• چکیده

در بازی سودوکو یک صفحه‌ی $n \times n$ شامل زیرمربع‌های $\sqrt{n} \times \sqrt{n}$ موجود ا ست. باید اعداد 1 تا n هر بار در خانه‌های هر سطر، ستون و زیرمربع از این صفحه به طور متمایز قرار بگیرند. یکی از روش‌های مرسوم حل این بازی استفاده از روش عقبگرد و در واقع بررسی تمام حالات مختلف مسئله است. در این مقاله با ا ستفاده از ار ضای محدودیت در هوش م صنوعی و فیلترینگ، راه حلی با زمان حل منا سب ارائه شده است.

واژه‌های کلیدی:

نظریه بازی، سودوکو، ارضای محدودیت، فیلترینگ در ارضای محدودیت

فهرست مطالب

صفحه

۰ چکیده ..... أ

۱ فصل اول: مقدمه ۱

۲ فصل دوم: طراحی حل‌کننده‌ی بازی ۳

۲-۱- روش عقبگرد ۴

۲-۲- فیلترینگ ارضای محدودیت ۴

۳ فصل سوم: شرح پیاده‌سازی ۶

۳-۱- منطق فایل `main` ۷

۳-۲- پیاده‌سازی کلاس `Sudoku` ۷

۴ فصل چهارم: جمع‌بندی و نتیجه‌گیری ۹

۵ منابع و مراجع ۲۷

۶ پیوست ۲۸

۱

فصل اول: مقدمه

در بازی سودوکو یک صفحه‌ی $n \times n$ شامل زیرمربع‌های $\sqrt{n} \times \sqrt{n}$ موجود ا ست. باید اعداد 1 تا n هر بار در خانه‌های هر سطر، ستون و زیرمربع از این صفحه به طور متمایز قرار بگیرند. در این گزارش فرض می‌کنیم فقط اعداد بتوانند در خانه‌های صفحه قرار بگیرند. در ابتدا با استفاده از روش عقبگرد[1] یک تابع برای حل این بازی نوشته و سپس با استفاده از فیلترینگ تابع حل‌کننده‌ی بازی بهبود داده می‌شود.

شرط فیلترینگ مطابق با همان قیدهای بازی انتخاب شده است. در ابتدا حل‌کننده بدون در نظر گرفتن قیدها، با شروع از اولین خانه‌ی صفحه و با در نظر گرفتن هر عددی برای هر کدام از مربع‌های جدول جلو می‌رفت، تا زمانی که یکی از قیدهای بازی نقض شود. در این صورت حل‌کننده به عقب برگشته و با اعداد دیگری از قبلی‌ترین خانه که آن را پر کرده بود دوباره به جلو حرکت می‌کرد.

با اعمال فیلترینگ در ارضای محدودیت، با هر بار پر کردن مربع‌ها، حل‌کننده با توجه به وضعیت به دست آمده از سطرها، ستون‌ها و مربعات، محدودیت‌هایی روی خانه‌های سطرها، ستون‌ها و مربعات تحت تاثیر قرار می‌دهد. این باعث می‌شود حل‌کننده در هنگام پر کردن مربع‌های بعدی صفحه‌ی بازی، لازم نباشد که همه‌ی اعداد را امتحان کند؛ در واقع فقط لازم است اعداد واقع در محدوده‌ی مجاز را امتحان کند.

[1] backtracking

۲ فصل دوم: طراحی حل‌کننده‌ی بازی

در این فصل مدل و منطق رفتار حل‌کننده‌ی بازی توضیح داده می‌شود. این مدل با توجه به ارضای محدودیت و همین طور به صورت کارآ پیاده‌سازی شده است. علت این کارآیی در ادامه شرح داده می‌شود.

