

### گزارش 1

مطابق کد شکل زیر می خواهیم سطح بالایی مکعب را 90 درجه ساعتگرد به سمت چپ بچرخانیم، ابتدا از حالات سمت چپ مکعب یک کپی نگه می داریم (چرا که می خواهیم آنها را بعدا جایگزین کنیم) سپس با توجه به اینکه ساعتگرد به سمت چپ می چرخانیم هر کدام از خانه ها یک واحد به سمت چپ شیفت داده می شود و خانه های سمت چپ با خانه سمت راست قلی جایگزین می شود و در انتها که به اولین خانه از سمت راست رسیدیم، آن را با خانه های اولی که کپی داشتیم (خانه های بالایی) جایگزین می کنیم:

```

57
58     # up to left
59
60     elif action == 4:
61         state[0:2, :] = np.rot90(state[0:2, :], -1)
62         start = np.copy(state[2, :])
63         state[2, :] = state[4, :]
64         state[4, :] = state[6, :]
65         state[6, :] = state[8, :]
66         state[8, :] = start

```

### گزارش 2

\* Iterative Deepening DFS :

test Case	searchTime	explored	expand	depth
①	1.217	5	57372	57349
②	323.59	7	14761872	14761844

### گزارش 3

این عدد برای مکعب روییک، به صورت تئوری نشان دهنده ای کمترین تعداد چابه جایی برای حل آن است که عدد آن برای مکعب روییک  $2^2 * 2$  برابر 11 حرکت است به این معنی که هر مکعب روییک  $2^2 * 2$  می تواند در 11 حرکت یا کمتر حل شود. همانطور که مشاهده می شود برای تست کیس های 1 و 2 این الگوریتم را آزمایش کردیم و همه دفعات جابجایی کمتر از 11 شد.

### گزارش 4

الگوریتم برای تست کیس های 3 و 4 اجرا شد و مشاهده شد بیشتر از عمق 8 توانایی جست و جو نداشته و فرآیند بسیار طولانی می شود و بعد از 5 دقیقه تازه به عمق 7 رسید که نتیجه می شود این تست کاملا بر عکس تست های 1 و 2 بوده است چرا که در تست های اولی به دلیل پیچیدگی کمتر در عمق بالاتر به جواب رسیدیم. همچنین هر چه حالت نهایی موجود در گراف پیچیده تر و action های بیشتری قبل از حالت نهایی که به تابع میدهیم داشته باشد، عمق جست و جو نیز بیشتر خواهد شد و به پیچیدگی زمانی جست و جو اضافه خواهد کرد و بر اساس god's number نتیجه میگیریم اگر تعداد حالات بیشتری داشته باشیم این الگوریتم چالش زمانی خواهد داشت و لزوما نمی تواند در کمتر از 11 حرکت مکعب را حل کند.

```

[3 1]
[4 3]]
0
1
2
3
4
5
6
7
8

```

## گزارش 5

در این عمل می خواهیم وجه بالایی مکعب را به صورت ساعتگرد 90 درجه بچرخانیم، طبق صورت heuristic، کل مکعب رو بیک از 8 مکعب کوچک تشکیل شده است که 4 تای رو بروی را شماره های 1 تا 4 و چهار تای پشتی را شماره های 5 تا 8 می دهیم. حال اگر مکعب را بچرخانیم چون فقط وجه بالا می چرخد، مکعب های شماره 1 و 2 و 5 و 6 جایجا می شوند و آنها را میکنیم و 90 درجه مطابق کد زیر index rotate می کنیم.

```

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

```

## گزارش 6

### \* A\* Search :

test Case	searchTime	explored	expand	depth
①	0.37008	4038	385	5
②	22.27	234165	24323	7
③	201.24	1884673	205260	8

## گزارش 7

از نظر کارایی، مشاهده می شود الگوریتم A\* در حالت کلی پیچیدگی زمانی کمتری نسبت به IDS دارد زیرا این الگوریتم از یک heuristic جهت رسیدن به هدف استفاده می کند و منجر می شود که مسیر هایی را که نزدیک به هدف هستند را جست و جو کند و از مسیر های پرت دور می شود در مقابل IDS که به صورت ناگاهانه تمامی مسیر های منتهی به هدف را تا یک عمق از پیش تعیین شده جست و جو می کند. از نظر هزینه در A\* اگر ضمانت شود که heuristic قابل قبول و consistent است، بهینه ترین جواب را خواهیم یافت اما در IDS جوابی که پیدا کردیم لزوماً بهینه نیست. از تست کیس 1 مثلاً متوجه می شویم با توجه به هیوریستیکی که داریم حدود بیش از 50 درصد کاهش زمان رسیدن به جواب داشتیم نسبت به الگوریتم IDS.

