www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

برنامه‌نویسی هوش مصنوعی
در JAVA

مؤلف

کاظم محمدی

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

سرشناسه : محمدی، کاظم، ۱۳۶۵
عنوان و نام پدیدآور : برنامه‌نویسی هوش مصنوعی در JAVA
مشخصات نشر : تهران: انتشارات ناقوس، ۱۳۹۴.
مشخصات ظاهری : ۱۹۰ ص.: مصور، جدول.
شابک : ۸-۷۵۴-۳۷۷-۹۶۴- ۹۷۸ بها : ۱۴۰.۰۰۰ریال
وضعیت فهرست‌نویسی : فیپای مختصر
یادداشت : فهرستنویسی کامل این اثر در نشانی http://opac.nlai.ir قابل دسترسی است.
شمارهٔ کتابشناسی ملی : ۳۷۶۶۱۱۲

انتشارات ناقوس

برای خرید Online به آدرس زیر مراجعه کنید:
www.naghoospress.ir

NAGHOOS PUBLICATION

چاپ اول

« در نوبت چاپ اول از دستگاه چاپ دیجیتال استفاده شده است »

S.M.S : ۳۰۰۰۴۵۲۳۲۳

نام کتاب	: برنامه‌نویسی هوش مصنوعی در JAVA
ناشر	: انتشارات ناقوس
مؤلف	: کاظم محمدی
چاپ اول	: ۱۳۹۴
تیراژ	: ۲۰۰ جلد
چاپ و صحافی	: نارنجستان
سرپرست صفحه‌آرا	: صدف پورعبدی
قیمت	: ۱۴۰.۰۰۰ ریال
شابک	: ۸-۷۵۴-۳۷۷-۹۶۴- ۹۷۸
ISBN	: 978-964-377-754-8

کلیه حقوق برای ناشر محفوظ است. تکثیر تمامی یا قسمتی از این اثر به‌صورت حروفچینی یا چاپ مجدد، چاپ افست، پلی‌کپی، فتوکپی و انواع دیگر چاپ ممنوع است و پیگرد قانونی دارد.

مراکز پخش:
۱- انتشارات ناقوس: میدان انقلاب، خیابان کارگرجنوبی، خیابان روانمهر، کوچه دولتشاهی، پلاک ۲
تلفن و فاکس : ۶۶۴۰۸۵۲۰ – ۶۶۴۰۱۷۹۸ – ۶۶۴۱۷۰۷۲ - ۶۶۴۶۶۷۹۳
۲- کتابفروشی گلریزان، ضلع جنوب‌شرقی میدان انقلاب، پلاک ۱۱ تلفن: ۶۶۹۷۶۲۳۰
۳- کتابفروشی الیاس: خیابان انقلاب، نبش فروردین تلفن: ۶۶۴۰۵۰۸۴

سپاسگزاری

ستایش خدای راست که پیوند دهندهٔ ستایش به نعمت است و نعمت به ستایش. او را بر نعمتهایش می‌ستاییم چنانکه بر بلایش، و از او بر نفس خود مدد خواهیم که در آنچه باید کاهل است و بر آنچه نباید عاجل، و خواهان بخشش او هستیم در آنچه علمش آن را در برگرفته و در کتابش بر شمرده. گواهی می‌دهیم که پرستش شونده‌ای جز خدای ما نیست، یکی است بی‌شریک، و محمد بندهٔ اوست و پیامبرش، که درود خدا بر او و خاندان پاکش باد.[1]

آرزوی سلامتی می‌خواهم پدرم را و غرق بوسه می‌کنم وجود مادرم را که می‌دانم نبوده است جز به دعای خیر ایشان، از ابتدای زندگی.

۱- برگرفته از خطبهٔ ۱۱۴ نهج‌البلاغه

فهرست مطالب

پیشگفتار: ۹
فصل اول: هیستوریک ۱۳
مقدمه: ۱۳
۱-۱- نمایش فضای وضعیت جستجو و عملیات‌های جستجو: ۱۳
۱-۲- یافتن مسیرها در مارپیچ‌ها ۱۵
۱-۳- یافتن مسیرها در گراف‌ها ۲۱
۱-۴- افزودن هیوریستک‌ها به جستجوی اولین وسعت ۲۹
۱-۵- جستجو و انجام بازی ۳۰
۱-۵-۱- جستجو آلفا-بتا ۳۰
۱-۵-۲- چارچوب جاوا برای جستجو و انجام بازی ۳۲
۱-۵-۳- tic-tac-toe که از الگوریتم جستجوی آلفا-بتا استفاده می‌کند ۳۸
۱-۵-۴- شطرنج که از الگوریتم جستجوی آلفا-بتا استفاده می‌کند ۴۳
فصل دوم: منطق واستدلال ۵۵
مقدمه: ۵۵
۲-۱- منطق ۵۶
۲-۱-۱- تاریخچه منطق ۵۷
۲-۲-۲- مثال‌هایی از انواع متفاوت منطق ۵۷
۲-۲- مروری بر PowerLoom ۵۸
۲-۳- اجرای PowerLoom به گونه تعاملی ۵۸
۲-۴- استفاده از APIهای PowerLoom در برنامه‌های جاوا ۶۲
۲-۵- پیشنهادهایی برای مطالعه بیشتر ۶۴
فصل سوم: وب معنایی(Semantic Web) ۶۵
مقدمه: ۶۵
۳-۱- مدل پایگاه داده‌ای منطقی ۶۶
۳-۲- RDF: فرمت داده‌های جامعه آماری ۶۷

