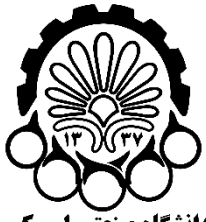
به نام خداوند بخشنده و مهربان



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

درس هوش مصنوعی

گزارش پروژه

ارضاء محدودیت

استاد درس:

دکتر قطعی

نام دانشجو:

امیررضا رادجو

۹۷۱۳۰۱۸

بهار ۱۴۰۰

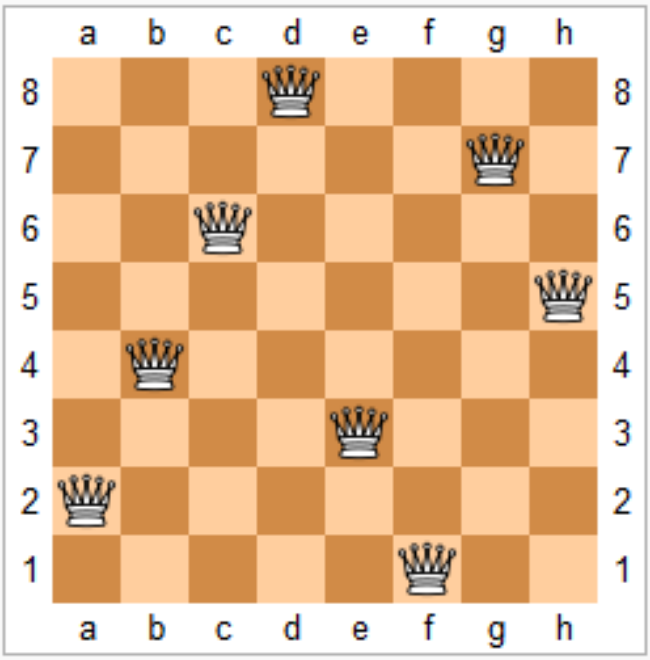
مساله N-Queen با روش‌های CSP :

مساله N-Queen به این صورت می‌باشد که در یک صفحه شطرنج از ما خواسته می‌شود تا تعداد n وزیر را طوری قرار دهیم که هیچ یک از آن‌ها همدیگر را تهدید نکنند.

حرکت وزیرها در بازی شطرنج به حالت‌های افقی عمودی و اریب می‌باشد.

بنابرین هیچ دو وزیری نمی‌توانند در یک ردیف ستون و یا قطری از صفحه شطرنج قرار داشته باشند بدون آنکه یکدیگر را تهدید کنند.

در عکس پایین نمونه‌ای از حالت قابل قبول در صفحه شطنج استاندارد را مشاهده میکنید.



One solution to the eight queens puzzle

همانطور که در بالا گفته شده در این حالت از صفحه بازی هیچ دو وزیری وجود ندارند که یکدیگر را تهدید کنند بنابرین به یک حالت ایده‌آل دست یافته‌ایم.

حال کمی راجع به الگوریتم مورد استفاده در حل این مساله صحبت میکنیم:

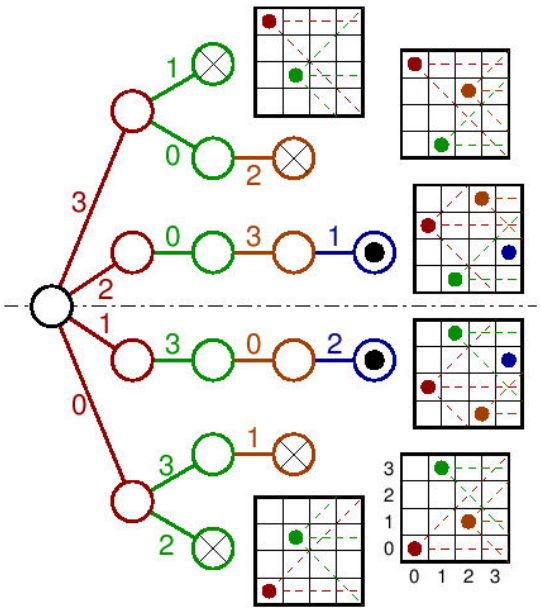
این مساله با دو روش مختلف backtracking و backtracking + MRV پیادهسازی شده است و در ادامه گزارش مقایسه این دو روش را خواهیم دید.

