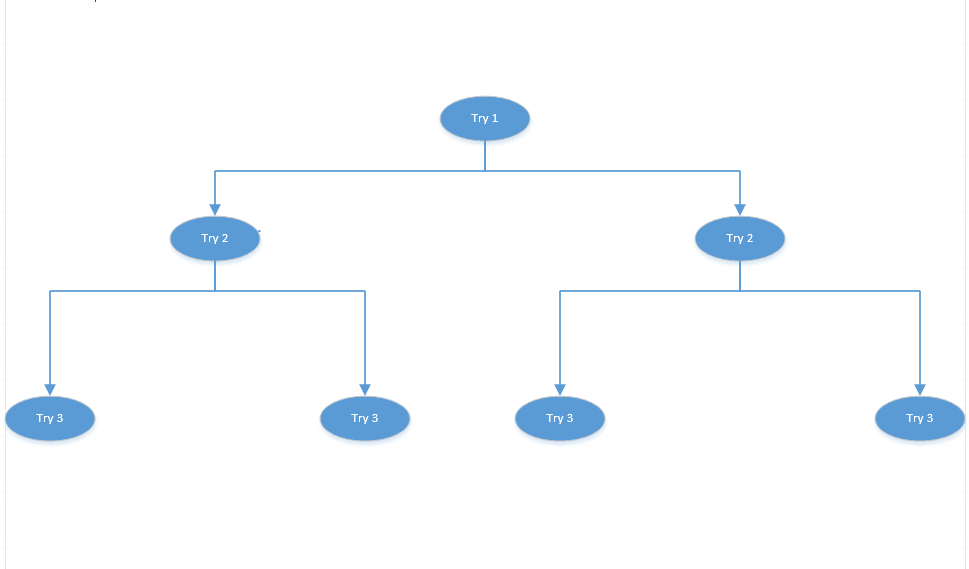
[Figure 8. Hàm viết nháp 7 15](file:///D:\project%20sudoku\sudoku\Doc\reportnew.docx#_Toc531337413)

# **Giới thiệu đồ án**

## **Lý do chọn đề tài**

* Sudoku là một trò chơi từng gây nghiện của các quốc gia phát triển. Độ phức tạp của trò chơi tăng dần khi các số cho trước giảm dần. Vì vậy để chinh phục trò chơi này dễ dàng hơn, người ta đã tìm thuật toán(quay lui) và lập trình nó trên máy tính. Nhờ vào sự phát triển của công nghệ mà máy tính có thể giải một ma trận Sudoku trong thời gian ngắn.
* Từ đó ta cũng có thể viết hoàn thiện một game Sudoku và cho phép người dùng điền vào giá trị và máy tính liên tục kiểm tra các giá trị đó.
* Sở dĩ nhóm chọn đề tài về game Sudoku là vì phần thuật toán của game vô cùng hấp dẫn, cần tư duy logic cao. Ngoài ra, các thành viên trong nhóm cũng từng rất hứng thú với trò chơi này trên giấy. Trò chơi thật sự khó, phải tốn nhiều thời gian và công sức để giải. Vì vậy giải trong vòng vài giây, đối với nhóm mang lại sức hấp dẫn rất lớn.

