گزارش کار پروژه ۱ مبانی و کاربرد های هوش کاربردی

گزارش ۱:

```
# up to left
elif action == 4:
    state[0:2, :] = np.rot90(state[0:2, :], -1)
    start = np.copy(state[2, :])
    state[2, :] = state[4, :]
    state[4, :] = state[6, :]
    state[6, :] = state[8, :]
    state[8, :] = start
```

ابتدا این کد می‌بایست وجه بالایی را به سمت چپ بچرخاند عملا خانه هایی که در دو درایه اول لیست ذخیره وجه های مکعب روبیک هستند را ۹۰ درجه بچرخانیم. برای این کار کد به این صورت عمل میکند که در مرحله اول وجه بالایی که شامل دو درایه اول است را با استفاده از np.rot90 به صورت ساعتگرد ۹۰ درجه می چرخانیم. سپس در مرحله دیگر می‌بایست بلوک هایی را درست کنیم که یک خانه آن فقط چرخیده است برای اینکار کافیست به صورت ساعتگرد ۲-> ۸ -> ۶ -> ۴ -> ۲ چون ۲ هم باید ریخت و هم باید برداشت برای همین ابتدا یک کپی ازش میگیریم که بعدا داخل ۸ بریزیم سپس مراحل ریختن درایه ها را آغاز میکنیم.

گزارش ۲:

Test case	Time	Explore	Expand	Answer depth
1	3.90651 S	57349	57378	5
2	240.14889s S	14761844	14761880	7

گزارش ۳:

عدد God's Number برابر است با حداکثر تعداد حرکات برای کامل کردن هر مکعب n * n است که این عدد برای مکعب دو در دو برابر است با ۱۱ (اما اگر عین مسئله ما هر نیم دور را یک حرکت در نظر بگیریم برابر ۱۴ میشود) این عدد برای مکعب با وجه ۳ هم برسی شده که ۱۵ سال زمان صرف پیدا کردن عدد God Number مکعب سه در سه شده است که برابر با ۲۰ است.

گزارش کار پروژه ۱ مبانی و کاربرد های هوش کاربردی

گزارش ۴:

خیر، الگوریتم IDS-DFS توانایی حل تست کیس های ۳و ۴ را در ۵-۶ دقیقه نداشت. دلیل این هم این است که رابطه عمق جستجو با تعداد گره های بررسی شده رابطه ای نمایی دارد و این رابطه نمایی به این صورت است که :

$$T(depth) \sim 12^{depth}S$$

و در ۵ دقیقه که برابر است با ۳۰۰ ثانیه حتی توانایی کامل کردن عمق ۷ام را نیز ندارد و تست کیس های ۳و ۴ بعد از بررسی عمق حل بیش از ۷ دارند.

با توجه به God's Number مکعب ۲ * ۲ که برابر است با ۱۴ با IDS-DFS و اردر زمانی که داریم 1283918464548864 ثانیه نزدیک به ۴۱ میلیون سال زمان میبرد تا این الگوریتم بتواند مکعب روبیکی که نیازمند عمق حل ۱۴ است را حل کند.

گزارش ۵:

```
    # up to left
    elif action == 4:
        location[:, 0, :] = np.rot90(location[:, 0, :], 1)
```

این قطعه کد میبایست که وجه بالایی رو به سمت چپ ۹۰ درجه انتقال دهد این کار را numpy خودش با استفاده از دستور np.rot90 انجام میدهد و اینگونه است که کافیست وجه سه بعدی را مشخص کنید و با ۱ به معنای جهت ساعتگرد آن وجه را میچرخاند. نحوه مشخص کردن وجه هم به اینگونه است که کافیست دومین عنصر لوکیشن را ۰ بگذاریم به معنای این است که اولین وجه جهت y را پاس بده به این تابع.

گزارش ۶:

A* :

Test case	Time	Explore	Expand	Answer depth
1	5.59195 S	12949	121406	9
2	17.97808 S	41058	378600	11
3	39.24870 S	90454	815415	12
4	32.36929 S	73703	672044	11

برای ID-DFS :

Test case	Time	Explore	Expand	Answer depth
1	3.90651 S	57349	57378	5
2	240.14889s S	14761844	14761880	7
3	NA	NA	NA	NA
4	NA	NA	NA	NA

گزارش ۷:

همانطور که مشاهده میکنید با اینکه در تست کیس های یک و دو عمق جواب به دست آمده نسبت به الگوریتم ID-DFS بیشتر است و حتی زمان تست یکم نیز بیشتر است اما این الگوریتم برای تست کیس هایی که نیازمند عمق زیادی برای جست و جو هستند بسیار سریع تر نسبت به الگوریتم ID-DFS عمل میکند.

