جزوه کامل هوش مصنوعی - جلسه چهارم

مرور و تکمیل فصل دوم: عامل‌ها و محیط‌های کاری

در ابتدای جلسه، پیش از ورود به مباحث جدید، مروری بر انواع عامل‌ها (Agents) و نحوه عملکرد آن‌ها در محیط کار صورت گرفت.

> **یادآوری: عامل واکنشی ساده (Simple Reflex Agent)**
>
> این عامل‌ها بر اساس یک جدول کامل از قوانین شرطی (Condition-Action Rules) عمل می‌کنند. برنامه‌نویس تمامی حالت‌های ممکن را در حافظه عامل تعبیه می‌کند. عامل محیط را حس (Sense) کرده، وضعیت فعلی را با جدول تطبیق می‌دهد و اقدام (Action) متناظر را اجرا می‌کند.

محدودیت‌های عامل واکنشی ساده:
این مدل با وجود سادگی، برای محیط‌های پیچیده و تصادفی (Stochastic) مانند رانندگی تاکسی به هیچ وجه مناسب نیست. دلیل این امر این است که محیط دارای بخش‌های غیرقابل مشاهده است و ما حافظه (Memory) نامحدودی نداریم که بتوانیم تمام حالت‌های ممکن در جهان را پیش‌بینی کرده و در یک عامل تعبیه کنیم. بنابراین، مدیریت چنین محیط‌هایی با این عامل امکان‌پذیر نیست.

۱. عامل‌های مبتنی بر مدل (Model-based Agents)

برای رفع محدودیت‌های عامل‌های ساده، عامل‌های مبتنی بر مدل معرفی می‌شوند. این عامل‌ها دارای حافظه هستند و وضعیت فعلی را با وضعیت‌های قبلی ترکیب می‌کنند.

- **عملکرد:** عامل درک می‌کند که جهان اطرافش چگونه در حال تکامل و تغییر است ($How\ the\ world\ evolves$).
- **ترکیب اطلاعات:** عامل وضعیت جدید (State) را با قوانین شرطی و همچنین اطلاعات ذخیره شده در حافظه خود ترکیب کرده و سپس تصمیم‌گیری می‌کند.

> **مثال کاربردی: راننده تاکسی**
>
> در تایم‌استپ (Time Step) قبلی، راننده ماشینی را در پشت سر خود حس کرده است. در استپ فعلی، آن ماشین دیگر پشت سرش نیست. عامل با مقایسه این دو وضعیت، تغییرات محیط را درک کرده و متوجه می‌شود که برای اقدام بعدی باید این تحولات را مد نظر قرار دهد.

۲. عامل‌های مبتنی بر هدف (Goal-based Agents)

در بسیاری از مواقع، عامل در وضعیت فعلی قابلیت انجام چندین اقدام مختلف را دارد (مانند استفاده از دست‌ها، پاها یا سنسورها برای حرکت در جهات مختلف).

- **نقش هدف (Goal):** هدف مشخص می‌کند که از میان تمام اقدامات ممکن، کدام اقدام باید انتخاب شود.
- **مثال:** اگر هدف رسیدن به دانشکده داروسازی باشد، عامل مسیر سمت راست را انتخاب می‌کند و اگر هدف دانشکده فیزیک باشد، مسیر شمال را انتخاب می‌کند. هدف، فیلتر نهایی برای انتخاب اقدام است.

۳. عامل‌های مبتنی بر سودمندی (Utility-based Agents)

گاهی اوقات چندین اقدام مختلف ما را به هدف می‌رسانند. در این حالت، عامل نیازمند یک معیار ارزیابی (سبک‌سنگین کردن) است تا بهترین مسیر را انتخاب کند.

- **مفهوم سودمندی:** چه چیزی عامل (یا کاربر) را خوشحال‌تر می‌کند؟ آیا رسیدنِ سریع‌تر مهم است؟ کم‌هزینه‌تر بودن؟ یا انتخاب یک مسیر مفرح و نشاط‌آور؟
- **عملکرد:** عامل اقداماتی را انتخاب می‌کند که میزان سودمندی (Utility) را بر اساس این معیارها بیشینه (Maximize) کند.

