گزارش پروژه اول درس هوش مصنوعی

اولدوز نیساری 810199505

بر اساس گراف دنیا سید که داریم و با توجه به خواسته های مسئله به ازای هر نود گراف و با توجه به داده های دیگر مسئله کلاسی به نام state تشکیل می دهیم و در روند حل مسئله درخت جستجو را بر اساس این استیت ها تشکیل می دهیم. هر استیت دارای feature های زیر است :

Position:

که نودی(مکانی که در گراف دنیا سید در آن قرار دارد) که در آن استیت در آن نود هستیم را مشخص می کند

Path:

مسیری که از نقطه شروع تا رسیدن به آن استیت طی شده است که آرایه ای است از استیت ها

Needed_time:

زمانی که برای گذر از آن استیت لازم است .این مقدار برای تمام استیت ها به صورت پیش فرض 0 است اما برای راس های صعب العبور این زمان برابر تعداد دفعاتی است که از آن گذشته است.

Visited_recp:

دیکشنری است که به تمام دستور پخت های موجود در گراف دنیا سید می شود بر حسب دیده شدن یا نشدن یک مقدار درست یا غلط نسبت میدهد.در ابتدا به همه دستورپخت ها مقدار falseنسبت داده می شود ، سپس با هر بار گذر از نودی از گراف دنیا سید که در آن دستور پختی است مقدار آن دستور پخت در آن استیت true میشود .

Visited_morid:

دقیقا مشابه دیشکنری دستور پخت هاست با این تفاوت که این بار در آن مرید ها و وضعیت راضی شدن آن ها قرار دارد.اگر مریدی تمام دستور پخت هایش دیده شود آن مرید راضی شده است.

Cost :

هزینه ای است که تا رسیدن به آن نود پرداخت شده است .

Pass_number :

تعداد دفعاتی که قبلا از استیتی با نود یکسان در گراف دنیا سید گذر کرده ایم ، که برای مدیریت رئوس صعب العبور به درد می خورد .

کلیات روند الگوریتم سرچ :

بعد از تشکیل این کلاس با ویژگی های گفته شده یک استیت اولیه بر اساس مکان اولیه سید که مشخص شده است ، مسیری که شامل هیچ استیتی نیست و دیشکنری های دستور پخت و مرید که هردو تماما false هستند ، هزینه اولیه 0 و pass number=0 می سازیم .

در هر یک از الگوریتم ها تا جایی که مجموعه فرانتیر خالی نشده باشد چرخه حذف شدن از مجموعه فرانتیر و بسط داده شدن ادامه دارد و اگر در مرحله ای به استیتی برسیم که همه مرید های آن راضی شده باشند به هدف مسئله رسیده ایم و در آنجا به الگوریتم پایان می دهیم. در واقع می توان گفت در مدل کردن این مسئله test goal بررسی تعداد مرید های راضی شده است ، اگه همه مرید ها راضی شده باشند به goal رسیده ایم. در هر بار روند حذف از مجموعه فرانتیر و بسط دادن چند گام طی می شود .

استیتی از مجموعه فرانتیر حذف می شود .استیت وارد تابعی به نام apply_state_changes می شود و در این تابع بنا بر این که در position آن استیت در گراف دنیا سید چه چیز هایی قرار دارد تغییراتی اعمال می شود . اگر دستور پختی در آن position باشد مقدار آن دستور پخت در دیشکنری position ها true می شود . اگر مریدی باشد به ازای تمام دستور پخت هایش بررسی می کنیم که آیا دستور پخت هایش دیده شده اند یا نه ، اگر دیده شده بودند مقدار آن مرید را هم در دیکشنری مرید ها true می کنیم .بعد از این مرحله اگر needed_time استیت برابر 0 بود وارد تابعی به نام add_children می شویم و با تشکیل استیت های فرزندان استیت بر اساس نود های همسایه position استیت در گراف دنیا سید آن ها را به مجموعه فرانتیر اضافه می کنیم و اگر نه از needed_time آن یک واحد کم می کنیم (چون عملا به اندازه یک واحد صبر کردیم) ، به هزینه یک واحد اضافه می کنیم و بار دیگر استیت را به مجموعه فرانتیر اضافه می کنیم .

باتوجه به توضیحاتی که در بخش قبل هم داده شد در این مسئله :

goal _state :

استیتی است که در آن مقدار تمام مرید ها در دیشکنری مرید ها true باشد.

Action :

رفتن به همسایه های نود position در گراف دنیا سید است .

