به نام خدا

گزارش پروژه ۱ درس هوش مصنوعی

استاد روشن فکر

میلاد اسرافیلیان

۹۷۳۱۰۰۷

فرموله سازی مسئله:

توضیح کلاس ها:

در بین هر سه فایل برای سوالات دو کلاس مشترک وجود دارد که توضیح آن ها به شرح زیر است:

- کلاس Card:

این کلاس جهت شبیه سازی و فرموله کردن هر کارت موجود در مسئله در نظر گرفته شده است که شامل دو attribute رنگ (color) و شماره (number) میباشد.

توابع کلاس:

تابع splitNumColor: جهت جدا سازی رنگ و شماره هر کارت که به عنوان ورودی یک رشته از ترکیب هر دو میگیرد.

دو تابع __eq__ و __hash__ را نیز برای این کلاس override می کنیم تا جهت مقایسه هر کارت با کارت دیگر رنگ و شماره آن کارت ها را در نظر بگیرد.

- کلاس State:

این کلاس جهت شبیه سازی هر node در گراف و در واقع هر حالت از بازی در هر لحظه از زمان ایجاد شده است. شامل چهار attribute است:
۱- stateList: جهت ذخیره کردن وضعیت کارت ها در تمام ستون ها به صورت یک لیست دو بعدی.
۲- parent: جهت ذخیره کردن والد هر فرزند و node ایجاد شده در گراف.
۳- action: عمل انجام گرفته که باعث شده است حالت بازی از حالت parent به حالت فعلی برسد.
۴- depth: عمق هر state یا node در گراف حالات

توابع کلاس:

تابع addAction: اضافه کردن عمل یا action ورودی به حالت.

دو تابع __eq__ و __hash__ را نیز override می کنیم تا در هنگام مقایسه دو شی از نوع state لیست حالات دو شیء را مقایسه کند و هنگام hash کردن نیز با توجه به محتویات کارت های هر ستون و ترتیب آن ها این hash کردن اتفاق بیفتد.

- توضیح سایر توابع مشترک:

تابع canBeAdded: چک می کند که آیا می توان کارت ورودی را به ستون ورودی اضافه کرد یا خیر بر اساس اینکه آیا کارت ورودی از آخرین کارت ستون ورودی کوچکتر است یا خیر و اینکه آیا ستون ورودی اگر خالی است هر کارتی می توان به آن اضافه کرد.

تابع sameColor: چک می کند که آیا کارت های لیست ورودی همگی دارای رنگ یکسان هستند یا خیر.

تابع trueOrder: چک میکند که آیا کارت های لیست ورودی به ترتیب نزولی قرار گرفته اند یا خیر.

تابع getTopCard: اگر لیست خالی نباشد آخرین کارت لیست ورودی را برمی گرداند در غیر این صورت False برمی گرداند.

تابع goalTest: با استفاده از دو تابع sameColor , trueOrder آیا همه کارت های لیست های موجود(ستون ها) در لیست دو بعدی ورودی دارای رنگ یکسان و ترتیب نزولی هستند یا خیر.

تابع findInFile: جهت خواندن ورودی تست کیس ها از فایل test.txt و قرار دادن آن ها در یک لیست دو بعدی از کارت ها.

هیوریستیک های استفاده شده در الگوریتم A*:

در این مسئله ما از دو هیوریستیک متفاوت استفاده می کنیم که هر دو قابل قبول اند جهت بهتر شدن الگوریتم در هر مرحله به عنوان هیوریستیک اصلی از ماکسیمم این دو هیوریستیک استفاده می کنیم که میدانیم قابل قبول است. راه حل کلی ما برای ایجاد این هیوریستیک ها ساده سازی مسئله و ریلکس کردن آن است به دو صورت زیر:

- هیوریستیک اول - رنگ:

در این هیوریستیک ریلکس کردن به این صورت خواهد بود که ترتیب اعداد را در نظر نمی گیریم. حال در این مسئله ساده شده تنها چیزی که مد نظر ماست این است که رنگ کارت های هر ستون یکسان باشد. برای این کار در هر ستون یا لیست پیمایش می کنیم و هر جا که رنگ کارت تغییر کند یکی به متغیر anomaly که در ابتدا صفر است اضافه می کنیم و این کار را برای تمام ستون ها ادامه می دهیم و در انتها مقدار متغیر anomaly را به عنوان هیوریستیک بر می گردانیم.
قابل قبول بودن این هیوریستیک از آنجا بر می آید که با توجه به اینکه در حالت هدف رنگ همه کارت ها در یک ستون یکسان است پس بدیهی است که حداقل به ازای هر کارتی که رنگی متفاوت دارد باید یک عمل یا action انجام شود تا به ستونی که کارت های آن همرنگ این کارت است منتقل شود و از آنجا که این مقدار کمتر مساوی حالت اصلی مسئله است پس این هیوریستیک قابل قبول است.

