تمرین کامپیوتری 1 هوش – جستجوی آگاهانه و ناآگاهانه:

در این پروژه ما میخواهیم با استفاده از الگوریتم های جستجوی آگاهانه و ناآگاهانه ، عمل جستجو را انجام دهیم و به مساله داده شده پاسخ دهیم. در بخش اول و دوم ، از الگوریتم های BFS,IDS (جستجوی ناآگاهانه) استفاده کرده و در بخش آخر نیز از الگوریتم A* (جستجوی آگاهانه)به همراه ضرایب مختلف استفاده می کنیم.

خواندن ورودی:

ما از یک لیست دو بعدی به ابعاد m*n بعنوان زمین اصلی استفاده کرده و محدودیت های داده شده (مثل نقطه شروع و پایان رهبر، مبدا و مقصد یاران و نقاط نظارت اورک ها) را روی آن اعمال می کنیم.

نقطه شروع و پایان گندالف در یک متغیر ذخیر شدند و در پیاده سازی الگوریتم های سرچ (به ترتیب برای نقطه شروع سرچ و بررسی شرط پایان یافتن جستجو)از آن استفاده خواهیم کرد.

نقاط حضور اورک ها نیز در جدول بصورت خانه های غیرقابل دسترس ذخیره شده و همچنین منطقه های تحت نظارت آنها ،هم در زمین اصلی و هم در یک دیکشنری (با کلید نام اورک و مقادیر خانه های تحت نظر آن اورک) ذخیره شده اند. علت استفاده از این دیکشنری، تغییر دادن و وجود محدودیت حضور گندالف در منطقه تحت نظارت اورک است که روی عمل جستجو توسط اورک تاثیر گذار است و علت مشخص کردن خانه های تحت نظارت درون زمین، مشخص شدن محدودیت آن خانه برای گندالف در هنگام جستجو و جلوگیری از سرچ کردن تکراری درون دیکشنری برای پیدا کردن آن خانه است.

نقاط مبدا و مقصد یاران نیز درون یک متغیر لیست مشخص شده اند. از این لیست در پیاده سازی goal ها درون توابع جستجو و کلاس State استفاده شده.

مدل سازی :

در هر سه الگوریتم جستجو، یک کلاس State تعریفکردیم که این کلاس نشان دهنده وضعیت و شرایط جستجو در هر مرحله است.

در این کلاس ، وضعیت جستجو در هر مرحله نشان داده می شود.

این کلاس دارای property هایی شامل اطلاعاتی مثل : طول مسیر طی شده ، خانه ای که گندالف (agent) در لحظه جاری، قدرت اورک ها در آن لحظه از جستجو، پدر مسیر طی شده(برای استفاده در چاپ مسیر طی شده) ، وضعیت دوستان ، همراه بودن/نبودن دوست با گندالف ، مقدار fValue برای جستجوی A* و یک ID است.

توجه: این ID با استفاده از داده هایی که نشان دهنده حالت های مشابه هستند Hash شده و از جستجوی حالت های تکراری جلوگیری می کند.

همچنین این کلاس دارای متدهای محاسبه مقدار F,G,H برای جستجوی A* و تابع هش برای محاسبه ID بوده و همچنین دارای دو متغیرstatic که نشان دهنده state های دیده شده و تعداد آن ها است ، می باشد.

استیت اولیه یا initialState ما برابر با کلاس State ای می باشد که طول مسیر طی شده آن 0 ، پدر آن None و خانه شروع آن نقطه شروع گندالف است، می باشد . همچنین مقادیر متغیر های قدرت اورک ها و خانه های دوستان، برابر با مقادیری است که ورودی گرفتیم.

بطور کلی در هر 3 الگوریتم، ما استیت اولیه را تعریف میکنیم و روی آن استیت ، همه move های ممکن را انجام میدهیم و چک میکنیم که آیا به استیت نهایی رسیدیم یا نه و در صورت رسیدن به استیت نهایی ، مسیر پیدا شده را چاپ می کنیم.

همچنین با استفاده از متغیر استاتیک حالت های ویزیت شده، از پیمایش خانه های تکراری جلوگیری می کنیم.