شرح الگوریتم ها :

:BFS

در این الگوریتم از یک صف به عنوان مجموعه فرانتیر استفاده می کنیم . در ابتدا initial state ای که ساخته ایم را به این مجموعه اضافه می کنیم .سپس تا جایی که به شرط مسئله یعنی راضی شدن تمام مرید ها نرسیدیم دقیقا روندی که در بخش کلیات الگوریتم سرچ توضیح دادیم را طی می کنیم .در این الگوریتم چون سطر به سطر پیش میرویم مطمئن هستیم که به لحاظ طول مسیر میتوانیم بهینه ترین حالت را پیدا کنیم . نکته قابل ذکر این است که با استفاده از همین خاصیت بهینه بودن در طول مسیر برای

رئوس صعب العبور کاری که انجام می دهیم این است که اگر زمان لازم برای گذر از آن صفر باشد در آن صورت از مجموعه فرانتیر حذف می کنیم در غیر این صورت بعد از حذف با کم کردن یک واحدی از زمان مورد نیاز برای گذر از آن ها ، آن ها را مجددا به مجموعه فرانتیر اضافه می کنیم و عملا در یک عمق جدید باز هم آن استیت را اضافه می کنیم به جای فرزندانش.

:IDS

از عمق 1 شروع می کنیم و هر بار با عمق محدود مشخص شده الگوریتم dfs با محدویدیت عمق را اجرا می کنیم .اگر به هدف مسئله که راضی شدن تمام مرید ها است در آن طول رسیدیم چرخه را متوقف می کنیم ، اگر نه یک واحد به عمق مشخص شده مان اضافه می کنیم و بار دیگر الگوریتم dfs را اجرا می کنیم.پیاده سازی این الگوریتم dfs با محدودیت عمق هم دقیقا مشابه حالت قبلی است با این تفاوت که این بار برای مجموعه فرانتیز به جای صف از استک استفاده می کنیم . در تابع add_children این بار متغیر عمق را هم می گیریم و هربار که می خواهیم فرزندی را اضافه کنیم طول مسیر آن استیت را که عملا تعداد اجداد آن استیت است را با طول عمق داده شده به آن می سنجیم و تا جایی که از عمق داده شده کمتر باشد ، فرزند را می افزاییم.

:*A

برای هر یک از استیت ها یک تابع هیوریستیک به این صورت تعریف می کنیم که مجموع تمام دستور پخت ها و مرید ها منهی تعداد استیت های در مسیر استیت که شامل مرید یا دستور پخت باشند .به ازای هر استیت این مقدار را به اضافه مقدار هزینه ای که تا رسیدن به آن استیت شده است را به هر استیت نسبت می دهیم . این بار برای مجموعه فرانتیر یک min heap در نظر می گیریم و استیت ها را به همراه مقداری که به آن ها نسبت دادیم تحت قالب یک tuple به مجموعه اضافه می کنیم و تعیین می کنیم که بر اساس مقدار نسبت داده شده در مجموعه مرتب شوند . با اختصاص min heap برای مجموعه فرانتیر اطمینان پیدا می کنیم که هر بار بهینه ترین استیت را برای بسط دادن انتخاب می کنیم (بر اساس مجموع هیوریستیکی که خودمان تعیین کردیم و هزینه ای که تاکنون برای آن شده است که اساس کار الگوریتم A* است).در خصوص مدیریت استیت هایی با نود صعب العبور هم مشابه حالت های قبلی اگر زمان مورد نیاز برای استیت 0 نبود بار دیگر به مجموعه فرانتیر با یک واحد کاهش زمانی و با یک واحد افزایش هزینه اضافه می کنیم.

:*Weighted A

این الگوریتم کاملا مشابه الگوریتم قبلی است . با این تفاوت که این بار با مشخص کردن یک مقدار آلفا نقش تابع هیورستیک را افزایش میدهیم .این مقدار آلفا در بخش هیورستیک مان ضرب می شود و تاثیر آن بخش را بیشتر می کند و من مقادیر 1.3 و 1.8 را برای آلفا انتخاب کردم .