عامل‌های یادگیرنده (Learning Agents)

این مدل یک عامل کاملاً مستقل و مجزا نیست، بلکه توانمندی «یادگیری» می‌تواند به مدل‌های قبلی (Reflex, Goal-based, Utility-based) اضافه شود. اگر اطلاعات تعبیه شده توسط طراح کافی نباشد، عامل با مانیتور کردن و جستجو (Exploration) در محیط، چیزهای جدیدی یاد می‌گیرد و اقدامات جدیدی را به دایره توانمندی‌های خود می‌افزاید.

اجزای یک عامل یادگیرنده

یک عامل یادگیرنده از چهار مؤلفه اصلی تشکیل شده است:
۱. **عنصر یادگیری (Learning Element):** وظیفه به‌روزرسانی دانش عامل را بر عهده دارد. این بخش قوانین جدید (Condition-Action Rules) را می‌سازد و به پایگاه دانش عامل اضافه می‌کند.
۲. **منتقد (Critic):** این مؤلفه معمولاً بر اساس بازخوردهای بیرونی (یا معیارهای عملکردی درونی) کار می‌کند. منتقد به عامل می‌گوید که عملکردش چقدر خوب یا بد بوده است و او را تشویق یا تنبیه می‌کند (ارزیابی بر اساس Performance Measure).
۳. **عنصر عملکرد (Performance Element):** همان بخش اجرایی است که پیشتر با آن آشنا شدیم و بر اساس وضعیت فعلی، اقدام را انتخاب می‌کند.
۴. **تولیدکننده مسئله (Problem Generator):** این بخش وظیفه دارد در مواجهه با هر وضعیتی کشف کند که چه مسئله جدیدی برای حل کردن وجود دارد.

نکته مهم درباره Problem Generator

همان‌طور که در انسان‌ها، توانایی «کشف مسئله» نشانه‌ای از هوش بالا و خلاقیت است (برخی سریع می‌خوانند، اما برخی مسئله‌های جدید کشف می‌کنند)، در عامل‌های هوشمند نیز این مؤلفه به آن‌ها اجازه می‌دهد تا برای یادگیری بیشتر، خود را درگیر چالش‌ها و اقدامات جدید کنند.

نمایش دانش و وضعیت (Knowledge Representation)

برای اینکه عامل بتواند وضعیت محیط (State) را درک کند و آن را در حافظه خود پیاده‌سازی (Implement) نماید، نیازمند یک ساختار برای نمایش دانش است. این نمایش از ساده تا پیچیده دسته‌بندی می‌شود:

1. **نمایش اتمیک (Atomic Representation):** ساده‌ترین حالت است. عامل درک می‌کند که یک وضعیت کلی یکپارچه وجود دارد و مستقیماً از وضعیت B به وضعیت C می‌رود.
2. **نمایش فاکتور شده (Factored Representation):** وضعیت به متغیرهای مجزا تجزیه می‌شود. مثلاً در یک اتاق، متغیرهای بولین یا مقداری داریم: وضعیت چراغ (روشن/خاموش)، وجود آفتاب، میزان دما (مثلاً ۲۵ درجه). تغییر از یک وضعیت به وضعیت دیگر با تغییر این متغیرها رخ می‌دهد.
3. **نمایش ساختاریافته (Structured Representation):** شبیه به برنامه‌نویسی شیءگرا (Object-Oriented) است. وضعیت شامل اشیاء (Objects) و روابط بین آن‌هاست. با تغییر وضعیت، نه تنها ویژگی‌های یک شیء (مثلاً رنگ آن از سبز به سفید)، بلکه روابط بین اشیاء نیز تغییر می‌کند.

مدل‌های معماری پیشرفته عامل‌ها

برای طراحی عامل‌های پیچیده، نیاز به مدل‌های ذهنی پیشرفته‌تری داریم. در اینجا با معماری‌های لایه‌ای و مدل‌های روان‌شناختی عامل‌ها آشنا می‌شویم.

