**فهرست مطالب**

[مقدمه 2](#_Toc534353450)

[عامل و محیط 2](#_Toc534353451)

[ساختار مسائل و راه حل ها 3](#_Toc534353452)

[نحوه پیاده سازی، روش ها و الگوریتم ها 5](#_Toc534353453)

[کلاس های اصلی و منبع 6](#_Toc534353454)

[کلاس مسئله 6](#_Toc534353455)

[کلاس گره 8](#_Toc534353456)

[کلاس حالت 9](#_Toc534353457)

[کلاس عمل 9](#_Toc534353458)

[جستجوگرها 9](#_Toc534353459)

[جستجوگر ابسترکت 9](#_Toc534353460)

# مقدمه

عامل های هوشمند واکنشی ساده، در محیط هایی که تعداد حالات آن زیاد است و در مسائلی که به مسائل جهان واقعی نزدیک است نمی توانند بازدهی مناسبی داشته باشند و در نتیجه دیگر سادگی آن ها کارآمد نخواهد بود. برای حل مسائل واقعی، می توانیم از عامل های مبتنی بر هدفی که اعمال و اقدامات خود را پیش از انجام مورد سنجش قرار می دهند استفاده کنیم. این عامل ها، حالت های محیط خود را به دید اتمیک نگاه می کنند؛ به این معنی که در هر حالت، حالت فعلی دیگر قابل تقسیم به جزء های کوچک تر نیست. در این جا به منظور سادگی فرض می شود که راه حل یک مسئله، همواره دنباله ای ثابت از عملیات مختلف است؛ عمل های عامل در طول جستجو برای جواب، در اثر ورود ادراکات مختلف عوض نخواهند شد. با این مفروضات، روش ها و الگوریتم هایی که برای جستجوی راه حل در مسائلی از این قبیل در نظر گرفته می شوند می توانند از نوع آگاهانه یا ناآگاهانه باشند که در ادامه به توضیح آن ها پرداخته شده است.

## عامل و محیط

عامل ها می توانند با دریافت ادراکات از محیط، با انجام عمل هایی روی محیط تاثیر گذاشته و حالت آن را عوض کنند و یا اینکه حالت کنونی خود در مقایسه با محیط را تغییر دهند. طبق تعریف عامل های هوشمند، می بایست فعالیت های این عامل ها منجر به بیشینه شدن معیار کارآیی[[1]](#footnote-1) آن ها شود. گاهی اوقات، عامل می تواند با داشتن هدفی برای خود و تلاش برای رسیدن به آن هدف، معیار کارآیی خود را آسان تر به میزان بیشینه برساند. اهداف می توانند با محدود مقاصد عامل از انجام عمل های مختلف، در سامان دهی رفتار عامل کمک به سزایی داشته باشند. در نتیجه فرموله کردن هدف و مقصد مورد نظر - براساس موقعیت و حالت کنونی عامل و معیار کارآیی آن - اولین قدم برای حل مسئله می باشد.

**هدف** برای ما، مجموعه ای از چند حالت محیط می باشد که عامل می تواند با انجام عمل های مختلف خود را در آن حالت(ها) قرار دهد و به مقصود خود دست یابد؛ برای این منظور عامل می بایست مشخص کند که چه عملیاتی باید انجام دهد. پیش از تصمیم درباره عمل خاصی، عامل باید بداند که باید چه نوع عمل هایی را برای انجام و چه حالت هایی از محیط را برای رسیدن باید در نظر بگیرد. فرموله کردن مسئله به معنی تعریف و توصیف دقیق حالت ها و عملیات ممکن با توجه به حالت های هدف در یک محیط است. سطح تجرید عملیات ها و همچنین میزان جزئیات توصیف حالت های محیط می تواند بسته به مسئله متفاوت باشد و باید حتما به این نکته توجه شود که ذکر جزئیات بیش از حد و یا کم تر از میزان قابل قبول می تواند به عدم موفقیت عامل در دستیابی به پاسخ و رسیدن به هدف شود.