همچنین طبق جدول میتوان دید که تفاوت زیاد و قابل توجه ای بین گره های اکسپلور و اکسپند الگوریتم A* است که در الگوریتم ID-DFS همچین چیزی رو مشاهده میکنیم و میبینیم که این اختلاف به مراتب کمتر است. همین دلیل باعث میشود که متوجه کارایی هیروستیکی که پیاده سازی کرده ایم است. این هیوریستیک عملا باعث میشود که تنها گره هایی را مورد بررسی قرار دهیم که احتمال رسیدن به جواب در آنها بیشتر است و به صورت کورکورانه تمامی گره ها را بررسی نمیکنیم.

این کاهش زمان عملا الگوریتم نمایی که در ID-DFS داشتیم که به ازای اینکه هر چقدر عمق جواب افزایش پیدا میکرد الگوریتم ما نیز به صورت نمایی اردر زمانیش زیاد میشد به یک الگوریتمی با اردری بسیار کمتر (در حال حاضر قابل اندازه گیری نیست و سوال هم همچین درخواستی نکرده) تبدیل کرده است.

گزارش کار پروژه ۱ مبانی و کاربرد های هوش کاربردی

گزارش ۸:

Bi-BFS

Test case	Time	Explore	Expand	Answer depth
1	0.02105 S	90	1077	5
2	0.12318 S	525	6296	7
3	0.55571 S	2341	28086	8
4	1.41567 S	6245	74935	9
5	10.03124 S	45967	551606	11
6	60.34277 S	299319	3591821	12
7	102.83214 S	546400	6556796	13

جدول مربوطه به دو الگوریتم دیگر در گزارش ۶ آمده و برای تست کیس ها ۵ تا ۷ این دو الگوریتم جوابی در زمان مناسب خروجی ندادند.

گزارش ۹:

دو الگوریتم *BI-BFS* , *ID-DFS* هر دو مبتنی بر یافتن کمینه عمق کار میکنند اما نحوه عملکرد این دو الگوریتم با همدیگر بسیار متفاوت است که همین باعث ایجاد اختلاف زمان حل بسیاری بین این دو الگوریتم میشود اما فرق این دو الگوریتم در چیست؟ فرق این دو الگوریتم در این است که در روش *ID-DFS* ما برای یافتن جواب دونه به دونه عمق را افزایش میدادیم و سپس چک میکردیم ایا جواب در آن عمق موجود هست یا نه الگوریتم *BI-BFS* نیز عملا همینکار را میکند اما با این تفاوت که این الگوریتم زیر مسئله را به دو بخش تقسیم میکند استدلال این است که اگر عمق جواب مسئله n باشد کافیست تا عمق n/2 را از استیت حل شده و تا عمق n/2 از استیت شروع را برسی کنیم تا به جواب مطلوب برسیم. روش A* که بر اساس هیروستیک عمل میکند روش عملکری متفاوتی دارد و مبنا بر اساس عمق در نظر نمیگیرد و بر اساس هیروستیک است این روش طبق داده های به دست آمده از روش *ID-DFS* بهتر عمل میکند اما همچنان در تست کیس هایی که عمق حل آنها بالا باشد از نظر زمانی بسیار کند است و روش *Bi-BFS* از آن سریع تر است. روش *Bi-BFS* نیز اگر عمق حل مسئله بالاتر برود نیز به مشکل زمان حل مسئله میخورد اما طبق گزارش ۳ میدانیم که گاد نامبر برای مکعب روبیک ۲ * ۲ برابر است با ۱۴ پس این الگوریتم زمان اجرای مناسب برای حل کل حالت های مکعب روبیک دو در دو را دارد.

گزارش ۱۰:

گزارش کار پروژه ۱ مبانی و کاربرد های هوش کاربردی

بله این الگوریتم توانست که مکعب های روبیک رندوم را طی۴ آزمایش کامل، در آزمایش هایی که عمق جست و جو ۱۲ و ۱۳ بود الگوریتم بین ۶۰ تا ۱۲۰ ثانیه برای حل زمان احتیاج داشت و در مواردی که ۹و ۸ بود عمق جواب این الگوریتم در کمتر از ۳ ثانیه جواب داد.