گزارش پروژه هوش مصنوعی

۱)

گزارش ۱: کد مربوط به action == 4 (که همون عمل نشون‌داده‌شده در شکل ۳ هستش) رو از فایل state.py بخونید و توضیح بدید که قدم به قدم چجوری حالت جدید ساخته شده.

```
    state[0:2, :] = np.rot90(state[0:2, :], -1)
```

در این خط ابتدا به وسیله تابع کتابخانه نامپای دو قسمت اول state که در واقع مربع های سطح بالایی میشوند را نود درجه ساعتگرد میچرخاند.

```
    start = np.copy(state[2, :])
```

در این خط دو مربع نصفه ی بالایی وجه سمت چپ را در یک متغیر نگه میدارد تا بتواند در آخر کار آنها را بر روی وجه پشتی قرار بدهد و از دست نرود.

```
    state[2, :] = state[4, :]
```

در این خط دو مربع نصفه ی بالایی وجه روبرو را روی دو مربع نصفه ی بالایی وجه چپ قرار میدهد.

```
    state[4, :] = state[6, :]
```

در این خط دو مربع نصفه ی بالایی وجه راست را روی دو مربع نصفه ی بالایی وجه روبرو قرار میدهد.

```
    state[6, :] = state[8, :]
```

در این خط دو مربع نصفه ی بالایی وجه پشت را روی دو مربع نصفه ی بالایی وجه راست قرار میدهد.

```
    state[8, :] = start
```

در این خط دو مربع نصفه ی بالایی وجه چپ را که قبلا در یک متغیر دیگر ذخیره کرده است روی دو مربع نصفه ی بالایی وجه پشتی قرار میدهد.

گزارش ۲: الگوریتم رو برای تست‌کیس‌های ۱ تا ۲ اجرا کنید و تعداد گره‌های explore و expand شده، عمق جواب و زمان جستجو (به ثانیه) رو توی یک جدول گزارش کنید.

```
G:\Ai\proje 1\Project-1-main - Copy>python main.py --testcase testcases/2.txt --method IDS-DFS
---------- START ----------
SOLVING...
0
1
2
3
4
5
6
7
explored =  7  expand =  14761872  depth =  14761844
actions: [9, 7, 9, 7, 4, 1, 6]
SOLVE FINISHED In 443.95225s.
--------- PRESS ENTER TO VISUALIZE --------
```

```
info: changed aspect ratio: 1.778 -> 1.778

G:\Ai\proje 1\Project-1-main - Copy>python main.py --testcase testcas
---------- START ----------
SOLVING...
0
1
2
3
4
5
explored =  5  expand =  57372  depth =  57349
actions: [11, 7, 9, 1, 6]
SOLVE FINISHED In 1.23327s.
--------- PRESS ENTER TO VISUALIZE --------
```

گزارش ۳: در مورد God's Number و مقدار اون برای مکعب روبیک دو در دو بخونید و مختصر در گزارش‌تون بنویسید. دقت کنید که در پیاده‌سازی ما تنها چرخش‌های ۹۰ درجه به عنوان اعمال در نظر گرفته شده، در نتیجه God's Number رو برای این مجموعه اعمال گزارش کنید.

به حداقل تعداد حرکات لازم برای حل یک مکعب روبیک که به بدترین شکل خراب شده است god number میگویند که برای مکعب های دو در دو ۱۴ میباشد.

گزارش ۴: الگوریتمی که پیاده‌سازی کردید رو برای تست‌کیس‌های شماره ۳ و ۴ اجرا کنید و حداکثر ۴-۵ دقیقه صبر کنید. آیا تو این مدت روش ID-DFS موفق بود تا جواب رو پیدا کنه؟ تعداد حالات موجود در گراف جستجوی این مسئله به طور حدودی چه رابطه‌ای با عمق جستجو داره؟ با توجه به مقدار God's Number، به نظرتون آیا الگوریتم ID-DFS توانایی حل مکعب روبیک با هر حالت اولیه دلخواهی رو داره؟

خیر. با هر بار اضافه شدن به عمق، گراف جستجو ۱۲ برابر میشود و به صورت نمایی زیاد میشود.

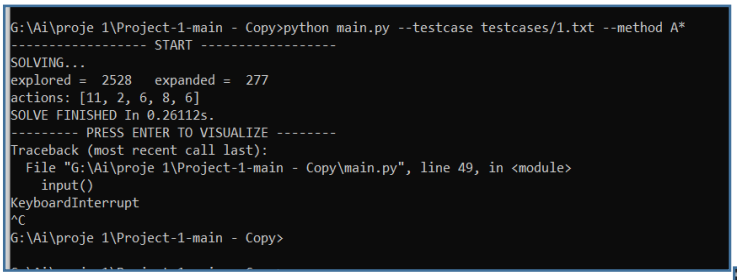
برای مکعب دو بعدی ممکن است با اینکه به زمان زیادی نیاز دارد ولی به طور کلی و در مدت کم امکان پذیر نیست.