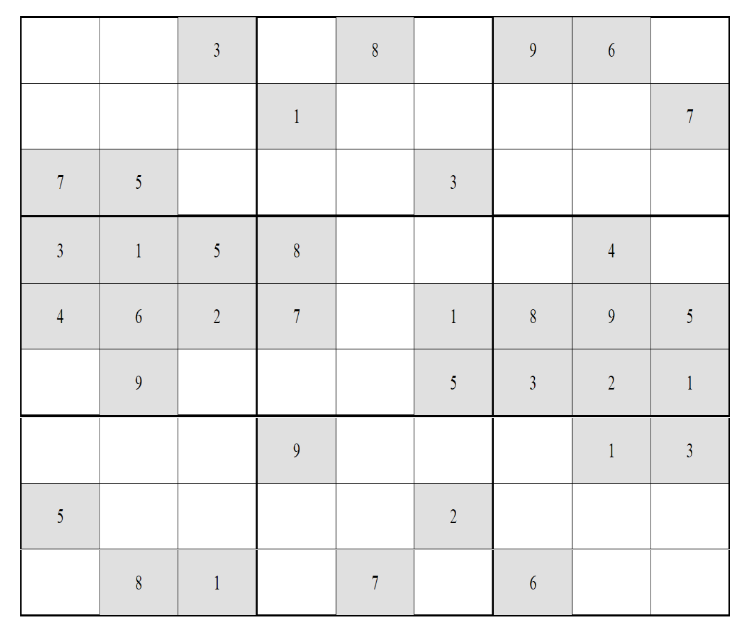
## **Thuật toán quay lui**

* Thuật toán quay lui là một thuật toán điển hình để giải các bài toán ứng dụng trong tin học. Bằng việc liệt kê các tình huống, thử các khả năng có thể cho đến khi tìm thấy một lời giải đúng, thuật toán quay lui chia nhỏ bài toán, lời giải của bài toán lớn sẽ là kết quả của việc tìm kiếm theo chiều sâu của tập hợp các bài toán con. Trong suốt quá trình tìm kiếm nếu gặp phải một hướng nào đó mà biết chắc không thể tìm thấy đáp án thì quay lại bước trước đó và tìm hướng khác kế tiếp hướng vừa tìm kiếm đó. Trong trường hợp không còn một hướng nào khác nửa thì thuật toán kết thúc.

#### Figure 1. Sơ đồ thuật toán quay lui

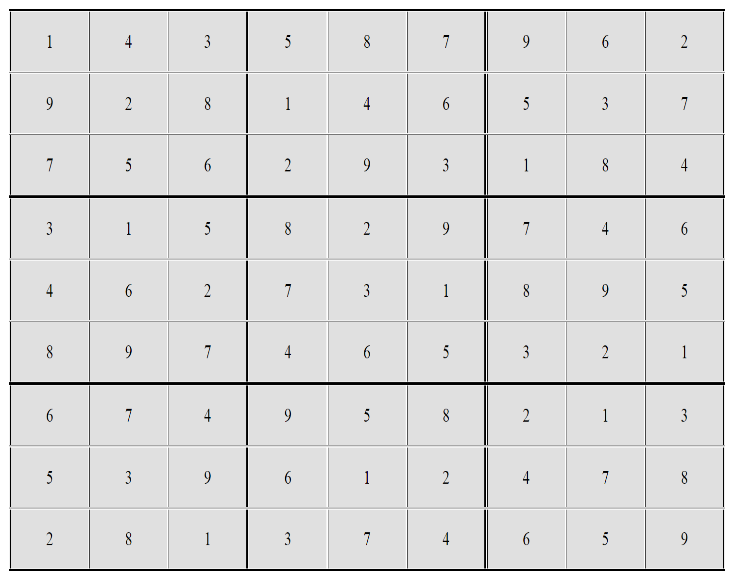
* Khác với thuật toán tham lam (cũng là điểm mạnh), thuật toán quay lui có điểm khác là nó không cần phải duyệt hết tất cả các khả năng, nhờ đó tránh được các khả năng không đúng nên có thể giảm được thời gian giải. Thuật toán quay lui thường được cài đặt theo lối đệ quy, hàm đệ quy được thực hiện để giải quyết các bài toán con để trả về kết quả của bài toán lớn. Mục đích của việc sử dụng hàm đệ quy là để thuật toán được rõ ràng, dễ viết, dễ hiểu hơn và cũng để bảo toàn các biến, các trạng thái lúc giải bài toán con.
* Thuật toán quay lui có thể được thể hiện theo sơ đồ cây tìm kiếm theo chiều sâu như hình trên. Từ hình vẽ, ta dễ dàng nhận thấy rằng:
  + - Ở 1 bài toán hiện tại (mỗi nốt), ta đi tìm lời giải cho bài toán đó. Ứng với lời giải, ta đi giải bài toán kế tiếp cho đến lúc bài toán gốc trở nên đầy đủ.
    - Lời giải của bài toán gốc thường là một lối đi từ gốc đến nốt cuối cùng (không có nốt con)

## **Giới thiệu bài toán ứng dụng**

* Sudoku là một trò chơi trí tuệ nổi tiếng, thu hút nhiều người tham gia đặc biệt là giới trẻ. Ra đời ở Nhật và không lâu sau đã trở nên cực kỳ phổ biến trên thế giới. Quy luật của trò chơi tương đối đơn giản, cho một bàn hình vuông được chia thành một lưới 81 ô nhỏ trên 9 hàng và 9 cột. 81 ô nhỏ đó lại được chia thành 9 vùng, mỗi vùng có 9 ô. Đề bài Sudoku là một bàn hình vuông như thế, trên đó tại một số ô, người ta đã điền sẳn một số giá trị.
* VD: Đây là một ma trận Sudoku chưa được giải, với các dòng, các cột, các vùng 3x3 chưa được điền đầy đủ.