۲-۱-روش عقبگرد

در روش عقبگرد از مربع اول صفحه‌ی بازی شروع به حرکت کرده و به مرور به صفحه‌های خالی رفته و به صورت بازگشتی مربع‌ها حل می‌شوند. به وضوح پیچیدگی زمانی این روش بسیار زیاد است، چون در هر خانه‌ی یک صفحه‌ی $n \times n$ می‌توان هر عدد از 1 تا n را امتحان کرد. در این روش تابع حل‌کننده را روی اولین خانه‌ی صفحه فراخوانی می‌کنیم. این تابع هر عدد مجاز برای آن خانه را امتحان کرده و اگر آن عدد یکی از قیدهای مذکور برای بازی را نقض کند، عدد بعدی را امتحان می‌کند. اما در غیر این صورت، یعنی اگر آن عدد هیچ قیدی را نقض نکند، به طور بازگشتی خانه‌ی بعدی (از چپ به راست و از بالا به پایین) را حل می‌کند. اگر خانه‌ی بعدی قابل حل باشد، در این صورت این خانه هم قابل حل بوده و یک سیگنال درست برمی‌گرداند. اگر خانه‌ی بعدی قابل حل نباشد، اعداد دیگر امتحان شده و اگر هیچ کدام از اعداد جواب خانه نباشند، یعنی آن خانه قابل حل نبوده و سیگنال نادرست برگردانده می‌شود.

۲-۲-فیلترینگ ارضای محدودیت

برای بهبود حل‌کننده‌ی مذکور باید در هر مرحله محدودیت‌هایی اعمال شود. این محدودیت‌ها با توجه به همان قیدهای بازی معین می‌شود. یعنی در همان روش عقبگرد بعد از امتحان هر عدد در مربعی که تابع حل‌کننده برای آن فراخوانی شده است، دیگر نمی‌توان همان عدد را در سطر، ستون و مربع متناظر با آن عدد قرار داد. یعنی مربعاتی وجود دارند که وقتی در ادامه حل‌کننده به آنها می‌رسد لازم نیست که همه‌ی اعداد را در آنها امتحان کند. پس می‌توان به کلی رویکرد را تغییر داد که در قسمت پیاده سازی با جزئیات بیشتر به آن اشاره شده است. یعنی به جای اینکه هر بار مجاز بودن هر عدد را که می‌خواهیم در خانه‌ای قرار دهیم بررسی کنیم، سه آرایه در نظر می‌گیریم که در خانه‌های هر آرایه مجموعه‌هایی هستند که در آنها اعداد غیرمجاز برای سطر، ستون و مربعی که آن خانه در آنها قرار گرفته است ذخیره می‌شوند. پس در هر بار قرار دادن عددی در یک خانه کافی است این عدد غیرمجاز نباشد، یا به عبارتی دیگر در

ســه مجموعه‌ی اعداد غیرمجاز متناظر نباشــد. به عنوان مثال برای قرار دادن عددی در خانه‌ی (1, 3)ام یک سودوکوی ۹×۹، اعدادی که عضو مجموعه‌ی خانه‌ی اول آرایه‌ی مربوط به اعداد غیرمجاز در سطرها، مجموعه‌ی خانه‌ی ســوم آرایه‌ی مربوط به اعداد غیرمجاز در ســتون‌ها و مجموعه‌ی خانه‌ی اول[۱] آرایه‌ی مربوط به اعداد غیرمجاز در مربعات نبا شند برگزیده می شوند. بقیه‌ی روش هم به صورت همان عقبگرد است.

[۱] فرض می‌کنیم indexing از صفر شروع شده باشد.

۳

فصل سوم: شرح پیاده‌سازی

پیاده سازی بازی و حل‌کننده‌ی آن در بستر گیت‌هاب در یک منبع خصوصی انجام شد. در ابتدا قسمت عقبگرد و پس از آن قسمت فیلترینگ نوشته شد که در اینجا به جزئیات آن می‌پردازیم.

