

به نام خدا

پروژه اول درس هوش مصنوعی

استاد نیک آبادی

محمد مهدی آقاجانی

پاییز ۹۵

نحوه مدل سازی پروژه و روابط حاکم بر کلاس ها

در این پروژه سه package ساخته شده است . که اولی مربوط به program و بعدی algorithm و بعدی data structures میباشد که هر یک را به اختصار توضیح میدهیم :

۱- problem

این بسته شامل کلاس پدر problem میباشد که abstract است و متد های مربوط به یک مساله مانند اولین حالت و یا حالت های بعدی و تعیین هدف بودن و تابع شهودی در این کلاس است که باید مابقی کلاس های فرزند این متد ها را پیاده سازی کنند.

همچنین کلاس های فرزند Robot و Queens وجود دارند که مربوط به هر یک از این مسایل میباشد.

۲- algorithm

این بسته شامل کلاس پدر algorithm است که حاوی لیست های f , e است و همچنین متد abstract ای دارد به نام apply که همه فرزندان باید آن را پیاده سازی کنند . ویژگی های مربوط به یک الگوریتم از قبیل تعداد گره های دیده شده یا تعداد گره های بسط داده شده و بیشترین مصرف حافظه و متد های , getter setter آن ها نیز در این کلاس ها قرار دارند .

اما هر یک از الگوریتم ها به نام خودشان دارای یک کلاس هستند که متد apply در آن ها پیاده سازی شده است که دارای دو حالت درختی و گرافی میباشد. برای اینکه یک الگوریتم را بر روی مساله اجرا کنیم کافی است متد apply آن را به همراه حالت اجرا (درختی یا گرافی) بر روی مساله صدا بزنیم (یکی از ورودی های این متد شی ای از جنس problem میباشد)

همچنین این بسته بر اساس جست وجو های آگاهانه یا نا آگاهانه دسته بندی شده است .

۳- data structures

این بسته شامل کلاس پدر node می باشد و کلاس های فرزند robot node , queens node در این بسته حضور دارند . یک node شامل اشاره گری به پدرش و همچنین هزینه دسترسی به خود است که متد

هایمربوط به آن‌ها نیز در آن‌ها پیاده‌سازی شده است . همچنین state نیز داخل node نگه داری می‌شود که برای هر مساله ساختمان داده مربوط به خود را دارد.

توضیح کلاس‌های root

۱- problem solving agent

این کلاس درواقع به عنوان حل کننده مسایل عمل میکند بدین صورت که یک مساله و یک الگوریتم میگیرد و آن الگوریتم را بر روی مساله اجرا میکند و سپس جواب بدست آمده را به کلاس response formatter میدهد تا این کلاس بتواند جواب را با فرمت درستی نمایش دهد.

۲- response formatter

مسئولیت درست و زیبا نمایش دادن جواب‌ها و اطلاعات مساله را دارد.

حال به سراغ گزارش سؤال‌ها می‌رویم :

تمرین اول : وزیر ها

در این مساله حالت‌ها را به صورت یک آرایه هشت تایی فرض کرده‌ایم به این صورت که هر درایه مکان یک وزیر در ستون ۱ ام را نشان میدهد. حالت اولیه به صورت $\langle ۷,۳,۰,۲,۵,۱,۴,۰ \rangle$ میباشد. هر حرکت به این صورت است که میتوان یک وزیر را در ستونش یک خانه به بالا و یا یک خانه به پایین آورد و تابع next state بر همین اساس لیستی از حالت‌های بعدی را به ما تحویل میدهد. تابع هدف هم چک میکند تا هیچ وزیری همدیگر را تهدید نکنند.

توضیح تابع شهودی

در این تمرین تابع شهودی تعداد سطر هایی است که هیچ وزیری در آن حضور ندارد . این تابع به وضوح برای حالت هدف برابر صفر است و همچنین قابل قبول نیز هست زیرا برای هر حالت از کمترین میزان هزینه رسیدن به هدف ، کمتر میباشد. در نتیجه اگر الگوریتم را به صورت درختی اجرا کنیم نتیجه بدست آمده بهینه خواهد بود.

خروجی ها

در بار اول مساله را با الگوریتم سطح اول به صورت درختی اجرا میکنیم و خروجی زیر مشاهده می شود:

```
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 30208
number of visited nodes : 407006
number of max used memory : 376798
-----
Cost of goal : 5

Path :
[7 ,3 ,0 ,2 ,5 ,1 ,5 ,0]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,0]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,1]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,2]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,3]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,4]

Process finished with exit code 0
```

4: Run 6: TODO 2: Version Control Terminal Messages Event Log
Compilation completed successfully in 1s 22ms (moments ago) 20:1 LF UTF-8 Git: master

سپس با الگوریتم Depth limited serach با عمق ۸ آن را اجرا کردیم که خروجی زیر حادث شد :

```
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 179
number of visited nodes : 270
number of max used memory : 96
-----
Cost of goal : 8

Path :
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,0]
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,1]
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,2]
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,3]
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,4]
[7 ,3 ,0 ,2 ,5 ,1 ,4 ,5]
[7 ,3 ,0 ,2 ,5 ,1 ,5 ,5]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,5]
[7 ,3 ,0 ,2 ,5 ,1 ,6 ,4]

Process finished with exit code 0
```