در کل برای هر استیت ما حداکثر 4 تا move داریم. (حرکت در 4 جهت)

در الگوریتم های جستجو، پس از تعریف استیت اولیه، این 4 حرکت در نظر میگیریم و با بررسی شرط های valid بودن آن حرکت (مثلا خانه درون جدول باشد یا اورک در آن نباشد یا بیشتر از قدرت اورک در آن محدوده از اورک حرکت نکرده باشیم)، گندالف را به آن خانه حرکت می دهیم و با توجه به خانه ای که گندالف به آن می رسد، یک استیت جدید تعریف می کنیم.

همچنین بدیهی است که با توجه به خانه ای که گندالف به آن می رسد، استیت متفاوتی تعریف می شود.مثلا ممکن است که نیاز به تغییر امتیاز های محدوده اورک ها باشد یا اورک به یک دوست برسد و آن را بردارد یا به خانه مقصد یک دوست برسد و حالت های دیگر.

استیت نهایی یا همان finalState برابر با استیتی است که در آن گندالف به خانه مقصد رسیده و همه دوستانش را به خانه هایشان رسانده و همچنین درطی مسیر نیز خطایی نکرده (مثلا در محدوده یک اورک، بیشتر از قدرت آن حرکت نکرده)

بطور کلی بعد از رسیدن به استیت نهایی و جواب،باید با استفاده از خواص کلاسState ، مسیر و طول مسیر پیمایش شده و تعداد استیت های ویزیت شده و جهت مسیر های پیمایش شده (چپ/راست/بالا/پایین) را خروجی دهیم.

توجه : بعد از پیدا کردن مسیر پیمایش شده با استفاده از تابع بازگشتی و parent هر آبجکت، از تابع نوشته شده PrintPath برای چاپ مسیر براساس حرکت های انجام شده L/U/R/D استفاده می کنیم.

الگوریتم جستجوی BFS :

این یک الگوریتم جستجوی ناآگاهانه و براساس جستجوی سطح اول است.

خروجی حاصل از جواب این الگوریتم ، مسیری بهینه و کامل (پیدا کردن جواب درصورت وجود داشتن) و از مرتبه زمانی و فضایی O(b^d) می باشد که در آن b و d به ترتیب حداکثر تعداد فرزندان و طول جواب بهینه است.

بطور خلاصه در این الگوریتم ما ابتدا استیت اولیه را به مجموعه مرزی اضافه میکنیم . سپس (از آنجایی که هزینه هر حرکت 1 است)تمامی حرکت های ممکن روی تک عنصر مجموعه مرزی را اعمال میکنیم و استیت های جدید را به مجموعه مرزی اضافه میکنیم.

این کار را تا انجایی ادامه میدهیم که یا تعداد عناصر مجموعه مرزی تمام شود یا به استیت نهایی برسیم.(هر کدام زودتر رخ داد) که درصورت رخ دادن حالت اول، جوابی وجود نخواهد داشت.

خروجی تست کیس ها برای این الگوریتم:

```
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_00.txt
BFS finished
pathIs:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRDURDURDURDURDRRRRRR
pathLenIs: 48
numberOfVisitedStates: 21259
execution Time: 198.38690757751465 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_01.txt
BFS finished
pathIs:DRRDRRDDDDDLLDLURUUUURRRRDLDLDDDLUURUURRRRDDDDLLLRRR
pathLenIs: 52
numberOfVisitedStates: 3999
execution Time: 30.92026710510254 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_02.txt
BFS finished
pathIs:RRRRRRRDDDDLLLLDLLLDDRUURURRRDRDDR
pathLenIs: 34
numberOfVisitedStates: 1113
execution Time: 8.976459503173828 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_03.txt
BFS finished
pathIs:DDRDRDDDDDLLDRURUUUUUUUURRRRRRRLLDDDDDLLDDDDLLLLURUUUURRRRRRRDDDDD
pathLenIs: 66
numberOfVisitedStates: 12252
execution Time: 101.08709335327148 ms
```

الگوریتم جستجوی IDS :

این یک الگوریتم جستجوی ناآگاهانه و براساس ترکیب جستجوی عمق اول و سطح اول است.به بیان دیگر ، این الگوریتم همان الگوریتم DFS است ولی دارای یک شرط اضافه به معنای حداکثر عمق قابل پیمایش است و در هر مرحله با افزایش این حداکثر عمق قابل پیمایش، یک عمق بیشتر برای جستجو حرکت میکنیم و این باعث رفع مشکلات الگوریتم جستجوی DFS (بهینه نبودن خروجی) می شود.