**محیط** برای تصمیم گیری در مورد انجام عملیات مختلف، عامل می بایست ابتدا بداند که در حالت کنونی خود مجاز به انجام چه عملیاتی است و نتیجه هر عمل منجر به رسیدن به کدام حالت از محیط می شود؛ سپس با بررسی هر عمل و با در نظر داشتن هدف، می تواند بهترین تصمیم را بگیرد. بنابراین فرض می شود که محیط **کاملا قابل مشاهده** است. همچنین در صورت نامحدود بودن عملیات قابل انجام در هر حالت، امکان تصمیم گیری برای عامل در عمل وجود نخواهد داشت. در نتیجه محیط م یبایست **گسسته** باشد. به این معنی که در هر حالت، تعداد محدودی عمل قابل انجام است. همچنین در فرآیند تصمیم گیری، عامل مورد نظر ما می بایست قادر به پیش بینی نتیجه حاصل از عمل خود باشد. در غیر این صورت قادر به تصمیم گیری نخواهد بود. به عبارت دیگر فرض می شود محیط **شناخته شده** است و عامل می داند که با انجام هر عمل از حالت فعلی به کدام حالت می تواند برود. همچنین عدم قطعیت اعمال به کلی نادیده گرفته می شود و هر عامل در صورت انجام عملی، حتما و الزاما خروجی مربوط به آن عمل را خواهد دید. بنابراین محیط **قطعی** است.

**جستجو** فرآیند یافتن دنباله ای از عملیات که منجر به رسیدن به هدف شود جستجوی راه حل نام دارد. یک الگوریتم جستجوی راه حل، مسئله ای (فضای حالت) را به عنوان ورودی گرفته و راه حل آن مسئله را در قالب دنباله ای از عملیات و اقدامات توصیف کرده و برم یگرداند.الگوریتم های مختلفی برای جستجوی راه حل در محیط های گوناگون وجود دارد که در ادامه به توضیح آن ها می پردازیم. هنگامی که راه حلی پیدا شد، می توان دنباله عملیات ذکر شده در راه حل را انجام داد؛ به این مرحله، مرحله اجرا گفته می شود. بنابراین برای رسیدن به هدف، می بایست ابتدا فرموله بندی مناسبی از مسائل و محیط ارائه دهیم، سپس راه حل را جستجو کنیم و در مرحله آخر، راه حل به دست آمده را برای اجرا به عامل بدهیم.

## ساختار مسائل و راه حل ها

همانطور که در قسمت قبل ذکر شد، به دلیل گرافی بودن فضای حالت مسئله، برای فرموله کردن هر مسئله، گره های گراف را در قالب داده ساختاری به نام گره ذخیره می کنیم که این داده ساختار دارای مشخصه های زیر است:

١. آدرس گره پدر: برای یافتن مسیری که منتهی به این گره شده می بایست آدرس گره پدر را در هر گره ذخیره کنیم. این مشخصه در یافتن دنباله حالت های جستجو شده در راه حل مناسب می باشد.

٢. حالت عامل: حالتی که عامل با رسیدن به گره فعلی در آن قرار می گیرد در این داده ساختار ذخیره می شود.

٣. عمل انجام شده: عملی که با انجام آن به گره فعلی رسیده ایم و گره فعلی در نتیجه انجام آن عمل در حالت گره پدر می باشد. این مشخصه برای به دست آوردن دنباله عملیات انجام شده در طی یک راه حل مناسب می باشد.

۴. هزینه مسیر تا این گره: هزینه ای که تا این رسیدن به این گره پرداخت شده است. این مولفه برای به دست آوردن هزینه مسیر راه حل مناسب است.

همچنین به دلیل گرافی بودن فضای حالت مسئله، می بایست برای هر مسئله موارد زیر را تعریف کنیم:

١. ساختار کلی حالت ها: اینکه حالت ها در قالب ماتریس، عدد، رشته و یا غیره بیان می شوند، می بایست توسط مسئله بیان شود.

٢. حالت اولیه: حالت اولیه در واقع نشان دهنده پیکربندی اولیه و حالت شروع عامل می باشد. این حالت در جستجوی راه حل به عنوان گره آغازین در نظر گرفته می شود و در راه حل پیدا شده نیز اولین گام خواهد بود.