۳-۱-منطق فایل main

این فایل یک تابع main() داشته که در ابتدا از کاربر ورودی می‌گیرد. ورودی گرفتن را می‌توان با تابع get_sudoku_from_file() که از فایل ورودی می‌گیرد و یا get_sudoku_from_commandline() که از خطر فرمان ورودی می‌گیرد انجام داد که خروجی هر دو یک سودوکو از کلاس Sudoku است که در ادامه توضیح داده خواهد شد. آنگاه متد بازگشتی solve(i, j) برای حل این سودوکو روی خانه‌ی اول صفحه فراخوانی شده که اگر درست برگرداند، یعنی سودوکو حل شده و در غیر این صورت یعنی غیرقابل حل است که در آن صورت Unsolvable CSP! به کاربر نمایش داده می‌شود. در نهایت زمان اجرای حل سودوکو نیز نمایش داده می‌شود که برای بررسی کدها روی سودوکوهای مختلف قرار داده شده است.

۳-۲-پیاده‌سازی کلاس Sudoku

این کلاس اندازه‌ی صفحه و یک آرایه‌ی دوبعدی را نگه می‌دارد که نمایانگر صفحه‌ی بازی است. به عنوان مثال اندازه‌ی یک بازی ۹×۹، ۱۹ است که صفحه‌ی آن یک آرایه‌ی دو بعدی است. همچنین سه آرایه هر کدام به طول اندازه‌ی صفحه در ابتدا ایجاد می‌کنیم؛ یکی برای سطرها، یکی برای ستون‌ها و در نهایت یکی برای مربع‌ها. هر عضو هر کدام از این آرایه‌ها مجموعه‌ای است که بیانگر اعداد غیرمجاز در سطر، ستون یا مربع متناظر با جایگاه آن عضو از آرایه است. به عنوان مثال اگر عضو سوم آرایه‌ی مربوط به مربع‌ها مجموعه‌ای شامل اعداد ۲، ۳ و ۵ باشد، یعنی هیچ خانه‌ای از مربع چهارم[1] نمی‌تواند شامل هر کدام از اعداد ۲، ۳ و ۵ باشد.

[1] شروع از صفر است.

برای پر کردن یا خالی کردن خانه‌ای از صفحه از تابع `set_entry(i, j, value)` استفاده می‌کنیم. این تابع در صورتی که مقدار `value` غیر صفر باشد، عضو فعلی خانه‌ی (i, j)ام را از آرایه‌ی اعداد غیرمجاز سطر، ستون و مربع متناظر با آن خانه پاک کرده و مقدار جدید را جایگزین می‌کند. اما اگر مقدار `value` صفر باشد لازم نیست که مقدار جدید در آرایه‌ی اعداد غیرمجاز متناظر قرار داده شود.

چند تابع کمکی و ساده در این کلاس وجود دارند، مثل تابع `move_to_next_cell(i, j)` که جهت حرکت حل‌کننده را مشخص می‌کند. تابع `is_empty(i, j)` هم خالی بودن خانه را مشخص می‌کند. همچنین تابع `is_valid(i, j)` که از تابع کمکی `has_conflict(i, j, p, q)` استفاده می‌کند در ابتدا پیاده سازی شده بود که مجاز بودن عدد جدید در یک خانه را با پیمایش سطرها، ستون‌ها و مربع‌ها بررسی می‌کرد. اما پس از پیاده سازی سه آرایه‌ی مربوط به اعداد غیرمجاز، دیگر لازم به آن نبود و اعداد غیرمجاز در مجموعه‌هایی قرار گرفته و این سرعت را بسیار بیشتر و الگوریتم را کارآتر می‌کرد.