خروجی حاصل از جواب این الگوریتم ، مسیری بهینه و کامل است.

بطور خلاصه در این الگوریتم ما ابتدا استیت اولیه را به مجموعه مرزی اضافه میکنیم . سپس الگوریتم DFS را تا عمق k با مقدار اولیه 0 اجرا میکنیم و هر مرحله مقدار k را یک واحد اضافه میکنیم تا یا به جواب برسیم ، یا k به حد بالایی که برای طول حداکثر جواب مشخص کردیم برسد(حالتی که جواب ندارد.)

خروجی تست کیس ها برای این الگوریتم:

```
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_00.txt
IDS finished
pathIs:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRDURDURDURDURDRRRRRR
pathLenIs: 48
numberOfVisitedStates: 127422
execution Time: 1.7654848098754883 s
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_01.txt
IDS finished
pathIs:RRRRDDDDDDLLDLURUUUURRRRDLDLDDDLUURUURRRRDDDDLLLRRR
pathLenIs: 52
numberOfVisitedStates: 23898
execution Time: 0.28518199920654297 s
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_02.txt
IDS finished
pathIs:RRRRRRRDDDDLLLLDLLLRDDUURURRRRDRDD
pathLenIs: 34
numberOfVisitedStates: 6594
execution Time: 0.09452390670776367 s
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_03.txt
IDS finished
pathIs:RRDDDDDDDDLLRDURUUUUUUUURRRRRRRLLDDDDDLLDDDDLLLLURUUUURRRRRRRDDDDD
pathLenIs: 66
numberOfVisitedStates: 74034
execution Time: 0.9450554847717285 s
PS E:\University\Term6\AI\CAs\CA1>
```

الگوریتم *A:

همانطور که می دانیم این روش یک الگوریتم جستجوی آگاهانه است و در این الگوریتم ما باید یک تابع heuristic تعریف و از آن برای پیدا کردن مسیر استفاده کنیم.

در صورتی که این تابع consistent باشد ، خروجی الگوریتم ما کامل و بهینه است و این الگوریتم نسبت به دو الگوریتم دیگر سریعتر خواهد بود.

ما تابع h را بسته با حالت گندالف ، بصورت زیر تعریف می کنیم:
اگر گندالف دوستی با خود همراه نداشت، تابع ما برابر با ماکزیمم (فاصله منهتن گندالف و دوست + دوست و مقصد دوست + مقصد دوست و مقصد گندالف) این مقدار برای همه دوستان است.

اگر گندالف با خود دوستی همراه داشت ، این مقدار برابر با مجموع فاصله منهتن گندالف و مقصد دوست + مقصد دوست و مقصد گندالف است.

اگر گندالف همه دوستانش را رسانده بود، این مقدار برابر با منهتن گندالف و مقصد گندالف است.

خروجی این تابع consistent است . چون اولا شرط تعریف شده برای تابع consistent برای آن صادق است و همچنین خروجی جواب ما بهینه است.

خروجی برای alpha=1 :

```
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_00.txt
A* finished
pathIs:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRDRUDRUDRUDRUDRRRRRR
pathLenIs: 48
numberOfVisitedStates: 1317
execution Time: 28.93853187561035 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_01.txt
A* finished
pathIs:DRRDRRDDDDLLDLDRRRUUURRUULLDDDLDDRRRRRUUUUDDDLDLLRRR
pathLenIs: 52
numberOfVisitedStates: 2786
execution Time: 50.86350440979004 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_02.txt
A* finished
pathIs:RRRRRRRLLLDDDLDLDLLDDRRUURURRDRDDR
pathLenIs: 34
numberOfVisitedStates: 315
execution Time: 4.986763000488281 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_03.txt
A* finished
pathIs:DDRDRDDLDLDDRDRRRRUUUURRURURUUULLLDDDLDLLLDDDDLDRRRRUUUUURRRRDDDDD
pathLenIs: 66
numberOfVisitedStates: 5854
execution Time: 106.71019554138184 ms
```

