TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐÀO TẠO CHẤT LƯỢNG CAO
NGÀNH CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

GIẢI SUDOKU BẰNG NGĂN XẾP

Th.s : Trần Công Tú
SVTH : Võ Minh Hiếu
MSSV : 17110136
SVTH : Lê Minh Tiến
MSSV : 17110236

TP. Hồ Chí Minh, tháng 11 năm 2017

MỤC LỤC

I. Giới thiệu đồ án 1
1. Lý do chọn đề tài 1
2. Thuật toán quay lui 1
3. Giới thiệu bài toán ứng dụng 2
II. Nội dung 4
1. Xây dựng giải thuật quay lui bằng phương pháp đệ quy. 4
2. Giải quyết vấn đề: Khử đệ quy 7
3. Giới thiệu về Stack-Ngăn xếp 7
4. Áp dụng Stack để khử đệ quy giải quyết vấn đề 7
5. Áp dụng Stack để viết hàm Undo và Redo 10
6. Chức năng hàm viết nháp 11
7. Các khó khăn khi gặp phải 16
8. Bảng phân công 16
III. Kết luận 17
1. Ưu điểm 17
2. Nhược điểm 18
3. Hướng phát triển 18

LỜI NÓI ĐẦU

Lời nói đầu tiên, nhóm xin cảm ơn quý thầy cô đã đọc bài báo cáo và chấm điểm đồ án cho nhóm. Nói sơ lược về nội dung báo cáo, thì bài báo cáo là tổng hợp những nội dung quan trọng mà nhóm sử dụng để làm đồ án. Nội dung được dựa trên những gì đã học của môn "Cấu trúc dữ liệu và giải thuật". Như tiêu đề ở trang bìa, nhóm đã lập trình game Sudoku bằng việc sử dụng các Stack để tiến hành các công việc liên quan. Dựa vào nền tảng lập trình trước đó kết hợp với tìm hiểu cái mới, nhóm quyết định viết trên ngôn ngữ mới JavaScript và chạy nó trên Web. Việc viết trên ngôn ngữ hoàn toàn mới và là lần đầu nhóm làm một đồ án có thời gian dài đến như vây, nên không thể tránh khỏi những sai sót. Kính mong quý thầy cô và bạn đọc thông cảm bỏ qua. Chân thành cảm ơn.

