مدرس: دکتر فاطمه منصوری

نیم‌سال اول ۱۴۰۲-۱۴۰۳

تهیه کننده : فاطمه حسین زاده

هوش مصنوعی

جزوه اول

عامل ها

مسئله اصلی در هوش مصنوعی، ساخت یک عامل منطقی است که دارای هدف و اولویت‌هایی می باشد. . این عامل در تلاش است اعمالی را اجرا کند که با توجه به اهداف، بهترین نتیجه را به همراه داشته باشد. هر عامل در یک محیط مخصوص قرار دارد که برای تعامل با آن از حسگرهای خود استفاده می کنند. محیط و عامل به همراه هم یک جهان را تشکیل می دهند. به عنوان یک مثال ساده، محیط عامل در بازی چکرز، تخته بازی چکرز است و هر حرکت مهره‌ها یک عمل به حساب می‌آید. به عنوان یک مثال دیگر، بازی شطرنج را در نظر بگیرید: در این بازی، بازیکن یک عامل است و هر حرکت مهره یک عمل محسوب می شود و صفحه شطرنج محیط عامل را تشکیل می دهد.

[در ادامه به بررسی انواع عامل های می پردازیم:

عامل واکنشی، عاملی است که به عواقب اعمال خود فکر نمی‌کند. او ترجیح می‌دهد عملی را انتخاب کند که وضعیت فعلی جهان را بهتر کند. این عامل‌ها معمولا توسط عامل‌های برنامه‌ریز، که مدلی از جهان را طراحی می‌کنند و اعمال مختلفی را که برای محیط شبیه سازی شده اجرا می‌کنند، شکست می‌خورند.

عامل برنامه‌ریز می‌تواند فرضیاتی برای عواقب عمل‌ها در نظر بگیرد و بهترین آنها را انتخاب کند. این یک شبیه‌سازی "هوش" است و معنی هوش در اینجا یعنی همان کاری که انسان هنگام تلاش برای تعیین کردن بهترین حرکت ممکن در هر موقعیتی انجام می‌دهد؛ مانند فکر کردن به آینده.[برای فکر کردن به آینده ابتدا فرضیاتی را در نظر می‌گیریم و عواقب انتخاب هر فرضیه را پیش-بینی می‌کنیم و سپس یکی از این فرضیات را انتخاب می‌کنیم و انجام می‌دهیم.]

[یک مثال ساده برای عامل واکنشی خودروهای خودران هستند، این خودروها با حسگرهای خود پیام هایی را از محیط دریافت می‌کنند، مثلا چراغ‌های ترمز ماشین جلویی روشن می‌شود یا پیاده‌روها باید تشخیص داده شود، هنگام رانندگی این مقادیر از طریق حسگرها به حافظه ماشین منتقل می‌شود.]

[عامل‌های برنامه‌ریزو عامل‌های واکنشی، دو نوع مختلف از عامل‌های هوشمند است که در سیستم‌های هوش مصنوعی استفاده می‌شوند. در ادامه تفاوت های این دو نوع عامل بیان شده است.

۱. رفتار هدفمند:

- عامل واکنشی: عامل واکنشی عمدتاً به تعاملات فوری براساس دریافت حسی خود علاقه‌مند است. این عامل با تطبیق مستقیم ورودی‌های حسی به اقدامات مناسب، بدون درنظر گرفتن پیامدها یا اهداف بلندمدت عمل می‌کند.

- عامل برنامه‌ریز: عامل برنامه‌ریز به اصطلاح، رفتار هدفمند دارد. این عامل با در نظر گرفتن وضعیت فعلی، وضعیت‌های آینده و اهداف مورد نظر، توالی اقداماتی را تولید می‌کند که به دست‌یابی به اهداف منجر خواهد شد.

۲. پردازش اطلاعات:

- عامل واکنشی: عموماً عامل واکنشی از قوانین ساده شرط-عمل یا نگاشت‌های از پیش‌تعیین شده برای تعیین اقدامات خود استفاده می‌کند. این عامل نمی‌تواند مدل داخلی جهان را حفظ کند و درباره آن استدلال کند.

- عامل برنامه‌ریز: عامل برنامه‌ریز یک مدل داخلی از جهان را حفظ می‌کند و از آن برای استدلال درباره اقدامات، نتایج و تأثیرات آن‌ها استفاده می‌کند. این عامل اطلاعات موجود را تجزیه و تحلیل کرده، برنامه‌ها را پیش‌بینی می‌کند و براساس اهداف و پیامدهای پیش‌بینی شده بهترین عمل را انتخاب می‌کند.

۳. تطبیق‌پذیری:

- عامل واکنشی: عوامل واکنشی در تطبیق‌پذیری محدود هستند زیرا بر اصول یا قوانین ثابتی تکیه می‌کنند. آنها براساس تجربه یا شرایط تغییر کننده، یاد نمی‌گیرند یا عملکرد خود را بهبود نمی‌بخشند.

- عامل برنامه‌ریز: عامل‌های برنامه‌ریز تطبیق‌پذیری بیشتری دارند. زیرا می‌توانند از تجارب گذشته یاد بگیرند و طبق آن برنامه‌های خود را تنظیم کنند. آنها می‌توانند استراتژی‌های مختلف را در نظر بگیرند، اثربخشی آنها را ارزیابی کنند و مدل‌های خود را بر اساس بازخورد یا اطلاعات جدید به‌روزرسانی کنند.