مدل BDI (Belief, Desire, Intention)

این مدل یکی از مدل‌های پایه در سیستم‌های چندعاملی (Multi-Agent Systems) و استدلال منطقی است.

- **باور (Belief):** شناخت و درک عامل از وضعیت محیط و جهان پیرامون.
- **تمایل/خواسته (Desire):** اهدافی که عامل دوست دارد به آن‌ها برسد.
- **قصد (Intention):** تصمیمی قطعی که عامل در لحظه برای انجام یک اقدام می‌گیرد.

پویایی در مدل BDI

باورهای عامل با سنس کردن محیط دائماً بازبینی (Revise) می‌شوند. ممکن است دو عامل باورهای یکسانی داشته باشند اما تمایلات متفاوتی نشان دهند؛ و یا حتی با تمایلات یکسان، بر اساس شرایط، قصدهای متفاوتی برای اقدام اتخاذ کنند.

معماری‌های لایه‌ای (Layered Architectures) و اولویت‌بندی نیازها

عامل‌ها مانند انسان‌ها دارای یک هرم اولویت‌بندی نیازها هستند (شبیه هرم مزلو). این اولویت‌ها رفتار نهایی عامل را تعیین می‌کنند.

مثال تعارض وظایف و اولویت‌بندی

فرض کنید یک عامل وظیفه انجام پژوهش، مدیریت اداری و کنترل ایمنی آتش‌سوزی یک کارخانه را بر عهده دارد. اگر در حین پژوهش، کارخانه آتش بگیرد، عامل نمی‌تواند بگوید «من مشغول پژوهشم!». در بالاترین اولویت، حفاظت از جان و فرار از خطر یا اطفاء حریق قرار دارد. تا زمانی که این نیاز (پایین‌ترین لایه هرم ولی با بالاترین اولویت بقا) رفع نشود، پرداختن به لایه‌های بالاتر (پژوهش) معنایی ندارد.

انواع معماری‌های لایه‌ای:

- **لایه واکنشی (Reactive Layer):** بالاترین اولویت و کمترین زمان پردازش را دارد (اقدام سریع در برابر خطر).
- **لایه برنامه‌ریزی (Planning Layer):** اولویت متوسط، برای تصمیم‌گیری‌های میان‌مدت.
- **لایه مدل‌سازی (Modeling Layer):** در پایین‌ترین سطح اولویت قرار دارد و برای یادگیری، درک تکامل جهان ($How\ the\ world\ evolves$) و تولید مسئله (Problem Generation) استفاده می‌شود.

لایه های ذکر شده می توانند به صورت **افقی** (Horizontal - تعامل مستقیم تمام لایه‌ها با محیط) یا **عمودی** (Vertical - عبور اطلاعات به صورت فیلتر شده از یک لایه به لایه دیگر به شکل One-pass یا Two-pass) پیاده‌سازی شوند.

✎ پرسش و پاسخ کلاس: جایگاه ماشین‌ها (Automata) در این مدل‌ها

در پاسخ به سوال دانشجویان مبنی بر اینکه «اتوماتا کل مدل عامل است یا بخشی از آن؟»، استاد توضیح دادند که اتوماتا (و ماشین تورینگ، شبکه‌های پتری، یا مدل‌های مارکوف) در واقع روش‌های **پیاده‌سازی (Implementation)** و طراحی عملکرد هر کدام از این لایه‌ها هستند. وضعیتی (State) که در اتوماتا تعریف می‌شود، وضعیت کل جهان نیست، بلکه وضعیت انتزاعی مختص به همان لایه‌ای است که در حال پردازش اطلاعات است.

ورود به فصل سوم: جستجو (Search) و حل مسئله

بخش عمده‌ای از هوش مصنوعی حول محور **جستجو (Search)** می‌چرخد. ما موقعیت‌ها را به شکل «مسئله» (Problem) تعریف می‌کنیم و حل آن مسئله نیازمند یافتن یک راه‌حل از طریق جستجو است.