DANH MỤC CÁC HÌNH

Hình I.2.1. Sơ đồ thuật toán quay lui 2

Hình I.2.1. Ma trận Sudoku chưa được giải 3

Hình I.2.2. Ma trận Sudoku chưa được giải 4

Hình II.1.1. Thuật toán giải Sudoku bằng quay lui 5

Hình II.4.1. Thuật toán giải bằng Stack 1 8

Hình II.4.2. Thuật toán giải bằng Stack 2 9

Hình II.4.3. Node của hàm giải 9

Hình II.5.1. Hàm Undo 10

Hình II.5.2. Hàm Redo 11

Hình II.5.2. Node của Undo, Redo 11

Hình II.6.1. Hàm viết nháp 1 12

Hình II.6.2. Hàm viết nháp 2 13

Hình II.6.3. Hàm viết nháp 3 14

Hình II.6.4. Hàm viết nháp 4 14

Hình II.6.5. Hàm viết nháp 5 15

Hình II.6.5. Hàm viết nháp 5 15

Hình II.6.6. Hàm viết nháp 6 15

Hình II.6.7. Hàm viết nháp 7 16

DANH MỤC CÁC BẢNG

Bảng 1. Bảng phân công..... 17

I. Giới thiệu đồ án

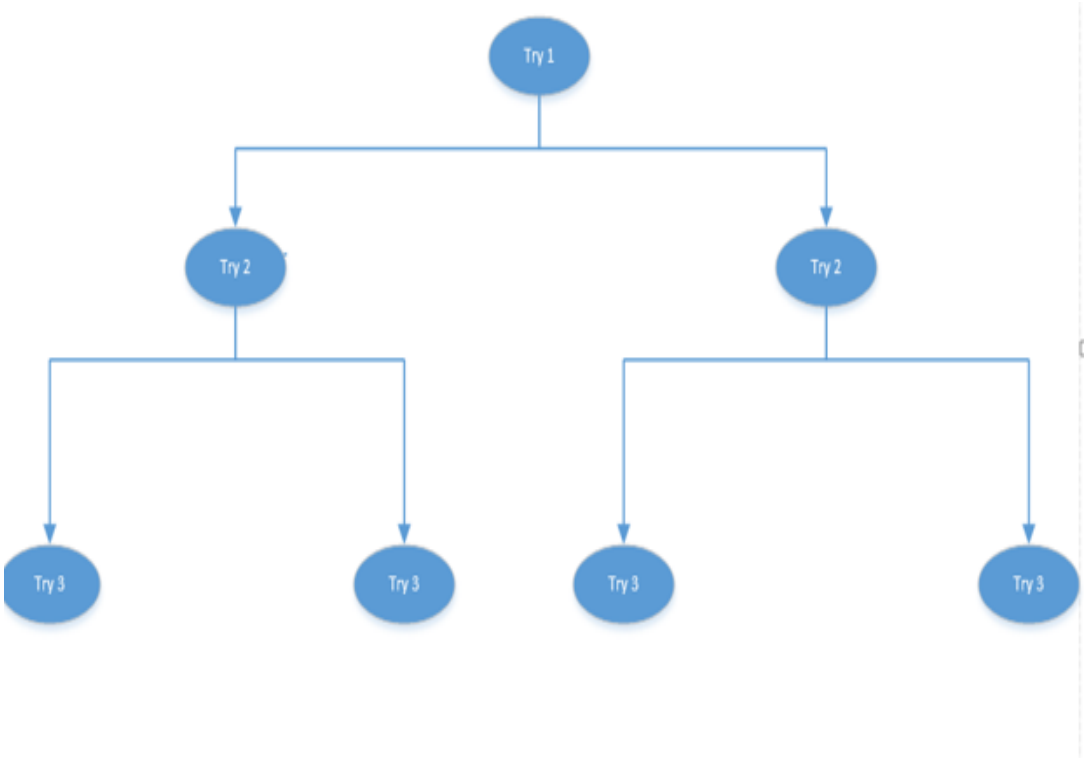
1. Lý do chọn đề tài

- 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 việc giải trong vòng vài giây, đối với nhóm mang lại sức hấp dẫn rất lớn.

2. 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.
- 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.



Hình I.2.1. Sơ đồ thuật toán quay lui

- 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).[1]

3. 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.[2] 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 sẵn 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.

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

Hình I.2.1. 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 quay lui.

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

Hình I.2.2. Ma trận Sudoku chưa được giải

II. Nội dung

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

- Như đã nói ở trên, thuật toán giải Sudoku căn bản là sử dụng phương pháp quay lui. Và điển hình của quay lui là 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[3] thuật toán giải Sudoku sử dụng đệ quy trên ngôn ngữ C++ với:
    - printSolution (S): In ma trận ra màn hình.
    - solveSudoku(): Hàm giải Sudoku.
    - checkValid(): Hàm kiểm tra.

```
void solveSudoku(int S[][9], int x, int y){
    if(y == 9){
        if(x == 8){
            printSolution(S);
            exit(0);
        } else {
            solveSudoku(S, x+1,0);
        }
    } else if(S[x][y] == 0){
        for (int k = 1; k <=9; k++){
            if(checkValid(S,x,y,k)){
                S[x][y] = k;
                solveSudoku(S, x, y+1);
                S[x][y] = 0;
            }
        }
    } else {
        solveSudoku(S,x,y+1);
    }
}

boolean checkValid(int S[][9], int x, int y, int k){
    for(int i = 0; i <9 ; i++){
        if(S[x][i] == k) return false;
    }
    for(int i = 0; i <9 ; i++){
        if(S[i][y] == k) return false;
    }
    int a = x/3, b = y/3;
    for(int i = 3*a; i < 3*a+3; i++){
        for(int j = 3*b; j < 3*b+3; j++){
            if(S[i][j] == k) return false;
        }
    }
    return true;
}
```

Hình II.1.1. Thuật toán giải Sudoku bằng quay lui

- Giải thích thuật toán:

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

Các biến được truyền vào bao gồm:

- int S[][9] : Ma trận suduku.

