

گزارش کار اول

برای مشاهده کد و خروجی اش به [code_report.html](#) مراجعه کنید.

اول به سه جدول مقایسه عملکرد الگوریتم‌ها نگاهی می‌کنیم.

زمان برای رسیدن به جواب			
	IDFS	BFS	A*
in1.csv	-	-	0.128
ln2.csv	-	-	0.066
ln3.csv	-	-	0.169
test_a.csv	0.254	7.168	0.009
test_b.csv	16.332	78.499	0.125
test_c.csv	165.012	-	0.037
Average	60.533	42.833	0.089

تعداد کل حرکات انجام شده

	IDFS	BFS	A*
in1.csv	-	-	63
ln2.csv	-	-	28
ln3.csv	-	-	77
test_a.csv	69045	19118	4
test_b.csv	4608473	240512	55
test_c.csv	49972179	-	14

تعداد حرکات مسیر جواب

	IDFS	BFS	A*
in1.csv	-	-	13
ln2.csv	-	-	11
ln3.csv	-	-	15
test_a.csv	3	3	4
test_b.csv	4	4	11
test_c.csv	5	-	5

بررسی و توضیح کلی مساله:

در این مساله هر استیتمان یکی از حالات هشت وزیر در شطرنج است و هدفمان پیدا کردن حالتی است که تعداد تهدیدها صفر باشد در توابع بخش [12] IN توابعی وجود دارند که با گرفتن یک جدول دو در هشت که موقعیت کنونی وزیرهاست. تعداد تهدیدها را می‌توانند خروجی دهند.

همچنین در بخش [3] IN تابع move قرار دارد که با گرفت جدول و شماره وزیر در جدول می‌تواند وزیر را در یکی از هشت جهت جابه‌جا کند به این شکل که جهت صفر جابه‌جایی به بالات است و بقیه جهت‌ها به شکل ساعت گرد ادامه پیدا می‌کنند. تابع pure_move برای وقتی است که از مجاز بودن جابه‌جاییمان اطلاع داریم (مثلا وقتی می‌خواهیم جابه‌جاییمان به حالت قبل برگردد) که دیگر در این تابع چک نمی‌شود جابه‌جایی مجاز است یا خیر که باعث افزایش سرعت می‌شود.

بررسی و توضیح هر یک از الگوریتم‌ها:

IDFS

این الگوریتم در بخش [6] IN پیاده‌سازی شده است. پیاده‌سازی آن به شکل بازگشتی است. از مزایا و خوبیهای آن مصرف کم رم برای انجام الگوریتم نسبت به دو الگوریتم دیگر است. و چون حافظه‌ای برای خانه‌های طی شده ندارد. حالات تکراری زیادی طی می‌کند ولی با این حال به علت استفاده نکردن از حافظه برای ذخیره‌سازی استیت‌ها می‌تواند هر استیت را در زمان کمتری بررسی کند و همانطور که در جداول مشخص است با این که تعداد استیت بیشتری بررسی کرده است ولی در زمان بسیار زودتری نسبت به BFS به جواب رسیده است. هرچند که هر چه عمق فاصله بیشتر شود این اختلاف کمتر می‌شود. چون درخت حالات عمیقتر شده و حالات تکراری به شکل نمایی زیاد می‌شوند.

BFS

این الگوریتم در بخش [9] IN پیاده‌سازی شده است. پیاده‌سازی آن یک حلقه بینهایت برای پیدا کردن جواب است. از یک صف برای ذخیره‌سازی استیت‌هایی که باید بررسی

شوند استفاده شده است و از یک دیکشنری برای استیت‌های ویزیت شده. علت استفاده از دیکشنری بازدهی بالاتر آن برای چک کردن وجود داشتن یک استیت بود. چون دیکشنری از لیست به عنوان `key` پشتیبانی نمی‌کند از یک تابع `my_hash` استفاده شده است که در ازای هر حالت جدول یک استرینگ ۸ کاراکتری برمی‌گرداند. چیزی که در این الگوریتم قطعی است این است که هر استیت فقط یک بار بررسی می‌شود ولی مشکل بارز آن استفاده از حافظه زیاد و به همین علت پایین آمدن سرعت برنامه به علت `save` و `load` است. همچنین در این الگوریتم نمی‌توان فاصله جواب را به سادگی فهمید و برای همین یک صف `move_count` برای این منظور در نظر گرفته شده است.

A*

این الگوریتم در بخش [13] IN پیاده‌سازی شده است. پیاده‌سازی آن دقیقاً همانند BFS است. با این تفاوت که به جای آن که استیت بعدی‌مان اولین عضو صف باشد. عضوی است که کمترین تعداد تهدید را داراست. یعنی تابع هیوریستیک آن تعداد تهدیدهاست. مزیت این تابع آن است که محاسبه آن راحت است و با این که با کمترین تعداد حرکت به جواب نمی‌رسد ولی در سریعترین زمان ممکن می‌تواند ما را به جواب برساند. سرعت این الگوریتم همانطور که در نتایج پیدااست به مقدار فاصله ما تا جواب نهایی بستگی ندارد و این نشان می‌دهد که در تعداد بالا تنها الگوریتمی که به خوبی کار می‌کند همین است.

این الگوریتم حافظه بیشتری به نسبت `dfs` می‌گیرد ولی چون سریع به جواب میرسد مقدار حافظه گرفته شده آن در مجموع زیاد نمی‌شود. از تفاوت فاحش آن در مجموع حرکات انجام شده نسبت به دو الگوریتم دیگر نیز نمی‌توان چشم‌پوشی کرد.