۴. افق زمانی:

- عامل واکنشی: عامل‌های واکنشی یک افق زمانی کوتاه دارند و بر تصمیم‌گیری فوری تمرکز دارند. آنها بدون در نظر گرفتن پیامدهای بلندمدت، به شرایط فعلی واکنش نشان می‌دهند.

- عامل برنامه‌ریز: عامل‌های برنامه‌ریز یک افق زمانی بلندتر دارند. آنها حالت‌های آینده محتمل، اقدامات ممکن و نتایج آنها را در نظر می‌گیرند. آنها اقداماتی را انتخاب می‌کنند که منجر به حالت‌های آینده مطلوب می‌شوند، حتی اگر برای رسیدن به آنها فراز و فرودهایی لازم باشد.

به طور خلاصه، عامل‌های واکنشی به طور مستقیم به حسگرها پاسخ می‌دهند، در حالی که عامل‌های برنامه‌ریز با توجه به جهان استدلال کرده، برنامه‌ها را تولید و اقدامات را بر اساس نتایج پیش‌بینی شده انتخاب می‌کنند. عامل‌های برنامه‌ریز یک مدل داخلی پیچیده‌تر دارند و می‌توانند استراتژی‌های خود را در طول زمان تطبیق دهند.]

محیط

برای اینکه وظایف محیط را تعریف کنیم، باید PEAS[1] را بشناسیم. مقیاس عملکرد(P)، سودی که عامل در تلاش برای افزایش آن است را توصیف می‌کند(مثلا توی مار بازی توی گوشی‌های نوکیا سودی که مار دنبالش بود امتیازی بود که با خوردن توپ‌ها به دست می‌آورد پس مقیاس عملکرد ماربازی توپ‌ها بودن). محیط(E) مشخص می‌کند که عامل در کجا عمل می‌کند و هر عمل چه تاثیری بر محیط می‌گذارد(محیط بازی ماربازی همون محیطی بود که در ان حرکت می‌کرد و دیواره‌هایی که توی بعضی مراحل وجود داشت موانعی بودند که در محیط وجود داشتند، پس همه آنها باهم محیط رو تشکیل می‌دادند.). عمل‌ها(A) و سنسورها(S) روش‌هایی هستند که عامل با استفاده از آنها بر روی محیط عمل می‌کند و اطلاعاتی از آن دریافت می‌کند(عمل‌ها در این بازی همان حرکت به چپ، راست، بالا و پایین بودند که با انجام آن مار تغییر مسیر می‌داد و شما برای عملکرد بهتر سعی می‌کردین به توپ‌ها نزدیک بشین در صورتی که مار به مانع یا خودش برخورد نکند، سنسورها یا همون حسگرها هربار موقعیت مار نسبت به دیوارها و توپ ها را (کلا موقعیت مار در محیط را) گزارش می کردند.)

طراحی یک عامل به شدت بستگی به نوع محیطی دارد که عامل در آن قرار دارد. ما می‌توانیم انواع محیط‌ها را به صورت زیر مشخص کنیم:

- محیط های نیمه مشاهده پذیر:

در محیط‌های نیمه مشاهده پذیر عامل اطلاعات کاملی از حالت‌ها ندارد و برای همین باید یک برآورد داخلی از وضعیت جهان داشته باشد. این کاملا در تضاد با محیط‌های مشاهده پذیر است که عامل دسترسی کامل به اطلاعات مربوط به حالت‌ها دارد.(همه بازی‌های کارتی که بازیکن قادر به دیدن همه کارت‌ها نیست و فقط کارت‌های مشخصی را می‌تواند ببیند دارای محیط‌های نیمه مشاهده‌پذیر هستند و بازی‌هایی مثل شطرنج و تخته‌نرد از بازی‌هایی هستند با محیط‌های مشاهده‌پذیر)

- محیط های تصادفی:

محیط‌های تصادفی در مدل‌های انتقال دچار عدم قطعیت می‌شوند، یعنی اینکه اگر عملی را برای رفتن به یک استیت خاص در نظر بگیریم ممکن است چندین نتیجه با احتمال های متفاوت داشته باشیم. این محیط‌ها در تضاد با محیط‌های قطعی هستند؛ در محیط های قطعی در صورت انتخاب یک عمل و یک استیت تنها یک اتفاق با احتمال صد در صد میافتد.(بازی هایی که دارای تاس هستند، بازی هایی با محیط های تصادفی می باشند. بازی شطرنج دارای محیط‌های تصادفی است چون با حرکت‌دادن هر مهره مشخص نیست بازیکن مقابل چه مهره‌ای را و چقدر به حرکت در می‌آورد. برای محیط‌های قطعی می‌توان به ماشین‌های خودران اشاره کرد)

[1] **P**erformance Measure, **E**nvironment, **A**ctuators, **S**ensors

- محیط‌های چند عاملی:

در محیط های چندعاملی، چندین عامل همزمان باهم عمل می‌کنند، برای همین ممکن است هر عامل بخواهد عمل‌های خود را به صورت تصادفی انتخاب کند تا عامل‌های دیگر قادر به تشخیص عمل بعدی او نباشند.(بازی‌های کارتی جزو بازی‌هایی با محیط چندعاملی هستند. فوتبال هم بازی چندعاملی است که هر تیم ۱۱ عامل دارد)

- محیط‌های ایستا و پویا:

اگر عامل عملی را انجام دهد و محیط تغییری نکند به این نوع محیط، ایستا می‌گویند. این محیط در تضاد با محیط پویاست؛ در محیط‌های پویا با هر حرکت عامل محیط تغییر می‌کند.