#### Figure 2. Ma trận Sudoku chưa được giải

* Sao khi được tiền đầy đủ thì các dòng, các cột, các vùng ô 3x3 sẽ được lắp đầy với các số từ 1 đến 9 ngẫu nhiên không trùng nhau.
* Ví dụ Ma trận bên dưới đã được giải bằng thuật toán Sudoku.



#### Figure 3. Ma trận Sudoku đã được giải

# **Nội dung**

## **Xây dựng giải thuật quay lui bằng phương pháp đệ quy.**

* Như đã nói ở trên, thuật toán Sudoku căn bản là sử dụng phương pháp quay lui. Và điển hình của quay lui và sử dụng đệ quy để giải quyết vòng lặp. Ưu điểm của đệ quy giúp cho việc code trở nên đơn giản hơn rất nhiều, đở tốn công sức để code.
* Ví dụ dưới đây là code thuật toán giải Sudoku bằng đệ quy được viết bằng C++ với :
  + 1. printSolution (S): In ma trận ra màn hình.
    2. solveSudoku(): Hàm giải Sudoku.
    3. checkValid(): Hàm kiểm tra.

#### 2.3.2 Thuật toán giải với đệ quy

* Giải thích thuật toán:

1. Hàm kiểm tra ( boollean checkValid ( int S[][9], int x, int y, int k) )

* Các biến được truyền vào bao gồm:
  + 1. int S[][9] : Ma trận suduku.
    2. int x : vị trí dòng
    3. int y : vị trí cột
    4. int k: số được người chơi điền vào.
* Ta kiểm tra đồng thời 3 đặt điểm của một ma trận suduku:
  + 1. Kiểm tra dòng (mỗi dòng gồm các chữ số từ 1 đến 9 không trùng nhau): Khởi tạo biến i chạy từ 0 đến 8 để kiểm tra số k vừa điền vào có trùng với giá trị vị trí dòng x cột i (S[x][i]) hay không. Nếu trùng thì trả về false.
    2. Kiểm tra cột (mỗi dòng gồm các chữ số từ 1 đến 9 không trùng nhau): Khởi tạo biến i chạy từ 0 đến 8 để kiểm tra số k vừa điền vào có trùng với với giá trị vị trí dòng i cột y (S[x][i]) hay không. Nếu trùng thì trả về false.
    3. Kiểm tra vùng 3x3: Kiểm tra xem trong vùng 3x3 với giá trị được thêm vào có trùng với các số có sẵn hoặc đã điền trước đó hay không. Nếu trùng thì trả về false.
    4. Nếu cả 3 đều kiện trên đều không trả về false thì trả về true (giá trị k được chấp nhận điền vào).
    5. Lưu ý: Giá trị k được điền vào chỉ đúng với trong thời điểm nhất định, có thể không đúng với toàn ma trận.
  1. Hàm giải thuật ( solveSudoku( int S[][9], int x, int y) )
* Thuật toán được lý giải khá đơn giản. Nếu tại vị trí S[x,y] = 0 ( chưa điền giá trị) thì máy sẽ tự điền tuần tự các giá trị từ 1 đến 9. Nếu giá trị thứ k đầu tiên trong tập giá trị từ 1 đến 9 thỏa checkValid = true thì k được điền vào ô [x,y].
* Sau đó ta gọi đệ quy với x và y+1( điền tiếp số k khác vào vị trí tiếp theo). Nếu không có giá trị nào từ 1 đến 9 điền được vào ô S[x,y+1], ta gán trở lại S[x,y+1] =0, rồi quay về vị trí trước đó (S[x,y]), xét tiếp các giá trị còn lại của k.
* Khi hết 1 dòng thì tăng x để xét tiếp dòng tiếp theo.
* Vòng lặp sẽ dừng và in ra màn hình ma trận khi ta giải được đến ô cuối cùng.