خروجی برای alpha=1.8

```
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_00.txt
A* finished
pathIs:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRDRUDRUDRUDRUDRRRRRR
pathLenIs: 48
numberOfVisitedStates: 427
execution Time: 7.9784393310546875 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_01.txt
A* finished
pathIs:DRRDRRLLDLDDLDDDRRRRUUURRUULLDDDLDDRRRRRUUUUDDDLDLLRRR
pathLenIs: 54
numberOfVisitedStates: 2778
execution Time: 40.498971939086914 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_02.txt
A* finished
pathIs:RRRRRRRLLLDDDLDLDLLDDRRUURURRDRDDR
pathLenIs: 34
numberOfVisitedStates: 198
execution Time: 3.008604049682617 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_03.txt
A* finished
pathIs:DDRDRDDLDLDDRDRRRRRRRRUUUULUURUUULLLLLLLLDDDRDDDDDLDRRRRUUUUURRRRDDDDD
pathLenIs: 70
numberOfVisitedStates: 2344
execution Time: 31.45623207092285 ms
PS E:\University\Term6\AI\CAs\CA1>
```

خروجی برای alpha=3

```
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_00.txt
A* finished
pathIs:RRRRRRRRRRRRRRRRRRRRRRRRRRRRRDRUDRUDRUDRUDRRRRRR
pathLenIs: 48
numberOfVisitedStates: 235
execution Time: 4.057884216308594 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_01.txt
A* finished
pathIs:DRRDRRLLDLDDLDDDRRRRUUURRUULLDDDLDDRRRRRUUUULLLDLDDDRRRR
pathLenIs: 56
numberOfVisitedStates: 2653
execution Time: 36.18979454040527 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_02.txt
A* finished
pathIs:RRRRRRRLLLLLDDRDDLDLLDDRRUURURRDRDDR
pathLenIs: 36
numberOfVisitedStates: 163
execution Time: 1.959085464477539 ms
PS E:\University\Term6\AI\CAs\CA1> python codes.py test_03.txt
A* finished
pathIs:RRRRRRRLLLLLLDDDRDDDDDLDRUUUUUUDDLDLDDDRRRRRRRRRUUUULUURUUULLLDDDLDRRRRDDDDD
pathLenIs: 76
numberOfVisitedStates: 1851
execution Time: 22.969961166381836 ms
PS E:\University\Term6\AI\CAs\CA1>
```

همانطور که در جواب ها و برای ضرایب مختلف میبینیم ،برای ضرایب کم (بین 1 تا 2) این خروجی همچنان بهینه است ولی برای ضرایب بیشتر ، مسیر خروجی دیگر بهینه نخواهد بود.

همچنین هر چه ضریب بیشتر شود، تعداد استیت های دیده شده و زمان رسیدن به جواب ، کمتر می شود.

به بیان دیگر با افزایش ضریب، سرعت را فدای دقت می کنیم.

تست کیس 1			
	طول جواب	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	48	21259	198ms
IDS	48	127422	1.7s
A_e	48	1317	28ms
alpha=1.8	48	427	7.9ms
alpha=3	48	235	4ms

تست کیس 2			
	طول جواب	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	52	3999	30ms
IDS	52	23898	0.3s
A_e	52	2786	50ms
alpha=1.8	54	2778	40ms
alpha=3	56	2653	36ms

تست کیس 3			
	طول جواب	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	34	1113	9ms
IDS	34	6742	0.09s
A_e	34	315	4.9ms
alpha=1.8	34	198	3ms
alpha=3	36	163	1.9ms

تست کیس 4			
	طول جواب	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	66	12252	101ms
IDS	66	74034	0.9s
A_e	66	5854	106ms
alpha=1.8	70	2344	31ms
alpha=3	76	1851	22ms