تابع `solve(i, j)` همان حل‌کننده‌ی بازی است که روی خانه‌ی `(i, j)`ام فراخوانی می‌شود. اگر این خانه از صفحه بیرون باشد، یعنی خارج از محدوده‌ی مجاز آرایه باشد، سیگنال درست برگردانده می شود که یعنی مسئله به درستی حل شده است. در غیر این صورت اگر خانه از قبل پر باشد آنقدر به خانه‌ی بعدی رفته تا سرانجام به یک خانه‌ی خالی برسد. آنگاه به ازای همه‌ی مقادیر مجاز خانه‌ی فعلی پر شده و حل خانه‌ی بعدی به صورت بازگشتی فراخوانی می شود. اگر این خانه‌ی بعدی به درستی حل شد سیگنال درست فرستاده شده و در غیر این صورت باقی مقادیر مجاز بررسی می شوند. در نهایت چنانچه هیچ کدام از مقادیر مجاز در خانه نمی‌توانستند قرار بگیرند سیگنال نادرست برگردانده می شود، یعنی آن خانه قابل حل شدن نیست؛ پس اگر خانه‌ی اول قابل حل نباشد، کل سودوکو غیرقابل حل است.

۴

فصل چهارم: جمع‌بندی و نتیجه‌گیری

برای بررســـی کارکرد کدها الگوریتم را روی چند جدول مختلف امتحان کردیم که در ادامه این مثال‌ها آورده می‌شوند. این مثال‌ها با کمک یک فایل نمونه آزمایش شده‌اند، اما می‌توان با استفاده از تابع دیگری که در پیاده‌سازی گفته شد آنها را امتحان کرد. همچنین در بعضی از حالت‌ها که زمان اجرا بیشتر طول می‌کشید، روی یک کامپیوتر شخصی سریع‌تر هم آزمایش انجام و گزارش شده است.

مثال اول همان سودوکویی است که در فایل راهنما آمده است. در این مثال $n = 9$ و $c = 34$ است.

ورودی:

```
9
34
0 1 2
0 2 6
0 6 8
0 7 1
1 0 3
1 3 7
1 5 8
1 8 6
2 0 4
2 4 5
2 8 7
3 1 5
3 3 1
3 5 7
3 7 9
4 2 3
4 3 9
4 5 5
```

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

خروجی:

```
Before solve:
0 2 6 0 0 0 8 1 0
3 0 0 7 0 8 0 0 6
4 0 0 0 5 0 0 0 7
0 5 0 1 0 7 0 9 0
0 0 3 9 0 5 1 0 0
0 4 0 3 0 2 0 5 0
1 0 0 0 3 0 0 0 2
5 0 0 2 0 4 0 0 9
```

```
0 3 8 0 0 0 4 6 0
After solve:
7 2 6 4 9 3 8 1 5
3 1 5 7 2 8 9 4 6
4 8 9 6 5 1 2 3 7
8 5 2 1 4 7 6 9 3
6 7 3 9 8 5 1 2 4
9 4 1 3 6 2 7 5 8
1 9 4 8 3 6 5 7 2
5 6 7 2 1 4 3 8 9
2 3 8 5 7 9 4 6 1
Execution time: 0.015997886657714844 seconds
```

جواب تولید شده مطابق همان جواب فایل بوده و همچنین زمان مناسبی نیز روی کامپیوتر شخصی اجرا شده است.

مثال دوم از سایت (1sudoku.com, n.d.) آورده شده که در واقع همان طور که در شکل ۱-۴ هم دیده می‌شود، n = 16 بوده و درجه‌ی Expert یعنی بالاترین درجه‌ی سختی برای آن انتخاب شده است. لازم به ذکر است که حروف A، B، C، D، E، F و G در شکل به ترتیب با اعداد ۱۰، ۱۱، ۱۲، ۱۳، ۱۴، ۱۵ و ۱۶ جایگزین شده‌اند.