- هیوریستیک دوم - عدد:

در این هیوریستیک نیز دو بار مسئله اصلی را ساده سازی یا ریلکس می کنیم:
اول رنگ کارت ها را در نظر نمی گیریم و تنها بر اساس اینکه آیا اعداد کارت ها به صورت نزولی هست یا خیر تصمیم می گیریم.
در مرحله بعد محدودیت های مسئله که شامل قرار دادن کارت با شماره کوچکتر روی شماره بزرگتر می شود را نیز در نظر نمیگیریم.
تنها مقایسه ما این است که با فرض اینکه اعداد قرار گرفته در این بخش بتوانند به صورت مرتب شده قرار بگیرند چه تعداد عمل باید انجام دهیم.

جهت پیاده سازی به صورت زیر عمل می کنیم:
ابتدا ماکسیمم مقدار موجود در آرایه را پیدا می کنیم اگر جایگاه آن در جایگاه iام آرایه اصلی نبود یکی به متغیر sum اضافه می کنیم حال در زیر آرایه ای به جز عنصر اول عملیات های بالا را تکرار می کنیم پس از اتمام مقادیر همه ستون ها را جمع کرده و به عنوان مقدار هیوریستیک بر می گردانیم.

قابل قبول بودن این هیوریستیک از آنجا بر می آید که با توجه به در نظر نگرفتن محدودیت ها در صورت نامرتب بودن اعداد باید کارت ها را به ستون های دیگر منتقل کرده و دوباره به ستون اصلی برگردانیم یا در صورت وجود ستون خالی آن کارت را در همانجا رها میکنیم پس به ازای یک کارت که در ستون نامرتب است حداقل ۱ اکشن برای مرتب سازی باید انجام دهیم کاری که در هیوریستیک ما نیز انجام می شود(به ازای هر کارت نامرتب حداکثر مقدار یک اضافه می شود).

هر دو هیوریستیک ما در واقع مکمل یکدیگرند به این صورت که اگر رنگ تمامی کارت ها در یک ستون یکسان باشد هیوریستیک رنگ مقدار 0 (با اینکه حالت هدف نیست) ولی هیوریستیک عدد مقداری بیشتر از صفر برمی گرداند.

بالعکس اگر ترتیب اعداد کارت های یک ستون به صورت نزولی باشند ولی رنگ آن ها یکی نباشد هیوریستیک عدد مقدار 0(با اینکه حالت هدف نیست) و هیوریستیک رنگ مقداری بیشتر از صفر برمی گرداند.
حال با ماکسیمم گرفتن از هر دو هیوریستیک و با توجه به اینکه هر دو قابل قبول اند هیوریستیکی قابل قبول و دقیق تر خواهیم داشت که هرجا یکی ضعیف عمل کند دیگری آن را پوشش می دهد.

سازگاری:
با توجه به اینکه اندازه هر عمل ۱ است و در هر تغییر یا هیوریستیک رنگ یکی کمتر میشود(به ستون همرنگ برده می شود کارت) یا هیوریستیک عدد یکی کمتر می شود(به ستونی با ترتیب درست برده می شود) و در صورتی که هر دو اتفاق بیفتد از آنجا که ماکسیمم را انتخاب میکنیم پس حداقل مقدار هیوریستیک فرزند یکی از هیوریستیک والد کمتر است اما با توجه به یک بودن مسیر از والد به فرزند مقدار f فرزند برابر والد می شود. چون حداقل ها را در نظر گرفتیم تمامی حالات دیگر بزرگتر یا مساوی حالت فعلی خواهند بود و میتوان نتیجه گرفت که مقدار f در طول هر مسیری غیر نزولی است پس هیوریستیک ما سازگار نیز هست و میتوان نتیجه گرفت که الگوریتم A* بهینه خواهد بود.