🕮 مفهوم برنامه‌ریزی پیشاپیش (Plan Ahead)

در مسائل جستجو، عامل قبل از اینکه هیچ اقدام فیزیکی انجام دهد یا وضعیت آینده را ببیند، به صورت آفلاین یک دنباله از اقدامات (Action Sequence) را برنامه‌ریزی می‌کند. پس از یافتن نقشه راه (مثلاً مسیر تهران تا شهر X)، عامل به مرحله اجرا (Run) می‌رود. راه‌حل در واقع همان نقشه خروجی است.

انواع جستجو (مقدمه)

در این فصل با دو دسته کلی از الگوریتم‌های جستجو روبرو هستیم:

1. **جستجوی ناآگاهانه (Uninformed Search):** عامل هیچ اطلاعاتی از مسیر پیش رو یا فاصله تا هدف ندارد. صرفاً مدل‌های مختلف را امتحان می‌کند. مثال‌ها:
   - جستجوی عمق‌اول (DFS)
   - جستجوی سطح‌اول (BFS)
   - جستجوی هزینه یکنواخت ($Uniform\ Cost\ Search$)
2. **جستجوی آگاهانه (Informed Search):** عامل دارای اطلاعاتی (مانند تخمین فاصله تا هدف) است. (مانند الگوریتم A^*).

هدف نهایی در این جستجوها، یافتن **راه‌حل بهینه (Optimal Solution)** است؛ یعنی بهترین راه حلی که عامل را با کمترین هزینه (Least Cost) به هدف برساند.

فرموله‌سازی مسئله جستجو (Problem Formulation)

برای اینکه یک عامل بتواند مسئله‌ای را با جستجو حل کند، باید آن مسئله را به صورت دقیق و ریاضی فرموله کنیم. هر مسئله جستجو دارای ۵ مؤلفه اساسی است:

۱. **فضای وضعیت‌ها (State Space - S):**
شامل تمام وضعیت‌های ممکنی است که عامل می‌تواند در دنیای مسئله در آن‌ها قرار بگیرد (مثلاً تمام شهرهای موجود در نقشه).

۲. **وضعیت اولیه (Initial State - S_0):**
نقطه شروع حرکت عامل کجاست. جستجو همیشه از یک S_0 مشخص آغاز می‌شود.

۳. **اقدامات ممکن (Actions - $A(s)$):**
در هر وضعیت مشخص s، چه اکشن‌هایی در اختیار عامل است؟ (مثلاً در یک شهر ممکن است چهارراه وجود داشته باشد و در شهری دیگر فقط بن‌بست).

۴. **مدل انتقال / مدل گذار (Transition Model):**
تابعی است که نتیجه اعمال یک اقدام در یک وضعیت را نشان می‌دهد. اگر عامل در وضعیت s باشد و اقدام a را انجام دهد، ماحصل آن رسیدن به وضعیت جدید s' خواهد بود:

$$Result(s, a) = s'$$

۵. **آزمون هدف (Goal Test):**
تابعی است که بررسی می‌کند آیا وضعیت فعلی s، همان وضعیت هدف ما هست یا خیر. خروجی این تابع از جنس $True$ یا $False$ است.

۶. **هزینه مسیر (Path Cost / Step Cost):**
برای یافتن راه‌حل بهینه، نیازمند تخصیص هزینه به هر اقدام هستیم. تابع هزینه مشخص می‌کند که حرکت از وضعیت s با اقدام a به وضعیت s' چقدر هزینه دارد:

$$c(s, a, s')$$

✨ فرموله‌سازی در محیط بازی پک‌من (Pacman)

- **فضای وضعیت‌ها (S):** شامل مختصات فعلی پک‌من، جهت نگاه او (Up, Down, Left, Right)، و وضعیت نقطه‌های خورده شده یا باقی‌مانده (مثلاً کیک‌ها و نقطه‌های درون بازی).
- **مدل انتقال:** اگر پک‌من در جهت $North$ حرکت کند، وضعیت بعدی او شامل مختصات جدید، تغییر جهت نگاه (Face = Up) و کاهش یک نقطه از نقشه (در صورت خوردن) است.
- **Goal Test:**
  - *حالت صریح (Explicit):* به عامل مجموعه‌ای از اعداد و مشخصات دقیق داده می‌شود. (مثلاً رسیدن دقیق به وضعیت شماره ۵ یا ۶ که در آن پک‌من رو به بالاست و همه نقاط خورده شده‌اند).
  - *حالت ضمنی (Implicit):* تعریف یک شرط کلی برای هدف. (مثلاً $s\ has\ no\ dots\ left$ - هر وضعیتی که در آن هیچ نقطه‌ای در نقشه باقی نمانده باشد، بدون توجه به جهت نگاه پک‌من).