ابتدا لازم است تا مقداری راجع به این دو الگوریتم توضیح دهیم.

الگوریتم backtracking:

این الگوریتم به این شکل عمل میکند که به ترتیب ستونها را پیمایش کرده و به ازای هر ستون حالتهای قابل قبول را انتخاب میکند. پس از انتخاب حالت قابل قبول به سراغ ستونهای بعدی میرود. در صورتی که در مسیر به مشکل برخورد و به ازای ستونی خانه قابل قبولی موجود نبود مسیر پیمایش شده را یک پله عقب میآید و خانه قابل قبول دیگری را انتخاب میکند. دلیل نامگذاری این روش به این اسم هم این مساله میباشد.

در شکل زیر میتوانید یک درخت حل مساله n-queen را مشاهده نمایید.



در اینجا میتوانیم یک درخت backtracking را برای مساله n queen در یک ماتریس ۴ در ۴ مشاهده کنیم. اما این الگوریتم قابلیت بهبودی نیز دارد.

یکی از روشهایی که باعث عملکرد بهتر این الگوریتم میشود اضافه کردن MRV به آن است.

در ادامه به توضیحاتی درباره ترکیب این دو الگوریتم میپردازیم.

الگوریتم backtracking + MRV:

این الگوریتم درواقع بهبود یافته‌ای از الگوریتم قبلی می‌باشد.

در MRV دو نکته به الگوریتم حل این مساله افزوده می‌شود. نکته اول این است که پیمایش ستون‌های جدول به ترتیب نمی‌باشد. به این معنا که بعد از انتخاب خانه مناسب برای یک ستون انتخاب بعدی الزاماً ستون مجاور آن نیست. بلکه ستونی انتخاب می‌شود که تعداد خانه‌های مجاز کمتری نسبت به باقی ستون‌ها داشته باشد. نکته دیگری که در این الگوریتم حایز اهمیت است آن است که اگر ستونی یافت شد که هیچ حالت قابل قبولی نداشته باشد الگوریتم شکست می‌خورد و به عقب برمیگردد تا خود را اصلاح نماید. این مساله باعث می‌شود که الگوریتم به پیمایش خود آن هم در حالتی که صددرصد شکست می‌خورد ادامه ندهد و همین نکته باعث خواهد شد که زمان نسبتاً زیادی ذخیره شود و این الگوریتم عملکرد بسیار بهتری نسبت به الگوریتم قبلی داشته باشد.

حال که کمی با عملکرد این دو الگوریتم آشنا شدیم به سراغ کد و توضیحات توابع به کار رفته در این کد خواهیم رفت و در هر بخش از نقاط ضعف و قوت هر کدام از قسمت‌های این دو الگوریتم صحبت خواهیم کرد و نتایج استفاده از آن‌ها را با یکدیگر مقایسه می‌کنیم.

تابع‌های استفاده شده:

تابع show_gameboard:

یک تابع ساده میباشد که ماتریس جواب را گرفته و آن را چاپ میکند. این تابع در هر دو الگوریتم مشترک است.

توابع number_of_collisions و number_of_collisions_full:

تابع اول برای الگوریتم عقبگرد ساده و تابع دوم برای الگوریتم عقبگرد به همراه MRV می‌باشد.

عملکرد این دو تابع به این شکل می‌باشد که به عنوان ورودی زمین بازی و مختصات یک نقطه خاص از زمین را گرفته و تعداد مهرهایی که آن نقطه از زمین را تهدید می‌کنند برمی‌گردانند.

