

I. Giới thiệu về thuật toán

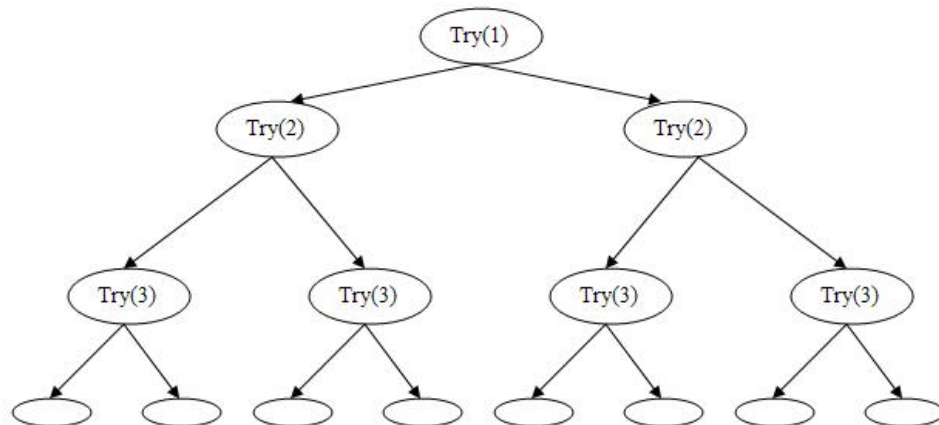
Thuật toán quay lui

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 phần tử. 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 nữa 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, mỗi lần gọi hàm đệ quy, hàm đệ quy được truyền một tham số (trong các tham số) là chỉ số của bài toán con, trong hàm sẽ cố gắng tìm lời giải cho bài toán con đó, nếu tìm thấy thì gọi hàm đệ quy để giải bài toán con tiếp theo hoặc là đưa ra đáp án bài toán lớn nếu đã đầy đủ lời giải, nếu không tìm thấy thì chương trình sẽ trở về điểm gọi hàm đó. 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 bên. Từ hình vẽ, ta dễ dàng nhận thấy rằng :



- Ở 1 bài toán hiện tại (mỗi nút), 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 trở gốc nên đầy đủ.
- Lời giải của bài toán gốc thường là một lối đi từ gốc đến nút cuối cùng (không có nút con)

II. 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 sẵn một số giá trị.

Ví dụ

Yêu cầu dùng các số từ 1 đến 9 để điền nốt vào các ô còn lại sao cho trên mỗi hàng, mỗi cột và mỗi vùng 9 ô, phải điền đầy đủ 9 số từ 1 đến 9. Như ở ví dụ trên thì đáp án sẽ là

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

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

III. Đặc tả cấu trúc dữ liệu và giải thuật

Cấu trúc dữ liệu

Dữ liệu sử dụng trong chương trình là dữ liệu kiểu mảng

```
int [,] row = new int[10, 10];
int [,] collum = new int[10, 10];
int [,] area = new int[10, 10];
```

```
int [,] AREA=new int[10,10];  
int [,] Area = new int[10, 10];
```

```
int[, ,] agree = new int[10, 10, 11];  
int[,] value = new int[10, 10];  
int[,] problem = new int[10, 10];
```

- Mảng row,collum, area là các mảng 2 chiều dùng để đánh dấu hàng, cột, vùng có thể đánh một số nào đó hay không, ví dụ row[2,3]=1 tức là hàng 2 có thể đánh số 3.
- Mảng AREA để là mảng cố định, AREA[i,j]=k nghĩa là ô hàng i, cột j là ở vùng k.
- Mảng Area là để phục vụ cho việc duyệt theo vùng, với các giá trị Area [i,j]=k nghĩa là vùng i, có ô trống thứ j là k.
- Mảng agree để đánh dấu trên từng ô trống một, có thể điền các giá trị nào. Cách diễn tả như sau.
 - o value[i,j,0]=k nghĩa là ô trống hàng i, cột j có k khả năng điền.
 - o Các giá trị value[i,j,1] value[i,j,2] value[i,j,3]... value[i,j,k] là các khả năng điền đó.
- Mảng value là mảng 2 chiều để chỉ giá trị đang được điền hiện tại của một bất kỳ. value[i,j]=a tức là ô hàng i, cột j đang được điền số agree[i,j,a]
 - o Ví dụ :
Nếu agree[1,2,0]=5 và 5 giá trị
agree[1,2,1]=1,
agree[1,2,2]=3,
agree[1,2,3]=6,
agree[1,2,4]=7,
agree[1,2,5]=9,
Và value[1,2]=3 có nghĩa là ô hàng 1, cột 2 đang được đánh số thứ 3 trong các khả năng, tức là đang được đánh số 7.
Cách đánh dấu này có điểm thuận lợi là dễ dàng duyệt các khả năng điền của một ô, cho phép loại bỏ khả năng bất kỳ khi biết khả năng đó là không thuận lợi.

IV. Thuật giải

1) Tổng quan

- Xác định bài toán
 - Input : Đề sudoku, bảng số cho bởi file hoặc do người dùng nhập trên giao diện GUI.
 - Output : Kết quả, lời giải sudoku (nếu có). Nếu đề bài có nhiều đáp án thì phải xuất được nhiều đáp án.
- Các cách xác định mỗi ô số bất kỳ.
 - Xác định theo số thứ tự từ 1 đến 81. Cách này ta sẽ sử dụng chủ yếu để duyệt toàn bộ 81 ô, hay để lấy đề bài, xuất kết quả ra giao diện người dùng.