(جهت اطمینان بیشتر در الگوریتم A* اگر مقادیر نزولی شود آن را غیر نزولی و هیوریستیک را سازگار میکنیم)

توضیح الگوریتم ها:

- الگوریتم BFS:

در کلاس BFS ابتدا مقدار اولیه یا initialState را در کلاس تعیین می کنیم.
در تابع breadthFirstSearch الگوریتم اجرا می شود:
در صورتی که تعداد رنگ ها از تعداد ستون ها بیشتر باشد الگوریتم Failure برمی گرداند.
ابتدا حالت اولیه را به frontier اضافه می کنیم.
در یک لوپ بی نهایت اعمال زیر را انجام می دهیم:
اگر چیزی در frontier باقی نمانده باشد Failure برمی گردانیم.
در غیر اینصورت اولین عنصر اضافه شده به frontier را بیرون می کشیم - FIFO Queue- سپس آن را به مجموعه visited اضافه می کنیم و به تعداد نود های بسط یافته یکی اضافه می کنیم. State انتخاب شده را به عنوان والد در نظر گرفته و با توجه به اعمال مجاز فرزندانش را تولید می کنیم.
در هر بار تولید فرزند به تعداد نود های ایجاد شده یکی اضافه کرده و اگر state ایجاد شده در مجموعه visited و frontier نباشد اکشن مربوطه را اضافه کرده و پس از مشخص کردن والد و عمق تابع goalTest را روی فرزند صدا میزنیم اگر حالت هدف باشد آن را برمی گردانیم در غیر این صورت آن را به مجموعه frontier اضافه می کنیم. و به سراغ فرزند بعدی می رویم.

- الگوریتم IDS:

در کلاس IDS ابتدا مقدار اولیه initialState را در کلاس تعیین می کنیم.
حال در تابع iterative_deepening_search در یک حلقه از لیمیت اولیه تعیین شده جلو رفته و تابع depth_limited_search را صدا میزنیم در جواب cutoff باشد آن را بر میگردانیم در غیر این صورت ادامه می دهیم.
در تابع depth_limited_search هر بار با لیمیت داده شده و حالت اولیه مسئله تابع recursive_DLS را صدا زده و نتیجه را برمی گردانیم.
در تابع recursive_DLS اگر حالت ورودی هدف باشد آن را برگردانده و اگر لیمیت صفر باشد cutoff برمی گردانیم در غیر اینصورت حالت ورودی را بسط می دهیم و فرزندانش را تولید میکنیم و یکی به expandedNodes اضافه می کنیم پس از

مشخص کردن والد اکشن و عمق حالت ایجاد شده یکی به createdNodes اضافه کرده و این بار با حالت تولید شده و کم کردن یک واحد از لیمیت دوباره recursive_DLS را صدا میزنیم این کار تا زمانی ادامه پیدا می کند یا به cutoff برخورد کنیم در این صورت به سراغ برادر نود میرویم اگر نتیجه برگردانده شده Failure نباشد آن را بر میگردانیم در غیر این صورت تابع را روی تمام فرزندان صدا میزنیم اگر نتیجه ای حاصل نشد اگر cutoff باشد آن را برمی گردانیم در غیر این صورت Failure برمی گردانیم و به این صورت الگوریتم تمام می شود.

- الگوریتم A*:

در کلاس AStar ابتدا مقدار اولیه initialState را در کلاس تعیین می کنیم.
حال در تابع AStarSearch ابتدا اگر حالت اولیه همان حالت هدف باشد آن را برمی گردانیم در غیر این صورت مقدار F آن را با استفاده از هیوریستیک هایی که توضیح داده شد و عمق آن حساب کرده و به frontier و visited اضافه می کنیم.
حال در یک حلقه بی نهایت اعمال زیر را انجام می دهیم:
اگر frontier خالی باشد Failure برمی گردانیم.
در غیر این صورت حالتی که کمترین F را در frontier دارد بیرون می کشیم و آن را برای بسط انتخاب می کنیم اگر حالت هدف باشد آن را برمی گردانیم در غیر این صورت فرزندان آن را تولید می کنیم پس از آنکه والد عمق و اکشن آن را مشخص کردیم جهت اطمینان بیشتر آن را سازگار کرده در صورتی که در frontier نباشد یا باشد ولی مقدار F جدید از قبلی کمتر باشد frontier را آپدیت می کنیم الگوریتم زمانی تمام می شود که حالت انتخاب شده از frontier حالت هدف باشد.