٣. حالت هدف: درواقع اینکه عامل در مورد مسئله مورد نظر به حالت درستی رسیده یا نه توسط این حالت مشخص می شود. مقصد نهایی عامل می بایست اینجا باشد. در مسائلی که نتوان حالت هدف را به طور دقیق بیان کرد می بایست شرایط حالت هدف ذکر شود.

۴. نتیجه هر عمل در حالت های مختلف: از آن جا که فرضکرده ایم که محیط ما کاملا قابل مشاهده است، مسئله باید گویای نتیجه هر عمل (حالت حاصل شده در ازای انجام عمل) در هر حالتی از فضای حالت مسئله باشد.

۵. عملیات ممکن در هر حالت: پیرو مورد قبلی، از آن جا که فرضکرده ایم که محیط ما کاملا قابل مشاهده است، مسئله باید گویای عملیات قابل انجام توسط عامل در هر حالتی از فضای حالت باشد.

۶. هزینه انجام هر عمل در هر حالت: عامل باید بداند که هزینه انجام هر عمل در هر حالتی که باشد چقدر خواهد بود.

٧. هزینه مسیری که تا کنون پرداخت شده: عامل باید بداند که تا کنون و تا رسیدن به حالت فعلی چه هزینه ای پرداخت کرده است.

٨. تابع شهودی هزینه: این تابع که در واقع تخمینی است از هزینه باقی مانده تا رسیدن به حالت هدف، در صورتی باید پیاده سازی شود که بخواهیم از روش های جستجوهای آگاهانه در یافتن پاسخ استفاده کنیم.

در هر پیمایش و جستجوی راه حل، گرهیی ساخته می شود که پدر آن گره فعلی بوده و تمامی خصوصیت های نام برده شده نیز برای آن ست می شوند. اگر در حین جستجوی راه حل به گرهی برسیم که رمقصد مورد نظر عامل باشد، برای به دست آوردن دنباله اعمال انجام شده از حالت اولیه، کافی است از گره انتهایی)مقصد(که در جستجوی راه حل ساخته شده، شروع کرده و تا زمانی که به گره مبدا نرسیده ایم دنباله اعمال و حالت ها را بسازیم.

# نحوه پیاده سازی، روش ها و الگوریتم ها

فریمورک پیاده­سازی به زبان جاوا بوده که از چهار بسته[[2]](#footnote-2) به شرح زیر تشکیل شده است.

* بسته resources: در این بسته، کلاس هایی که به طور مستقیم با پیمایش گراف و ساخته شدن راه حل در ارتباط هستند قرار دارند.
* بسته searcher: هر جستجوگر که با الگوریتمی منحصر به فرد شروع به جستجو در فضای حالت مسئله می کند، در این بسته قرار داده شده است. این بسته حاوی کلاسی تحت عنوان Searcher است که به صورت انتزاعی[[3]](#footnote-3) پیاده سازی شده است. برای ثابت نگه داشتن واسط میان جستجوگرها و فضای حالت مسئله، تمامی جستجوگرها می­بایست از کلاس Searcher به ارث برده شوند[[4]](#footnote-4).
* بسته utilities: شامل کلاس های کمکی درکاربا فریم ورک است. درزمان ویراستاری این گزارش، تنها کلاس GSException موجود است. این بسته بیشتر برای استفاده در آینده و افزودن کلاس های کمکی بیشتر به کار برده می شود. به عنوان مثال می توان در آینده کلاسی تحت عنوان تحلیل گر عملکرد جستجوگرها ساخت و در این بسته قرار داد.
* بسته main: حاوی برنامه اصلی و مدل­های مسئله می­باشد.

توضیحات کلاس های موجود در هر بسته و نحوه پیاده سازی آن ها در ادامه ذکر می شود. شایان ذکر است تمامی توضیحات به صورت کامنت در کد منبع این پروژه نیز قرار دارند و توضیحات ذیل برای شفاف سازی بیشتر است.