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81

- Xác định bằng [hàng, cột], Với cách xác định này, ta dễ dàng duyệt theo chiều cột, theo hàng ta có bảng sau.

[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]	[1,7]	[1,8]	[1,9]
[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]	[2,8]	[2,9]
[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]	[3,8]	[3,9]
[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]	[4,8]	[4,9]
[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]	[5,8]	[5,9]
[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]	[6,8]	[6,9]
[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]	[7,8]	[7,9]
[8,1]	[8,2]	[8,3]	[8,4]	[8,5]	[8,6]	[8,7]	[8,8]	[8,9]
[9,1]	[9,2]	[9,3]	[9,4]	[9,5]	[9,6]	[9,7]	[9,8]	[9,9]

- Xác định bằng {vùng, số thứ tự trong vùng}, Cách xác định này cho phép ta duyệt theo vùng, ta có bảng

{1,1}	{1,2}	{1,3}	{2,1}	{2,2}	{2,3}	{3,1}	{3,2}	{3,3}
{1,4}	{1,5}	{1,6}	{2,4}	{2,5}	{2,6}	{3,4}	{3,5}	{3,6}
{1,7}	{1,8}	{1,9}	{2,7}	{2,8}	{2,9}	{3,7}	{3,8}	{3,9}
{4,1}	{4,2}	{4,3}	{5,1}	{5,2}	{5,3}	{6,1}	{6,2}	{6,3}
{4,4}	{4,5}	{4,6}	{5,4}	{5,5}	{5,6}	{6,4}	{6,5}	{6,6}

{4,7}	{4,8}	{4,9}	{5,7}	{5,8}	{5,9}	{6,7}	{6,8}	{6,9}
{7,1}	{7,2}	{7,3}	{8,1}	{8,2}	{8,3}	{9,1}	{9,2}	{9,3}
{7,4}	{7,5}	{7,6}	{8,4}	{8,5}	{8,6}	{9,4}	{9,5}	{9,6}
{7,7}	{7,8}	{7,9}	{8,7}	{8,8}	{8,9}	{9,7}	{9,8}	{9,9}

- Để ý cách 1 và cách 2 dễ dàng chuyển đổi qua lại lẫn nhau bởi công thức
Chuyển từ dạng 1 sang dạng 2 $x \rightarrow [i,j]$

```
int toi(int x)
{
    return (x-1)/9+1;
}
int toj(int x)
{
    return (x-1)%9+1;
}
```

Chuyển từ dạng 2 sang dạng 1

```
int tox(int i, int j)
{
    return (i - 1) * 9 + j;
}
```

Đối với dạng 3 thì ta lưu thành mảng AREA để sử dụng, trong mảng này, mỗi giá trị $AREA[i,j]$ = số thứ tự ở dạng 1.

2) Vấn đề

Chương trình giải dựa trên thuật giải quay lui. Tư tưởng của thuật giải này là chi nhỏ bài toán lớn thành các bài toán phần tử, giải bài toán phần tử đó, ứng với mỗi trường hợp giải được của bài toán phần tử đó, ta đi tìm lời giải cho bài toán phần tử tiếp theo cho đến khi bài toán lớn trở nên đầy đủ.

<Khởi tạo các thông số cần thiết>

```
void Try(int i){
    <Nếu cấu hình hiện tại là đáp ứng đủ yêu cầu thì xuất ra và thoát>
    <Duyệt các khả năng có thể có ở vị trí i>{
        <Đánh dấu là đã cấu hình ở vị trí i>
        <Gọi hàm Try(i+1)>
        <Hủy đánh dấu ở trên>
    }
}
```

Ta nhận xét rằng,

- Khi đang xét vị trí thứ i , ta đưa ra các phương án để tiếp tục xét vị trí $i+1$, có những phương án sẽ làm cho vị trí $i+1$, $i+2 \dots$ có thể tìm tiếp phương án và dẫn đến kết quả cuối cùng (Gọi là phương án khả thi) và có những phương án sẽ không có kết quả (gọi là phương án bất khả thi). Vậy thì để chương trình chạy nhanh, ta cần phải loại bỏ các phương án bất khả thi càng nhiều càng tốt.
- Thuật toán được biểu diễn bằng đệ quy nhưng nếu ta cài đặt bằng đệ quy thì sẽ không có lợi, phải sử dụng bộ nhớ stack, gọi hàm đệ quy nhiều lần, điều đó làm

chương trình sẽ chạy rất chậm, Vì vậy ta có thể cài đặt bằng không đệ quy nhưng mà tin thần giải thì vẫn dựa trên phương pháp đệ quy. Để làm được điều này, cần phải có cách sao cho có thể di chuyển và thử các khả năng trên các ô được dễ dàng.

3)Giải quyết vấn đề

- **Vấn đề 1. Ta đánh dấu các khả năng điền số và tìm cách loại bỏ các khả năng bất khả thi. Các bước làm như sau.**

➤ Khởi tạo, tất cả các ô trống đều có 9 khả năng điền từ 1 đến 9

1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9	1..9

➤ Tạo 1 stack để lưu các ô đã được điền. Cứ mỗi ô $[i,j]$ đã được điền giá trị NewValue, ta tiến hành các thao tác như sau.

