به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

هوش مصنوعی

گزارش پروژه شماره ۱

نام و نام خانوادگی: سجاد علی‌زاده

شماره دانشجویی: ۸۱۰۱۹۷۵۴۷

تعاریف اولیه

تعریف استیت: هر حالت صفحه را یک استیت در نظر میگیریم. یعنی مکانی که بدن مار در آن قرار دارد و مکان دانه‌های باقی مانده در کلاس استیت جای میگیرند. همچنین اطلاعات اضافه‌تری مانند مسیر طی شده و یا سود را میتوان در کلاس استیت قرار داد.

تعریف سود: ارزش دانه‌هایی که توسط مار خورده شده به عنوان سود یا profit در نظر گرفته شده است.

تعریف هزینه: تعداد حرکاتی که مار انجام داده هزینه پرداخت شده است به عبارت دیگر طول مسیری که مار طی کرده برابر با هزینه است.

تعریف استیت نهایی: استیتی که در آن هیچ دانه ای نباشد. به عبارت دیگر سود ماکسیمم شود.

تعریف اکشن: به هر حرکت مار به یکی از چهار جهت یک اکشن میگوییم.

حرکت رو به بالا با (1,0) حرکت رو به پایین با (-1,0) حرکت به راست (0,1) و حرکت به چپ (0,-1) در نظر گرفته شده است. یعنی اگر به بالا به صفحه نگاه کنیم خانه (0,0) در بالا و چپ صفحه قرار دارد. یعنی صفحه به شکل زیر است:

	0	1	2	…
0				
1				
2				
3				
…				

مثلا اگر مار در خانه (1,1) قرار داشته باشد و یک واحد به سمت پایین برود به خانه (0,1) و اگر به سمت بالا برود به خانه (2,1) خواهد رسید.

BFS

با دستور زیر برنامه را اجرا میکنیم:

```
python3 BFS.py tests/test1.txt
```

که آرگومان اول آدرس فایل تست میباشد.

همانطور که مشخص است در این الگوریتم ابتدا استیت آغاز در یک صف اضافه می‌شود و با توجه به حرکت هایی که مار میتواند انجام دهد تعدادی استیت جدید به وجود می‌آید که در انتهای یک صف اضافه می‌شود.

خروجی تست اول به شکل زیر است:

```
→  codes python3 BFS.py tests/test1.txt
Length: 12
Path: ULUURRRURUUR
Number of states: 1805
Number of not reapeted states: 890
Time: 90.693951 ms
```

خروجی تست دوم به شکل زیر است:

```
→  codes python3 BFS.py tests/test2.txt
Length: 15
Path: DRULLDDDDLDLLLL
Number of states: 31333
Number of not reapeted states: 13811
Time: 1644.085646 ms
```

خروجی تست سوم به شکل زیر است:

```
→  codes python3 BFS.py tests/test3.txt
Length: 25
Path: DRUUURURRUURRRDRRULLLDDLL
Number of states: 45979
Number of not reapeted states: 19032
Time: 2370.269537 ms
```

IDS

با دستور زیر برنامه را اجرا میکنیم:

```
python3 IDS.py tests/test1.txt
```

که آرگومان اول آدرس فایل تست می‌باشد.

در این روش ابتدا DFS با عمق پیاده سازی شده است. در این روش یک آرگومان عمق نیز وجود دارد. این آرگومان بیانگر این است که جستجو تا چه عمقی ادامه پیدا کند. فلذا جستجو با عمق‌های مختلف (از یک تا بی نهایت) انجام میپذیرد و در صورت پیدا شدن جواب مقدار آن بازگردانده میشود در غیر این صورت در حلقه گیر می‌افتیم.

خروجی تست اول به شکل زیر است:

```
→  codes python3 IDS.py tests/test1.txt
Length: 12
Path: ULUURRRURUUR
Number of states: 25394
Number of not reapeted states: 12901
Time: 767.583609 ms
```

خروجی تست دوم به شکل زیر است:

```
→  codes python3 IDS.py tests/test2.txt
Length: 15
Path: DRULLDDDDLDLLLL
Number of states: 249523
Number of not reapeted states: 120999
Time: 8961.174965 ms
```

خروجی تست سوم به شکل زیر است:

```
→  codes python3 IDS.py tests/test3.txt
Length: 25
Path: DRUUURURRUURRRDRRULLLDDLL
Number of states: 805412
Number of not reapeted states: 342858
Time: 29989.042044 ms
```

A* 1

با دستور زیر برنامه را اجرا میکنیم:

```
python3 AStar1.py tests/test1.txt
```

که آرگومان اول آدرس فایل تست می‌باشد.