- int x : vị trí dòng
- int y : vị trí cột
- int k: số được người chơi điền vào.

Ta kiểm tra đồng thời 3 đặt điểm của một ma trận suduku:

- 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.
- 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.
- 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.
- Nếu cả 3 đều kiện trên đều không trả về false thì trả về true (giá trị k được chấp nhận điền vào).
- Lưu ý: Giá trị k được điền vào chỉ đúng với trong thời điểm nhất định, có thể không đúng với toàn ma trận.

b. 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.

2. Giải quyết vấn đề: Khử đệ quy

- Mặc dù đệ quy là một phương pháp phổ biến trong lập trình, tuy nhiên nó chứa rất nhiều hạn chế như: Vô cùng tốn bộ nhớ khi xử lý dữ liệu lớn, dẫn đến quá trình xử lý diễn ra một cách chậm chạp. Đó là điều tối kị nếu ta muốn có một chương trình tối ưu hóa tốt. Vấn đề đặt ra, ta cần có một phương pháp nào đó để "khử đệ quy". Do đó, nhóm đã dùng Stack-Ngăn xếp để tiến hành khử đệ quy.
- Đây là một trong những cách khử đệ quy phổ biến.

3. 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 hoặc lấy ra trong Stack đều thực hiện theo cơ chế "Vào sau ra trước".[4]
- 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 hoặc 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).

4. Áp dụng Stack để khử đệ quy giải quyết 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 game Sudoku được viết bằng JavaSript (solveSu ()) :

```
//hàm giải
var SolveSu = function () {
  for (var rowIndex = 0; rowIndex < n; rowIndex++) {

    for (var colIndex = 0; colIndex < n; colIndex++) {
      //
      var k = 1;
      if (solveSudokuArr[rowIndex][colIndex] === "") {
        //nếu là ô trống thì làm.
        while (k <= 9) {
          if (isOK(rowIndex, colIndex, k)) {
            //nếu đúng thì ghi số đó vào mảng,lưu vào stack
            solveSudokuArr[rowIndex][colIndex] = k;
            sudokuStack.push(new StackNode(rowIndex, colIndex, k));
            k = 0;
            //kt số tiếp theo:
            break;
          } else {
            if (k < 9) //nếu k chưa bằng 9 thì
            {
              k++;
              continue;
            } else {
              //là khi k=9 mà không thỏa
              var lastNode = sudokuStack.pop(); //gọi số trước của nó
              if (lastNode) { //kt stack còn k?
                if (lastNode.value !== 9) //nếu ô trc khác 9 thì tăng k
                {
                  solveSudokuArr[rowIndex][colIndex] = ""; //set ô trước
                  rowIndex = lastNode.curRow; //gán lại ô trc đó
                  colIndex = lastNode.curCol; //gán lại ô trước đó
                  k = lastNode.value + 1; //tăng k lên +1 và giải tiếp
                  continue; //giải tiếp
                } else // nếu ô trước bằng 9 thì skip ô này. qua ô trước
                {
                  solveSudokuArr[rowIndex][colIndex] = "";
                  rowIndex = lastNode.curRow; //gán lại ô trc đó
                  colIndex = lastNode.curCol; //gán lại ô trc đó
                  solveSudokuArr[rowIndex][colIndex] = ""; //set ô đang
                  lastNode = sudokuStack.pop(); //set ô trước đó bằng 0
                  rowIndex = lastNode.curRow; //gán lại ô trc đó
                  colIndex = lastNode.curCol; //gán lại ô trc đó
```