- محیط‌های فیزیکی:

در صورتی که محیط فیزیکی شناخته شده‌ای داشته باشیم عامل‌ها با مدل های انتقال آشنا هستند(حتی اگر تصادفی باشد) و عامل از این ویژگی می‌تواند هنگام برنامه‌ریزی استفاده کند. اگر محیط فیزیکی شناخته شده نباشد عامل باید برای یادگیری محیط پویای ناشناخته عمل های مختلفی را انجام دهد.

فضای حالت و مسئله جستجو

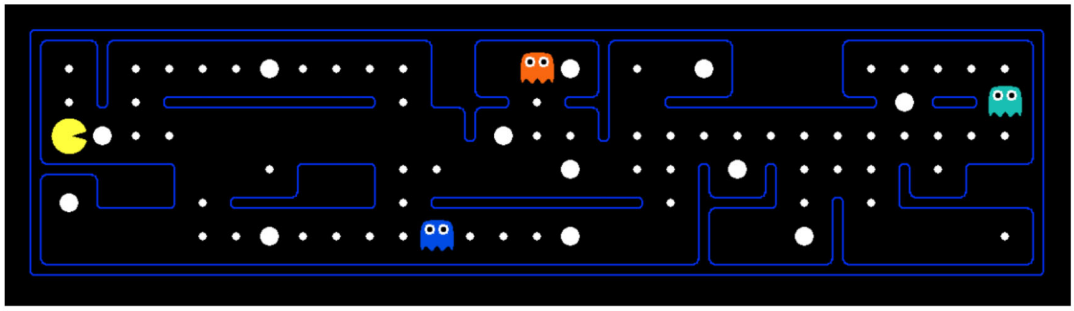
برای ساخت یک عامل منطقی برنامه‌ریز، به راهی نیاز داریم که بتوان به صورت ریاضی، محیطی که عامل در آن قرار دارد را توصیف کرد. برای این کار باید به بیان این مسئله از جستجو بپردازیم که-با توجه به حالتی که عامل در حال حاضر در آن قرار دارد- چطور می‌شود به حالت جدیدی رسید به صورتی که هدف‌های خود را به بهترین شکل ممکن ارضا کند؟

اساس تشکیل یک مسئله جستجو به صورت زیر است:

- فضای حالت: مجموعه‌ای از تمام حالت‌های ممکن که احتمال اتفاق افتادن آنها در جهان مشخص شده، امکان‌پذیر است.
- مجموعه‌ای از اعمال(عمل‌ها) قابل دسترس در هر حالت
- مدل انتقال: زمانی که عملی خاص در حالت فعلی انجام می‌شود حالت بعدی را به عنوان خروجی می‌دهد(نتیجه عملی که در حالت فعلی صورت گرفته است).
- هزینه هر عمل: هنگامی که عملی را انجام می‌دهیم برای انتقال عامل از حالتی به حالت دیگر رخ می‌دهد.
- حالت شروع: حالت اولیه‌ای که عامل در آن قرار دارد.
- آزمون هدف: تابعی است که حالتی را به عنوان ورودی می‌گیرد و مشخص می‌کند که آن حالت، حالت هدف است یا نه.

یک مسئله جستجو در ابتدا با در نظر گرفتن حالت شروع حل می‌شود، سپس با استفاده از روش های هزینه اعمال و انتقال به جستجوی فضای حالت می‌پردازیم، ما به صورت مکرر شاخه های مختلف یک درخت را تا هدف بررسی می‌کنیم و سپس مسیری از حالت شروع به حالت هدف پیدا می‌کنیم. اینکه در چه موقعیتی کدام حالت انتخاب شود با استفاده از یک استراتژی از پیش تعیین شده تعریف شده است که در ادامه به بررسی این استراتژی‌ها می‌پردازیم.

قبل از اینکه مسئله جستجو را ادامه دهیم، باید تفاوت بین حالت جهانی و حالت جستجو را بدانیم. یک حالت جهانی شامل همه اطلاعاتی است که در یک حالت وجود دارد، اما حالت جستجو فقط شامل اطلاعاتی از جهان است که برای برنامه‌ریزی لازم است. برای نشان دادن این مفاهیم به سراغ بازی پکمن می‌رویم. بازی پکمن ساده است: پکمن باید در یک ماز حرکت کند و تمام گلوله های غذایی کوچک را بخورد بدون اینکه توسط روح‌ها خورده شود، اگر پکمن یکی از گلوله های بزرگ را بخورد برای مدتی در برابر ارواح مصون می‌شود و قادر به خوردن ارواح برای گرفتن امتیاز بیشتر می‌شود.