در این الگوریتم استیت بعدی که قرار است expand شود بر اساس مجموع هزینه ای که تا آن استیت کردیم و هزینه ای که تخمین میزنیم تا رسیدن به هدف بپردازیم (heuristic) انتخاب می‌شود به صورتی که استیتی که انتخاب میشود کمترین این مقدار را داشته باشد. برای حفظ سرعت لیست frontier را مرتب شده نگه میداریم و در هر مرحله مقدار عنصر اول آن که کمترین مقدار مجموع هزینه پرداخت شده و پیش بینی هزینه را دارد انتخاب می‌کنیم.

هزینه ای که پرداخته‌ایم در اصل برابر تعداد حرکت هایی است که انجام داده‌ایم. یعنی طول مسیر پیموده شده همان هزینه است

در این مسئله مقدار heuristic در هر استیت ارزش دانه های باقی مانده است.

این مقدار admissible است. زیرا مار برای خوردن هر دانه حداقل یک حرکت نیاز دارد و برای خوردن تمام دانه ها حداقل به اندازه ارزش دانه‌ها باید حرکت کند. (مثلا اگر ارزش دانه های باقیمانده ۶ است حداقل ۶ حرکت باید انجام دهیم تا تمام دانه ها خورده شود)

این مقدار consistent است. فرض کنید در استیت الف هستیم و با یک اکشن (که حرکت به یکی از چهار جهت است) به استیت ب میرویم. در این جابجایی یا به ارزش یک سود میکنیم یا اصلا سود نمیکنیم (دانه خورده میشود یا نمیشود) در هر دوی این حالات (سود یکی اضافه شود یا برابر با قبل بماند) سودی که ممکن است داشته باشیم از هزینه ای که پرداخته ایم (یک حرکت با که هزینه یک در پی دارد) کمتر و یا مساوی است. به عبارت دیگر اختلاف heuristic هر دو استیت حداکثر می‌تواند یک باشد که از مقدار هزینه پرداختی (که قطعا یک است) کمتر و یا مساوی است. پس consistent است.

خروجی تست اول به شکل زیر است:

```
→  codes python3 AStar1.py tests/test1.txt
Path Length: 12
Path: ULDDLDLLDDLL
Number of states: 1655
Number of not reapeted states: 969
Time: 106.833458 ms
```

خروجی تست دوم به شکل زیر است:

```
→  codes python3 AStar1.py tests/test2.txt
Path Length: 15
Path: RDLLULDDDDDLLLL
Number of states: 22974
Number of not reapeted states: 11112
Time: 1448.431015 ms
```

خروجی تست سوم به شکل زیر است:

```
→  codes python3 AStar1.py tests/test3.txt
Path Length: 25
Path: DRUUURURRUURRRDRRULLLDDLL
Number of states: 37651
Number of not reapeted states: 16668
Time: 2236.622810 ms
```

A * 2

با دستور زیر برنامه را اجرا میکنیم:

python3 AStar2.py tests/test1.txt

که آرگومان اول آدرس فایل تست می‌باشد

تنها تفاوت این حالت با حالت قبل در تابع heuristic آن است.

تابع heuristic به شرح زیر است:

فاصله سر مار از تمام دانه ها را محاسبه میکنیم. (دقت کنید میتوانیم از دیوار عبور کنیم) بیشترین فاصله از میان فاصله های ایجاد شده تابع heuristic است. (توجه کنید فاصله منهتنی محاسبه میشود) به عبارت دیگر حداقل تعداد حرکاتی که مار باید انجام دهد تا به دورترین دانه موجود در صفحه برسد.

این تابع admissible است زیرا مار برای خوردن تمام دانه ها حداقل باید دورترین دانه را خورده باشد و این همان heuristic در نظر گرفته شده است.