- Đánh dấu trên hàng, cột và vùng chứa ô $[i,j]$ là giá trị NewValue đã được điền và không được điền nữa bằng cách đặt các giá trị $row[i,j, NewValue]$, $collum[i,j, NewValue]$, $area[i,j, NewValue]$ là bằng 0
- Đặt giá trị ở vùng $[i,j]$ như sau. $Agree[i,j,0]=1$ (Chỉ 1 giá trị có thể điền), $Agree[i,j,1]= NewValue$, $value[i,j]=1$.
- Với các ô trên hàng, cột, vùng chứa ô đó (tất nhiên phải khác ô đó) ta loại bỏ khả năng điền số NewValue ra khỏi tập các khả năng.

Ví dụ ô $[1,4]$ đã được điền số 1

2..9	2..9	2..9	1	2..9	2..9	2..9	2..9	2..9
1..9	1..9	1..9	2..9	2..9	2..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	2..9	2..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9
1..9	1..9	1..9	2..9	1..9	1..9	1..9	1..9	1..9

- Ta tiến hành tìm số chưa từng được xét đưa vào stack và cũng được xử lý như trên cho đến khi không tìm thấy ô nào nữa. Đó là các ô thỏa mãn

- Là ô duy nhất của hàng (hoặc cột, hoặc vùng) có thể điền một số nào đó

Ví dụ. hàng 4 chỉ có thể điền số 5 tại ô ở cột 6 (dấu x), như vậy ta sẽ xóa khả năng điền được số 5 tại các ô đánh dấu (-),

				5				
-	-	-	4	-	x	1	2	3
			-		-			
		5	-		-			
					-			
					-			
					-			

- Là ô chỉ có một khả năng điền. Ở ví dụ sau, ô [5,6] (Đánh dấu x) chỉ có thể điền số 7. Nhờ đó các ô đánh dấu (-) có thể được loại bỏ số 7 ra khỏi tập phương án,

					-			
					4			
					5			
			1	2	3			
9	8	-	-	-	X	-	6	-
					-			
					-			
					-			
					-			

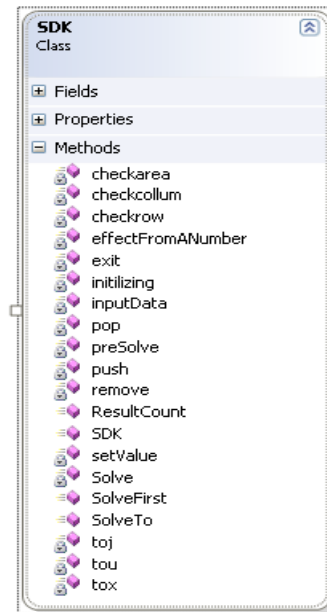
Ngoài ra nếu như khi kiểm tra có hàng, cột hay vùng nào đó mà không thể điền được một giá trị nào đó thì kết quả sẽ là không có nghiệm. Hoặc có 1 ô nào đó không có khả năng điền, sudoku cũng sẽ không có nghiệm.

- Vấn đề 2.** Khử đệ quy trong thuật toán. Tư tưởng chính, ta sử dụng một biến điều khiển việc di chuyển lên trước, ra sau giữa các ô số. Để việc di chuyển dễ dàng, ta dùng cách xác định ô số thứ nhất. Biến điều khiển add sẽ mang giá trị 1 nếu cần phải tiến lên phía trước, -1 nếu quay về phía sau. Biến index để chỉ vị trí hiện tại.

- Thuật toán cơ bản để tìm ra đáp án được tóm tắt như sau :

- Bước 1 : index = 1 (Đang xét ô 1) add = 1 (đang tiến).
- Bước 2 : Lặp trong khi vị trí tiếp ta xét là lớn hơn 0
 - Nếu index == 82 (nghĩa là từ ô 1 đến ô 81 đã đúng), ta đưa ra kết quả và thoát.