علت تفاوت این دو تابع نداشتن ترتیب در انتخاب ستون در الگوریتم دوم می‌باشد. به این صورت که در تابع اول تنها نیاز است تهدیدها با سمت نقطه انتخابی بررسی شود در صورتی که در تابع دوم این بررسی باید در کل زمین انجام شود چون در حالت اول ما مطمئن هستیم که در سمت راست مختصات ورودی مهره‌ای نیست ولی در تابع دوم این قضیه صدق نمی‌کند.

بنابراین می‌توان گفت در اینجا الگوریتم اول کمی سریعتر عمل خواهد کرد.

تابع numberofFreePosition:

این تابع به این شکل عمل می‌کند که زمین و یک ستون را به عنوان ورودی دریافت کرده و تعداد نقاط قابل انتخاب را برمی‌گرداند. این تابع در انتخاب شماره ستون در الگوریتم دوم کاربرد دارد.

تابع colMinFree:

این تابع نیز مربوط به الگوریتم دوم می‌باشد.

این تابع به این شکل عمل می‌کند که زمین بازی و ستون‌های انتخاب نشده را دریافت کرده و به کمک تابع numberOfFreePosition ستونی را که در نوبت بعدی باید انتخاب شود را به الگوریتم برمی‌گرداند.

توابع backtrackSolver و backtrackMRVSolver:

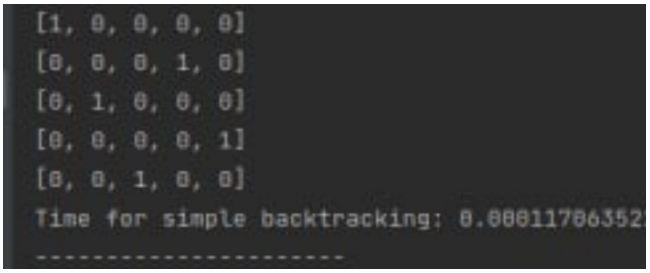
عملکرد این دو تابع در قسمت‌های قبلی گزارش بیان شده است. اما برای یادآوری و به طور خلاصه این دو تابع درواقع توابع هسته‌ای و عملیاتی الگوریتم می‌باشند و به این شکل می‌باشند که ستون‌ها را پیمایش نموده و در هر کدام وزیر را در جای مناسب قرار می‌دهند و در صورت بروز مشکل به عقب برمی‌گردند. تفاوت این دو الگوریتم در ترتیب انتخاب ستون‌ها و زمان بازگشتن به عقب می‌باشد.

با توجه به توضیحاتی که در مورد الگوریتم ها داده شد انتظار می‌رود الگوریتم شماره دو عملکرد بهتری نسبت به الگوریتم اول داشته باشد. برای دیدن نتیجه دو الگوریتم را در زمین‌هایی با سایز های مختلف آزمایش میکنیم.

زمینی به سایز ۵:

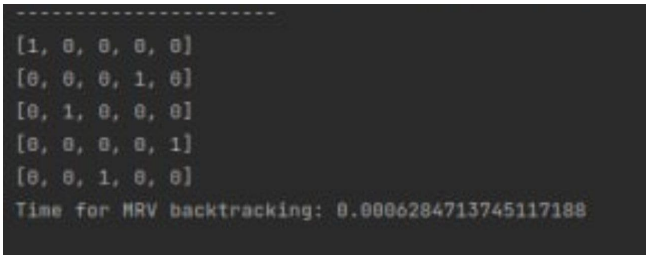
حالت عقب گرد ساده

0,0001



عقب گرد + MRV

0,0008

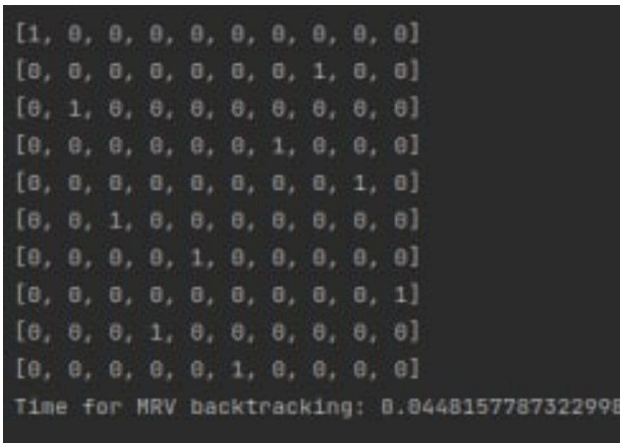


در این حالت الگوریتم اول عملکرد بهتری از خود نشان میدهد.