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

۳-۳- گسترش RDF با طرح RDF ۷۰
۴-۳- زبان جستجو SPARQL ۷۱
۵-۳- استفاده نمودن ازSesame ۷۵
۶-۳- OWL: زبان آنتولوژی وب ۷۸
۷-۳- ارائه دانش و REST ۷۹
فصل چهارم: سیستم‌های خبره ۸۱
مقدمه: ۸۱
۱-۴- سیستم‌های تولید ۸۳
۲-۴- زبان قوانین Drools ۸۳
۳-۴- استفاده نمودن از Drools درجاوا ۸۵
۴-۴- سیستم خبره Drools مثال: دنیای بلوک‌ها ۸۹
۱-۴-۴- مدل‌های شیء POJO برای مثال دنیای بلوک‌ها ۹۰
۲-۴-۴- قوانین Drools برای مثال دنیای بلوک‌ها ۹۳
۳-۴-۴- کد جاوا برای مثال دنیای واقعی ۹۷
۵-۴- سیستم خبره Drools مثال: سیستم روزمره کمکی ۹۹
۱-۵-۴- مدل‌های شی برای مثال کمک روزانه ۹۹
۲-۵-۴- قوانین Drools برای مثال کمک روزمره ۱۰۲
۳-۵-۴- کد جاوا برای یک مثال از کمک روزمره ۱۰۴
۶-۴- نکاتی برای پیش‌نویس ساختن سیستم‌های خبره ۱۰۵
فصل پنجم: الگوریتم ژنتیک ۱۰۷
مقدمه: ۱۰۷
۱-۵- نظریه ۱۰۷
۲-۵- کتابخانه جاوا برای الگوریتم‌های ژنتیک ۱۱۰
۳-۵- یافتن حداکثر مقدار تابع ۱۱۳
فصل ششم: شبکه‌های عصبی ۱۱۹
مقدمه: ۱۱۹
۱-۶- شبکه‌های عصبی Hopfield ۱۲۱
۲-۶- کلاس‌های جاوا برای شبکه‌های عصبی Hopfield ۱۲۲

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

۳-۶- تست کلاس شبکه عصبی Hopfield ۱۲۴
۴-۶- شبکه‌های عصبی عقب گرد ۱۲۷
۵-۶-کتابخانه کلاس درجاوا برای انتشار عقب‌گرد ۱۳۰
۶-۶- افزودن اندازه حرکت آنی برای سرعت بخشیدن به تمرین back-prop ۱۳۸
فصل هفتم: یادگیری ماشین با Weka ۱۴۱
مقدمه: ۱۴۱
۱-۷- استفاده از نرم‌افزار کاربردی GUI ۱۴۲
۲-۷- استفاده از Weka در خط دستور تعاملی ۱۴۴
۳-۷- تعبیه‌نمودن Weka در نرم‌افزار کاربردی جاوا ۱۴۶
فصل هشتم: پردازش زبان آماری NLP ۱۵۱
مقدمه: ۱۵۱
۱-۸- نشانه‌گذاری، ریشه‌یابی و قسمتی از متن برچسب‌گذاری گفته‌ها ۱۵۲
۲-۸- استخراج موجودیت نامگذاری شده از متن ۱۵۶
۳-۸- استفاده از پایگاه‌دادهای زبانی WordNet ۱۵۹
۱-۳-۸- استفاده از مثال کتابخانه JAWS WordNet ۱۵۹
۴-۸- خوشه‌بندی متن ۱۶۶
۵-۸- هجی کردن صحیح ۱۷۰
۱-۵-۸- کتابخانه GUI ASpell و Jazzy ۱۷۰
۲-۵-۸- الگوریتم هجی کردن Peter Norvig ۱۷۲
۳-۵-۸- توسعه دادن الگوریتم Norvig با استفاده از زوج آماره‌های کلمه ۱۷۶
۶-۸- مدل‌های مارکوف پنهانی ۱۸۰
۱-۶-۸- مدل‌های مارکوف پنهانی تمرینی ۱۸۳
۲-۶-۸- استفاده نمودن از مدل مارکوف تمرین‌شده برای متن برچسب ۱۸۸
منابع فارسی ۱۹۳
منابع انگلیسی ۱۹۳

پیشگفتار

هوش مصنوعی **(artificialintelligence)** را باید عرصه پهناور تلاقی و ملاقات بسیاری از دانش‌ها، علوم، و فنون قدیم و جدید دانست. ریشه‌ها و ایده‌های اصلی آن را باید در فلسفه، زبان‌شناسی، ریاضیات، روان‌شناسی، نورولوژی، و فیزیولوژی نشان گرفت. نام هوش مصنوعی در سال ۱۹۶۵ میلادی به عنوان یک دانش جدید ابداع گردید. البته فعالیت درزمینه این علم از سال ۱۹۶۰ میلادی شروع شده بود.

هوش مصنوعی که همواره هدف نهایی دانش رایانه بوده‌است، اکنون در خدمت توسعه علوم رایانه نیز است. زبان های برنامه نویسی پیشرفته، که توسعه ابزارهای هوشمند را ممکن می‌سازند، پایگاههای داده‌ای پیشرفته، موتورهای جستجو، و بسیاری نرم‌افزارها و ماشینها از نتایج پژوهش‌های هوش مصنوعی بهره می‌برند.

یکی از زبان‌های برنامه‌نویسی که وجود دارد **JAVA** نام دارد. این زبان برنامه‌نویسی یک زبان برنامه‌نویسی شئ‌گراست که برای اولین بار توسط جیمز گاسلینگ در شرکت سان مایکروسیستمز ایجاد شد و در سال ۱۹۹۵ به عنوان بخشی از سکوی جاوا منتشر شد. زبان جاوا شبیه به **++C** است اما مدل شیءگرایی آسان‌تری دارد و از قابلیت‌های سطح پایین کمتری پشتیبانی می‌کند. یکی از قابلیت‌های اصلی جاوا این است که مدیریت حافظه را بطور خودکار انجام می‌دهد.