- Nếu ô hiện tại chỉ có 1 khả năng thì tiếp tục vòng lặp. (Ta không xét những ô này), những ô này luôn có value[,] bằng 1
 - Nếu ô hiện tại có giá trị chính là giá trị tối đa, nghĩa là không thể tăng lên nữa, thì ta cho nó về giá trị nhỏ nhất, cho add=-1, tiếp tục vòng lặp, lùi về ô trước nó.
 - Ta cần thay đổi giá trị trên ô số hiện thời nên giá trị cũ bị bỏ. Những ràng buộc của giá trị cũ đối với hàng, cột và vùng bị bỏ.
 - Tìm giá trị trong tập khả năng có thể điền cho ô hiện tại, giá trị đó phải đạt 2 yêu cầu :
 - Chưa được điền trong hàng, cột, vùng chứa nó.
 - Xây dựng một cấu hình sudoku theo chiều hướng tăng theo thứ tự từ điển, nghĩa là ta phải tìm một giá trị lớn hơn hoặc bằng giá trị hiện thời.
 - Nếu ở bước trên
 - Tìm thấy giá trị thỏa mãn thì ta tạo ràng buộc đã điền giá trị trong cột, hàng, ô chứa ô số đó. Cho add=1, sau đó tiếp tục vòng lặp để tiến tiếp ô phía sau.
 - Nếu không tìm thấy thì ta cho add=-1, tiếp tục vòng lặp ta xét ô liền trước nó.
- Thực tế, ta không chỉ cần tìm 1 đáp án mà ta còn phải đếm số đáp án, xem đáp án không phải là đầu tiên, nên từ thuật toán cơ bản trên, ta có một số điều chỉnh như sau.
- Ta tạo biến **ResultCount** để đếm số đáp án đã tìm được, cứ hề tìm thấy đáp án thì ta tăng biến này thêm 1.
 - Ta thêm tham số count cho hàm, tham số về số kết quả
 - Nếu là -1 tức là đếm số kết quả, gộp tham số này, ta không bao giờ xuất kết quả, khi ResultCount lớn hơn tối đa số kết quả cần tìm hoặc không tìm thấy kết quả nào nữa thì ta trả về giá trị của **ResultCount**.
 - Nếu là 0 là in ra kết quả đầu tiên, chương trình sẽ chạy như thuật toán cơ bản. Nếu tìm thấy kết quả, chương trình sẽ xuất ra kết quả đó và trả về giá trị 1, nếu không thấy, trả về 0;
 - Nếu là một số dương thì ta sẽ in ra kết quả nếu **ResultCount == count**, Nếu tìm thấy kết quả, chương trình sẽ xuất ra kết quả đó và trả về giá trị 1, nếu không thấy, trả về 0;
- Chương trình giải. Do được cài đặt trên C#, thuật toán được cài đặt trong lớp SDK, lớp này có các phương thức như sau quan trọng:
- `public SDK(int[] _pro)` : Tạo ra đối tượng trên lớp SDK, khởi tạo các giá trị cho các biến cần thiết cho các hàm giải.
 - `private bool inputData()` : Đưa đề bài có dạng là mảng hai chiều để khởi tạo các mảng dùng cho việc giải như mảng agree[,], mảng value[,].



Ngoài ra, hàm cũng phát hiện ra trường hợp đề bài có mâu thuẫn (như trong 1 hàng có 2 ô cùng giá trị...), lúc đó đề bài không có đáp án.

- `private void preSolve()`: Giải thủ công, bước giải này là để chuẩn bị nhân hạn chế các trường hợp không dẫn tới đáp án cho hàm giải chính.
- `int Solve(int count)`: Giải đề bằng thuật toán quay lui, tham số `int count` là để điều khiển công việc của hàm như giải xem đáp án có số thứ tự nào đó, đếm số đáp án. Tùy thuộc tham số `count` mà hàm sẽ trả về các giá trị khác nhau.
- `public bool SolveFirst()`: Hàm này gọi hàm `Solve(0)` để tìm ra đáp án đầu tiên. Khi đó hàm `Solve(0)` sẽ trả về 1 nếu tìm thấy, 0 nếu không tìm thấy.
- `public bool SolveTo(int)` : Hàm gọi hàm `Solve(int)` để giải đến một đáp án nào đó.
- `public int ResultCount()` : Hàm gọi hàm `Solve(-1)` để đếm số đáp án của đề bài. Khi đó hàm `Solve` sẽ giải cho đến khi không tìm thấy đáp án nữa và trả về tổng số đáp án hoặc giải đến đáp án thứ 10.000 và kết luận đề có quá nhiều đáp án, trả về -1.

