پروژه شماره ۲

حل مساله گذر از رودخانه
با استفاده از روش جستجوی عمق محدود

استاد: جناب آقای دکتر فیضی درخشی

دانشجو: پارسا یوسفی نژاد محمدی

شماره دانشجویی: ۱۴۰۰۵۳۶۱۱۰۴۸

مقدمه

مساله گذر از رودخانه یکی از مسائل قدیمی و کلاسیک دنیای هوش مصنوعی است، این مسئله از نمونه‌های مسائل ارضای محدودیت یا همان CSPها است. علاوه بر آن، این مساله یک تست هوش و معمای فکری جذاب و سرگرم کننده هم می‌تواند باشد. این معما اصالت چینی دارد و همچنین تستی برای محک زدن iQ به شمار می‌رود.

هدف پروژه

در این مسئله پلیس و دزد و پدر و مادر و ۲ دختر و ۲ پسر در یک سمت ساحل رودخانه قرار گرفته‌اند، هدف ما این است که ضمن ارضای برخی قوانین و محدودیت‌ها، تمامی افراد را به سمت دیگر ساحل تنها به کمک یک قایق با ظرفیت حداکثر دو نفر ببریم. برای جستجوی روش و پیمایش صحیح انجام این عمل در دنیای هوش مصنوعی، نیاز داریم که مساله خود را به مساله جستجوی درخت فضای حالات، مدل سازی کنیم. بدین منظور در ابتدا باید فضای حالات و عملگرها و حالت ابتدایی و نهایی را تعریف و به اصطلاح مدل کنیم، که این کار در تمرین اول صورت گرفته و در سامانه آموزشی قرار داده شده است.

قوانین عبور از رودخانه

- ظرفیت قایق حداکثر ۲ نفر می‌باشد.
- تنها افرادی که می‌توانند قایق را برانند، پدر، مادر و پلیس می‌باشند.
- پدر نباید با هر یک از دختران بدون حضور مادر، تنها بماند.
- مادر نباید با هر یک از پسران بدون حضور پدر، تنها بماند.
- دزد نباید با هیچ یک از اعضای خانواده بدون حضور پلیس باقی بماند

شرح پروژه

همانطور که می‌دانیم، حل این معما در هوش مصنوعی به روش جستجوی درخت فضای حالات میسر می‌باشد. پروژه را به زبان پایتون کدنویسی کرده‌ایم که شامل توابع و بخش‌های گوناگونی می‌باشد.

بخش اصلی این پروژه تابع جستجوی فضای حالات به روش جستجوی عمق محدود یا همان DLS است. از حالت ابتدایی که حالتی است که تمامی افراد در سمت چپ رودخانه قرار دارند شروع به تولید فرزندان و پیمایش عمقی نودهای تولید شده میکنیم و تا عمق محدودی که مدنظر سوال است حرکت می‌کنیم و درصورت پیدا نشدن جواب در آن عمق، دیگر فرزندان نود آن عمق را تولید نمی‌کنیم و تنها آن را از پشته‌مان که با کمک لیست ساخته شده است، حذف میکنیم. سپس همین روند را تا زمانی که حالتی برای تست هدف کردن وجود داشته باشد انجام می‌دهیم و در انتها لیست حالات پیمایش شده را بر چاپ کردن بر میگردانیم. شیوه و نحوه پیاده سازی صحیح توابع اصلی همچون: بررسی اعتبار حالت، تولید فرزندان یک حالت و جستجوی عمق محدود، از اهداف اصلی این پروژه می‌باشند.

همچنین برای راحتی کار و افزایش خوانایی توابع، بجای استفاده از اندیس‌ها از اسامی در ورودی ایندکس آرایه حالتمان استفاده کرده‌ایم، بدین صورت که در ابتدا به ترتیب به اندیس و مکان قرارگیری آن فرد در آرایه متغیری را که همان اسم آن کاراکتر است را اختصاص داده‌ایم تا از پس بجای استفاده از اعداد، از نام آن افراد جهت دسترسی به خانه‌ای که در آن قرار دارند استفاده کنیم.