یکی از ویژگی‌های جاوا قابل حمل بودن آن است. یعنی برنامه نوشته شده به زبان جاوا باید به طور مشابهی در کامپیوترهای مختلف با سخت‌افزارهای متفاوت اجرا شود؛ و باید این توانایی را داشته باشد که برنامه یک بار نوشته شود، یک بار کامپایل شود و در همه کامپیوترها اجرا گردد. به این صورت که کد کامپایل شده جاوا را ذخیره می‌کند، اما نه به‌صورت کد ماشین بلکه به‌صورت بایت‌کد جاوا. دستورالعمل‌ها شبیه کد ماشین هستند، اما با ماشین‌های مجازی که به طور خاص برای سخت‌افزارهای مختلف نوشته شده‌اند، اجرا می‌شوند. در نهایت کاربر از سکوی جاوا نصب شده روی ماشین خود یا مرورگر وب استفاده می‌کند. کتابخانه‌های استاندارد یک راه عمومی برای دسترسی به ویژگی‌های خاص فراهم می‌کنند. مانند گرافیک، نخ‌کشی و شبکه. در بعضی از نسخه‌های ماشین مجازی جاوا، بایت‌کدها می‌توانند قبل و در زمان اجرای برنامه به کدهای محلی کامپایل شوند. فایده اصلی استفاده از بایت‌کد، قسمت کردن است. اما ترجمه کلی یعنی برنامه‌های ترجمه شده تقریباً همیشه کندتر

www.SourceSara.com سایت سورس سرا - آموزش برنامه نویسی

از برنامه‌های کامپایل شده محلی اجرا می‌شوند. این شکاف می‌تواند با چند تکنیک خوش‌بینانه که در کاربردهای **JVM** ، کم شود. یکی از این تکنیک‌ها **JIT** است که بایت‌کد جاوا را به کد محلی ترجمه کرده و سپس آن را پنهان می‌کند. در نتیجه برنامه خیلی سریع‌تر نسبت به کدهای ترجمه شده خالص شروع و اجرا می‌شود .بیشتر**VM**های پیشرفته، به‌صورت کامپایل مجدد پویا، در آنالیز**VM**، رفتار برنامه اجرا شده و کامپایل مجدد انتخاب شده و بهینه‌سازی قسمت‌های برنامه، استفاده می‌شوند. کامپایل مجدد پویا می‌تواند کامپایل ایستا را بهینه‌سازی کند. زیرا می‌تواند قسمت**hotspot** برنامه و گاهی حلقه‌های داخلی که ممکن است زمان اجرای برنامه را افزایش دهند را تشخیص دهد. کامپایل **JIT** و کامپایل مجدد پویا به برنامه‌های جاوا اجازه می‌دهد که سرعت اجرای کدهای محلی بدون از دست دادن قابلیت انتقال افزایش پیدا کند.

تکنیک بعدی به عنوان کامپایل ایستا شناخته شده‌است؛ که کامپایل مستقیم به کدهای محلی است مانند بسیاری از کامپایلرهای قدیمی. کامپایلر ایستای جاوا، بایت‌کدها را به کدهای شی محلی ترجمه می‌کند.

کارایی جاوا نسبت به نسخه‌های اولیه بیشتر شد. در تعدادی از تست‌ها نشان داده شد که کارایی کامپایلر**JIT** کاملاً مشابه کامپایلر محلی شد. عملکرد کامپایلرها لزوماً کارایی کدهای کامپایل شده را نشان نمی‌دهند. یکی از پیشرفت‌های بی نظیر در در زمان اجرای ماشین این بود که خطاها ماشین را دچار اشکال نمی‌کردند. علاوه بر این در زمان اجرای ماشینی مانند جاوا وسایلی وجود دارد که به زمان اجرای ماشین متصل شده و هر زمانی که یک استثنا رخ می‌دهد، اطلاعات اشکال زدایی که در حافظه وجود دارد، ثبت می‌کنند.

آنچه که دراین کتاب به آن می پردازیم برنامه‌نویسی بصورت کاربردی برای هوش مصنوعی در زبان **JAVA** می‌باشد .این کتاب شامل یک سری از مثال‌هایی می‌باشد که بصورت کد شده آورده‌ام و کد تست شده ونوشته شده در جاوا نیز در سی دی کتاب می باشد.این کتاب مخصوص دانشجویان مقطع کارشناسی ارشد ودکتری مهندسی کامپیوتر گرایش هوش مصنوعی وهمچنین دانشجویان کارشناسی ارشد ودکتری رشته مهندسی مکاترونیک گرایش طراحی ربات‌ها وسیستم‌های مکاترونیکی می‌باشد. قابل ذکر است که کسانی تمایل دارند درزمینه هوش توزیع شده فعالیت کنند نیز می‌توانند ازاین کتاب استفاده کنند.

مطالبی که دراین کتاب به آن می‌پردازیم به شرح زیر می‌باشد:

فصل ۱: در این فصل مطالعاتی درمورد هیستوریک در دو دامنه شبکه‌های ۲بعدی (مثال‌های مارپیچ) و گراف‌ها انجام شده است.

فصل۲: دراین فصل نمایش دانش و منطق و استدلال با استفاده از سیستم **Power-Loom** شرح داده شده است.

فصل ۳: این فصل به**SemanticWeb** پرداخته است. شما دراین فصل نحوه استفاده از داده‌های **RDF** و **RDFS** را برای نمایش دانش و نحوه استفاده از سیستم معروف **Semantic Web** منبع باز **Sesame** یاد خواهیدگرفت.