## **Giải quyết vấn đề: Khử đệ quy**

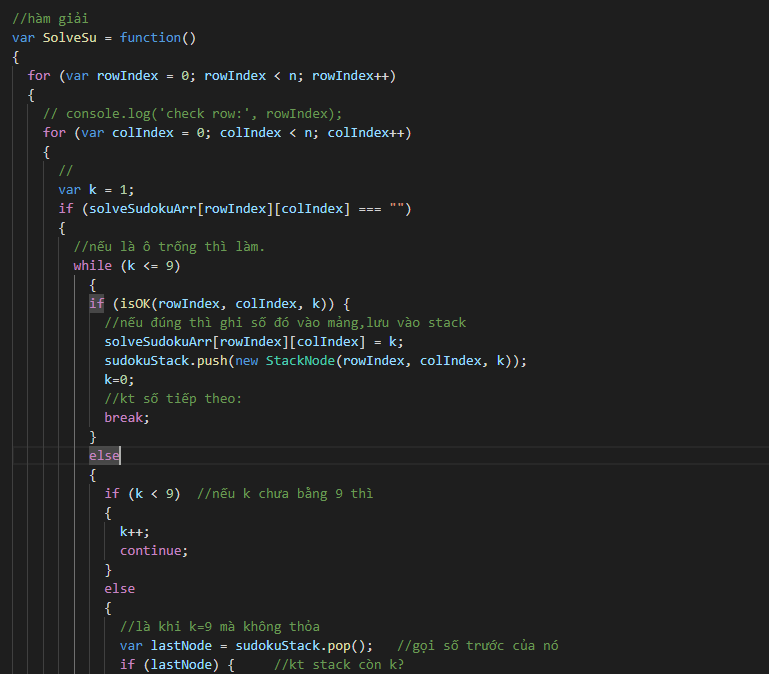
* Vì trong chương trình “Cấu trúc dữ liệu và giải thuật” không có chương về đệ quy, nên ta sẽ giải thuật toán quay lui bằng phương pháp “Khử đệ quy”. Cụ thể ta sẽ khử đệ quy bằng Stack-Ngăn xếp.
* Đây là một trong những cách khử đệ quy phổ biến.

## **Giới thiệu về Stack-Ngăn xếp**

* Stack là một vật chứa ( container ) các đối tượng làm việc theo cơ chế LIFO ( Last In First Out ) nghĩa là thêm một đối tượng or lấy ra trong Stack đều thực hiện theo cơ chế “Vào sau ra trước”.
* Các đối tượng được thêm vào bất cứ lúc nào, nhưng chỉ được lấy ra phần tử đầu.
* Stack có thể được viết bởi mảng một chiều hoăc danh sách liên kết.
* Các tính năng chính của stack bao gồm: push() – Thêm 1 giá trị vào stack, pop() - Lấy giá trị đầu ra khỏi Stack, peek() – Truy xuất giá trị đầu của Stack( không xóa giá trị đó ).

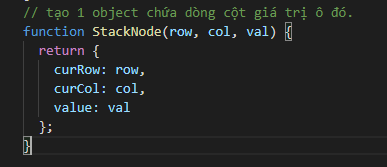
## **Áp dụng Stack để khử đệ quy giải quyết bài vấn đề**

* Thuật toán quay lui là điền một số vào ô sau đó chuyển sang ô khác, điền tiếp. Nếu không hợp lệ thì quay về ô cũ điền giá trị khác.
* Vì bản chất Stack là vật chứa (container) nên sao khi điền một số, ta sẽ lưu nó vào trong Stack (push), muốn lấy nó ra ngoài để sử dụng (pop)
* Như vậy khi áp dụng vào thuật toán giải sudoku ta giải thích như sau : Máy tính vẫn điền ngẫu nhiên từ 1 đến 9, và kiểm tra từng số có phù hợp với ma trận hay không? Nếu hợp lệ thì viết số đó vào ô và push vào Stack, tiếp đó là duyệt vị trí tiếp theo của ma trận. Nếu không thỏa thì pop ra. Quay ngược về vị trí cũ làm tiếp.
* Code khử đệ quy bằng Stack để giải Sudoku được viết bằng JavaScript SolveSu():



#### 2.4.1. Thuật toán giải với Stack 1

#### 2.4.2. Thuật toán giải Stack 2

* Giải thích về các hàm:
  + - SolveSu = function() : Hàm giải Sudoku.
    - StackNode : Khái báo 1 Node chứa dòng, cột, giá trị của ô nào đó.

2.4.2. Thuật toán giải Stack 3

* + - lastNode: là node cuối cùng của Stack.
    - Vì trong JavaScript đã có sẳn Stack nên không cần viết thêm các hàm pop(), push().