نحوه مدل سازی مساله

در این مساله، هر یک از حالات، چیدمان‌ها مختلف افراد در دو ساحل کنار رودخانه می‌باشند. یک آرایه ۹ تایی که شامل تمام افراد و همچنین مکان قرارگیری قایق می‌باشد، ایجاد میکنیم. خانه‌ها با ایندکس‌ها ۰ تا ۷ را به ترتیب به پلیس، دزد، پدر، مادر، دختر اول، دختر دوم، پسر اول و پسر دوم اختصاص می‌دهیم و خانه شماره آخر را برای جهت قرارگیری قایق تخصیص می‌دهیم.

در داخل هر یک از خانه های این آرایه عدد صفر یا یک قرار می‌گیرد که نماینده جهت قرار گرفتن آن عنصر در سمت چپ (صفر) و یا سمت راست (یک) می‌باشد.

هریک از حالات مختلف این آرایه، حالت جدیدی برای معما بوده که می‌تواند معتبر و یا نامعتبر باشد. برای مساله فوق، می‌توان دسته عملگرهای یکتایی و یا دوتایی داشته باشیم، با انجام این عملگر که به درون تابع GenerateAllValidStates پیاده سازی شده است، قادر به تولید حالات جدید هستیم. ورودی این برنامه، تنها یک حالت ابتدایی که به صورت پیش‌فرض، حالت همه افراد در چپ (آرایه ۹ تایی از صفر) می‌باشد است و خروجی برنامه، برگرداندن پیمایش صحیح جستجو از حالت شروع تا حالت پایانی که همان هدف ما است، می‌باشد.

تشریح ساختار کد پروژه

برای پیاده سازی این برنامه، نیاز به توابع و ساختمان داده‌های متنوع و گوناگونی داریم، که به اختصار هر کدام را در این بخش تشریح میکنیم.

تابع Show() : ورودی این تابع، یک حالت از مساله به صورت آرایه ۹ تایی می‌باشد و خروجی آن نمایش دادن آن حالت به صورت گرافیکی در ترمینال برنامه است.

تابع ShowPath() : وظیفه این تابع، نمایش دادن ترتیبی و با اندکی تاخیر زمانی هر یک از حالات می‌باشد، این تابع یک لیست از حالت های مختلف را دریافت و ضمن فراخوانی تابع TellMove() به صورت انیمیشن‌وار حالات و توضیحی درباره تغییر حالت را به نمایش می‌گذارد.

تابع Clear(): این تابع وظیفه پاک کردن محتوای خروجی ترمینال برنامه را بر عهده دارد.

تابع IsValid(): این تابع بررسی میکند که آیا حالت ورودی‌اش، حالتی معتبر و سازگار با قوانین مساله می‌باشد یا خیر و در انتها جواب True و یا False را به عنوان نتیجه برمی‌گرداند.
درون این تابع در ابتدا سعی کردیم که تداخل‌های دختران را بررسی کنیم و بعد از آن تداخل‌های پسران و در نهایت تداخل‌های ممکنی که با سایر افراد و دزد امکان اتفاق افتادن دارد را بررسی کرده‌ایم

تابع IsGoal() : بررسی می‌کند که آیا حالت ورودی داده شده، حالت نهایی و هدف (سمت راست بودن تمامی افراد = ۱ بودن ۹ درایه از آرایه) می‌باشد یا خیر؟.

تابع GenerateAllValidStates() : یکی از توابع اصلی برنامه بوده و کار آن تولید تمامی فرزندان معتبر یک حالت ورودی می‌باشد و در آخر تمام حالات پدر را در غالب یک لیست از حالات بر می‌گرداند. این تابع در هر مرحله از تولید فرزندی، معتبر بودن آن حالت و تکراری نبودن آن را هم بررسی میکند و بعد از آن در صورت ارضای محدودیت‌های مسئله، آن را به لیست نهایی اضافه می‌کند.

تابع TellMove() : این تابع با گرفتن دو حالت جدید و قبلی، سعی در توصیف تغییر موقعیت افراد میکند و متنی مبنی بر اینکه کدام اشخاص به کدام جهت ساحل رفته‌اند، برمی‌گرداند. این تابع بدلیل اینکه در ابتدا در مدل سازی مسئله جهت قرارگیری هر کاراکتر را با عدد صفر و یک تعیین

کردیم، میتواند تنها با کم کردن متناظر هر خانه از دو حالت جدید و قدیم پیشرو، به این مورد پی ببرد که کدام کاراکتر به کدام مکان از ساحل رفته است و سپس اقدام به تولید متنی مبنی بر همین تغییر حالت افراد و جهت قرارگیری قایق بکند.