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

فصل۴: فصل چهارم شما را با سیستم‌های قانون‌مدار یا تولید(سیستم‌های خبره) آشنا می‌سازد. مادراین فصل از سیستم **Drools** استفاده خواهیم کرد تا سیستم‌های ساده تخصصی را برای حل مسایل دنیای بلوک‌ها و برای شبیه‌سازی سیستم کمکی بهره میگیریم.

فصل ۵: این فصل رابه الگوریتم‌ها ژنتیک اختصاص دادم، و از کتابخانه موجود در جاوا کمک گرفتم تا مسئله آزمایشی را حل کند. این فصل با پیشنهادهایی برای پروژه‌هایی پایان می‌یابد که برای یادگیری شما درمورد الگوریتم ژنتیک در جاوا وکتابخانه‌های جاوا مؤثر می‌باشد.

فصل۶: شبکه‌های عصبی **Hopfield** و **BackPropagation** معرفی می کندو علاوه بر کتابخانه‌های جاوا که شما می‌توانید در پروژه‌های شخصی خود استفاده کنید، ما از دو نرم‌افزار کاربردی جاوای **swing** محور استفاده خواهیم کرد تا متوجه بشویم که شبکه‌های عصبی چگونه آموزش داده می‌شوند.

فصل ۷: شما را با پروژه **GPLedWeka** آشنا می‌سازد. **Weka** بهترین **toolkit** برای حل محدوده وسیعی از مسایل یادگیری ماشین می‌باشد که دراین فصل به آن می‌پردازیم.

فصل۸: چندین تکنیک پردازش زبان طبیعی آماری (**NLP**) را دراین فصل توضیح دادم پردازش متن (علامت‌گذاری، شاخه داشتن، و تعیین نمودن قسمتی از درس دادن)، استخراج موجودیت نامگذاری‌شده از متن، استفاده از پایگاه داده‌ای واژگانی **WordNet**، واگذاری خودکار منگوله‌ها به متن، دسته‌بندی متن، ۳ رویکرد متفاوت برای درست نوشتن و خودآموز کوتاه روی مدل‌های مارکوف نیز شرح داده شده است.

برای اینکه بتوانید از مطالب و مثال‌های این کتاب به خوبی، بهره ببرید از **CD** کتاب که مثال‌های نوشته شده کتاب به زبان جاوا درآن موجود است استفاده کنید.

لازم است که من پشاپیش به دلیل اشتباهات لفظی ومعنایی احتمالی، پوزش بطلبم. وامیدوارم که مطالب این کتاب گام مؤثری در رشد علمی ویادگیری شما در زمینه هوش مصنوعی و برنامه‌نویسی هوش مصنوعی باشد.

باتشکر
کاظم محمدی
زمستان۱۳۹۳

فصل اول

هیستوریک

مقدمه

تحقیقات اولیه در **AI** بر بهینه‌سازی الگوریتم‌های جستجو تأکید می‌کند. این رویکرد مفاد بسیاری را ایجاد نموده چون کارهای **AI** بسیاری می‌توانند بگونه‌ای مؤثر با تعریف نمودن فضاهای وضیعت و استفاده از الگوریتم‌های جستجو برای تعریف و جستجوی درخت‌های جستجو در این فضای وضعیت حل شوند. مسائل جستجو با استفاده از هیوریستک‌های حل شده ساخته شدند تا از عرصه‌های جستجورا در درخت‌های جستجو محدود نماید. این کاربرد از هیوریستیک‌ها مسایل حل نشده را به مسایل قابل حلی با به خطر انداختن کیفیت راه‌حل‌ها تبدیل می‌کند؛ این تبادل در پیچیدگی محاسباتی کمتر، برای راه‌حل بهینه‌ای یک الگوی طراحی استاندارد برای برنامه‌ریزی در**AI** شده‌است. در این فصل به راه‌حل بهینه که برای محاسبات سریع‌تر و نتایج بهتر می‌پردازیم؛ اغلب، با دسته‌بندی نمودن داده‌های اضافی می‌توانیم زمان جستجو را سریع‌تر نماییم و جستجوهای آتی را در همان فضای جستجو حتی کارآمدتر نماییم. حال سؤال این است که محدودیت‌های جستجو چیست؟ در اوایل، جستجوی بکار رفته برای مسایلی مانند بررسی کننده و شطرنج بودند که محققان را در درک نمودن دشواری نوشتن نرم‌افزار به اشتباه انداخت که کارهایی را در دامنه‌هایی اجرا می‌کند که نیازمند به دانش دنیای کلی هستند یا برای محیط‌های پیچیده ودر حال تغییراقدام‌های لازم را انجام می‌دهند. این انواع مسایل معمولاً نیازمند به درک و آنگاه پیاده‌سازی دانش خاص و ویژه دامنه می‌باشد. دراین فصل،ما از۳ دامنه مسئله جستجو برای بررسی نمودن الگوریتم‌های جستجو استفاده خواهیم نمود که عبارتند از: یافتن مسیر در یک مارپیچ، یافتن مسیر در یک گراف، و جستجوس آلفا-بتا در بازهای شطرنج و **tic-tac-toe**.