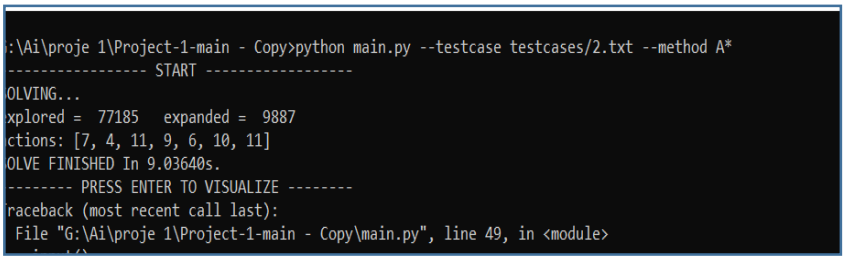
۲)

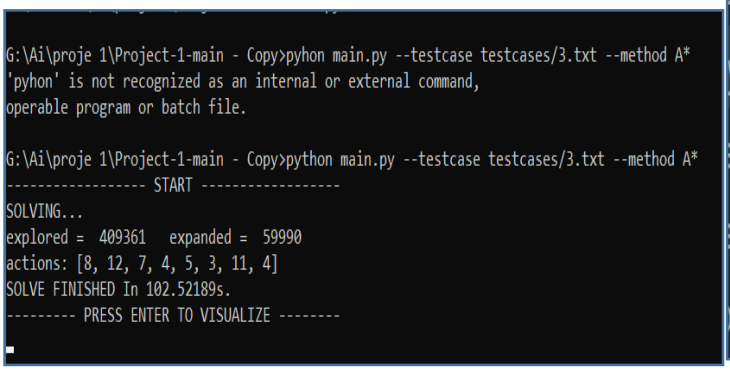
گزارش ۵: کد مربوط به action == 4 رو از فایل location.py بخونید و توضیح بدید که چجوری آرایه location جدید ساخته میشه. این عمل مطابق شکل ۳ و شکل ۴ (ب) است.

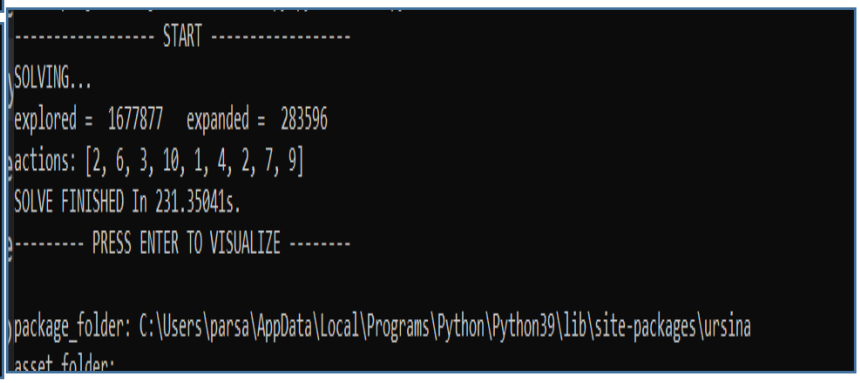
در آرایه سه بعدی فقط باید درایه های بخش بالایی جابجا شوند بنابراین در بعد دوم آرایه فقط ایندکس صفر را عوض میکنیم و آنرا نود درجه ساعتگرد میچرخانیم.

گزارش ۶: الگوریتم رو برای تست‌کیس‌های ۱ تا ۴ اجرا کنید و تعداد گره‌های **explore** و **expand** شده، عمق جواب و زمان جستجو رو توی یک جدول و کنار نتایج **ID-DFS** گزارش کنید.









گزارش ۷: به طور مختصر نتایجی که از **ID-DFS** گرفتید رو با ***A** مقایسه کنید. چقدر هیوریستکی که پیاده‌سازی کردید منجر به کاهش زمان جستجو و گره‌های **explored** شد؟

هم از نظر زمانی و هم از نظر تعداد گره های جستجو شده به مقدار بسیار زیاد و قابل توجه ای کم شده است.

۳)

گزارش ۸: الگوریتم رو برای تست‌کیس‌های ۱ تا ۷ اجرا کنید و تعداد گره‌های explore و expand شده، عمق جواب و زمان جستجو رو توی یک جدول و **کنار نتایج قبلی** بیارید.

Id dfs	1	2
explore	57372	14761872
expand	57349	14761872
time	1	444

A*	1	2	3	4
explore	2528	77185	409361	1677971
expand	277	9887	59990	283605
time	0.26	9	102	231

BiBFS	1	2	3	4	5	6	7
explore	90	501	1945	4805	25723	106851	183004
expand	991	4842	16588	39569	186432	679623	1135830
time	0.1	0.27	0.75	1.3	8.6	150	84

گزارش ۹: به طور مختصر نتایجی که از این ۳ تا روش گرفتید رو مقایسه کنید.

در حالت کلی در روش bi bfs کاهش بسیار چشم گیری مشاهده میشود و خیلی سریعتر به جواب میرسیم در حالی که در روش A* نسبت به ۲ن ضعیفتر است ولی از id dfs بهتر است.

گزارش ۱۰: اگر تا اینجای مسیر رو به درستی پیش اومده باشید، می‌بینید که روش با Bi-BFS می‌تونید خیلی سریع سخت‌ترین چینش‌های روبیک رو حل کنید. حالا می‌تونید کامند زیر رو اجرا کنید تا چینش‌های رندوم تولید بشه و با الگوریتم شما حل بشه:

```
python main.py --method BiBFS
```

۳-۴ بار این دستور رو اجرا کنید. آیا الگوریتم‌تون می‌تونه این روبیک‌های رندوم رو حل کنه؟

بله