Hình II.4.1. Thuật toán giải bằng Stack 1

```
            solveSudokuArr[rowIndex][colIndex] = ""; //set ô đang xét =0
            lastNode = sudokuStack.pop(); //set ô trước đó bằng 0
            rowIndex = lastNode.curRow; //gán lại ô trc đó
            colIndex = lastNode.curCol; //gán lại ô trc đó
            solveSudokuArr[rowIndex][colIndex] = ""; //set ô đang xét =0
            k = lastNode.value + 1; //tăng k và giải tiếp
            continue;
          }
        } else {
          console.log("Buoc di cuoi cung");

          return -1;
        }
      }
    }
```

Hình II.4.2. Thuật toán giải bằng 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.

```
// tạo 1 object chứa dòng cột giá trị ô
function StackNode(row, col, val) {
  return {
    curRow: row,
    curCol: col,
    value: val
  };
}
```

Hình II.4.3. Node của hàm giải

  - lastNode: là node cuối cùng của Stack.
  - isOK (rowIndex, colIndex, k): hàm kiểm tra từ từng giá trị xem thỏa điều kiện của ma trận hay không.
  - Vì trong JavaScript đã có sẵn Stack nên không cần viết thêm các hàm pop(), push().

5. Áp dụng Stack để viết hàm Undo và Redo

Trong Sudoku, trường hợp người chơi lựa chọn số không phù hợp với ma trận và muốn quay về số trước đó, vì có thể những số trước đó hợp lý hơn. Vì vậy, chức năng Undo và cả Redo ra đời để giải quyết vấn đề trên:

- Cả hai chức năng trên đều nó những đặc điểm đặc trưng của Stack. Vì vậy, nó sẽ được thực hiện bằng Stack với hai mảng 1 chiều (1 dùng cho Undo, 1 dùng cho Redo) tương tác qua lại với nhau.
- Với Undo, khi người chơi nhập bất kì số X nào, thì ta push số X vào trong Stack thứ nhất, khi người chơi muốn Undo thì ta sẽ pop số X ra ngoài, để quay lại số trước và đồng thời số X sẽ được push vào Stack thứ hai với mục đích Redo khi cần.
- Hàm undoButton:

```
var undoButton = function () {
  //hàm undo.
  if (undoStack != 0) {
    var obj = undoStack.pop();
    var preobj = undoStack[undoStack.length - 1];
    redoStack.push(obj); //redostack;
    var container = undoSelectedCell.pop(); //truy xuất DOM để xuất hiện lên web
    redoSelectedCell.push(container);
```

Hình II.5.1. Hàm Undo

- Giải thích:

  1. Với undoStack: Ta sẽ lấy giá trị cuối của mảng undoStack. Sau đó cập nhật lại mảng undoStack (mất 1 length). Tiếp tục lấy nó bỏ vào mảng redoStack.

  2. Với undoSelectedCell: chỉ là 1 hàm truy xuất DOM để xuất hiện lên web.

- Hàm redoButton:

```
var redoButton = function () {
  //hàm redo
  if (redoStack != 0) {
    var obj = redoStack.pop();
    undoStack.push(obj);
    var container = redoSelectedCell.pop();
    undoSelectedCell.push(container);
```

Hình II.5.2. Hàm Redo

- Giải thích: Tương tự với undoStack, redoStack cũng có chức năng tượng tự.

- Với mỗi giá trị lấy ra hay thêm vào các Stack sẽ bao gồm: dòng, cột, giá trị cuối, giá trị trước cuối.

```
var obj = { //tạo object để lưu vào stack
  RowIndex: rowIndex,
  ColIndex: colIndex,
  value: value,
  preValue: prevValue
};
```