۱-۱- نمایش فضای وضعیت جستجو و عملیات‌های جستجو:

از یک نمایش درخت جستجو در مثال‌های گراف و مارپیچ در این فصل استفاده می‌کنیم. درخت‌های جستجو مرکب از گره‌هایی هستند که مکان‌هایی را در فضای وضعیت و اتصال‌ها به دیگر گره‌ها تعریف می‌کنند. برای برخی از مسایل کوچک، درخت جستجو می‌تواند به آسانی در حالت ایستا تعیین و مشخص شود، برای مثال، زمان اجرای جستجو در مارپیچ‌های بازی، ما می‌توانیم درخت جستجو را

www.SourceSara.com سایت سورس سرا - آموزش برنامه نویسی

برای کل فضای وضعیت مارپیچ محاسبه و ذخیره نماییم. برای مسایل بسیاری، کامل برشمردن درخت جستجو برای فضای وضعیت کار غیرممکنی است پس باید عملگرهای جستجورا برای گره متوالی تعریف کنیم که برای یک گره مفروض همه گره‌هایی را تولید می‌کند که می‌توانند از گره کنونی در یک گام جستجو شوند؛ برای مثال، در بازی شطرنج ما احتمالاً نمی‌توانیم درخت جستجو را برای همه بازی‌های احتمالی شطرنج برشمریم، پس ما عملگر جستجو برای گره متوالی را تعریف می‌کنیم که با توجه به یک موقعیت صفحه (نمایش داده شده توسط گرهی در درخت جستجو) همه حرکت‌های احتمالی را یا برای خانه‌های سفید یا سیاه محاسبه می‌کند. حرکت‌های احتمالی شطرنج توسط عملگر جستجوی گره بعدی محاسبه می‌شوند و توسط گره‌های تازه محاسبه‌شده‌ای نمایش داده می‌شوند که به گره قبلی متصل می‌شوند. توجه کنید که حتی زمانیکه شمردن همه درختهای جستجو، همانند مثال مارپیچ بازی، کار ساده‌ای است، ما هنوز باید بخواهیم تا درخت جستجو را به طور پویا تولید کنیم همان‌طور که در این فصل انجام خواهیم داد.

برای محاسبه درخت جستجو از گراف استفاده می‌کنیم. گراف‌ها را به صورت گره‌ای با اتصال‌هایی بین برخی از گره‌ها نمایش خواهیم داد. برای حل کردن پازل‌ها و برای بازی مرتبط با جستجوی مرتبط، موقعیت‌هایی را در فضای جستجو با اشیای جاوا به نام گره‌ها نمایش خواهیم داد. گره‌ها حاوی آرایه‌هایی از مرجع‌ها هم برای گره‌های والد و هم فرزند می‌باشند. فضای جستجو استفاده‌کننده از این نمایش گره می‌توانند به عنوان گراف یا درخت جهت داری دیده شود. گره‌ای که هیچ گره والدی ندارد، گره ریشه می‌باشد و همه گره‌هایی که هیچ گره بچه‌ای ندارند، گره برگ نامیده می‌شوند. عملگرهای جستجو برای حرکت از یک نقطه در فضای جستجو به نقطه دیگری استفاده می‌شوند. ما فضاهای جستجوی کوانتایی در این فصل بررسی می‌کنیم، اما فضاهای جستجو نیز می‌توانند در برخی نرم‌افزارهای کاربردی مستمر باشند. اغلب فضاهای جستجو یا خیلی بزرگ هستند یا بی‌نهایت می‌باشند. در این مورد، ما به طور ضمنی فضای جستجو استفاده‌کننده از برخی الگوریتم هابرای توسعه دادن فضا از موقعیت مرجع در این فضا تعریف می‌کنند. شکل ۱.۱ نمایش‌هایی را از فضای جستجو هم به صورت گره‌های متصل در گراف و هم به صورت شبکه دوبعدی با بردارهای نماینده حرکت احتمالی از نقطه مرجع علامت‌گذاری شده توسط **R** نشان می‌دهد.

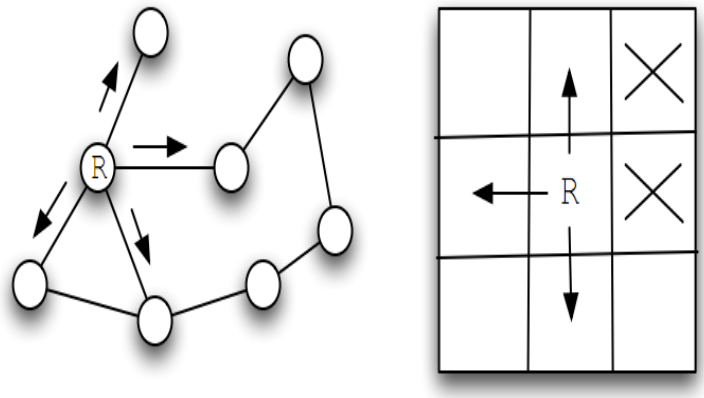
زمانیکه ما فضای جستجو را به صورت آرایه دو بعدی مشخص می‌کنیم، عملگرهای جستجو از نقطه مرجع در فضای جستجو از مکان شبکه مخصوصی به مکان شبکه مجاور حرکت خواهند نمود. برای برخی از نرم‌افزارهای کاربردی، عملگرهای جستجو به حرکت به بالا/پایین/چپ/راست محدود می‌شوند و در دیگر نرم‌افزارهای کاربردی عمگرها به طور مضاعفی از روی قطر به مکان مرجع حرکت می‌کنند.

زمانیکه ما فضای جستجو استفاده کننده از نمایش گره را مشخص می‌کنیم، عمگرهای جستجو می‌توانند از نقطه مرجع به سمت پایین به هر گره بچه یا به سمت بالا به گره والد حرکت نمایند. برای فضاهای جستجو که به طور ضمنی ارائه می‌شوند، عملگرهای جستجو نیز مسئول تعیین کردن گره‌های بچه قانونی، اگر باشد، از نقطه مرجع می‌باشند.