4) Chương trình giải

```
class SDK
{
    int index=0 , add=1 , i=0, j=0;
    public int MaxIndexResultCanFind;
    public int[] Result = new int[82];
    public int MaxIndex
    {
        get { return MaxIndexResultCanFind; }
        set { MaxIndexResultCanFind = value; }
    }
    bool HaveResult = true;
    const int size = 9;
    int tox(int x)
    {
        return (x-1)/size+1;
    }
    int toj(int x)
    {
        return (x-1)%size+1;
    }
}
```

```
int tou(int x, int y)
{
    return (x - 1) * size + y;
}
int [,] row = new int[10, 10];
int [,] collum=new int[10,10];
int [,] area=new int[10,10];
int [,] AREA=new int[10,10];

int [,,] agree=new int [size+1,size+1,11];
int [,] value=new int [size+1,size+1];
int [,] Area=new int [size+1,size+1];
int [,] problem=new int[size+1,size+1];
int[] stack=new int [500];
int top = 0;
void push(int a)
{
    top++;
    stack[top] = a;
}
int pop()
{
    top--;
    return stack[top + 1];
}
void initilizing()//Khởi tạo các giá trị cần thiết
{
    int i, j,
    for (i = 1; i <= size; i++) for (j = 1; j <= size; j++)
        Area[i,j] = 0;
    for (i = 1; i <= size; i++)
        for (j = 1; j <= size; j++)
        {
            row[i,j] = collum[i,j] = area[i,j] = 1;
            AREA[i,j] = 1 + ((i - 1) / 3) * 3 + (j - 1) / 3;
            for (k= 1; k <= size; k++) agree[i,j,k] = k;
            agree[i,j,10] = problem[i,j] = 0;
            agree[i,j,0] = size;
            problem[i,j] = value[i,j] = 0;
            Area[AREA[i,j],0]++;
            Area[AREA[i,j],Area[AREA[i,j],0]] = tou(i, j);
        }
}
} //Remove a number from list of available number
```

```
bool remove(int i, int j, int value)
{
    int k = 1;
    while ((agree[i,j,k] < value) && (k <= agree[i,j,0])) k++;
    if ((agree[i,j,k] == value) && (k <= agree[i,j,0]))
    {
        while (agree[i,j,k] != 0)
        {
            agree[i,j,k] = agree[i,j,k + 1];
            k++;
        }
        agree[i,j,0]--;
        return agree[i,j,0] == 1;
    }
    return false;
}

void effectFromANumber(int i, int j)
{
    int u, v;
    for (u = 1; u <= size; u++)
        for (v = 1; v <= size; v++)
        {
            if (!(u == i) && (v == j))
                if ((u == i) || (v == j) || (AREA[u,v] ==
AREA[i,j]))//
                    if (remove(u, v, agree[i,j,1]))
                        push(tou(u, v));
        }
}

bool exit(int i, int j, int v)
{
    int k;
    for (k = 1; k <= agree[i,j,0]; k++) if (agree[i,j,k] == v)
return true;
    return false;
}

void setValue(int i,int j,int newValue){
    int k;
    agree[i,j,1]=newValue;
    agree[i,j,0]=1;
    for(k=2;k<=10;k++) agree[i,j,k]=0;
    row[i,newValue]=collum[j,newValue]=area[AREA[i,j],newValue]=0;
    value[i,j]=1;
}
```

```
void checkrow()//Kiểm tra trên một hàng
{
    int i, j, value, o=0, count = 0;
    for (i = 1; i <= size; i++) for (value = 1; value <= 9; value++)
    {
        if (row[i,value] == 0) continue;
        for (count = 0, j = 1; j <= size; j++)
            if (exit(i, j, value))
            {
                o = j;
                count++;
            }
        if (count == 0)
        {
            HaveResult = false;
            return;
        }
        if ((count == 1))
        {
            setValue(i, o, value);
            push(tou(i, o));
        }
    }
}

void checkcollum()//Kiểm tra trên cột
{
    int i, j, value, o=0, count;
    for (j = 1; j <= size; j++) for (value = 1; value <= 9; value++)
    {
        if (collum[j,value] == 0) continue;
        for (count = 0, i = 1; i <= size; i++) if (exit(i, j,
value))
        {
            o = i;
            count++;
        }
        if (count == 0)
        {
            HaveResult = false;
            return;
        }
        if ((count == 1) && (agree[o,j,0] != 1))
        {
            setValue(o, j, value);
            push(tou(o, j));
        }
    }
}

void checkarea()//Kiểm tra trên vùng
{
    int k, value, o=0, count, b, i, j;
    for (k = 1; k <= 9; k++) for (value = 1; value <= 9; value++)
```

```
{
    if (area[k,value] == 0) continue;
    for (count = 0, b = 1; b <= size; b++)
    {
        i = tox(Area[k,b]);
        j = toj(Area[k,b]);
        if (exit(i, j, value)) { o = Area[k,b]; count++; }
    }
    if (count == 0)
    {
        HaveResult = false;
        return;
    }

    if (count == 1)
    {
        setValue(tox(o), toj(o), value);
        push(o);
    }
}

void preSolve()
{
    int k;
    while (top != 0)
    {
        checkrow();
        checkcollum();
        checkarea();
        k = pop();
        effectFromANumber(tox(k), toj(k));
        if (!HaveResult) return;
    };
}

int Solve(int count)
{
    int ResultCount = 0;
    do{
        index+=add;
        if(index==size*size+1)
        {
            ResultCount++;
            add = -1;
            if (count == -1)//Khi cần đếm số nghiệm
            {
                if (ResultCount > MaxIndexResultCanFind)
                    return -1;//Nếu như có quá nhiều nghiệm thì thôi
                continue;
            }
        }
    }

    else
    {
```

```

        if (ResultCount >= count)//Nếu đã tìm thấy nghiệm
thứ count
        {
            for (int ii = 1; ii <= 9; ii++)
                for (int jj = 1; jj <= 9; jj++)
                    Result[tou(ii, jj)] = agree[ii, jj],
value[ii, jj]];
            return 1;
        }
        continue;//Khi cần tìm tiếp nghiệm khác
    }
    i=tox(index);j=toj(index);

    if (agree[i, j, 0] == 1)
    {
        value[i, j] = 1;
        continue;
    }

    if(value[i,j]!=0) row[i, agree[i, j, value[i, j]]]=collum[j, agree[i, j, value[i, j]]
]=area[AREA[i, j], agree[i, j, value[i, j]]]=1;
        if((add<0) && (value[i, j]==agree[i, j, 0])) {
            value[i, j]=0;continue;
        }
        for(value[i, j]++;value[i, j]<=agree[i, j, 0];value[i, j]++)

    if((row[i, agree[i, j, value[i, j]]]!=0) && (collum[j, agree[i, j, value[i, j]]]!=0) &&
(area[AREA[i, j], agree[i, j, value[i, j]]]!=0))
    {

        row[i, agree[i, j, value[i, j]]]=collum[j, agree[i, j, value[i, j]]]=area[AREA[i, j],
agree[i, j, value[i, j]]]=0;

            add=1;break;
        }
        if(value[i, j]>agree[i, j, 0])
    {
        value[i, j]=0;add=-1;
    }
    }while(index+add>0);
    return count== -1?ResultCount:0;
}

//=====
private int[] pro=new int [82];
private bool inputData()
{
    bool run=true;
    int i,j,a;
    for(i=1;i<=size;i++)
        for(j=1;j<=size;j++){
            a = pro[tou(i, j)];
            if(a!=0){
                if(!exit(i, j, a))// agree[i, j, 1]
run= false ;//Can not solve
                setValue(i, j, a);
            }
        }
    }
}