Hình II.5.2. Node của Undo, Redo

6. 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.

```
var handleDraftCellClick = function (innderDiv, index) {
  return function () {

    createDraftsBlank(index);
    var rowIndex = parseInt(selectedCell[0].getAttribute("data-row"));
    var colIndex = parseInt(selectedCell[0].getAttribute("data-col"));
    //nếu số vừa nhập thỏa 1 div nào đó:
    // if (selectedCell[0].getAttribute("pos") == index)
    //   console.log(selectedCell[0]);
    var value = index + 1;
    if (value < 10) {
      var row = Math.floor(parseInt(index) / 3);
      var col = parseInt(index) % 3;
      // if (selectedCell[0].textContent!="") return 0;
      if (sudokuArray[rowIndex][colIndex] != "") return 0;
      if (parseInt(selectedCell[0].children[row].children[col].innerText) == value)
      {
        selectedCell[0].children[row].children[col].innerHTML = "";
      } else {
        if (checkCell(rowIndex, colIndex, value) == 1) //nếu thỏa thì cho ghi
          selectedCell[0].children[row].children[col].innerHTML = value;
      }
    } else if (value == 10) //nút xóa
    {
      //nhấp vào del thì sẽ kiểm tra có sô ở ô đang nhấp k.
      if (sudokuArray[rowIndex][colIndex] == "") {
        //xóa innerhtml
        selectedCell[0].innerHTML = "";
        //xóa trong mảng.
      }
    }
  };
};
```

Hình II.6.1. Hàm viết nháp 1

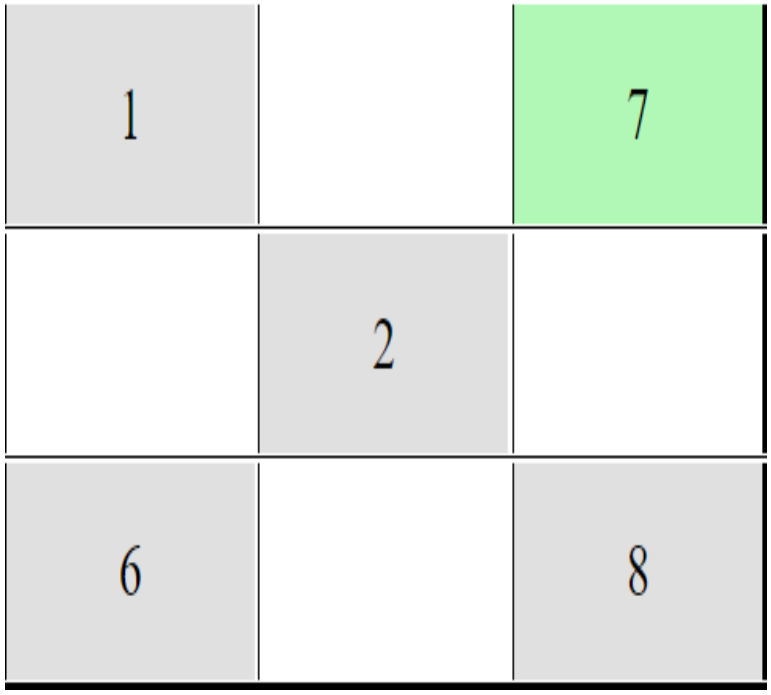
Ý nghĩa code:

- 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.

```
var createDraftsBlank = function (index) {
  var container = selectedCell[0];
  //kt nếu có  mark rồi hay chưa
  if (container.hasChildNodes() != true) {
    if (container.classList.value.indexOf("wrong") > 0) /
      container.classList.remove("wrong");
    //nếu chưa có con  thì tạo
    //container.classList.add("mark");
    //tạo 9 ô.
    for (var i = 0; i < 3; i++) {
      var indivRow = document.createElement("div");
      indivRow.setAttribute("class", "row-drafts");
      indivRow.setAttribute("rowPos", i);
      for (var j = 0; j < 3; j++) {
        var indivCol = document.createElement("div");
        indivCol.setAttribute("class", "cell-drafts");
        indivCol.setAttribute("colPos", j);
        indivRow.appendChild(indivCol);
      }
      container.append(indivRow);
    }
  } else {
    if (container.hasChildNodes() === true) {
      //nếu đã tạo rồi thì để yên.
      return 0;
    }
  }
};
```