## کلاس های اصلی و منبع

همانطور که گفته شد، این کلاس ها به طور مستقیم با پیمایش فضای حالت مسئله و ساخته شدن راه حل ارتباط دارند. همگی این کلاس ها در بسته resources قرار گرفته اند و احتمال دارد که با ورود به فصل های بعدی کتاب و گسترش یافتن انواع مسئله ها و پیچیده تر شدن ساختار راه حل و نحوه جستجو، ساختار بعضی از کلاس ها تغییر یابد.

### کلاس مسئله

این کلاس در فایل Problem.java به صورت ابسترکت پیاده­سازی شده و تنها دارای یک متد غیرابسترکت است. توابع این کلاس عبارتند از:

**تابع حالت اولیه** که در واقع تولیدکننده حالت ابتدایی و آغازین مسئله است. این تابع همچنین به ساختار حالت های مسئله قالب نیز می دهد و نوع داده آن ها را مشخص می کند.

*/\*\*  
 \* Getting the problem's initial state in which we start the search.  
 \*  
 \** ***@return*** *State the initial state.  
 \*/*public abstract State initialState();

**تابع تست هدف** این تابع برای بررسی اینکه آیا حالت داده شده هدف است یا نه مورد استفاده قرار می گیرد.

*/\*\*  
 \* Determine whether the given state is the goal of the problem or not.  
 \*  
 \** ***@param*** *s State to review.  
 \** ***@return*** *boolean the answer.  
 \*/*public abstract boolean goalTest(State s);

**تابع عمل های ممکن در هر حالت** عملیات ممکن در هر حالت توسط این تابع معلوم شده و برگردانده می شود. در این جا تمامی اعمال در نتیجه ظاهر نمی شوند و فقط اعمالی که قابلیت اجرا در حالت ورودی را دارند در نتیجه برگردانده می شوند.

*/\*\*  
 \* The actions list which can be performed while the agent is in state s.  
 \*  
 \** ***@param*** *s State to review.  
 \** ***@return*** *Vector set of actions available in state s.  
 \*/*public abstract List<Action> actions(State s);

**تابع معلوم کننده نتیجه هر عمل در یک حالت** به ازای عمل ورودی و قابل انجام در حالت ورودی، حالت نتیجه شدهدر صورت انجام آن عمل در حالت ورودی برگردانده می شود.

*/\*\*  
 \* Returns the result node of an action performed on agent when it was in state s.  
 \*  
 \** ***@param*** *s State the state in which the agent is.  
 \** ***@param*** *a Action the action to perform.  
 \** ***@return*** *State with parent n.  
 \*/*public abstract State result(State s, Action a);

**تابع هزینه عمل** هزینه انجام یک عمل در حالت ورودی را برمی گرداند. فرض براین است که عمل ورودی در حالت ذکر شده قابل انجام است.

*/\*\*  
 \* Returns the cost of action a in state s.  
 \*  
 \** ***@param*** *s State in which the action will be performed.  
 \** ***@param*** *a Action to be performed  
 \** ***@return*** *double cost of the action.  
 \*/*public abstract double actionCost(State s, Action a);

**تابع هزینه مسیر** هزینه مسیری که از حالت اولیه تا گره فعلی)دربردارنده حالت فعلی(طی شده را برمی گرداند.

*/\*\*  
 \* Returns the path in leaded to node n.  
 \** ***@param*** *n Node the leaf node.  
 \** ***@return*** *double path from root to leaf node n.  
 \*/*public abstract double pathCost(Node n);

**تابع شهودی** فقط باید در صورتی پیاده سازی شود که از جستجوگرهای آگاهانه می خواهیم استفاده کنیم

*/\*\*  
 \* Calculate the heuristic value for state s.  
 \*  
 \** ***@param*** *s {****@link*** *State} the state to evaluate.  
 \** ***@return*** *int heuristic value.  
 \*/*public abstract double heuristic(State s);

**تابع بازگرداننده راه حل** این تابع به صورت غیر ابسترکت پیاده سازی شده و به ازای هر ورودی که از جنس یک گره می باشد، حالت هایی که در هنگام جستجوی راه حل به دست آمده بودند را برمی گرداند. این تابع صرفا یک تابع کمکی است که به دست آوردن راه حل یک لایه تجرید بیشتر داشته باشد و آسان تر باشد. از ذکر جزئیات پیاده سازی این تابع پرهیز می شود؛ در صورت داشتن ابهام به کد منبع پروژه رجوع شود.