## گزارش 8

Iterative Deepening Search	TestCase	SearchTime	Explored	Expand	Depth
①	1.27	5	57372	57349	
②	323.59	7	14761 872	14761 844	
A*	①	0.37008	4038	385	5
②	22.27	234165	24323		7
③	201.24	18846 73	2052 60		8
④	—	—	—	—	—
Bidirectional Breadth first Search	①	0.04204	90	991	90
②	0.130	501	4842	501	
③	0.4557	1945	16588	1945	
④	1.31119	4805	39569	4805	
⑤	16.5660	25723	186432	25723	
⑥	36.636	106851 23	6796 23	106851	
⑦	—	—	—	—	—

## گزارش 9

در هر سه الگوریتم بدیهی است هر چه تست کیس پیچیده تری یا تعداد حالات بیشتری داریم پیچیدگی زمانی نیز افزایش پیدا می کند. که این در هر سه الگوریتم صدق می کند مثلا در تست کیس های 3 به بعد برای  $A^*$  افزایش زمان قابل توجهی داشتیم. اما برای مقایسه این سه الگوریتم، الگوریتم BiBFS همانطور که میبینیم از همه سریع تر عمل می کند چرا که این الگوریتم از گره پایانی و گره ابتدایی همزمان bfs اجرا می کند و مثلا در نقطه تلاقی جواب بهینه را بر میگرداند که باعث می شود به لحاظ زمانی پیچیدگی کمتری داشته باشد اما با توجه به اینکه همزمان دو dfs داریم انجام می دهیم، نیاز است دو ساختمان داده مجزا برای هر یک داشته باشیم، این الگوریتم با اینکه ناآگاهانه است اما مشاهده می شود بهتر از  $A^*$  که آگاهانه است عمل می کند. اما الگوریتم IDS با اینکه پیچیدگی حافظه کمتری دارد اما نسبت به دو مورد دیگر، کند است. همچنین طراحی یک هیوریستیک admissible برای  $A^*$  می تواند چالش برانگیز باشد و هیوریستیک این الگوریتم تاثیر زیادی روی کارکرد بهینه آن دارد.

## گزارش 10

آزمایش بار اول:

```
⑧ 003@pu0@80@0000@@@... u explored = 48940 expanded = 346899 depth = 48940
    actions: [3, 6, 4, 6, 2, 3, 5, 5, 7, 5, 4]
    SOLVE FINISHED In 24.68389s.
----- PRESS ENTER TO VISUALIZE -----
```

آزمایش بار دوم:

```
⑧ 003@pu0@80@0000@@@... u explored = 32 expanded = 380 depth = 32
    actions: [1, 6, 8, 11, 7]
    SOLVE FINISHED In 0.01708s.
----- PRESS ENTER TO VISUALIZE -----
```

آزمایش بار سوم:

```
⑧ 003@pu0@80@0000@@@... u explored = 629 expanded = 6007 depth = 629
    actions: [3, 6, 3, 7, 5, 4, 7]
    SOLVE FINISHED In 0.16379s.
----- PRESS ENTER TO VISUALIZE -----
```

آزمایش بار چهارم:

```
⑧ 003@pu0@80@0000@@@... u explored = 7852 expanded = 62794 depth = 7852
    actions: [5, 2, 4, 2, 10, 11, 10, 12, 8]
    SOLVE FINISHED In 1.98888s.
----- PRESS ENTER TO VISUALIZE -----
```

با توجه به آزمایش ها متوجه می شویم الگوریتم BiBfs می تواند مکعب روییک با چینش های رندوم را حل کرده و تضمین دهد که جواب دارد بنابراین این الگوریتم کامل است.