گزارش پروژه اول درس مبانی هوش مصنوعی

الهام رازی-۹۷۳۱۰۱۹

فرموله سازی مساله:

- حالات: حالات مختلف چیدمان کارت ها در بخش های زمین بازی.
- حالت اولیه: هر حالتی میتواند حالت اولیه باشد. حالت اولیه در ورودی برنامه داده میشود.
- اعمال: در هر مرحله، تنها یک کارت از بالای یک دسته از k دسته در زمین بازی برداشته میشود و میتواند بر روی کارتی که مقدارش بیشتر است قرار داده شود. در واقع آرایه ای از دوتایی هایی که شماره خانه مبدا و شماره خانه مقصد هستند را داریم. یعنی مثلا (۱ ، ۲) این معنی را دارد که از خانه دوم زمین بازی، کارت را در خانه ۱ قرار میدهیم. (که البته این عمل در صورتی انجام میشود که مجاز باشد.)
- آزمون هدف: بررسی این که در هر بخش از زمین بازی، کارت ها همرنگ و به صورت نزولی مرتب شده باشند.(لزوما تمام کارت های یک رنگ از 1 تا n در یک بخش قرار نمیگیرند.)
- هزینه مسیر: در حل مساله با الگوریتم *A، هزینه هر عمل، ۱ در نظر گرفته شده است.

*در پیاده سازی هر ۳ بخش، برخی از قسمت ها مشترک است و تفاوت در توابع جستجو است.

هر بخش، یک کلاس Node دارد که در آن حالت گره و والد آن مشخص شده است. تابع هدف نیز در این کلاس تعریف شده است. decks نیز همان k بخش زمین بازی و حالت کارت های آن است که به صورت یک لیست دو بعدی تعریف شده است(حالت)

تابع check نیز برای این است که مرتب بودن هر کدام از بخش های زمین را بررسی میکند و از آن در تابع هدف استفاده شده است.

تابع action نیز برای هر گره، فرزندان را تولید میکند و با استفاده از آرایه‌ی action ها، عملیات مجاز را بر روی گره اعمال کرده و فرزندان آن را به دست می آورد.

سوال اول:

پیاده سازی الگوریتم: در الگوریتم bfs، در هر مرحله، کم عمق ترین گره برای بسط انتخاب میشود؛ یعنی اولین گره موجود در مجموعه مرزی(frontier). برای پیاده سازی آن از یک صف fifo استفاده میکنیم که در پایتون به صورت یک لیست آن را تعریف کرده ایم که در هر مرحله، اولین عنصر از آن را برمیداریم.

در پیاده سازی، گره اولیه به تابع bfs پاس داده میشود. اگر حالت اولیه حالت هدف باشد، جستجو به پایان می‌رسد. در غیر این صورت، گره به لیست frontier اضافه میشود.

تا زمانی که لیست frontier خالی نشده، هر بار سطحی ترین گره را از لیست برداشته، آن را به لیست explored اضافه کرده و فرزندان آن را تولید میکنیم. هرکدام از فرزندان را هنگام تولید، اگر در frontierیا explored نبودند، با تابع هدف چک میکنیم. اگر حالت هدف بود، کار جستجو به پایان میرسد. در غیر این صورت، آن را به انتهای صف frontier اضافه میکنیم. این کار را برای همه action ها انجام می‌شود تا تمام فرزندان گره والد تولید و بررسی شود.

سوال دوم:

برای پیاده سازی این بخش، از تابع dls استفاده شده است. در این الگوریتم، عملیات جستجوی اول عمقی(dfs) تا عمق مشخصی انجام میشود. در الگوریتم ids، این عمق را یکی یکی در صورت پیدا نشدن جواب تا عمق مشخص شده در dls، افزایش میدهد. در این الگوریتم از مزایا dfs و bfs، یعنی بهینه بودن و پیچیدگی فضایی کم بهره میبرد.

dlsرا به صورت بازگشتی پیاده میکنیم؛ شرط بیس آن به این صورت است که اگر به حداکثر عمق رسیدیم و پاسخی پیدا نکردیم، false برمیگردانیم. در غیر این صورت، نود های یک عمق بیشتر را جستجو میکنیم.

در ids، dls را برای عمق های d تا بینهایت که d از کاربر گرفته میشود تکرار میکنیم. در هر کدام از عمق هایی که به حالت هدف برسیم، عملیات جستجو را به پایان میرسانیم.

سوال سوم:

در این قسمت، از الگوریتم A* استفاده میکنیم که بر خلاف دو الگوریتم پیشین، به صورت آگاهانه انجام شده و برای آن هیوریستیک تعریف میکنیم. در کلاس Node این بخش، فیلد heuristic و هزینه مسیر هم اضافه شده است که هزینه مسیر برای هر Node، هزینه مسیر Node والد به علاوه‌ی ۱ است.