4: Run 6: TODO 2: Version Control Terminal Messages Event Log
Compilation completed successfully in 1s 43ms (moments ago) 23:1 LF UTF-8 Git: master

و در آخر هم با الگوریتم Astar که خروجی زیر را گرفتیم :



- دقت کنید که در الگوریتم اول به دلیل زمان بر بودن زیاد حالت اولیه ساده تر در نظر گرفته شده ولی با این حال میزان گره های بسط داده شده خیلی بیشتر از دو حالت بعدی ست . هم چنین در دو حالت بعدی مشاهده میکنید که الگوریتم ASTAR به صورت بهینه عمل کرده و راه کوتاهتری را انتخاب نموده است هرچند که گره های بیشتر را نسبت به حالت دوم بسط داده است .

تمرین دوم : مسیریابی روبات

در این مساله حالت را به صورت یک عدد صحیح در نظر گرفتیم به این صورت که خانه های جدول شماره گذاری شده اند و اولین خانه شماره ۰ و آخرین خانه (در حالت ۵*۵) برابر ۲۴ است پس هر حالت بیانگر حضور ربات در یکی از خانه های جدول میباشد.

حال به سراغ خروجی ها می رویم :

در ابتدا جست و جوی هزینه یکنواخت را اجرا میکنیم :

```
AI_1 - [~/Desktop/Desktop/projects/JAVA/AI_1] - [AI_1] - ~/Desktop/Desktop/projects/JAVA/AI_1/src/algorithms/uninformed_search/UniformCostSearch
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AI_1 src algorithms uninformed_search UniformCostSearch
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 24
number of visited nodes : 24
number of max used memory : 5
-----
Cost of goal : 8

Path :
[0]
[5]
[10]
[15]
[20]
[21]
[22]
[23]
[24]

Process finished with exit code 0
4: Run 6: TODO 9: Version Control Terminal 0: Messages Event Log
Compilation completed successfully in 1s 192ms (moments ago) 23:1 LF: UTF-8 Git: master
```

سپس عمق اول را اجرا میکنیم :

```
AI_1 - [~/Desktop/Desktop/projects/JAVA/AI_1] - [AI_1] - ~/Desktop/Desktop/projects/JAVA/AI_1/src/Main.java - IntelliJ IDEA 2016.2.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AI_1 src Main
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 18
number of visited nodes : 23
number of max used memory : 10
-----
Cost of goal : 10

Path :
[0]
[1]
[2]
[3]
[4]
[9]
[14]
[13]
[18]
[23]
[24]

Process finished with exit code 0
4: Run 6: TODO 9: Version Control Terminal 0: Messages
Compilation completed successfully in 1s 91ms (moments ago) 25:1 LF+ UTF-8 Git: master
```

و حال جست و جوی دو طرفه را اجرا مینماییم :

همانطور که میبیند در گره [۱۶] دو طرف به یکدیگر پیوسته اند :

```
AI_1 - [~/Desktop/Desktop/projects/JAVA/AI_1] - [AI_1] - ~/Desktop/Desktop/projects/JAVA/AI_1/src/Main.java - IntelliJ IDEA 2016.2.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 18
number of visited nodes : 23
number of max used memory : 5
-----
Cost of goal : 8

Path :
[0]
[5]
[10]
[15]
[16]
[16]
[17]
[18]
[19]
[24]

Process finished with exit code 0
4: Run 6: TODO 9: Version Control Terminal 0: Messages
Compilation completed successfully in 1s 23ms (moments ago) 24:1 LF UTF-8 Git: master
```

و در نهایت ASTAR را اجرا مینماییم :

```
AI_1 - [~/Desktop/Desktop/projects/JAVA/AI_1] - [AI_1] - ~/Desktop/Desktop/projects/JAVA/AI_1/src/Main.java - IntelliJ IDEA 2016.2.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 24
number of visited nodes : 24
number of max used memory : 8
-----
Cost of goal : 8

Path :
[0]
[5]
[10]
[15]
[16]
[17]
[18]
[23]
[24]

Process finished with exit code 0
4: Run 6: TODO 9: Version Control Terminal 0: Messages
Compilation completed successfully in 1s 8ms (moments ago) 23:1 LF UTF-8 Git: master
```


توضیح: همانگونه که میبینید مطابق انتظار جست و جوی عمق اول بهینه نیست و مسیری که حداقل در ۸ گام میتوان پیمود را در ۱۰ گام میپیمایداز طرفی نسبت به Astart و uniform cost search گره های کمتری را بسط میدهد. همچنین جست و جوی دوطرفه که از دو طرف به صورت سطح اول جست و جو میکند به صورت گرافی اجرا شده در نتیجه گره زیادی را بسط نمیدهد.

مسیر قابل پیمایش نیز برای الگوریتم های مختلف متفاوت است که در تصاویر قابل مشاهده است