ورودی:

```
16
120
0 3 13
0 6 8
0 7 16
0 8 1
0 9 14
```

```
0 12 2
1 3 14
1 4 5
1 5 3
1 10 11
1 11 10
1 12 4
2 2 7
2 3 6
2 4 4
2 6 14
2 7 12
2 8 15
2 9 8
2 11 2
2 12 9
2 13 3
3 0 8
3 1 1
3 2 4
3 6 11
3 9 7
3 13 13
3 14 14
3 15 6
4 1 10
4 2 6
```

```
4 5 4
4 7 8
4 8 11
4 10 2
4 13 15
4 14 9
5 1 8
5 4 11
5 5 16
5 10 15
5 11 14
5 14 12
6 0 9
6 2 1
6 3 16
6 7 7
6 8 8
6 12 3
6 13 6
6 15 11
7 0 4
7 2 11
7 4 9
7 6 13
7 9 6
7 11 12
7 13 10
```

```
7 15 2
8 0 14
8 2 9
8 4 15
8 6 5
8 9 11
8 11 16
8 13 8
8 15 4
9 0 13
9 2 15
9 3 11
9 7 14
9 8 10
9 12 12
9 13 5
9 15 9
10 1 16
10 4 12
10 5 8
10 10 13
10 11 5
10 14 7
11 1 7
11 2 8
11 5 11
11 7 13
```

```
11 8 4
11 10 9
11 13 16
11 14 1
12 0 15
12 1 9
12 2 14
12 6 16
12 9 5
12 13 1
12 14 2
12 15 13
13 2 2
13 3 10
13 4 3
13 6 15
13 7 11
13 8 13
13 9 12
13 11 1
13 12 5
13 13 9
14 3 1
14 4 6
14 5 9
14 10 10
14 11 15
```

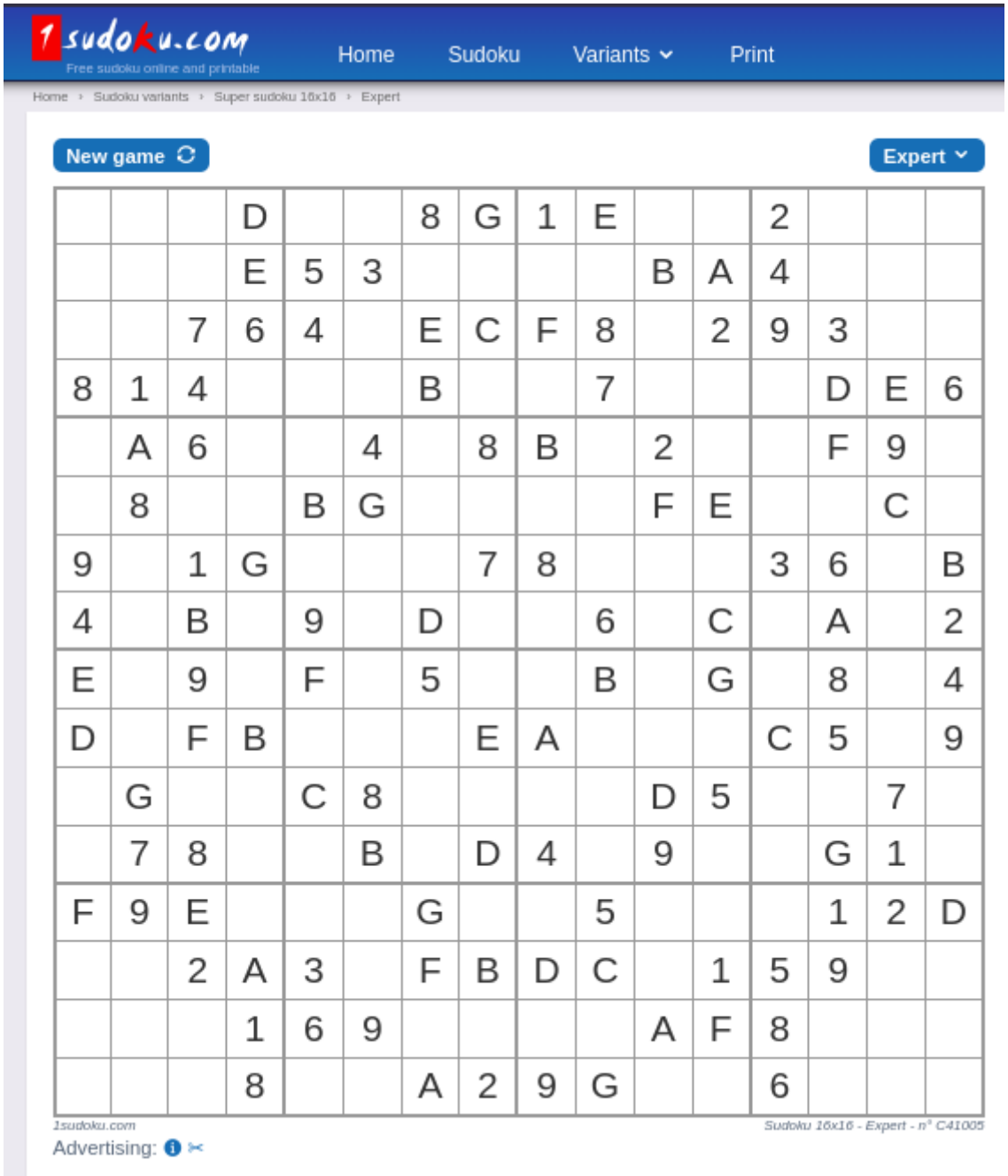
```
14 12 8
15 3 8
15 6 10
15 7 2
15 8 9
15 9 16
15 12 6
```