در این بخش از کلاس، بحث پیرامون اجزای فرموله‌سازی مسئله (به‌ویژه تابع انتقال و هزینه مسیر) با یک پرسش کلاسی به چالش کشیده شد.

✎ پرسش و پاسخ چالشی کلاس (سوال آقای مرادی)

سوال: با توجه به اینکه در یک محیط قطعی (Deterministic)، هر اقدام (a) در یک وضعیت مشخص (s)، ما را دقیقاً به یک وضعیت جدید و یکتای (s') می‌رساند، چرا در فرموله‌سازی مسئله، تابع $Transition\ Model$ و تابع $Path\ Cost$ از یکدیگر تفکیک شده‌اند؟ آیا نمی‌توان هزینه را مستقیماً در خود اکشن یا انتقال لحاظ کرد؟

روند بررسی استاد:
استاد برای پاسخ به این سوال تلاش کردند تا سناریوهای مختلفی را متصور شوند:

- در یک جدول پیاده‌سازی کامل، ما ستون‌هایی برای وضعیت فعلی ($State$)، اقدام ($Action$)، وضعیت بعدی ($Next\ State$) و هزینه ($Cost$) داریم. عامل برای محاسبه هزینه کل راه‌حل ($Solution\ Cost$) نیازمند جمع زدن (سیگما) این هزینه‌هاست.
- آیا ممکن است یک اقدام واحد از یک وضعیت، ما را به دو وضعیت مختلف با هزینه‌های متفاوت ببرد؟ در محیط دترمینیستیک خیر؛ مثلاً اقدام a هزینه ۵ دارد، اقدام b هزینه ۶.
- استاد با تشبیه‌های فیزیکی (مانند وارد کردن فشار بر یک جسم که همزمان باعث تغییر شکل و بالا رفتن دما می‌شود) و معماری کامپیوتر (مدل‌های $SIMD$ و $MIMD$) سعی در توجیه این تفکیک داشتند، اما در نهایت اذعان کردند که به دلیل خستگی، تمرکز کافی برای یافتن یک مثال نقض دقیق در لحظه وجود ندارد.

نتیجه‌گیری موقت: در فرموله‌سازی استاندارد هوش مصنوعی، عامل به هر دو تابع به عنوان اطلاعات ورودی جداگانه نیاز دارد. یکی برای درک اینکه "به کجا می‌روم" ($Result$) و دیگری برای محاسبه "چقدر هزینه دارد" ($Cost$). این تفکیک به عامل اجازه می‌دهد بر اساس استراتژی‌های مختلف (گاهی فقط بر اساس رسیدن به هدف و گاهی بر اساس بهینه‌سازی هزینه) تصمیم‌گیری کند. تفکیک این دو، معماری عامل را از پیاده‌سازی‌های خاص (Implementation) مستقل نگه می‌دارد.

(همچنین در پاسخ به سوال خانم پوشبان تأکید شد که در نمایش گراف ساده، همسایه بودن دو وضعیت به معنای وجود حداقل یک یال (Action) بین آن‌هاست، فارغ از اینکه طول یا هزینه آن یال چقدر باشد).

بررسی یک مسئله نمونه: مسیریابی در رومانی (Romania Touring)

برای درک بهتر فضای جستجو، از مسئله کلاسیک مسیریابی بین شهرهای رومانی استفاده می‌شود. این مسئله نمونه بارزی از **جستجوی ناآگاهانه (Uninformed Search)** است.