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

۲-۱- یافتن مسیرها در مارپیچ‌ها

برنامه مثال مورد استفاده در این بخش در دایرکتوری **scr/search/maze** برابر است با **MazeSearch.java** و ما فرض می‌کنیم که شما کل مثال فایل موجود در **CD**را برای این کتاب که درسی دی موجود است و فایل‌های منبع را در مکان راحت و مناسبی قرار داده‌اید.



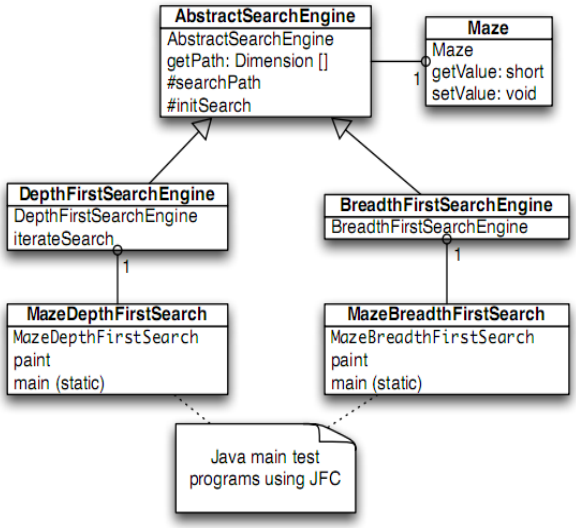
شکل۱-۱:نمایش گراف جهت دار

درشکل ۱.۱ نمایش گراف جهت دار در سمت چپ نشان داده شده است و نمایش شبکه دو بعدی (یا مارپیچ) در سمت راست نشان داده شده است. در هر دو نمایش، حرف **R** برای نمایش موقعیت کنونی (یا نقطه مرجع) استفاده می‌شود و نوک بردار حرکت‌های قانونی تولیدشده توسط عملگر جستجو را نمایش می‌دهند. در نمایش مارپیچ، دو سلول شبکه علامت‌گذاری‌شده با **X** نشان می‌دهند که عملگر جستجو نمی‌تواند این محل شبکه را تولید نماید.

شکل ۱.۲ نمودار کلاس **UML** را برای کلاس‌های جستجو مارپیچ نشان می‌دهد.جستجو برای اولین عمق و اولین وسعت می باشد. کلاس مبنای انتزاعی **Abstract Search Engine** حاوی کد و داده‌های مشترک و رایجی است که مورد نیاز هر دو کلاس **Depth First Search** و **Breadth First Search** می‌باشد. کلاس **Maze** برای ثبت داده‌ها برای مارپیچ دو بعدی استفاده می‌شود، که از جمله آن مکان‌های شبکه‌ای حاوی دیواره یا موانع می‌باشند. کلاس**Maze** ۳ مقدار صحیح کوتاه ایستای مورد استفاده را برای نشان دادن موانع، محل شروع و محل پایان را تعریف می‌کند. کلاس **Maze** جاوا فضای جستجو را تعریف می‌کند. این کلاس آرایه دو بعدی را با اعداد صحیح کوتاهی تخصیص می‌دهد تا وضعیت هر محل شبکه را در **Maze** ارائه دهند. هر زمانیکه ما نیاز داریم تا یک زوج عدد صحیح را ذخیره کنیم، ما از نمونه کلاس جاوا استاندارد **java.qwt.Dimention** استفاده خواهیم نمود، که دو مولفه داده‌ای صحیحی را داراست: ۱عرض و۲ ارتفاع. هر زمانیکه ما نیاز داشته باشیم تا محل شبکه **x-y** را ذخیره کنیم، ما شیء با بعد جدیدی را ایجاد می‌کنیم (اگر مورد نیاز باشد)، و مختصات **x** را در

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

عرض.بعد و مختصات y را در ارتفاع ذخیره می‌کنیم. همانطور که در سمت راست شکل ۱.۲ دیده می‌شود، عملگر برای حرکت روی فضای جستجو از مختصات‌های مفروض x-y انتقال به هر محل شبکه مجاوری تهی است را مجاز می‌داند. کلاس Maze نیز حاوی محل x-y برای مکان شروع (start Loc) و محل مقصد (goalLoc) می‌باشد. توجه کنید که برای این مثال، کلاس Maze مکان شروع را برای مختصات‌های شبکه 0-0 تنظیم می‌کند (گوشه سمت چپ بالاتر مارپیچ برای پیگیری ویژگی‌ها) و گره نهایی در (عرض-۱)- (ارتفاع-۱) (گوشه سمت راست پایین‌تر با ویژگی‌های زیر).



شکل۲-۱: نمودار کلاس UML برای کلاس‌های جاوا درجستجو مارپیچ

کلاس انتزاعی و مجرد Abstract Search Engine کلاس مبنایی برای هر دو کلاس جستجو اولین عمق (از یک دسته برای ذخیره‌سازی حرکت‌ها استفاده می‌کند) Depth First Search Engine و کلاس جستجو اولین وسعت (از یک صف را برای ذخیره‌سازی حرکت‌ها استفاده می‌کند) Breadth First Search Engine می‌باشد. ما با داشتن نگاهی به داده‌های مشترک و رفتار تعریف‌شده در Abstract Search Engine کار را آغاز می‌کنیم. سازنده کلاس دارای دو استدلال مورد نیاز می‌باشد: ۱(عرض و۲) ارتفاع مارپیچ، که در سلول‌های شبکه اندازه‌گیری شدند. سازنده نمونه‌ای را از کلاس مارپیچ با اندازه مطلوب را تعریف می‌کند و آنگاه روش مطلوبیت initSearch را برای تخصیص اشیای آرایه searchPathofDimension فراخوانی نماید، که برای ثبت مسیر پیمایش‌شده در کل مارپیچ استفاده خواهد شد. کلاس مبنای انتزاعی نیز دیگر روش‌های مطلوب را بصورت زیر تعریف می‌کند:

- هم‌مرتبه‌ها (بعد۱، بعد۲)- بررسی کنید ببینید که اگر دو استدلال از نوع بعد مشابه هستند.
- getPossibleMoves (محل بعد)- آرایه‌ای از اشیای بعد را بازمی‌گرداند و می‌توانند برای شکل‌دهی محل معین و مشخصی حرکت داده شوند. این کار عملگر حرکت را پیاده‌سازی می‌کند. اکنون ما به

www.SourceSara.com سایت سورس سرا - آموزش برنامه نویسی

رویه جستجوی اولین عمق نگاهی خواهیم داشت. سازنده برای کلاس مشتق‌شده **DepthFirstSearchEngine** سازنده کلاس مبنا را فراخوانی می‌کند و آنگاه مسئله جستجو را با فراخوانی روش **iterateSearch** حل می‌کند. ما به طور مفصل نگاهی به این روش می‌اندازیم. استدلال‌ها برای **iterate Search** محل کنونی و عمق جستجوی کنونی را مشخص و معین می‌سازد:

```
private void iterateSearch(Dimension loc, int depth)
```

متغیر کلاس **isSearching** برای متوقف نمودن جستجو استفاده می‌شود، که از حل‌های بیشتری اجتناب می‌کند، به محض اینکه یک مسیر به هدف یافت می‌شود.

```
if (isSearching == false) return;
```

مقدار مارپیچ را با عمقی برای تنها اهداف نمایش تنظیم می‌کنیم:

```
maze.setValue(loc.width, loc.height, (short)depth);
```

در اینجا، روش **getPossibleMoves** کلاس برتر را برای گرفتن هر آرایه از مربع‌های همسایه احتمالی استفاده می‌کنیم که می‌توانستیم با او حرکت کنیم؛ آنگاه روی ۴ حرکت ممکن، حلقه تشکیل می‌دهیم (مقدار پوچ در آرایه حرکت غیرقانونی را نمایش می‌دهد):

```
Dimension [] moves = getPossibleMoves(loc);
for (int i=0; i<4; i ++)        {
if (moves[i] == null) break; // out of possible moves
                                // from this location
```

حرکت بعدی را در آرایه مسیر جستجو ثبت کنید و بررسی کنید ببینید آیا انجام می‌شود:

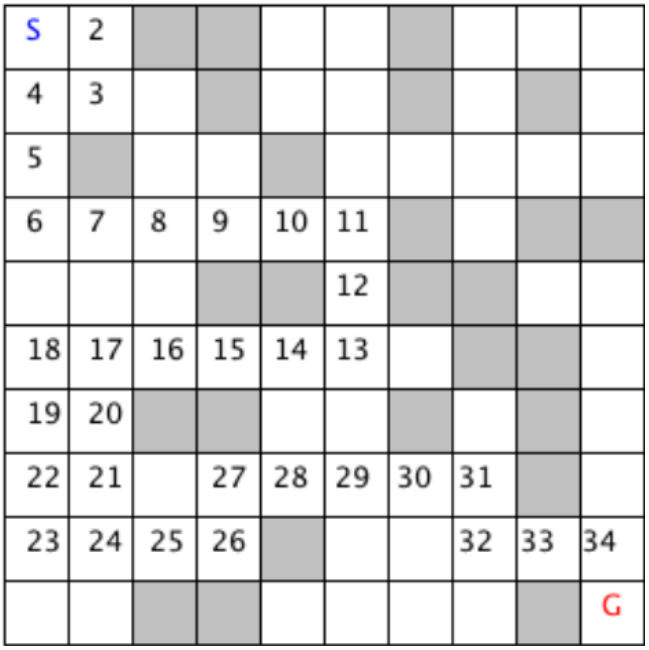
```
searchPath[depth] = moves[i];
if (equals(moves[i], goalLoc))      {
System.out.println("Found the goal at+ "
                        moves[i].width+
 " ,“ + moves[i].height);
isSearching = false;
maxDepth = depth;
return;
} else {
```

اگر حرکت بعدی احتمالی حرکت نهایی نیست، به طور بازگشتی دوباره روش **iterate Search** را فراخوانی می‌کنیم، اما از این مکان جدید کار را آغاز می‌کنیم و عمق در تلاقی با دیگری را افزایش می‌دهیم:

```
iterateSearch(moves[i], depth + 1);
if (isSearching == false)     return;
```

www.SourceSara.com
سایت سورس سرا - آموزش برنامه نویسی

}



شکل ۳-۱: استفاده از جستجوی اولین عمق برای یافتن عمق در مارپیچ راه‌حل غیربهینه را می‌یابد.

شکل ۱.۳ نشان می‌دهد جستجوی اولین عمق چقدر ضعیف می‌تواند مسیری را بین محل‌های آغاز و پایان در مارپیچ بیابد.

مارپیچ یک شبکه ۱۰ در ۱۰ است. حرف **S** محل آغاز را در گوشه بالاتر سمت چپ علامت‌گذاری می‌کند و موقعیت نهایی با **G** در گوشه سمت راست پایین‌تر شبکه علامت‌گذاری می‌شود. سلول‌های بسته شبکه با رنگ خاکستری کمرنگ نقاشی رنگ می‌شوند. مسئله پایه‌ای با جستجوی اولین عمق این است که موتور جستجو اغلب جستجو نمودن را در جهت بدی آغاز خواهد نمود اما هنوز انتهای مسیر را حتی با توجه به آغازی ضعیف می‌یابد. مزیت جستجوی اولین عمق نسبت به جستجوی وسعت این است که جستجوی اولین عمق نیازمند به حافظه خیلی کمتری می‌باشد. خواهیم دید که حرکت‌های ممکن برای جستجوی اولین عمق روی پشته ذخیره می‌شوند (آخرین ساختار داده‌ای که وارد می‌شود، ابتدا خارج می‌شود) و حرکت‌های ممکن برای جستجوی اولین وسعت در صف ذخیره می‌شوند (اولین ساختار داده‌ای وارد شود، ابتدا خارج می‌شود).