توضیحات تابع heuristic انتخاب شده :

برای هیوریستیک هر استیت این گونه عمل می کنیم که برای استیت شروع هیورستیک را برابر مجموع تعداد مرید ها و تعداد دستور پخت ها قرار می دهیم .هر استیت جدید را هیورستیکش را برابر هیوریستیک پدرش قرار می دهیم اگر نود position استیت در گراف دنیای سید شامل دستور پخت یا مرید بود یک واحد از آن کم می کنیم . واضح است از آنجایی که این هیورستیک حداکثر به تعداد استیت های بین دو استیت تغییر می کند و این تعداد از حداقل هزینه بین دو استیت که تعداد یال های بین هاست کمتر است ، الگوریتم consistent است .

نتایج BFS برای سه تست کیس:

```
1->3->4->5->7->10->11->9->7
cost 8
visited_states 38
0.002791643142700195
```

```
28->19->13->3->11->24->9->23->28->23->5->7->28
cost 12
visited_states 4045
10.994732141494751
```

```
40->42->38->24->31->45->30->48->41->18->1->19->43->49->47->49->9->34->25->50->12->15
cost 21
visited_states 3607
4.559550762176514
```

نتایج IDS:

```
1->3->10->11->10->7->5->7->8
cost 9
visited_states 22
0.012009143829345703
```

```
9->10->2->4->12->3->7->5->8
cost 8
visited_states 48
0.05098485946655273
```

```
13->11->10->11->7->2->3->2->9->4->1->13->11->10->5->12->5->10->11
cost 19
visited_states 367
2.7462828159332275
```

نتایج Astar :

```
1->3->4->5->7->10->11->9->8
cost 8
visited_states 40
0.004576921463012695
```

```
28->19->13->3->11->24->9->22->28->22->5->7->29
cost 12
visited_states 4037
10.015943765640259
```

```
40->42->38->24->31->45->30->48->41->18->1->19->43->49->47->49->9->34->25->50->12->16
cost 21
visited_states 3559
4.755645275115967
```

نتایج weighted Astar با آلفا 1.3 :

```
1->3->4->5->7->10->11->9->8
cost 8
visited_states 40
0.016281366348266 6
```

```
28->19->13->3->11->24->9->22->28->22->5->7->29
cost 12
visited_states 4037
9.261064767837524
```

```
40->42->38->24->31->45->30->48->41->18->1->19->43->49->47->49->9->34->25->50->12->16
cost 21
visited_states 3559
3.016017198562622
```

نتایج weighted Astar با آلفا 1.8 :

```
1->3->4->5->7->10->11->9->8
cost 8
visited_states 40
0.0032253265380859375
```

```
28->19->13->3->11->24->9->22->28->22->5->7->29
cost 12
visited_states 4037
9.216155052185059
```

```
40->42->38->24->31->45->30->48->41->18->1->19->43->49->47->49->9->34->25->50->12->16
cost 21
visited_states 3559
2.986661911010742
```

در بررسی الگوریتم ها از لحاظ بهینه بودن :

الگوریتم BFS که به دلیل نوع پیمایش آن همواره جواب بهینه را میدهد ، الگوریتم IDS هم با شرط محدودیتی که دارد جواب بهینه را می دهد ، اما اگر بدون محدودیت بود بهینه بودن آن از بین می رفت.
در الگوریتم Astar هم با توجه به consistent بودن هیورستیکمان الگوریتم بهینه می شود .الگوریتم weighted Astar همیشه بهینه نیست اما نکته ای که دارد این الگوریتم با نزدیک تر کردن هیورستیک ما به هزینه واقعی سرعت را بالا می برد که این مسئله به وضوح هم قابل دیدن است .

در کنار این مسائل قابل بیان است که به لحاظ حافظه ای الگوریتم IDS به لحاظ گام به گام پیش رفتن خطی است اما بقیه الگوریتم ها نمایی هستند .

Test 1	پاسخ مسئله	تعداد استیت های دیده شده	میانگین زمان اجرا
BFS	8	38	**0.05**
IDS	9	22	**0.01**
*A	8	38	**0.002**
Weighted A* 1	8	39	**0.01**
Weighted A* 2	8	37	**0.05**

Test 2	پاسخ مسئله	تعداد استیت های دیده شده	میانگین زمان اجرا
BFS	8	4045	**12.08**
IDS	8	48	**0.04**
*A	12	2654	**9.31**
Weighted A* 1	13	699	**9.13**
Weighted A* 2	13	106	**9.11**

Test 3	پاسخ مسئله	تعداد استیت های دیده شده	میانگین زمان اجرا
BFS	21	3607	**4.13**
IDS	19	367	**2.61**
*A	21	3334	**4.72**
Weighted A* 1	21	2971	**2.99**
Weighted A* 2	21	2080	**2.95**