شرایط مسئله (دیدگاه عامل):
عامل ما (مانند یک راننده رالی بدون نقشه کلی) هیچ دیدی از کل کشور رومانی ندارد. او تنها زمانی که به یک شهر می‌رسد، می‌تواند شهرهای مجاور (همسایه) و فاصله تا آن‌ها را مشاهده کند. مانند این است که عامل فقط یک چراغ‌قوه دارد که تنها یک قدم جلوتر را روشن می‌کند.

فرموله‌سازی مسئله رومانی:

1. **فضای وضعیت‌ها ($State\ Space$):** شهرهای رومانی. به دلیل نام‌های پیچیده، هر شهر با حرف اولش نمایش داده می‌شود (مثل $A, B, C, \dots$) که یک نمایش (Representation) یکتاست.
2. **وضعیت اولیه ($Initial\ State$):** شهری که عامل سفر را از آنجا آغاز می‌کند (مثلاً شهر A).
3. **اقدامات ($Actions$):** حرکت به سمت یکی از شهرهای همسایه. (مثلاً شهر S ممکن است ۴ همسایه داشته باشد، پس ۴ اکشن در دسترس است).
4. **مدل انتقال ($Transition\ Model$):** انجام اقدام حرکت به سمت شهر همسایه، باعث می‌شود وضعیت عامل به آن شهر تغییر کند.
5. **آزمون هدف ($Goal\ Test$):** از نوع صریح ($Explicit$) است. عامل در هر شهر بررسی می‌کند: "آیا اینجا بخارست (B)

است؟" (خروجی: $True/False$).

6. **هزینه مسیر ($Path\ Cost$)**: مسافت بین شهرها (Distance). این هزینه افزایشی ($Additive$) است و فرض بر این است که هزینه هر گام بزرگتر یا مساوی صفر است ($Step\ Cost \geq 0$).

> ⚠ **اهمیت استراتژی جستجو (مثال الگوریتم حریصانه)**
>
> فرض کنید استراتژی عامل استفاده از روش "حریصانه" ($Greedy$) باشد؛ یعنی در هر شهر، همسایه‌ای را انتخاب کند که کمترین مسافت (هزینه) را تا آنجا دارد.
> استاد با نشان دادن نقشه‌ای از اروپا توضیح دادند که این روش ممکن است عامل را از نروژ به سوئد، سپس آلمان، بلژیک، هلند و انگلیس ببرد و یک سفر ۷ ساعته را به یک مسیر طولانی دور اروپا تبدیل کند!
> **نتیجه:** استراتژی‌های جستجو همواره بهترین و بهینه‌ترین مسیر را تضمین نمی‌کنند. ما در جستجو به دنبال یافتن الگوریتم‌هایی هستیم که ما را با کمترین هزینه به هدف برسانند.

گراف جستجو در برابر درخت جستجو (Graph vs. Tree Search)

یکی از مفاهیم بسیار حیاتی در هوش مصنوعی، تفاوت بین گراف وضعیت‌های فیزیکی و درخت جستجویی است که عامل در ذهن خود (یا در حافظه کامپیوتر) می‌سازد.

فضای مسئله می‌تواند یک گراف کوچک باشد، اما **درخت جستجو (Search Tree)** حاصل از آن می‌تواند بی‌نهایت بزرگ یا دارای عمق بی‌نهایت باشد.

تفاوت وضعیت ($State$) و گره ($Node$)

این دو مفهوم به هیچ وجه یکسان نیستند:

- **وضعیت ($State$)**: یک پیکربندی فیزیکی و واقعی در دنیای مسئله است (مثلاً "بودن در شهر اصفهان").
- **گره ($Node$)**: یک ساختار داده (Data Structure) در پیاده‌سازی الگوریتم جستجو است.

یک وضعیت واحد می‌تواند متناظر با صدها گره مختلف در درخت جستجو باشد.