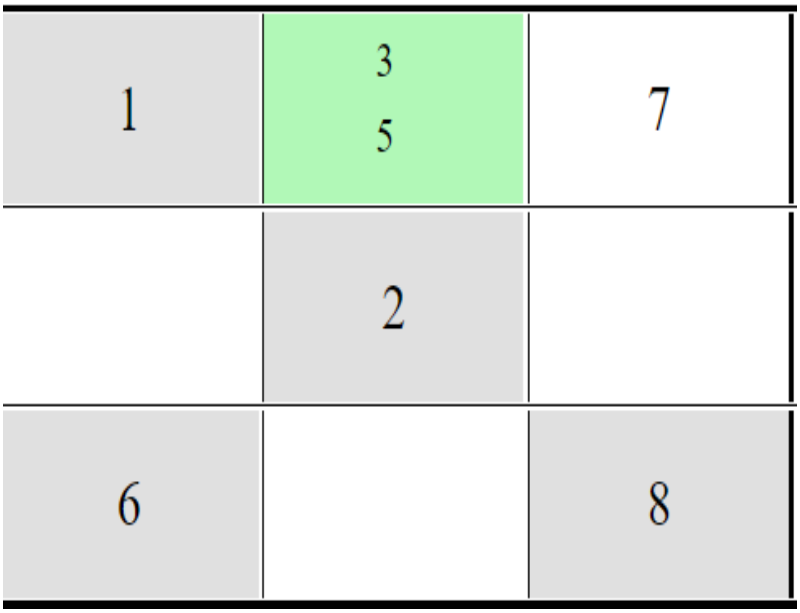
Hình II.6.2. Hàm viết nháp 2

- Ví dụ:



Hình II.6.3. Hàm viết nháp 3

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



Hình II.6.4. Hàm viết nháp 4

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

- Bước 2:

- Lấy địa chỉ ô đang nhấp.

```
var rowIndex = parseInt(selectedCell[0].getAttribute("data-row"));
var colIndex = parseInt(selectedCell[0].getAttribute("data-col"));
```

Hình II.6.5. Hàm viết nháp 5

- Bước 3:

```
if (value < 10) {
  var row = Math.floor(parseInt(index) / 3);
  var col = parseInt(index) % 3;
```

Hình II.6.5. Hàm viết nháp 5

```
if (sudokuArray[rowIndex][colIndex] != "") return 0;
if (parseInt(selectedCell[0].children[row].children[col].innerText) == value)
{
  selectedCell[0].children[row].children[col].innerHTML = "";
} else {
  if (checkCell(rowIndex, colIndex, value) == 1) //nếu thỏa thì cho ghi
    selectedCell[0].children[row].children[col].innerHTML = value;
}
```

Hình II.6.6. Hàm viết nháp 6

- Xử lý eventlistener khi người dùng nhấp từ 1 đến 9.
- 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:

```
    }
  } else if (value == 10) //nút xóa
  {
    //nhấp vào del thì sẽ kiểm tra có số ở ô đang nl
    if (sudokuArray[rowIndex][colIndex] == "") {
      //xóa innerhtml
      selectedCell[0].innerHTML = "";
      //xóa trong mảng.
    }
  }
};
```

Hình II.6.7. Hàm viết nháp 7

- Nút xóa tất cả số ghi nháp.
- Vì cho nút xóa là số 10. Nên ta dùng if để bắt nó.

7. Các khó khăn khi gặp phải

- Trong lúc viết khử đệ quy bằng Stack, đã gặp 1 vấn đề khá nghiệm trọng, máy chỉ giải được 1 số ma trận dễ, 1 số lại ko giải được. Nhóm đã mất 3 tuần để tiến hành debug kiểm tra lỗi gặp phải. Nguyên nhân xảy ra bug là do : nếu thuật toán chạy đến số 9 và số 9 phù hợp với ma trận, thì nó không push vào trong Stack mà nhảy qua ô khác để giải tiếp.
- Viết truy xuất DOM (Document Object Modal) khá phức tạp vì phạm vi của DOM rất rộng, tìm hiểu lâu.
- Hàm sinh ngẫu nhiên ma trận Sudoku, chưa thể chọn ngẫu nhiên những vị trí để phù hợp với điều kiện game.
- Do hàm thời gian không có sẵn, nên việc tìm hiểu và viết nó tốn nhiều thời gian của nhóm.