برای حل این مساله، از هیوریستیک های مختلفی استفاده شد؛ مانند شمردن تعداد inversion ها، تعداد رنگ ها در هر بخش زمین، و .. . در نهایت، برای محاسبه هیوریستیک، در یک بخش، اگر کارتی مقدارش از بعدی ها کمتر بود یا رنگ آن با اولین کارت متفاوت بود یا مقدارش به صورت پشت سر هم نبود، مقدار فاصله اش تا بالای دسته کارت را به هیوریستیک اضافه میکنیم؛ زیرا متوجه میشویم که این کارت در جای خود قرار نگرفته(حداقل نه در این حالت) و حداقل باید کارت های رویی آن برداشته شوند. بنابراین این هیوریستیک قابل قبول هم خواهد بود، چون ما حداقل تعداد حرکات را در نظر گرفتیم و در عمل تعداد اعمال بیشتری نیاز است.

برای پیاده سازی، از ساختمان داده‌ی دیکشنری پایتون برای frontier استفاده کردیم. مقدار key، میزان تابع ارزیاب است و value، گره مربوطه به آن است. یک لیست هم برای نگهداری حالت های جستجو شده داریم؛ چرا که جستجوی ما به صورت گرافی خواهد بود.

در هر مرحله(تا زمانی که frontier خالی نشده)، node ای را برای بسط انتخاب میکنیم که کمترین میزان f(n) را دارد. با کمک تابع action، فرزندان آن را تولید میکنیم. اگر فرزند تولید شده، حالت هدف بود، عملیات جستجو به پایان میرسد. در غیر این صورت، اگر حالت آن در لیست explored نبود، آن را به همراه جمع میزان هیوریستیک و هزینه مسیر تا به اینجا(که هزینه مسیر گره والد به علاوه یک است) را در دیکشنری frontier اضافه میکنیم.

حال برای یکی از تست کیس های داده شده به هر سه قسمت، نتایج را مقایسه میکنیم:

ورودی هر سه:

5 3 5

5a 4a 1a 3a 2a

1b

1c

5c 4c 3c

5b 4b 3b 2b 2c

برای هرکدام، پاسخی که به دست آمده، تعداد گره های تولید شده، عمق پاسخ و گره های بسط داده شده را بررسی میکنیم.(عمق اولیه برای ids را ۵ وارد کرده ایم)

	حالت پیدا شده	گره های بسط داده شده	گره های ایجاد شده	عمق
BFS	5a 4a 2a 1a 3a 5c 4c 3c 2c 1c 5b 4b 3b 2b 1b	1156	2704	5
IDS	5a 4a 2a 1a 3a 5c 4c 3c 2c 1c 5b 4b 3b 2b 1b	84605	1691946	6
A*	5a 4a 3a 2a 1a 5c 4c 3c 2c 1c 5b 4b 3b 2b 1b	6	38	6

در A* چون سعی بر این است که در هر مرحله بهترین انتخاب را داشته باشیم و به قولی هوشمند عمل کنیم، عملکرد بهتری داشته. جستجوی IDS هم بر خلاف دو مورد دیگر، به صورت درختی است، رکوردی از گره هایی که قبلا تولید شده نگهداری نمیکند، به همین جهت گره های تکراری را چندین بار تولید میکند و بسط میدهد.(هرچند که همین امر به کاهش پیچیدگی فضایی آن کمک میکند اما از لحاظ زمانی هزینه بر است)

البته باید به خاطر داشته باشیم که قابل قبول بودن هیوریستیک ارائه شده، در جستجوی گرافی A* لزوما مسیر بهینه را بر نمیگرداند و در این مثال عمقی که برگردانده با دوتای دیگر برابر شده است.

تست کیس دیگری را هم بررسی میکنیم(در این جا عمق اولیه برای ids، مقدار ۱۱ در نظر گرفته شده)

4 3 3

1a 2a 3b

3a

1c 1b 3c

2b 2c

	حالت پیدا شده	گره های بسط داده شده	گره های ایجاد شده	عمق
BFS	1a 3a 2a 3b 2b 1b 3c 2c 1c	707	946	11
IDS	1a 3a 2a 3b 2b 1b 3c 2c 1c	1125566	13506656	11
A*	1a 3b 2b 1b 3a 2a 3c 2c 1c	106	317	53

همانطور که در جدول بالا مشاهده میشود، دو الگوریتم bfs و ids یک حالت را اعلام کرده اند. این دو، پاسخ را در یک عمق پیدا کرده اند و هر دو بهینه هستند اما A* با وجود این که سریعتر به پاسخ میرسد، لزوما بهینه ترین مسیر را نمی یابد. البته با بهبود هیوریستیک و یا جستجوی درختی، بهینه(تر) خواهد شد.