خروجی:

```
Before solve:
00 00 00 13 00 00 08 16 01 14 00 00 02 00 00 00
00 00 00 14 05 03 00 00 00 00 11 10 04 00 00 00
00 00 07 06 04 00 14 12 15 08 00 02 09 03 00 00
08 01 04 00 00 00 11 00 00 07 00 00 00 13 14 06
00 10 06 00 00 04 00 08 11 00 02 00 00 15 09 00
00 08 00 00 11 16 00 00 00 00 15 14 00 00 12 00
09 00 01 16 00 00 00 07 08 00 00 00 03 06 00 11
04 00 11 00 09 00 13 00 00 06 00 12 00 10 00 02
14 00 09 00 15 00 05 00 00 11 00 16 00 08 00 04
13 00 15 11 00 00 00 14 10 00 00 00 12 05 00 09
00 16 00 00 12 08 00 00 00 00 13 05 00 00 07 00
00 07 08 00 00 11 00 13 04 00 09 00 00 16 01 00
15 09 14 00 00 00 16 00 00 05 00 00 00 01 02 13
00 00 02 10 03 00 15 11 13 12 00 01 05 09 00 00
00 00 00 01 06 09 00 00 00 00 10 15 08 00 00 00
00 00 00 08 00 00 10 02 09 16 00 00 06 00 00 00
After solve:
```

```
03 15 05 13 07 06 08 16 01 14 04 09 02 12 11 10
16 02 12 14 05 03 01 09 06 13 11 10 04 07 15 08
10 11 07 06 04 13 14 12 15 08 16 02 09 03 05 01
08 01 04 09 02 10 11 15 05 07 12 03 16 13 14 06
07 10 06 05 01 04 12 08 11 03 02 13 14 15 09 16
02 08 13 03 11 16 06 10 07 09 15 14 01 04 12 05
09 12 01 16 14 15 02 07 08 10 05 04 03 06 13 11
04 14 11 15 09 05 13 03 16 06 01 12 07 10 08 02
14 06 09 02 15 07 05 01 12 11 03 16 13 08 10 04
13 03 15 11 16 02 04 14 10 01 07 08 12 05 06 09
01 16 10 04 12 08 09 06 14 15 13 05 11 02 07 03
15 09 14 07 08 12 16 04 03 05 06 11 10 01 02 13
06 04 02 10 03 14 15 11 13 12 08 01 05 09 16 07
11 13 16 01 06 09 07 05 02 04 10 15 08 14 03 12
12 05 03 08 13 01 10 02 09 16 14 07 06 11 04 15
Execution time: 3.5928616523742676 seconds
```

حل این سودوکو حدود ۴ ثانیه روی یک کامپیوتر شخصی و به مراتب کمتر، یعنی حدود ۱ ثانیه روی کامپیوتر دیگر طول کشید که باز هم عدد مناسبی است. حل آن نیز با وارد کردن اعداد به دست آمده توسط برنامه در سایت مورد نظر بررسی شد که درست بود. هرچند با بررسی خروجی به دست آمده نیز می‌توان از صحت آن اطمینان حاصل کرد.