### کلاس گره

این کلاس در فایل Node.java پیاده­سازی شده و وظیفه نگهداری اطلاعات مربوط به مسیری که تا حالت کنونی طی شده است را دارد. شرط مساوی بودن یک گره با گره دیگر نیز مساوی بودن حالت های این دو گره است. در برخی جستجوها علاوه بر خصوصیات ذکر شده در قسمت های قبل، نیازمند دانستن عمق گره کنونی در گراف نیز هستیم که این خصوصیت از بررسی تعداد گره های موجود در مسیر منتهی به این گره به آسانی قابل حصول است و در داخل این کلاس نیز به عنوان یک تابع پیاده سازی شده است.

### کلاس حالت

این کلاس در فایل State.java پیاده­سازی شده و دارای یک شی [[5]](#footnote-5) به نام وضعیت [[6]](#footnote-6) و فقط به خاطر ساختارمند کردن هرچه بیشتر به عنوان یک کلاس پیاده سازی شده است. خصیصه وضعیت همان حالت مورد نظر است که به صورت اتمیک است.

### کلاس عمل

این کلاس در فایل Action.java پیاده­سازی شده و تنها دارای دو خصیصه data, cost می باشد که اولی توضیحات و داده های مربوط به عمل است و دومی هزینه انجام عمل می باشد. همانند کلاس حالت، این کلاس نیز صرفا برای ساختارمند کردن بیشتر پروژه، پیاده سازی شده است.

## جستجوگرها

این کلاس­ها در بسته searcher قرار داشته و وظیفه یافتن راه حل برای مسئله پاس داده شده به آن ها در هنگام ساخته شدن را دارند.

### جستجوگر ابسترکت

همانطور که ذکر شد، تمامی جستجوگرها می بایست از کلاس ابسترکتی به نام Searcher به ارث برده شوند. این کلاس در فایل Searcher.java قرار داشته و و دلیل اهمیت آن نیز دارا بودن اینترفیس و واسط بسیار ساده و همچنین خصیصه هایی که مورد نیاز تمامی جستجوگرهاست، می باشد. این کلاس دارای یک متد ابسترکت است که وظیفه آن جستجوی راه حل در مسئله داده شده است. هر جستجوگر می بایست جداگانه و با توجه به الگوریتم مورد نظر خود، این متد را پیاده سازی کند.

*/\*\*  
 \* The actuator function and main responsibility of this class.  
 \*  
 \** ***@return*** *Null if no solution found | {****@link*** *Node} containing a solution.  
 \** ***@throws*** *GSException if the frontier is not initialized.  
 \*/*public abstract Node search() throws GSException, InterruptedException;

برای ایجاد هر نمونه جستجوگر می بایست مسئله را در تابع سازنده به جستجوگر پاس دهیم و سپس متد search() را فراخوانی کنیم. به علاوه، این کلاس دارای متغیرهای دیگری نیز هست که برای شمارش تعداد گره های مشاهده شده، تعدا گره های بسط داده شده و حداکثر حافظه مورد استفاده در طول جستجو به کار برده می شوند.