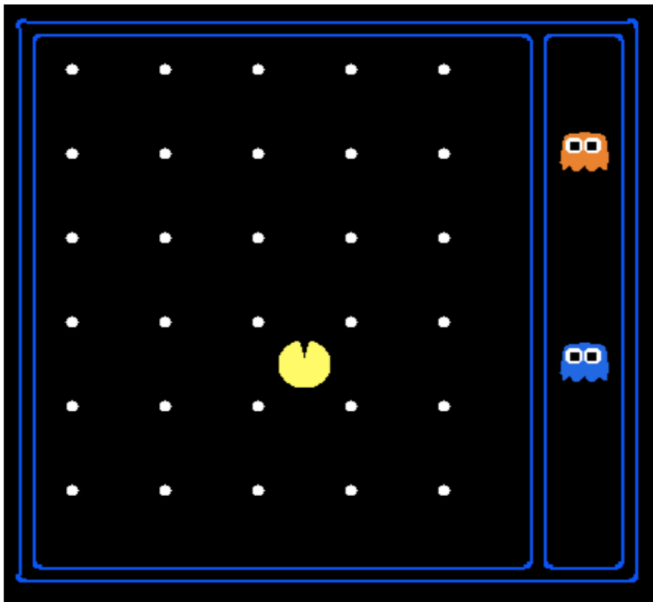
بیایید حالتی را در نظر بگیریم که فقط شامل پکمن و گلوله های غذایی است. ما می‌توانیم دو مسئله جستجو متفاوت را در این سناریو مطرح کنیم: مسیریابی و خوردن همه گلوله‌های غذایی. مسیریابی تلاش می‌کند مسیر بهینه بین دو نقطه (x_1,y_1) و (x_2,y_2) را پیدا کند در حالی که خوردن همه گلوله های غذایی تلاش می‌کند تا پکمن در کمترین زمان همه گلوله های غذایی را بخورد. در ادامه، حالت‌ها، اعمال، مدل انتقال و آزمون هدف برای هر دو مسئله ذکر شده است:

- مسیریابی:
  - حالت: موقعیت(x,y)
  - اعمال: شمال، جنوب، شرق، غرب
  - مدل انتقال(رفتن به حالت بعدی): به روزرسانی موقعیت
  - آزمون هدف: آیا موقعیت (x,y) حالت هدف است؟
- خوردن همه گلوله های غذایی:
  - حالت: موقعیت(x,y) و گلوله های غذایی
  - اعمال: شمال، جنوب، شرق، غرب
  - مدل انتقال(رفتن به حالت بعدی): به روزرسانی موقعیت و گلوله های غذایی
  - آزمون هدف: آیا همه گلوله های غذایی خورده شده اند؟

باید توجه داشت که اطلاعات ما در جستجو مسیر کمتر از جستجو برای پیدا کردن گلوله های غذایی است، چون در حالت دوم ما باید علاوه بر مسیر اطلاعاتی راجع به تعداد گلوله ها و موقعیت آنها هم داشته باشیم. یک حالت جهانی همچنان می‌تواند اطلاعات بیشتری داشته باشد، اطلاعاتی مانند کل مسافت طی شده توسط پکمن یا تمام حالت های بازدید شده توسط پکمن.

اندازه فضای حالت

سوال مهمی که اغلب هنگام تخمین زمان اجرای محاسباتی در حل یک مسئله جستجو پیش می‌آید اندازه فضای حالت است این کار اصولا با اصل شمارش اتفاق میافتد. این اصل بیان می‌کند که اگر n متغیر داشته باشیم که مقادیر $x_1,...,x_n$ را در خود جای دهند آنگاه تعداد کل حالت‌ها برابر با $x_1 \cdot x_2 \cdot ... \cdot x_n$ است. برای نشان دادن بهتر این مفهوم از مثال پکمن استفاده می‌کنیم:



فرض کنید متغیرها و احتمالاتشان به صورت زیر است:

- موقعیت پکمن: پکمن می‌تواند در 120 موقعیت مختلف (x,y) قرار داشته باشد و تنها یک پکمن در بازی قرار دارد.
- مسیر پکمن: پکمن می‌تواند به شمال، جنوب، شرق و یا غرب حرکت کند پس در مجموع ۴ احتمال برای پکمن وجود دارد.
- موقعیت روح‌ها: دو روح در بازی وجود دارند که هر کدام می‌توانند در 12 موقعیت مختلف (x,y) قرار داشته باشند.
- گلوله های غذایی: در مجموع 30 گلوله غذایی وجود دارد که این گلوله ها می‌توانند خورده شوند یا خورده نشوند.