			D			8	G	1	E			2			
			E	5	3					B	A	4			
		7	6	4		E	C	F	8		2	9	3		
8	1	4				B			7				D	E	6
	A	6			4		8	B		2			F	9	
	8			B	G					F	E			C	
9		1	G				7	8				3	6		B
4		B		9		D			6		C		A		2
E		9		F		5			B		G		8		4
D		F	B				E	A				C	5		9
	G			C	8					D	5			7	
	7	8			B		D	4		9			G	1	
F	9	E				G			5				1	2	D
		2	A	3		F	B	D	C		1	5	9		
			1	6	9					A	F	8			
			8			A	2	9	G			6			

شکل۱-۴ سودوکو با n = 16 و c = 120 و درجه‌ی سختی بیشینه‌ی Expert

مثال بعدی از ویکی‌پدیا در شکل ۲-۴ و پاسخ آن در شکل ۳-۴ آمده است.

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

شکل۲-۴ سودوکو با $n = 9$ و $c = 30$

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

شکل۳-۴ پاسخ سودوکوی شکل ۲-۴

ورودی:

```
9
30
0 0 5
0 1 3
0 4 7
1 0 6
```

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

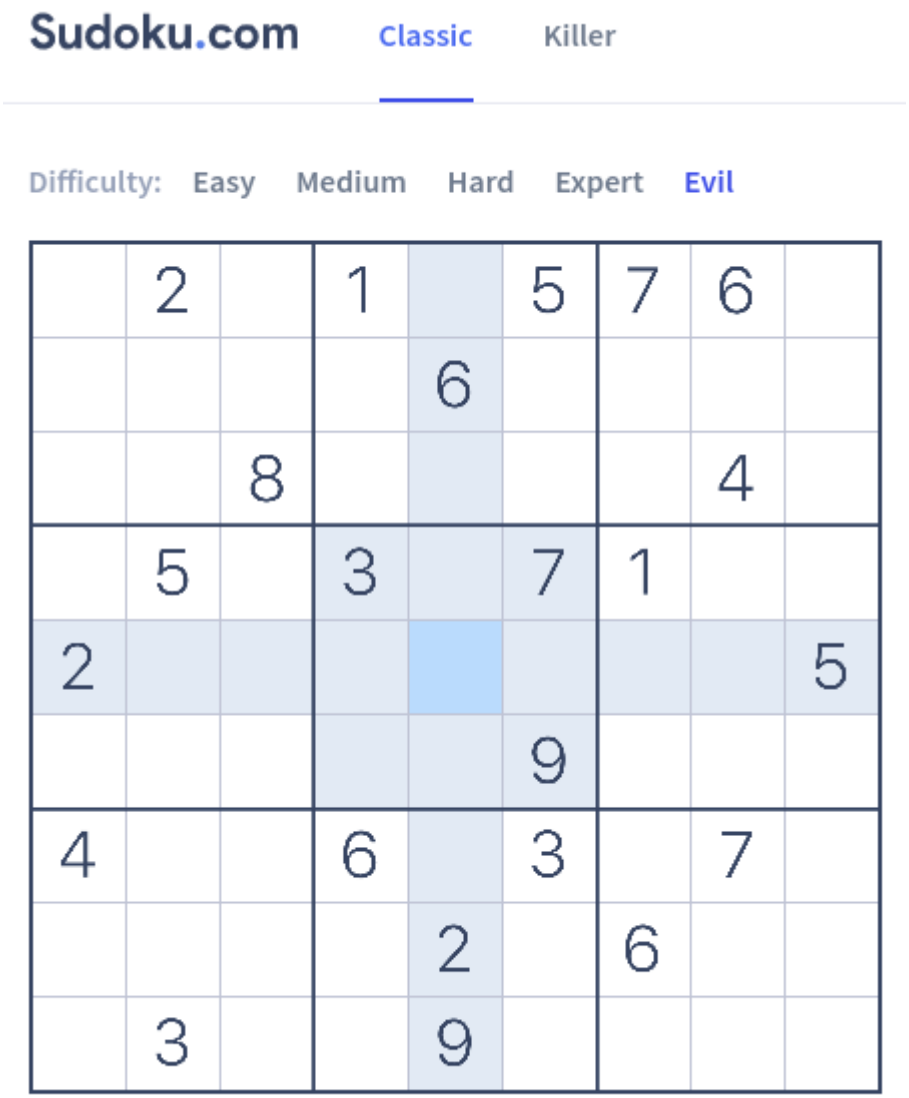
```
8 8 9
```

خروجی:

```
Before solve:
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
After solve:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
Execution time: 0.3300044536590576 seconds
```

زمان اجرای این برنامه در کامپیوتر شخصی کندتر حدود ۳۳۰ میلی‌ثانیه و در کامپیوتر شخصی سریع‌تر حدود ۳۰ میلی‌ثانیه است که باز هم زمان مناسبی است.

مثال بعدی از (Sudoku.com, n.d.) با درجه‌ی سختی بیشینه‌ی Evil در شکل ۴-۴ آورده شده و پاسخ برنامه با (Sudoku Solutions, n.d.) بررسی شده که در شکل ۴-۵ آمده است.