این تابع consistent است. وقتی از یک استیت به استیت دیگر می‌رویم هزینه ای که پرداخت می-کنیم برابر یک است و اختلاف heuristic این دو استیت حداکثر میتواند یک باشد. زیرا در بدترین حالت ممکن است در استیت دوم یک گام به دورترین دانه نزدیک شویم که این مقدار کوچکتر یا مساوی هزینه پرداخت شده است. به عبارت دیگر با انجام یک حرکت فاصله از دورترین دانه در همه حالات حداکثر یک واحد افزایش می‌یابد که کمتر از هزینه پرداخت شده است

خروجی تست اول به شکل زیر است:

```
→  codes python3 AStar2.py tests/test1.txt
Path Length: 12
Path: ULUURRRRUUUR
Number of states: 1114
Number of not reapeted states: 617
Time: 78.200340 ms
```

خروجی تست دوم به شکل زیر است:

```
→  codes python3 AStar2.py tests/test2.txt
Path Length: 15
Path: RDLLULDDDDDLLLL
Number of states: 1092
Number of not reapeted states: 785
Time: 115.336657 ms
```

خروجی تست سوم به شکل زیر است:

```
→  codes python3 AStar2.py tests/test3.txt
Path Length: 25
Path: RDUUURURRURRURRRDLLULDDLL
Number of states: 16406
Number of not reapeted states: 7157
Time: 1065.821171 ms
```

Weighted A*

در این تابع از heuristic اول استفاده میکنیم.

با دستور زیر برنامه را اجرا میکنیم:

python3 AStar2.py tests/test1.txt alpha

که آرگومان اول آدرس فایل تست و آرگومان دوم مقدار α میباشد.

در این الگوریتم صرفا مقدار heuristic محاسبه شده در مقدار ثابت α ضرب میشود. این عمل ممکن است بهینه بودن الگوریتم A* را به خطر بیاندازد ولی اگر α به درستی انتخاب شود میتواند سرعت را بیشتر کند.

تست اول به ازای α برابر ۲:

```
→  codes python3 WeightedAStar.py tests/test1.txt 2
Path Length: 12
Path: ULUURRRURUUR
Number of states: 443
Number of not reapeted states: 326
Time: 34.577847 ms
```

تست دوم به ازای α برابر ۲:

```
→  codes python3 WeightedAStar.py tests/test2.txt 2
Path Length: 15
Path: RDLLULDDDDDLLLL
Number of states: 9587
Number of not reapeted states: 5883
Time: 738.703489 ms
```

تست سوم به ازای α برابر ۲:

```
→  codes python3 WeightedAStar.py tests/test3.txt 2
Path Length: 25
Path: DRUUURURRUURRRDRRULLLDDLL
Number of states: 21097
Number of not reapeted states: 10006
Time: 1299.370289 ms
```

تست اول به ازای α برابر ۵:

```
→  codes python3 WeightedAStar.py tests/test1.txt 5
Path Length: 12
Path: ULDDLDLLDDLL
Number of states: 73
Number of not reapeted states: 72
Time: 7.505178 ms
```

تست دوم به ازای α برابر ۵:

```
→  codes python3 WeightedAStar.py tests/test2.txt 5
Path Length: 17
Path: DLURRDDDDLLLDLLLL
Number of states: 1838
Number of not reapeted states: 1487
Time: 189.342499 ms
```

تست سوم به ازای α برابر ۵

```
→  codes python3 WeightedAStar.py tests/test3.txt 5
Path Length: 26
Path: DRUUURURRULDRRUURRDRRUULDL
Number of states: 811
Number of not reapeted states: 645
Time: 79.341888 ms
```

مشاهده می‌شود در این روش سرعت به طرز چشمگیری افزایش پیدا کرده است. تعداد استیت های دیده شده و زمان به طرز محسوسی کاهش پیدا کرده‌اند. اما در حالت دوم (یعنی α برابر ۵) که مقدار α بیش از حد بزرگ شد باعث شد که جواب بهینه به دست نیاید. وقتی heuristicها در مقدار ثابتی ضرب میشوند باعث میشود به مقدار هزینه واقعی که باید بپردازیم نزدیک تر شوند اما اگر این مقدار ثابت بیش از حد بزرگ باشد همانطور که مشاهده شد تخمین هزینه از هزینه واقعی بیشتر میشود و جواب بهینه به دست نمی‌آید.