مقایسه سه الگوریتم:

تست کیس آزمایش شده مطابق زیر است:

```
5 3 5
5g 1r 2g
5r 4r 3r
4g 1g 2r
1b 2b 3g
#
```

- BFS:

~~~
mies@mies-X541UVK:~/Desktop/AIProject/Intro-To-AI-Card-Game-Project$ python3 BFS.py

Elapsed Time is: 0 minutes and 7.869791507720947 seconds

````````````````````````````````````````````````````````````````````````````````````````````````````
Final State Is:

5g 3g 2g

5r 4r 3r 2r 1r

4g 1g

1b

2b
````````````````````````````````````````````````````````````````````````````````````````````````````
Actions:
Moved Card 2g From Pile 1 To Pile 5
Moved Card 2r From Pile 3 To Pile 2
Moved Card 1r From Pile 1 To Pile 2
Moved Card 3g From Pile 4 To Pile 1
Moved Card 2g From Pile 5 To Pile 1
Moved Card 2b From Pile 4 To Pile 5

````````````````````````````````````````````````````````````````````````````````````````````````````
Depth is: 6

````````````````````````````````````````````````````````````````````````````````````````````````````
Created Nodes Are: 11768

````````````````````````````````````````````````````````````````````````````````````````````````````
Expanded Nodes Are: 1393

````````````````````````````````````````````````````````````````````````````````````````````````````
~~~

- IDS(initial Limit = 0):

```
mies@mies-X541UVK:~/Desktop/AIProject/Intro-To-AI-Card-Game-Project$ python3 IDS.py
0
``````````````````````````````````````````````````````````````````````````````````````````````````````````````

Elapsed Time is: 1 minutes and 44.76685690879822 seconds

``````````````````````````````````````````````````````````````````````````````````````````````````````````````
Final State Is:

5g 3g 2g

5r 4r 3r 2r 1r

4g 1g

1b

2b

``````````````````````````````````````````````````````````````````````````````````````````````````````````````
Actions:
Moved Card 2g From Pile 1 To Pile 5
Moved Card 2r From Pile 3 To Pile 2
Moved Card 1r From Pile 1 To Pile 2
Moved Card 3g From Pile 4 To Pile 1
Moved Card 2g From Pile 5 To Pile 1
Moved Card 2b From Pile 4 To Pile 5
``````````````````````````````````````````````````````````````````````````````````````````````````````````````
Depth is: 6

``````````````````````````````````````````````````````````````````````````````````````````````````````````````
Created Nodes Are: 202135

``````````````````````````````````````````````````````````````````````````````````````````````````````````````
Expanded Nodes Are: 23228

``````````````````````````````````````````````````````````````````````````````````````````````````````````````
```
```

- A*:

```
mies@mies-X541UVK:~/Desktop/AIProject/Intro-To-AI-Card-Game-Project$ python3 AStar.py
Elapsed Time is: 0 minutes and 1.3435871601104736 seconds
`````````````````````````````````````````````````````````````````````````````````````````````````
Final State Is:
5g 3g 2g
5r 4r 3r 2r 1r
4g 1g
1b
2b
`````````````````````````````````````````````````````````````````````````````````````````````````
Actions:
Moved Card 2g From Pile 1 To Pile 5
Moved Card 2r From Pile 3 To Pile 2
Moved Card 1r From Pile 1 To Pile 2
Moved Card 3g From Pile 4 To Pile 1
Moved Card 2g From Pile 5 To Pile 1
Moved Card 2b From Pile 4 To Pile 5
`````````````````````````````````````````````````````````````````````````````````````````````````
Depth is: 6
`````````````````````````````````````````````````````````````````````````````````````````````````
Created Nodes Are: 2225
`````````````````````````````````````````````````````````````````````````````````````````````````
Expanded Nodes Are: 272
`````````````````````````````````````````````````````````````````````````````````````````````````
```

| | BFS | IDS | A* |
|---|---|---|---|
| Depth | 6 | 6 | 6 |
| createdNodes | 11768 | 202135 | 2225 |
| expandedNodes | 1393 | 23228 | 272 |
| Time | 0:7 | 1:44 | 0:1 |