شکل۴-۴ سودوکو با n = 9 و c = 23

	1	2	3	4	5	6	7	8	9
A	9	2	4	1	3	5	7	6	8
B	5	7	3	4	6	8	9	1	2
C	1	6	8	9	7	2	5	4	3
D	8	5	9	3	4	7	1	2	6
E	2	4	7	8	1	6	3	9	5
F	3	1	6	2	5	9	4	8	7
G	4	9	5	6	8	3	2	7	1
H	7	8	1	5	2	4	6	3	9
I	6	3	2	7	9	1	8	5	4

شکل۵-۴ حل سودوکوی شکل ۴-۴

ورودی:

```
9
23
0 1 2
0 3 1
0 5 5
0 6 7
0 7 6
```

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

خروجی:

```
Before solve:
0 2 0 1 0 5 7 6 0
0 0 0 0 6 0 0 0 0
```

```
0 0 8 0 0 0 0 4 0
0 5 0 3 0 7 1 0 0
2 0 0 0 0 0 0 0 5
0 0 0 0 0 9 0 0 0
4 0 0 6 0 3 0 7 0
0 0 0 0 2 0 6 0 0
0 3 0 0 9 0 0 0 0
After solve:
9 2 4 1 3 5 7 6 8
5 7 3 4 6 8 9 1 2
1 6 8 9 7 2 5 4 3
8 5 9 3 4 7 1 2 6
2 4 7 8 1 6 3 9 5
3 1 6 2 5 9 4 8 7
4 9 5 6 8 3 2 7 1
7 8 1 5 2 4 6 3 9
6 3 2 7 9 1 8 5 4
Execution time: 0.4730360507965088 seconds
```

زمان اجرای این برنامه در کامپیوتر شخصی کندتر حدود ۴۷۰ میلی‌ثانیه و در کامپیوتر شخصی سریع‌تر حدود ۲۰۰ میلی‌ثانیه است که باز هم زمان مناسبی است.

پس الگوریتم در زمان مناسب و به طور کارآ بازی‌های بسیار سخت سودوکو در ابعاد بالا را حل می‌کند و با توجه به منطق پیاده شده می‌توان به صحت آن اطمینان داشت.

۵ منابع و مراجع

(n.d.). Retrieved from 1sudoku.com: https://1sudoku.com/

(n.d.). Retrieved from Sudoku.com: https://sudoku.com/

(n.d.). Retrieved from Sudoku Solutions: https://www.sudoku-solutions.com/