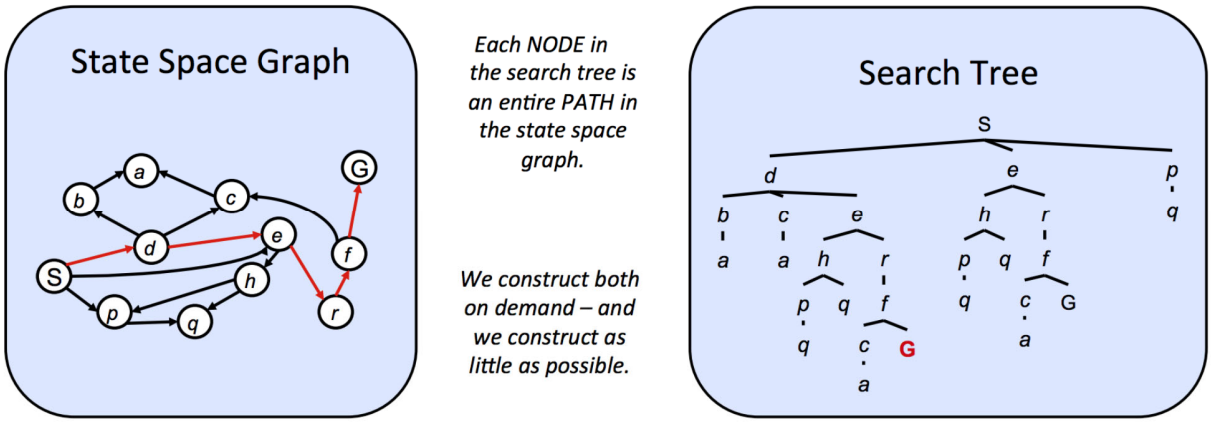
با استفاده از اصل شمارش ما 120 موقعیت برای پکمن، 4 عمل که پکمن در هر استیت می‌تواند استفاده کند، 12×12حالت برای دو روح موجود در بازی و 2^{30}حالت برای هر گلوله غذایی داریم(هر گلوله غذایی می‌تواند خورده شود یا دست نخورده باقی بماند) پس در مجموع اندازه فضای حالت ما برابر است با $120 \times 4 \times 12 \times 12 \times 2^{30}$.

گراف فضای حالت و درخت جستجو

اکنون که ایده فضای حالت و چهار مولفه آن را به صورت کامل تعریف کردیم وقت آن رسیده که به حل مسئله جستجو بپردازیم. آخرین مطلبی که باید برای حل مسئله جستجو بدانید گراف فضای حالت و درخت جستجو است.

همان‌طور که می‌دانید یک گراف مجموعه‌ای از گره‌ها و یال‌هاست که گره‌ها به وسیله یال‌ها به هم متصل می‌شوند. همچنین یال ها ممکن است دارای وزن باشند. یک گراف فضای حالت شامل **حالت‌هایی** است که گره‌های گراف را تشکیل می‌دهند و یال‌های گراف از هر حالت به فرزندانش وصل می‌شود. این یال‌ها نشان دهنده **اعمال** هستند و وزن هر یال نشان دهنده **هزینه**‌ای است که باید بپردازد تا از هر حالت با عملی که انجام می‌دهد به حالت بعدی برود. معمولا این گراف‌ها بزرگ‌تر از آن هستند که بشود در حافظه نگهداری کرد(حتی بازی ساده‌ای مثل پکمن که در بالا به آن اشاره شد گرافی با 10^{13} حالت مختلف دارد) اما خوب است که با آنها آشنا باشید و هنگام حل مسائل آنها را در نظر داشته باشید. همچنین مهم است که بدانید در یک گراف فضای حالت، هر حالت دقیقاً یک بار نشان داده می شود - نیازی به نشان دادن بیشتر نیست و دانستن این موضوع هنگام تلاش برای استدلال در مورد مسائل جستجو کمک زیادی می کند.

برخلاف ساختار گراف، ساختار درخت‌های جستجو چنین محدودیتی در تعداد دفعات تکرار حالت‌ها ندارند و ممکن است یک حالت چندین بار در یک درخت جستجو تکرار شود. اگرچه هر درخت جستجو خود نوعی گراف است و از گره ها و یال ها تشکیل می‌شود اما به دلیل اینکه هر دو نقطه در درخت جستجو از حالت شروع تا حالت پایان باید دارای مسیر جداگانه باشد باعث تکرار بعضی حالت‌ها می‌شود. شکل زیر گراف فضای حالت و درخت جستجوی آن را نشان می‌دهد:



مسیر مشخص شده در گراف فضای حالت (S → d → e → r → f → G) را با دنبال کردن S تا G در درخت جستجو خواهید یافت. به طور مشابه، هر مسیر از گره شروع به هر گره دیگری، در درخت جستجو با یک مسیر از ریشه S به گره های دیگر خواهد بود. از آنجایی که اغلب بیشتر از تنها یک راه برای رسیدن از یک حالت به حالت دیگر وجود دارد معمولا حالت ها چند بار در درخت جستجو تکرار می‌شوند. در این صورت **درختان جستجو بزرگتر یا مساوی با گراف فضای حالت خود هستند.**

ما قبلا مشخص کردیم که گراف‌های فضای حالت بسیار بزرگ هستند و حتی برای مسائل ساده هم نمی‌توان آنها را در حافظه نگهداری کرد. اکنون سوالی که پیش می‌آید این است که چگونه می‌توانیم محاسبات مفیدی را از روی این ساختارها به دست

آوریم؟ پاسخ این سوال به این مسئله بستگی دارد که ما چگونه فرزندان حالت کنونی را محاسبه می‌کنیم-در ابتدا فقط حالت هایی را که فورا به آن نیاز داریم محاسبه می‌کنیم و حالت های جدید را بر اساس نیاز با استفاده از اعمال و هزینه های آنها بررسی خواهیم کرد-

معمولا مسائل جستجو با استفاده از درخت‌های جستجو به صورتی حل می‌شوند که چند گره در یک زمان ذخیره شوند و پس از بازدید، گره‌های فرزندشان را جایگزین کنیم تازمانی که به حالت هدف برسیم. روش های مختلفی برای تعیین نحوه انتخاب گره‌ها وجود دارد و ما در ادامه به معرفی این روش‌ها می‌پردازیم:

جستجوی ناآگاه

در پروتکل استاندارد برای یافتن برنامه‌ای که با آن از حالت شروع به حالت هدف برویم، یک حد خارجی از برنامه جزئی که از درخت جستجو به دست آمده‌اند نگه می‌داریم. سپس حد را هربار با حذف کردن یک راس(که به کمک استراتژی تعیین شده انتخاب می‌شود) از یک برنامه جزئی درون حد و جایگزین کردن آن با فرزندانش بسط می‌دهیم. حذف و جایگزین کردن یک عنصر روی حد با فرزندانش به معنی کنار گذاشتن برنامه‌ای با طول n و به جای آن در نظر گرفتن تمامی برنامه های با طول n+1 که از آن ریشه می‌گیرند می‌باشد. این روند را تا جایی ادامه می‌دهیم که یک حالت هدف از حد حذف شود که در این صورت نتیجه می-گیریم برنامه جزئی‌ای که حالت هدف از آن حذف شد مسیری برای رفتن از حالت حالت شروع به حالت هدف است.(برای درک بهتر الگوریتم دایجکسترا را در نظر بگیرید.)

به طور کاربردی، اکثر پیاده‌سازی‌های چنین الگوریتم هایی اطلاعاتی در راس ذخیره می‌کنند که شامل اطلاعاتی راجع به راس پدر، فاصله تا راس و حالت درون راس است. رویه‌ای که شرح دادیم به عنوان جستجوی درختی شناخته شده که شبه کد آن در زیر نمایش داده شده است:

```
function TREE-SEARCH(problem, frontier) return a solution or failure
    frontier← INSERT(MAKE-NODE(INITIAL-STATE[problem]), frontier)
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        end if
        for each child-node in EXPAND(problem, node) do
            add child-node to frontier
        end for
    end while

    return failure
```

تابع EXPAND که در شبه کد ظاهر می شود، تمام گره های ممکنی را که می توان از یک گره مشخص با در نظر گرفتن تمام عمل‌های موجود به آنها دسترسی پیدا کرد، برمی‌گرداند.

```
function EXPAND(problem, node) yields nodes
    s← node.STATE
    for each action in problem.ACTIONS(s) do
        s′ ← problem.RESULT(s, action)
        yield NODE(STATE=s′, PARENT=node, ACTION=action)
    end for
```

[جستجوی ناآگاه یک دسته از الگوریتم‌های جستجو استفاده شده در هوش مصنوعی و علوم کامپیوتر است که به فضای مسئله بدون دانش یا راهنمایی قبلی درباره مسئله می‌پردازد. این الگوریتم‌ها تنها بر اساس عملیات و حالت‌های در دسترس مسئله عمل می‌کنند.

در زیر تعدادی از الگوریتم‌های جستجوی ناآگاه رایج آورده شده است:

۱. جستجوی اول سطح[۲]: همه گره‌های همسایه در سطح فعلی را قبل از حرکت به گره‌های در سطح بعدی بررسی می‌کند.

۲. جستجوی اول عمق[۳]: تا جایی که امکان دارد در طول هر شاخه پیش می‌رود و سپس به عقب برمی‌گردد. این الگوریتم قبل از بررسی گره‌های همسایه، به صورت عمیق‌تر درخت جستجو را بررسی می‌کند. DFS تضمینی برای پیدا کردن مسیر کوتاه ندارد.

۳. جستجوی عمق محدود با تکرار[۴]: با ترکیب مزایای BFS و DFS با انجام مجموعه‌ای از DFS با محدودیت عمق و افزایش تدریجی حداکثر عمق تا پیدا شدن هدف. (برای خواندن بیشتر می‌توانید به اینجا مراجعه کنید.)

۴. جستجوی هزینه یکنواخت[۵]: گره با کمترین هزینه مسیر تا این لحظه را گسترش می‌دهد. این الگوریتم هزینه رسیدن به هر گره را در نظر می‌گیرد و مسیری با کمترین هزینه جمعی را انتخاب می‌کند.

---

[2] BFS
[3] DFS
[4] IDDFS
[5] UCS

۵. جستجوی با محدودیت عمق[6]: شباهتی به DFS دارد اما حداکثر عمق بررسی را محدود می‌کند. این الگوریتم با تعیین یک حداکثر عمق، مسیرهای بی‌نهایت را از بین می‌برد. (برای خواندن بیشتر می‌توانید به اینجا مراجعه کنید.)

الگوریتم‌های جستجوی ناآگاه می‌توانند در فضاهای مسئله کوچک یا در صورت نزدیک بودن حالت هدف به حالت اولیه موثر باشند. با این حال، در فضاهای مسئله بزرگ یا پیچیده ممکن است کارآیی آنها کاهش یابد یا نتوانند به یافتن یک راه‌حل برسند. در چنین مواردی، الگوریتم‌های جستجوی آگاه که از هیوریستیک یا دانش خاص صنعتی استفاده می‌کنند، معمولاً بهتر و کارآمدتر هستند که در جزوه‌های بعد درباره آن‌ها صحبت می‌کنیم.]

زمانی که ما از موقعیت حالت‌های هدف در درخت جستجو اطلاعی نداریم مجبور می‌شویم برای جستجو در درخت موجود استراتژی خود را از یکی از تکنیک های جستجوی ناآگاه انتخاب کنیم.