8. Bảng phân công

STT	Tên công việc	Võ Minh Hiếu	Lê Minh Tiến
1	Tìm hiểu về Stack-Ngăn xếp	x	x
2	Viết hàm kiểm tra	x	

3	Viết nút undo, redo bằng Stack và reset	x	x
4	Viết hàm giải Sudoku bằng Stack	x	x
5	Viết nút ghi nháp		x
6	Thêm từng level cho game	x	
7	Thêm hàm tính thời gian		x
8	Làm giao diện cho web		x
9	Lưu tên người chơi		x
10	Test game và sửa lỗi	x	x
11	Viết báo cáo	x	
12	Soạn Power Point	x	

Bảng 1. Bảng phân công

III. Kết luận

1. Ưu điểm

Bất cứ một chương trình nào cũng sẽ có những ưu điểm riêng để mang lại trãi nghiệm tốt nhất cho người chơi.

- Game được lập trình trên Web (HTML, CSS, JavaScript) vì:
  + Dễ dàng chơi ở bất cứ mọi nơi, bất cứ thiết bị (PC, Smartphone, Tablet, Laptop,…) mà không cần phải cài đặt thêm bất cứ chương trình nào.
  + Tiêu tốn dung lượng bộ nhớ ít hơn so với lập trình trên Winform.
- Nhờ có chức năng viết nháp nên người chơi có thể giải nhanh hơn.
- Chức năng Hint giúp người chơi có thêm các gợi ý bổ ích trong quá trình chơi
- Giao diện đơn giản, dễ nhìn, dễ chơi.

2. Nhược điểm

- Với những ma trận có nhiều trường hợp, thì hàm giải chỉ giải được 1 trường hợp.
- Chương trình chỉ chạy được với các trình duyệt phổ biến như Chrome, Edge, Cốc Cốc,v.v. Có một số trình duyệt chạy bị lỗi như Firefox, Tor Browser, Opera Browser, UC Browser, v.v.
- Không có Internet thì sẽ ko chơi được.
- Code chưa tối ưu hóa tốt nên ngốn nhiều Ram của máy.
- Hiện tại game chỉ lưu tên người chơi, bảng xếp hạng trên Local Storage của máy cá nhân.

3. Hướng phát triển

- Chức Hint sẽ chỉ giải miễn phí 1 lần. Sau đó muốn giải thêm phải xem quảng cáo hoặc trả tiền/point cho việc đó.
- Khi click vào một ô nào đó, nó sẽ highlight dòng, cột, vùng mà ô đó phụ thuộc, để dễ dàng rà soát các số cho phù hợp.
- Bảng xếp hạng (Điểm-Time, Tên người chơi, Hạng) sẽ được lưu trên Sever với DatabaseUser để bất cứ người nào cũng có thể tìm thông tin của mình ở bất cứ đâu, và cũng có thể thêm thông khác vào.
- Thay đổi giao diện để thân thiện và đẹp hơn
- Tối ưu hóa thuật toán để chạy nhanh và mượt hơn.

---

[1] Lớp 07T4,Nhóm 12A, Khoa công nghệ thông tin, Đại học Bách Khoa Đà Nẵng
[2] Tham khảo và chỉnh sửa từ https://vi.wikipedia.org/wiki/Sudoku
[3] https://viblo.asia/p/thuat-toan-quay-lui-backtracking-bJzKmLbD59N
[4] Giáo trình Cấu trúc dữ liệu và giải thuật Đại Học Khoa Học Tự Nhiên TP.HCM