> ✨ **مثال:**
>
> شما در شهر اصفهان هستید (وضعیت S_1). از اصفهان به تهران می‌روید و دوباره به اصفهان برمی‌گردید.
> اگرچه از نظر فیزیکی دوباره در همان وضعیت اصفهان هستید، اما در درخت جستجو، این اصفهانِ جدید یک **گره جدید** است. زیرا:
> ۱. ممکن است زمان تغییر کرده باشد (گره اول اصفهان در صبح بود، گره دوم اصفهان در عصر است).
> ۲. مسیر رسیدن (Parent Node) متفاوت است.
> ۳. عمق درخت ($Depth$) و هزینه طی شده ($Cost$) برای این گره جدید با گره قبلی تفاوت دارد.

اجزای یک گره ($Node$) در درخت جستجو:
علاوه بر خودِ وضعیت ($State$)، یک گره شامل اطلاعات زیر است:

- گره والد ($Parent$)
- اقدامی که باعث رسیدن به این گره شده ($Action$)
- هزینه مسیر تا این نقطه ($Path\ Cost$ که معمولاً با g نشان داده می‌شود)
- عمق گره در درخت ($Depth$)

تولید حلقه بی‌نهایت

در یک گراف جستجو، ما هر وضعیت را یک بار می‌بینیم. اما در درخت جستجو، اگر عامل اجازه داشته باشد وضعیت‌های تکراری را دوباره ویزیت کند (عدم به خاطرسپاری وضعیت‌های دیده شده)، یک گراف کوچک می‌تواند به یک درخت با عمق بی‌نهایت تبدیل شود (گرفتار شدن در حلقه رفت و برگشت بین دو شهر).

مسئله جورچین هشت‌تایی (Puzzle-8) و تابع بسط (Expand)

یکی دیگر از مسائل کلاسیک جستجو، پازل ۸تایی است.

- **وضعیت‌ها:** چیدمان کاشی‌ها (Tiles) در صفحه. (هر چیدمان متفاوت، یک وضعیت مستقل است).
- **اقدامات:** به جای حرکت دادن کاشی‌ها، در هوش مصنوعی اقدام را "حرکت دادن خانه خالی" (مربع خالی) به سمت چپ، راست، بالا یا پایین تعریف می‌کنیم.
- **آزمون هدف:** رسیدن به یک چیدمان مرتب و مشخص (مثلاً خانه خالی در گوشه پایین سمت راست قرار گیرد).
- **هزینه:** هر حرکت یک واحد هزینه دارد.

نحوه ساخت درخت جستجو برای پازل ۸:
فرض کنید در وضعیت اولیه، خانه خالی در وسط ردیف پایین است. عامل می‌تواند آن را به ۳ جهت (چپ، راست، بالا) حرکت دهد. پس این وضعیت اولیه، در درخت جستجو ۳ فرزند (Child) خواهد داشت.

برای جلوگیری از بازگشت به عقب (مثلاً حرکت خانه خالی به بالا و سپس بلافاصله برگشت به پایین)، عامل نیازمند **حافظه اضافه** است تا وضعیت‌های ویزیت شده را در آن ذخیره کند.

تابع بسط ($Expand\ Function$)

در حین پیش‌روی در درخت جستجو، عامل در هر گره، تابعی به نام $Expand$ را صدا می‌زند.
وظیفه این تابع این است که با در نظر گرفتن وضعیت فعلی گره و اقدامات مجاز، گره‌های فرزند (گره‌های بعدی در دسترس) را تولید و به عامل معرفی کند تا عامل بتواند مسیر خود را ادامه دهد.

پاسخ به سوال آقای کاظمی

سوال: آیا اکشن‌های قبلی باید در هر گره ذخیره شوند (Save بشوند)؟
پاسخ: بله. بسته به نوع استراتژی جستجو، از آنجایی که هر گره نمایانگر یک مسیر تکامل یافته از وضعیت اولیه تا آن نقطه است، برای استخراج راه‌حل نهایی (دنباله اقدامات)، باید مشخص باشد که این گره از چه مسیری و با چه اکشن‌هایی به دست آمده است. البته در سطح پیاده‌سازی پیشرفته، ممکن است بتوان ساختارهای داده را خلاصه‌تر و بهینه‌تر طراحی کرد که حافظه کمتری اشغال کنند.