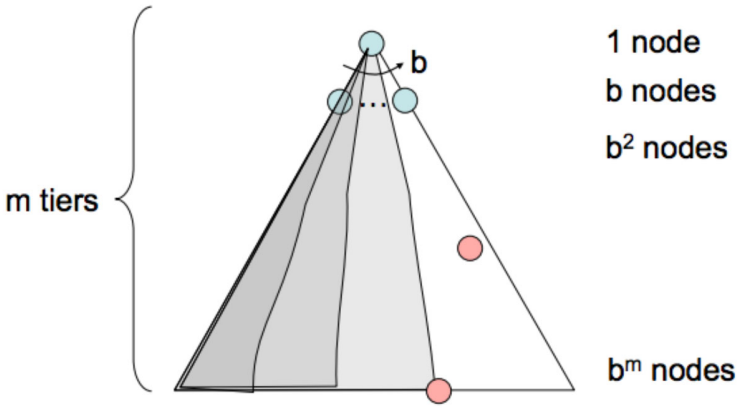
اگر بخواهیم سه استراتژی برای این کار نام ببریم آن سه استراتژی عبارتند از: جستجوی اول سطح، جستجوی اول عمق و جستجو با هزینه یکنواخت. همراه با هر استراتژی، برخی از ویژگی های ابتدایی استراتژی نیز ارائه شده است:

- **کامل بودن** هر استراتژی جستجو: اگر راه حلی برای مسئله جستجو وجود داشته باشد، آیا استراتژی موجود برای یافتن آن با منابع محاسباتی بی نهایت تضمین شده است؟
- **بهینه بودن** هر استراتژی جستجو: آیا ضمانتی وجود دارد که استراتژی موجود مسیری با کمترین هزینه را پیدا کند که به حالت هدف منتهی شود؟
- **عامل انشعاب**: هربار با جایگزین شدن گره های فرزند تعداد گره‌های موجود افزایش میابد$O(b)$ و در عمق k تعداد گره‌ها برابر است با $O(b^k)$
- حداکثر عمق برابر با m است.
- اولین عمق درخت(ریشه درخت) s است.

- جستجوی اول عمق

- توضیح: جستجوی اول عمق(DFS) یک استراتژی برای جستجو است که همیشه عمیق‌ترین گره مرزی را از گره شروع انتخاب می‌کند.
- نمایش حد: حذف کردن عمیق‌ترین گره و جایگزین کردن آن با گره‌های فرزندش نشان‌دهنده این است که اکنون فرزندان آن گره عمیق‌ترین عمق را دارند –عمق گره‌های فرزند یکی بیشتر از عمق گره پدر است- این بدین معناست که ما برای پیاده‌سازی DFS به ساختاری نیاز داریم که همیشه گره‌های تازه اضافه‌شده دارای اولویت بیشتری باشند. ساختار داده پشته آخرین ورودی/اولین خروجی(LIFO) دقیقا همین‌کار را می‌کند.

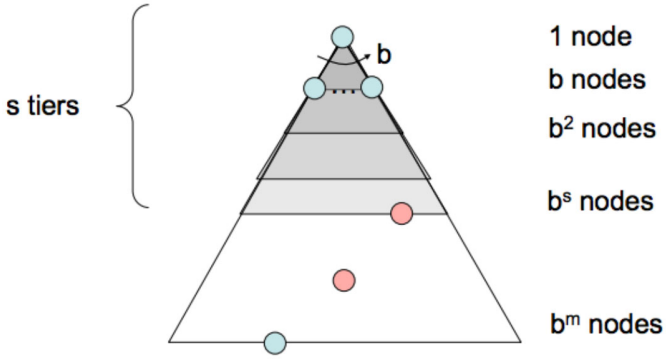
[6] DLS



- کامل بودن: جستجوی اول عمق کامل نیست. اگر در گراف فضای حالت دور داشته باشیم به این معنی‌است که شاخه درخت هیچ وقت تمام نمی‌شود و بی‌نهایت بار تکرار می‌شود برای همین ممکن است DFS بعضی اوقات برای یافتن راه‌حل در درخت جستجوی بی‌نهایت گیر کند.
- بهینگی: DFS به سادگی سمت چپ‌ترین برگ درخت جستجو را بدون توجه به هزینه‌های مسیر برمی‌دارد و این کار اصلا بهینه نیست.
- پیچیدگی زمانی: در بدترین حالت DFS مجبور است تمام گره‌های درخت جستجو را بازدید کند. اگر حداکثر عمق درخت برابر با m باشد پیچیدگی زمانی برابر است با $O(b^m)$.
- پیچیدگی فضایی: در بدترین حالتDFS، b گره در عمق m را نگهداری می‌کند. پس پیچیدگی فضایی برابر است با $O(mb)$.