پیاده سازی تابع DLS به دو شیوه بازگشتی نیز در پیوست موجود می‌باشد.

تابع Iterative_DLS() : تابع اصلی برنامه می‌باشد و وظیفه اصلی برنامه را که جستجوی فضای حالات است را بر عهده دارد. شیوه کار این تابع، انجام جستجوی DLS به روش غیربازگشتی یا همان Iterative بوده و ورودی آن، حالت آغازین و عدد ثابت محدودیت عمق و یک پرچم جهت تعیین نوع خروجی که یا تمام حالات های پیمایش شده در این جستجو باشد و یا اینکه تنها حالت‌هایی را برگرداند که ما را مستقیم از مقصد به هدف می‌رسانند.

این الگوریتم دقیقا شیوه کار کلی الگوریتم جستجوی درخت فضای حالات را که خوانده‌ایم را دارد، با این تفاوت که محدودیت عمق در DFS به آن اضافه شده است و از سربار حافظه و پردازنده جلوگیری می‌کند. متغیر ثابت محدودیت عمق، به صورت پیشفرض ۲۰ قرار داده شده است.

همچنین بدلیل اینکه نیاز داریم که پیمایشی را که انجام داده‌ایم را به عنوان خروجی برگردانیم، از جفت تاپل‌های عمق و حالت برای هر نود از درختمان استفاده کرده‌ایم تا بتوانیم به والد هر نود دسترسی داشته باشیم و در آخر بتوانیم پیمایش صحیح را برگردانیم.

تابع FilterFinalAnswerStates(): این تابع زمانی فراخوانی می‌شود که ما قصد داریم که خروجی تابع Iterative_DLS()، تنها پیمایش‌هایی باشند که مستقیما ما را از حالت شروع به حالت پایانی، میرسانند، زمانی این تابع فراخوانی میشود که پرچم مربوط به آن در تابع DLS، False شده باشد، در غیر این صورت از این تابع استفاده نخواهد شد.

این تابع برای ورودی لیستی از تاپل‌های عمق و حالت را دریافت میکند و در آخر با اعمال شرایطی، لیست فوق را فیلتر میکند و لیستی از حالات پاسخ درست را بر می‌گرداند.

جمع‌بندی پروژه دوم

در ابتدای پروژه به معرفی و بررسی هدف این پروژه پرداختیم، سپس مسئله را به صورت دقیق تشریح کردیم و قوانین معمار را مطرح کردیم، در ادامه به تشریح پروژه و شیوه پیاده سازی آن در درس هوش مصنوعی پرداختیم و نحوه مدل سازی مساله به درون دنیای گراف را بیان کردیم و پس از آن به بررسی اجمالی هر یک از توابع کد پروژه پرداختیم و آن ها را به تفصیل مورد بررسی قرار دادیم و شیوه کار هر یک را بیان کردیم.

- در بخش اول این پروژه توانستیم که حالتی دلخواه را فارغ از هرگونه محدودیتی، از ورودی دریافت و آن را به صورت گرافیکی در خروجی ترمینال نمایش بدهیم.
- در بخش دوم توابع جدیدی من جمله **ShowPath()** و **IsGoal()** و **IsValid()** و همچنین **GenerateAllValidStates()** را پیاده سازی کردیم.
- در بخش سوم، تابع جستجوی فضای حالت به روش **DLS** را به صورت بازگشتی پیاده سازی نمودیم.
- در بخش چهارم، در نهایت تابع جستجوی **DLS** را به روش غیر بازگشتی و **Iterative** پیاده‌سازی کردیم.

درنهایت توانستیم که برنامه‌ای کامل جهت جستجو برای یافتن پاسخ مساله گذر از رودخانه با روش جستجوی عمق محدود ارائه دهیم، که این پاسخ در عمق ۱۷ (۱۷ پیمایش) پیدا شد.