کلاس مشتق‌شده **BreadthFirstSearch** مشابه با رویه **DepthFirstSearch** با یک تفاوت عمده مشابه است. از محل جستجو مشخص‌شده، همه حرکت‌های ممکن را محاسبه می‌کنیم، و حرکت آزمایشی ممکن را در یک زمان می‌سازیم. ساختار داده‌ای صف را برای ذخیره کردن حرکت‌های احتمالی استفاده می‌کنیم، که حرکت‌های احتمالی را روی پشت سر صف قرار می‌دهیم همان‌طور که آنها

www.SourceSara.com سایت سورس سرا - آموزش برنامه نویسی

محاسبه می‌شوند، و حرکت‌های آزمایشی را از جلوی صف به طرف خود می‌کشند. اثر جستجوی اولین وسعت این است که به طور یکنواخت از گره آغازگر منتشر می‌شود تا اینکه گره نهایی یافت می‌شود. سازنده کلاس برای **BreadthFirstSearch** سازنده کلاس سوپر فراخوانی می‌کند تا مارپیچ را آغاز نماید و آنگاه از روش کمکی **doSearchOn2Dgrid** برای اجرای جستجوی اولین وسعت برای آرمان استفاده می‌کند. نگاهی به کلاس **BreadthFirstSearch** با کمی جزییات خواهیم داشت. جستجوی اولین وسعت به جای پشته از صف استفاده می‌کند (جستجوی اولین عمق) تا حرکت‌های احتمالی را ذخیره می‌کنند. کلاس **DimensionQueue** ساختار داده‌های استاندارد پیاده‌سازی می‌کند که نمونه‌هایی از بعد کلاس را پیش می‌برد. روش **doSearchOn2Dgrid** بازگشتی نمی‌باشد، از حلقه‌ای برای افزودن موقعیت‌های جدید جستجو تا پایان نمونه کلاس **DimensionQueue** و برای حذف و آزمون محل‌های جدیدی از جلوی صف استفاده می‌کند. آرایه دو بعدی، **allReadyVisited** ما را از جستجوی دوباره همان محل باز می‌دارد. جهت محاسبه کوتاه‌ترین مسیر بعد از اینکه هدف یافت شود، از آرایه قبلی و پیشین استفاده می‌کنیم:

```
private void doSearchOn2DGrid ()   {
    int width = maze.getWidth();
    int height = maze.getHeight();
    boolean alReadyVisitedFlag [] []=
                new boolean[width][height];
Dimension predecessor [] []=
                    new Dimension[width][height];
DimensionQueue    queue=
                new DimensionQueue();
for (int i=0; i<width; i++)     {
    for (int j=0; j<height; j++)  {
alReadyVisitedFlag [i] [j]  =  false;
predecessor[i] [j]  =  null;

        }
}
```

جستجو را با تنظیم پرچم از قبل دیده شده برای محل آغاز حلقه تا مقدار حقیقی و افزودن محل آغاز به انتهای صف شروع می‌کنیم:

```
alReadyVisitedFlag[startLoc.width][startLoc.height]
        = true;
queue.addToBackOfQueue(startLoc);
Boolean   success  =  false;
```

www.SourceSara.com سایت سورس سرا - آموزش برنامه نویسی

در شیء **Dimension** در جلوی صف نگاه سریعی می‌اندازیم و محل‌های مجاور را به موقعیت کنونی در مارپیچ اختیار می‌کنیم:

```
Dimension head = queue.peekAtFrontOfQueue();
Dimension [] connected =
                    getPossibleMoves(head);
```

روی هر حرکت ممکن حلقه می‌نویسیم، اگر حرکت احتمالی معتبر باشد (یعنی پوچ نیست) و اگر از قبل محل حرکت احتمالی را بازدید نکرده باشیم؛ آنگاه حرکت احتمالی را به انتهای صف اضافه می‌کنیم و آرایه قبلی و پیشین برای محل‌های جدیدی برای آخرین مربع بازدید شده تنظیم می‌کنیم (نوک مقدار از جلوی صف می‌باشد). اگر هدف را بیابیم، حلقه را می‌شکنیم:

```
for (int i=0; i<4; i ++)    {
   if (connected[i]  ==  null) break;
   int w = connected [i].  width;
   int h = connected [i].  height;
   if (alReadyVisitedFlag[w][h] == false )    {
      alReadyVisitedFlag [w] [h] = true;
      predecessor [w] [h] = head;
      queue.addToBackOfQueue(connected[i]);
      if (equals(connected[i], goalLoc))        {
         success = true;
break outer;   //   we are done
    }
  }
}
```

محل را در جلوی صف پردازش می‌کنیم (در نوک متغیر)، پس آن را حذف می‌کنیم:

```
queue.removeFromFrontOfQueue();
}
```

اکنون که خارج از حلقه اصلی هستیم، نیاز داریم تا از آرایه قبلی استفاده کنیم تا کوتاه‌ترین مسیر را اختیار کنیم. توجه کنید که آرایه **searchPath** را با ترتیبی معکوس که از محل هدف شروع می‌شود، پر کنیم:

```
maxDepth = 0;
if (success)      {
      searchPath[maxDepth++] = goalLoc;
      for (int i=0; i<100; i++)       {
```