```

```
        effectFromANumber(i, j);
        push(tou(i, j));
        problem[i, j]=a;
    };
}
return run;
}
public SDK(int[] _pro)
{
    initilizing();
    pro = _pro;
    MaxIndexResultCanFind = 10000;
    HaveResult = inputData();
    index = 0; add = 1; i = 0; j = 0;

    top = 1; // Top of stack
    preSolve();
}
public bool SolveFirst() { //Beginning Solve

    return (!HaveResult) ? false : Solve(0) == 1;
}
public bool SolveTo(int _index) {
    return (!HaveResult) ? false : Solve(_index) == 1;
}
public int ResultCount() {
    return (!HaveResult) ? 0 : Solve(-1);
}
}
```


VI. Giới thiệu phần mềm giải

1) Tổng quan

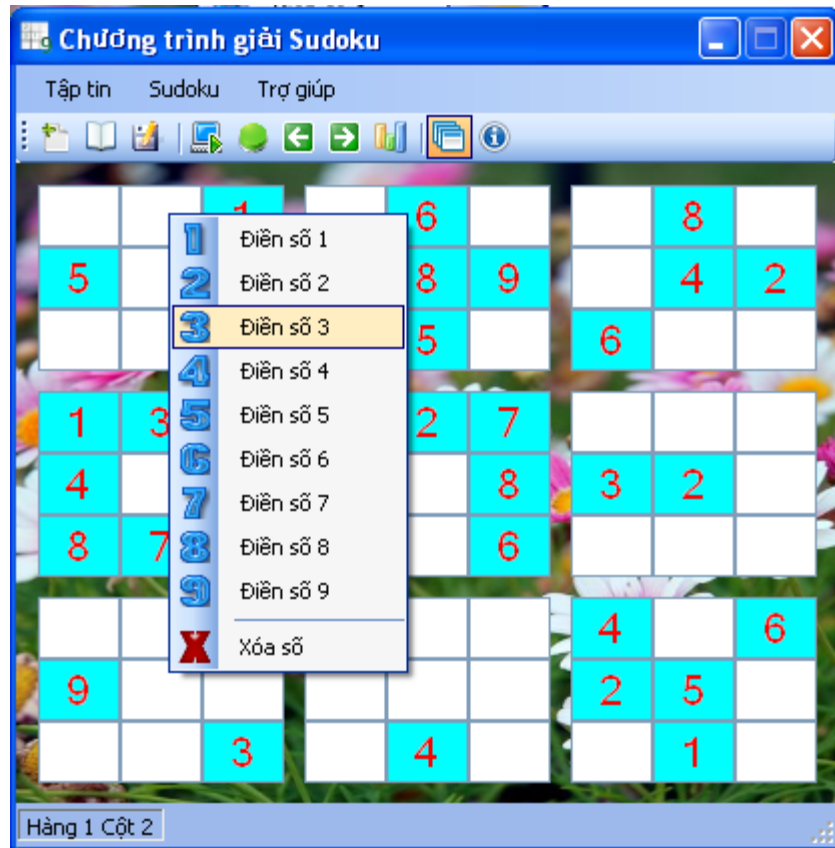
Chương trình được viết bằng ngôn ngữ C#, chạy trên nền tảng Framework .Net nên để chạy được, yêu cầu hệ thống cài sẵn Framework .Net 2.0 trở lên.

Giao diện chương trình

- Trên cùng là hệ thống menu
- Dưới đó là thanh công cụ
- Khu vực lớn nhất là vùng nhập / hiển thị Sudoku
- Dưới cùng là thanh trạng thái