- جستجوی اول سطح

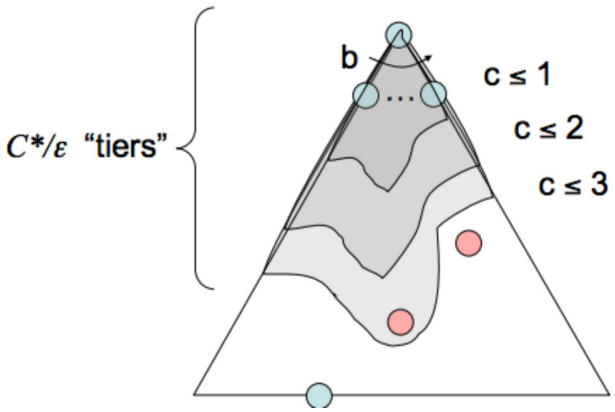
- توضیح: جستجوی اول سطح یک استراتژی برای جستجو است که همیشه کم‌عمق‌ترین گره را از گره شروع انتخاب می‌کند.
- نمایش حد: اگر ما بخواهیم گره‌هایی با عمق کمتر را زودتر از گره‌های با عمق بیشتر بازدید کنیم باید گره‌ها را به ترتیب ورودشان ببینیم. ما ساختار داده‌ای می‌خواهیم که قدیمی‌ترین ورودی در صف را تولید کند. برای این کار BFS از صف اولین ورودی اولین خروجی استفاده می‌کند که دقیقا همان کار خواسته شده را انجام می‌دهد.



- کامل بودن: اگر راه‌حلی در درخت وجود داشته باشد پس عمق سطحی ترین گره s باید متناهی باشد. پس BFS باید حتما آن عمق را بررسی کند، پس BFS کامل است.
- بهینگی: BFS بهینه نیست چون بدون در نظر گرفتن هزینه مسیر صرفا هر سطح درخت را بررسی می‌کند و اگر راه حلی پیدا نکرد به سطح بعدی می‌رود. تنها زمانی که BFS بهینه است زمانی است که هزینه همه یال های درخت باهم برابر باشند که در این صورت مسئله تبدیل به نوع خاصی از مسئله جستجوی هزینه یکنواخت می‌شود که در ادامه به توضیح آن می‌پردازیم.
- پیچیدگی زمانی: در بدترین حالت ما باید $1 + b + b^2 + \cdots + b^s$ گره را جستجو کنیم. از آنجایی که در هر عمق ما به همه گره‌ها سر می‌زنیم اگر تا عمق s بررسی کنیم پیچیدگی زمانی برابر است با $O(b^s)$.
- پیچیدگی فضایی: حد خارجی در بدترین حالت شامل همه گره‌ها در سطح مربوط به کم عمق ترین راه حل است. از آنجایی که کم‌عمق‌ترین راه حل در عمق s قرار دارد پیچیدگی فضایی برابر است با $O(b^s)$.

- جستجوی هزینه یکنواخت

- توضیح: جستجوی هزینه یکنواخت، آخرین استراتژی ما، استراتژی‌ای است که همیشه کمترین هزینه را برای بازدید گره‌ها از گره شروع انتخاب می‌کند.
- نمایش حد: برای نمایش حد استراتژی UCS انتخابِ ما معمولاً یک صف اولویت مبتنی بر پشته است که برای انتخاب گره v باید هزینه مسیر از ابتدا تا گره v بررسی شود. این صف اولویت به سادگی خود را تغییر می‌دهند تا کمترین هزینه مسیر را پیدا کند.



- کامل بودن: جستجوی هزینه یکنواخت کامل است، چون اگر راه‌حلی برای مسئله وجود داشته باشد حتما کوتاه‌ترین راه‌حلی برای آن وجود دارد که این راه‌حل توسط UCS پیدا می‌شود.
- بهینگی: UCS بهینه است اگر تمام یال‌های درخت غیرمنفی باشند. از آنجایی که گره‌ها را به ترتیب افزایش هزینه مسیر بررسی می‌کنیم، تضمین می‌کنیم که کم‌هزینه‌ترین مسیر را برای رسیدن به یک حالت هدف پیدا کنیم.
استراتژی مورد استفاده در جستجوی هزینه یکنواخت با الگوریتم Dijkstra یکسان است و تفاوت اصلی این است

که UCS به جای یافتن کوتاه‌ترین مسیر برای همه حالت‌ها، با یافتن یک حالت راه‌حل پایان می‌یابد. داشتن گرافی با یال‌هایی که دارای هزینه منفی هستند بهینگی این استراتژی را بهم می‌زند.

- پیچیدگی زمانی: اگر هزینه مسیر C^*باشد و حداقل هزینه بین دو گره در گراف برابر ε باشد در این صورت ما باید تقریبا همه گره‌ها را از عمق 1 تا C^*/ε بررسی کنیم. بنابراین پیچیدگی زمانی برابر است با $O(b^{\frac{C^*}{\varepsilon}})$
- پیچیدگی فضایی: حد خارجی تقریبا شامل همه گره‌ها در سطح کمترین هزینه خواهد بود پس پیچیدگی فضایی برابر است با $O(b^{\frac{C^*}{\varepsilon}})$.

مهم است که بدانید سه استراتژی ذکر شده در بالا اساساً یکسان هستند - فقط در استراتژی بازدید گره‌ها متفاوت هستند، و شباهت‌های آنها توسط شبه‌کد جستجوی درختی ارائه شده در بالا مشخص می شود.

منابعی برای داشتن اطلاعات کامل‌تر:

کتاب Artificial Intelligence: A Modern Approach نوشته Stuart Russell and Peter Norvig

کورس آنلاین دانشگاه استنفورد با نام Introduction to Artificial Intelligence در کورسرا