زمین به سایز 10:

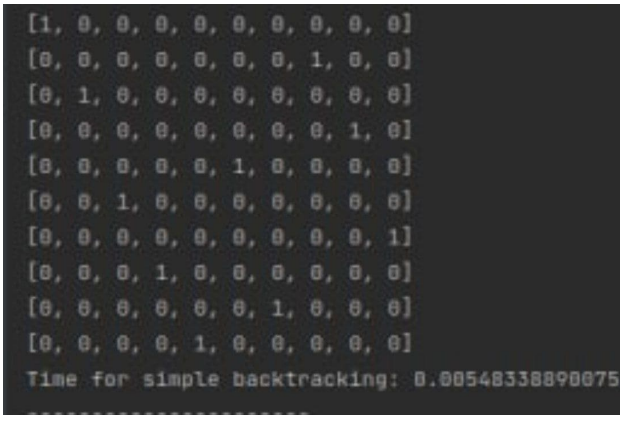
عقب گرد ساده

0,0054



عقب گرد + MRV

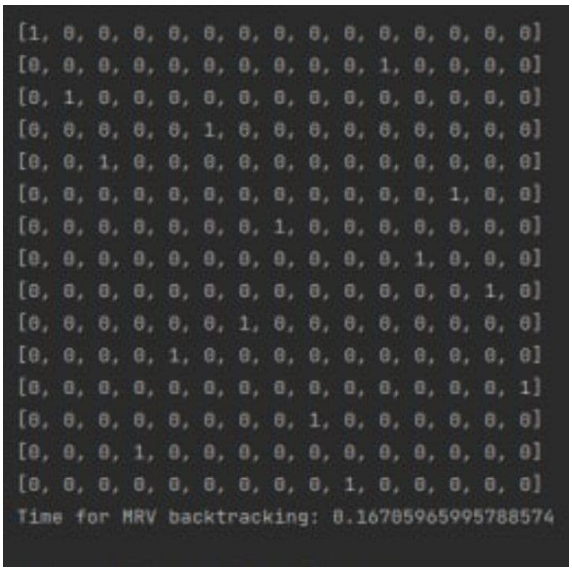
0,0448



زمین به سایز 15:

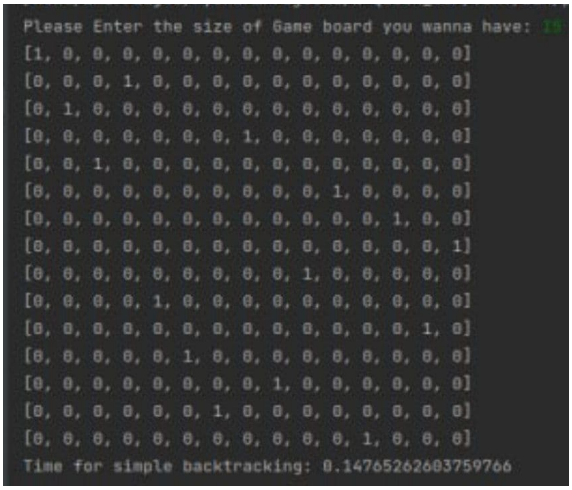
عقب گرد ساده

0,14



عقب گرد + MRV

0,16



زمین به سایز 18:

عقب گرد ساده

5,37

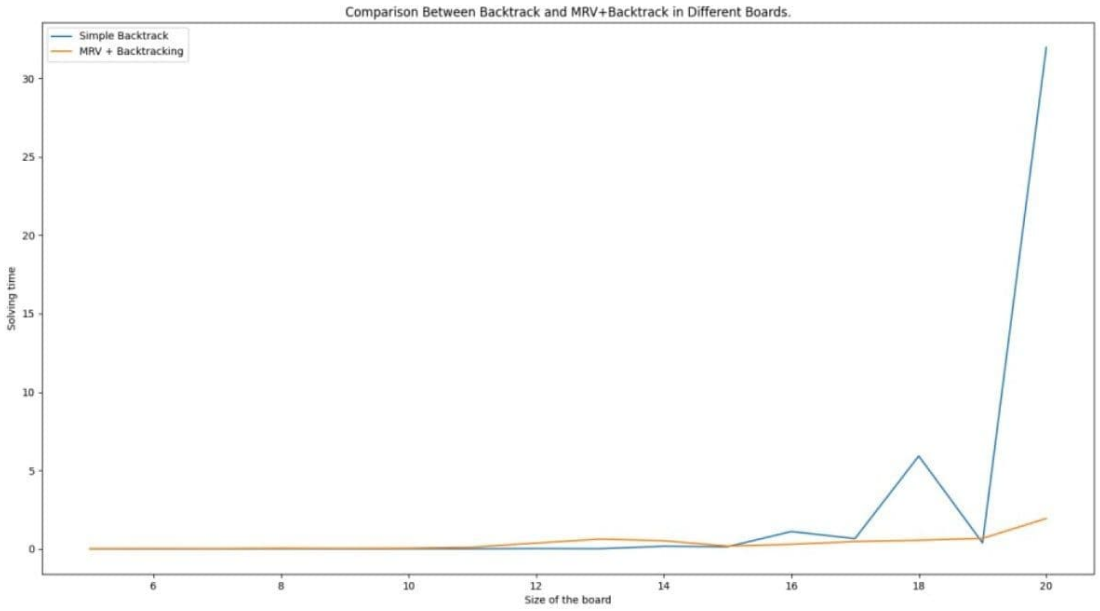
```
Please Enter the size of Game board you wanna have: 18
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
Time for simple backtracking: 5.372766494750977
----------------------
```

عقب گرد + MRV

0,54

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
Time for MRV backtracking: 0.545020580291748
```

نمودار مقایسه ای:



با توجه به نتایج به دست آمده مشاهده می‌کنیم الگوریتم شماره یک در زمین‌های کوچکتر عملکرد بهتری دارد که به دلیل عملکرد بهتر تابع number_of_collision و همچنین محاسبه نکردن برخی توابعی که در الگوریتم دوم وجود دارد می‌باشد. ولی هر چه سایز زمین افزایش می‌یابد بهینگی الگوریتم دوم نسبت به الگوریتم اول بیشتر می‌شود و انجام محاسبات اضافه تر در الگوریتم دوم بر عقب‌گردهای متوالی در الگوریتم اول حالت بهینه‌تری می‌سازد.

یک نکته جالب دیگری که در تست‌ها و نمودارها به چشم می‌آمد این بود که الگوریتم شماره یک در زمین‌هایی با سایز فرد عملکرد بهتری نسبت به همسایه‌های خود با شماره زوج دارد. این مساله را می‌توانید در نمودار بالا در خانه‌هایی با سایز ۱۵ و ۱۷ و ۱۹ به وضوح مشاهده نمایید.

این تست برای زمین‌هایی با اندازه ۲۱ و ۲۳ نیز انجام شد و دیده شد که مقدار زمانی که این الگوریتم برای حل کردن مساله نیاز دارد به طور قابل توجهی از همسایه‌های کنار خود کمتر است.

به طور مثال این الگوریتم زمین با سایز ۲۱ را در ۱,۵ ثانیه حل کرد در صورتی که در زمینی با سایز ۲۰ این عدد بیشتر از ۳۰ ثانیه بوده است.

کد مربوط به این گزارش را میتوانید در لینک زیر مشاهده نمایید:

https://github.com/amirradjou/N-Queen_With_CSP

References:

C how to program deitel

جزوه دکتر فاطمه موسوی

کتاب هوش مصنوعی راسل