جداول

تست کیس اول:

	فاصله جواب	مسیر حروف	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا بر حسب میلی ثانیه
BFS	12	ULUURRRURUUR	1805	890	91
IDS	12	ULUURRRURUUR	25394	12901	743
A* 1	12	ULDDLDLLDDLL	1655	969	104
A* 2	12	ULUURRRRUUUR	1114	617	78
Weighted $\alpha = 2$	12	ULUURRRURUUR	443	326	34
Weighted $\alpha = 5$	12	ULDDLDLLDDLL	73	72	7

تست کیس دوم:

	فاصله جواب	مسیر حروف	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا بر حسب میلی ثانیه
BFS	15	DRULLDDDDLDLLLL	31333	13811	1638
IDS	15	DRULLDDDDLDLLLL	249523	120999	8831
A* 1	15	RDLLULDDDDDLLLL	22974	11112	1464
A* 2	15	RDLLULDDDDDLLLL	1092	785	112
Weighted $\alpha = 2$	15	RDLLULDDDDDLLLL	9587	5883	743
Weighted $\alpha = 5$	17	DLURRDDDDLLLDLLLL	1838	1487	193

تست کیس سوم:

	فاصله جواب	مسیر حروف	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	زمان اجرا بر حسب میلی ثانیه
BFS	25	DRUUURURRUURRRDRRULLLDDLL	45979	19032	2389
IDS	25	DRUUURURRUURRRDRRULLLDDLL	805412	342858	29036
A* 1	25	DRUUURURRUURRRDRRULLLDDLL	37651	16668	2227
A* 2	25	RDUUURURRURRURRRDLLULDDLL	16406	7157	1077
Weighted $\alpha = 2$	25	DRUUURURRUURRRDRRULLLDDLL	21097	10006	1308
Weighted $\alpha = 5$	26	DRUUURURRULDRRUURRDRRUULDL	811	645	79

با توجه به جداول بالا درمی‌یابیم اگر مقدار α به خوبی انتخاب شود میتواند تاثیر زیادی بر روی سرعت ما بگذارد اما بد انتخاب شدن آن بهینه بودن را به خطر می اندازد. انتخاب عدد مناسب کار بسیار دشواری است و بستگی به heuristic استفاده شده نیز دارد. به عنوان مثال، اگر heuristic بسیار نزدیک به مقدار واقعی در نظر گرفته شود با ضرب کردن آن در یک عدد ثابت ممکن است برای تعداد زیادی از استیت‌ها مقدار آن از مقدار هزینه واقعی بیشتر شود و پاسخ به دست آمده بهینه نباشد. اگر سرعت برای ما مهم‌تر از بهینه بودن پاسخ باشد انتخاب آلفای بزرگتر می‌تواند مارا در این امر یاری کند.

روش IDS روش سریعی نیست و میتوان با روشهای بهتری به پاسخ بهینه رسید. حتی با استفاده از BFS که روش دیگری از جستجوی ناآگاهانه است پاسخ سریعتری به دست می‌آید.

heuristic دوم از نظر نزدیک بودن به واقعیت از heuristic اول بهتر است زیرا سریع‌تر به نتیجه رسیده است. به عبارت دیگر تخمین دقیق‌تری است و به واقعیت نزدیک‌تر است اما پیاده سازی آن دشوارتر است.

اگر استیت هایی که میبینیم را نگه نداریم تا از تکرار آنها جلوگیری کنیم (مفهوم مجموعه explored) سرعت به شدت کاهش می‌یابد با دقت در اختلاف تعداد استیت دیده شده و تعداد استیت مجزای دیده شده میتوان این موضوع را درک کرد.

در کل اگر بخواهیم عواملی نظیر سختی پیاده‌سازی و بهینه بودن جواب و حافظه مصرفی (که از روی استیت‌های دیده شده قابل مقایسه است) و زمان اجرا را در نظر بگیریم روش A* از سایر روش‌ها بهتر بوده است و با انتخاب یک تخمین نزدیک به واقعیت می‌توان سرعت را بالا برد و حافظه مصرفی را کم کرد. از طرف دیگر روش IDS هیچ آورده‌ای برای ما ندارد و از نظر هزینه هم به صرفه نیست. زیرا حافظه مصرفی آن نسبت به سایر الگوریتم‌ها به شدت زیاد است.