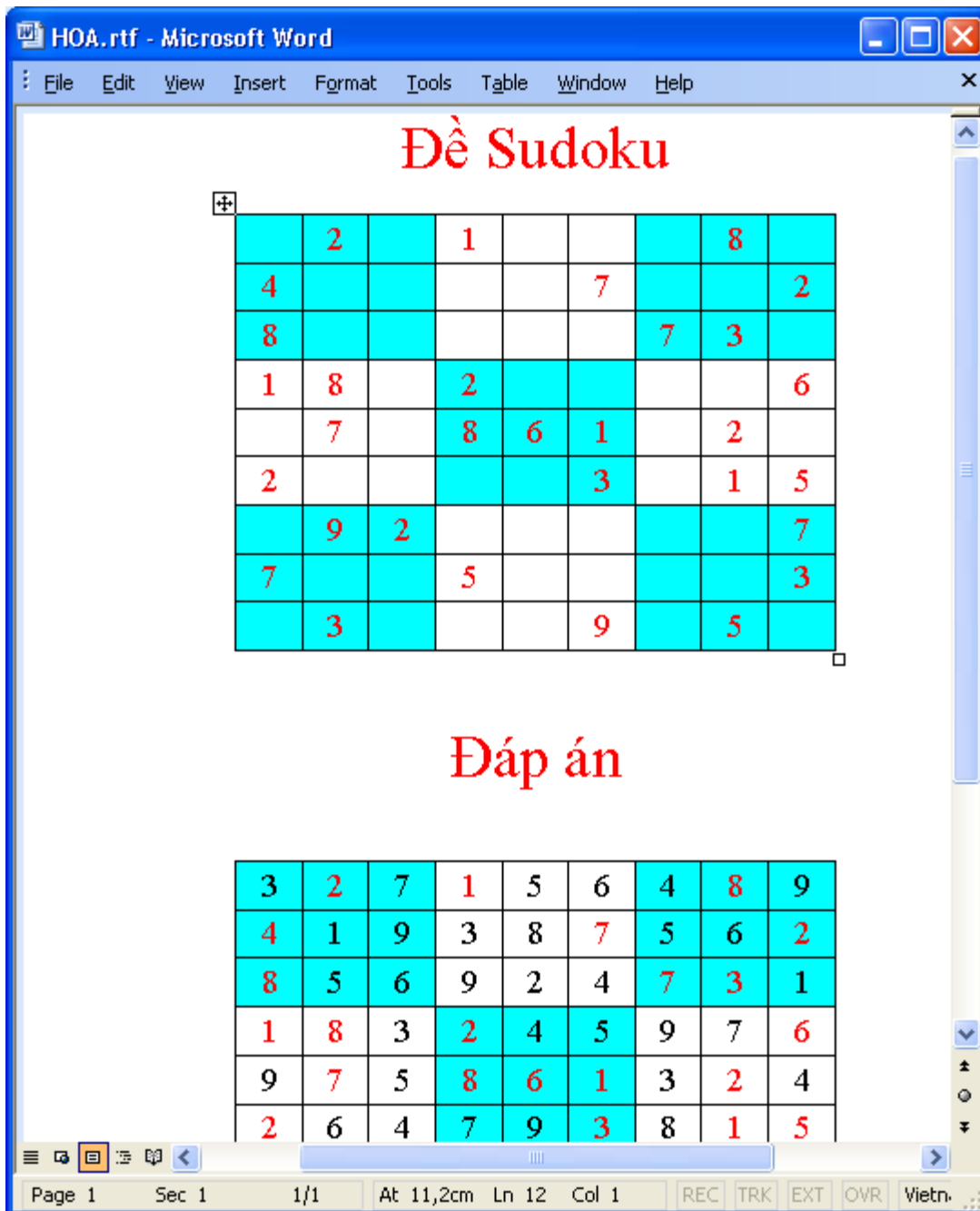
Giao diện chung



Giao diện nhập dữ liệu



Giao diện xem đáp án



The screenshot shows a Microsoft Word document titled "HOA.rtf". The document contains a Sudoku puzzle titled "Đề Sudoku" and its solution titled "Đáp án".

Đề Sudoku (Puzzle):

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

Đáp án (Solution):

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

Giao diện Microsoft Office Word (Tắt hết Toolbar) khi mở tập tin xuất bản

2) Những chức năng chính :

Tạo mới : Xóa tất cả các text, sẵn sàng để nhập đề bài mới.

Mở từ tập tin : Mở đề bài từ tập tin đã được lưu sẵn.

Lưu ra tập tin : Lưu đề bài ra tập tin để có thể mở vào thời gian khác.

Thông kê : Thiết lập các tùy chọn cho chương trình

Xuất bản : Lưu bài Sudoku dưới dạng Rich Text Format (.rtf) để mở bằng Winword hoặc html để mở bằng trình duyệt web.

Tùy chọn : Thiết lập một số thông số để chương trình hoạt động

Giải đáp án : Giải đề bài đang nhập, xem đáp án đầu tiên tìm thấy (nếu có).

Xem đáp án thứ : Giải đề đến đáp án do người dùng chỉ định và xuất ra (nếu có)

Đáp án trước : Giải và xem đáp án liền trước đáp án đang xem.

Đáp án sau. Giải và xem đáp án liền sau đáp án đang xem (nếu có)

Phục hồi : Trở lại đề bài để có thể chỉnh sửa đề bài được dễ dàng.

Thông kê : Xem các số liệu về đề Sudoku đang hiển thị.