## **Áp dụng Stack để viết hàm Undo và Redo**

## **Chức năng hàm viết nháp**

* Với Sudoku, người chơi rất khó khăn trong việc lựa chọn số phù hợp để điền vào, vì một ô có thể có rất nhiều con số phù hợp. Vấn đề đặt ra là chúng ta phải tìm một phương pháp để người chơi có thể viết tất cả các số khả thi trong một ô để người chơi so sánh các số và chọn ra con số thích hợp. Vì vậy chức năng viết nhập ra hiện.

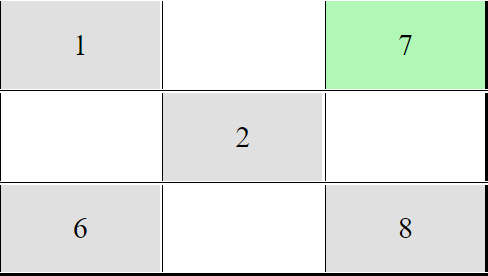
#### 2.5.1. Hàm viết nháp 1

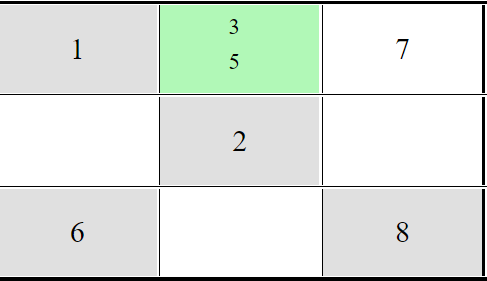
* Ý nghĩa code :
  + - Bước 1: Đầu tiên ta gọi hàm createDraftsBlank: Hàm này chỉ đơn giản là tạo ra 9 ô nhỏ để điền nháp và nếu ô đang nhấp đã có số “KHÔNG GHI NHÁP” rồi thì sẽ không tạo ra 9 ô nhỏ nữa. Cho đến khi ô đó không có số thì mới được tạo.

#### Figure 4. Hàm viết nháp 2

* + - Ví dụ:

Ngay ô xanh lá đã có số KHÔNG GHI NHÁP thì không được tạo 9 ô nhỏ.

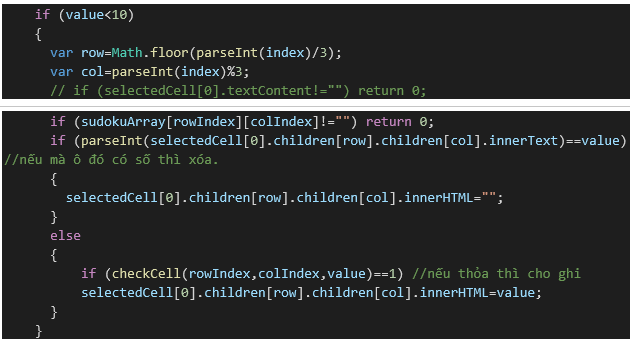


Còn dây là ô trống. Nên hàm createDraftsBlank sẽ chạy, tạo ra 9 ô.

#### Figure 5. Hàm viết nháp 4

* Bước 2: Lấy địa chỉ ô đang nhấp.

#### Figure 6. Hàm viết nháp 5

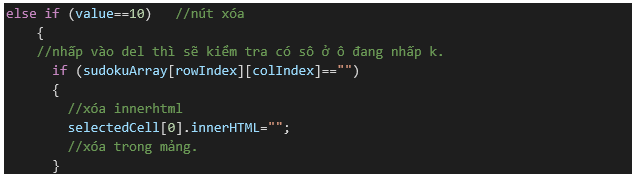
* Bước 3: Xử lý eventlistener khi người dùng nhấp từ 1 đến 9.

#### Figure 7. Hàm viết nháp 6

Hàm này xử lý người dùng nhập số vào. Nếu bấm thêm 1 lần nữa sẽ xóa số đã điền. Điền vào ô html đã lấy địa chỉ. Ngoài ra ta còn lồng vào hàm kiểm tra số có hợp lý không để người dùng điền vào.

* Bước 4: Nút xóa tất cả số ghi nháp.

Vì cho nút xóa là số 10. Nên ta dùng if để bắt nó.

1. ******Các khó khăn khi gặp phải**

#### Figure 8. Hàm viết nháp 7

1. **Bảng phân công**

# **Kết luận**

## **Ưu điểm**

## **Nhược điểm**

## **Hướng phát triển**