# آزمایشات و نتایج

## p1a

## p1b

## p1c

## p2a

BFS:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Path Cost: 4.0 #Visited Nodes: 17, #Expanded Nodes: 48, #Maximum Memory Cells Used: 34  
resources.Node@131245a  
0 1 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 0 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 0  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 0  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 6  
7 8 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## p2b

DFS:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Path Cost: 32.0 #Visited Nodes: 33, #Expanded Nodes: 90, #Maximum Memory Cells Used: 59  
resources.Node@131245a  
0 1 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
4 1 2  
0 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
4 1 2  
7 5 3  
0 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
4 1 2  
7 5 3  
8 0 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
4 1 2  
7 0 3  
8 5 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
4 0 2  
7 1 3  
8 5 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 4 2  
7 1 3  
8 5 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
7 4 2  
0 1 3  
8 5 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
7 4 2  
8 1 3  
0 5 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
7 4 2  
8 1 3  
5 0 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
7 4 2  
8 0 3  
5 1 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
7 0 2  
8 4 3  
5 1 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 7 2  
8 4 3  
5 1 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
8 7 2  
0 4 3  
5 1 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
8 7 2  
5 4 3  
0 1 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
8 7 2  
5 4 3  
1 0 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
8 7 2  
5 0 3  
1 4 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
8 0 2  
5 7 3  
1 4 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 8 2  
5 7 3  
1 4 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 8 2  
0 7 3  
1 4 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 8 2  
1 7 3  
0 4 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 8 2  
1 7 3  
4 0 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 8 2  
1 0 3  
4 7 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 0 2  
1 8 3  
4 7 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 5 2  
1 8 3  
4 7 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 5 2  
0 8 3  
4 7 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 5 2  
4 8 3  
0 7 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 5 2  
4 8 3  
7 0 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 5 2  
4 0 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 0 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 0  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 0  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 6  
7 8 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## P2c

DepthLimit = 2

DFS:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
NOT FOUND #Visited Nodes: 22, #Expanded Nodes: 28, #Maximum Memory Cells Used: 15

DepthLimit = 5

DFS:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Path Cost: 4.0 #Visited Nodes: 62, #Expanded Nodes: 82, #Maximum Memory Cells Used: 31  
resources.Node@131245a  
0 1 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 0 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 0  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 0  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 6  
7 8 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## p2d

A\*:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Path Cost: 4.0 #Visited Nodes: 40, #Expanded Nodes: 101, #Maximum Memory Cells Used: 67  
resources.Node@131245a  
0 1 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 0 2  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 0  
4 5 3  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 0  
7 8 6  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 2 3  
4 5 6  
7 8 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## p3a

UCS:   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Path Cost: 12.0 #Visited Nodes: 62, #Expanded Nodes: 304, #Maximum Memory Cells Used: 96  
resources.Node@131245a  
0 0 0 0 0 1 0 0  
0 0 0 0 0 0 0 0  
0 -1 0 0 0 0 0 0  
0 0 0 0 0 -1 0 0  
-1 0 0 -1 0 0 0 0  
0 0 0 0 -1 0 0 0  
0 0 0 0 0 0 -1 0  
0 0 0 0 2 0 0 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
0 -1 0 0 0 0 1 0  
0 0 0 0 0 -1 0 0  
-1 0 0 -1 0 0 0 0  
0 0 0 0 -1 0 0 0  
0 0 0 0 0 0 -1 0  
0 0 0 0 2 0 0 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
0 -1 0 0 0 0 0 0  
0 0 0 0 0 -1 0 0  
-1 0 0 -1 0 0 0 1  
0 0 0 0 -1 0 0 0  
0 0 0 0 0 0 -1 0  
0 0 0 0 2 0 0 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
0 -1 0 0 0 0 0 0  
0 0 0 0 0 -1 0 0  
-1 0 0 -1 0 0 0 0  
0 0 0 0 -1 1 0 0  
0 0 0 0 0 0 -1 0  
0 0 0 0 2 0 0 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
0 -1 0 0 0 0 0 0  
0 0 0 0 0 -1 0 0  
-1 0 0 -1 0 0 0 0  
0 0 0 0 -1 0 0 0  
0 0 0 0 0 0 -1 0  
0 0 0 0 3 0 0 0  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## p3b

## p4

## p5

## p6

1. Performance measure [↑](#footnote-ref-1)
2. package [↑](#footnote-ref-2)
3. abstract [↑](#footnote-ref-3)
4. inherit [↑](#footnote-ref-4)
5. Object [↑](#footnote-ref-5)
6. State [↑](#footnote-ref-6)