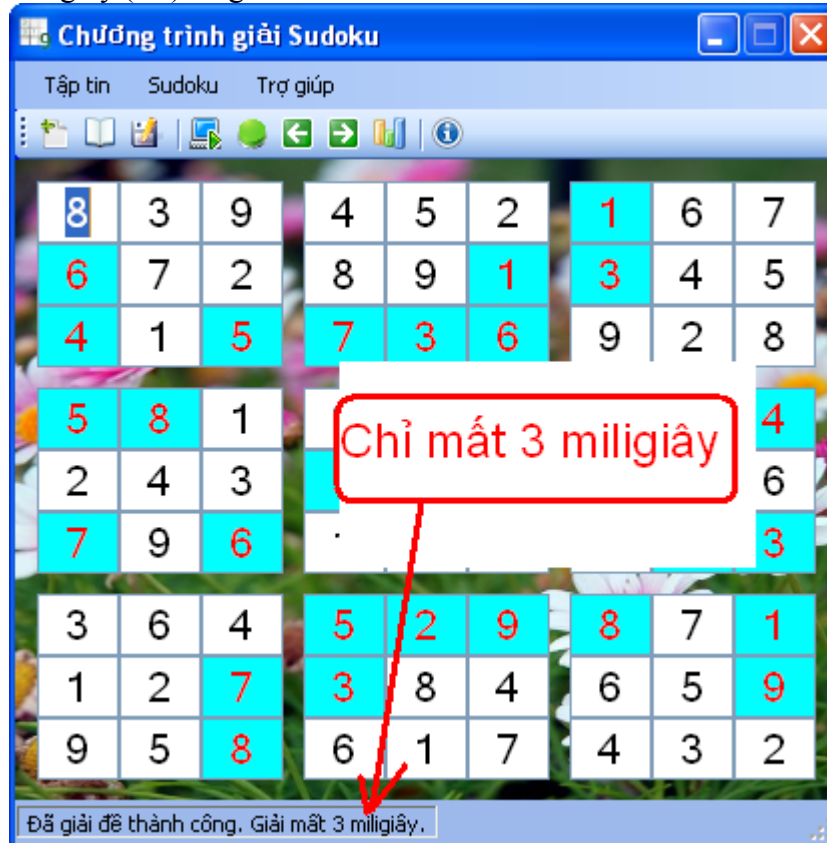
3) Bảng phím tắt

Thứ tự	Phím tắt	Chức năng
1	Ctrl + N	Tạo mới (New)
2	Ctrl + S	Lưu Sudoku vào tập tin (Save As)
3	Ctrl + O	Mở đề bài từ file (Open)
4	F5	Giải và xem đáp án đầu tiên
5	F6	Xem đáp án thứ
6	F11	Xem đáp án trước
7	F12	Xem đáp án sau
8	F1	Trợ giúp
9	Alt + R	Phục hồi lại đề bài (Recovery)
10	Alt + S	Thông kê về Sudoku (Static)
11	Shift + F1	Thông tin
12	Right, Enter, Tab	Di con trỏ đến ô kế tiếp
13	Left, Shift + Tab	Di con trỏ đến ô trước
14	Up	Di con trỏ đến ô phía trên
15	Down	Di con trỏ đến ô phía dưới
16	Home	Di con trỏ đến ô đầu hàng
17	End	Di con trỏ đến ô cuối hàng
18	Page up	Di con trỏ đến ô hàng trên cùng
19	Page down	Di con trỏ đến ô hàng dưới cùng
20	1,2,3,4,5,6,7,8,9	Điền số
21	Space, Backspace, Delete	Xóa trống một ô

VI. Thử nghiệm, đánh giá

Chương trình hoạt động theo yêu cầu, xử lý được tất cả các trường hợp của sudoku như có một đáp án, có nhiều đáp án hoặc không có đáp án, chương trình đều đưa ra được kết luận đúng với một tốc độ khá cao. Điều đó có thể thấy ở các điểm sau :

- Nạp một đề bài bình thường, sử dụng chức năng giải. Xem thời gian giải (đã tích hợp bộ đếm thời gian trong chương trình), thường một đề bài chỉ mất vài miligiây (ms) để giải. Xem hình.



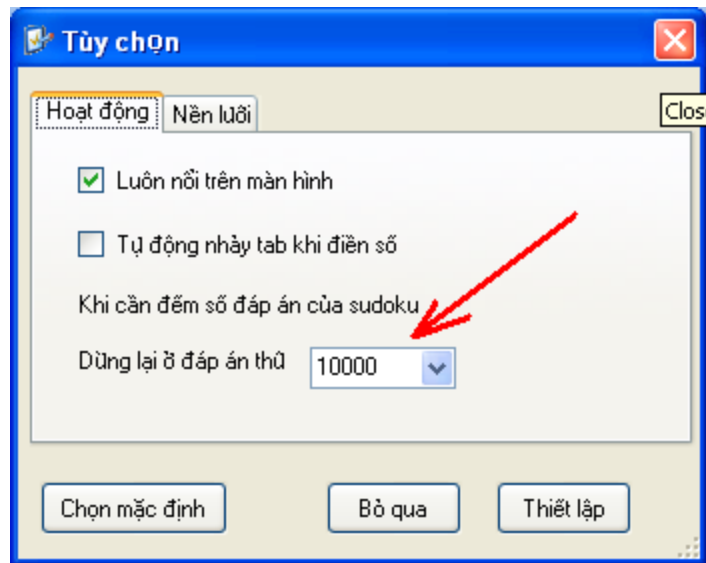
- Trường hợp không có lời giải, như minh họa, đưa ra kết quả là gần như tức thời.



- Trường hợp có nhiều đáp án (nếu sử dụng chức năng giải, chương trình sẽ đưa ra đáp án đầu tiên), sử dụng chức năng thống kê để biết số đáp án



- Nếu đề bài có quá nhiều đáp án thì chương trình chỉ giải đến đáp án thứ 10000 (mười ngàn), có thể thay đổi con số này bằng cách vào Menu Tập tin và chọn "Tùy chọn"



-
- Khi nhập một Sudoku trống (không điền sẵn ô nào cả) thì chương trình có thể giải đến đáp án thứ 1000000 (1 triệu) trong vòng không đầy nửa phút. Thử nghiệm có thể vào Menu Sudoku và chọn “Xem đáp án thứ”



Mục lục:

I. Giới thiệu về thuật toán.....	1
Thuật toán quay lui	1
II. Giới thiệu bài toán ứng dụng :	2
III. Đặc tả cấu trúc dữ liệu và giải thuật.....	2
IV. Thuật giải.....	4
1) Tổng quan.....	4
2) Vấn đề	5
3) Giải quyết vấn đề	6
4) Chương trình giải	9
VI. Giới thiệu phần mềm giải	16
1) Tổng quan.....	16
2) Những chức năng chính :	19
3) Bảng phím tắt.....	20
VI. Thử nghiệm, đánh giá	21

Liên lạc với mình qua Gmail : voquanghoa@gmail.com để có mã nguồn