هوش مصنوعی

University of Applied Science And Technology

Branch of Telecommunication Company of East Azerbaijan

Winter 2014 – 2015

دانشگاه جامع علمی-کاربردی

واحد مخابرات آذربایجان شرقی (تبریز)

جزوه درسی هوش مصنوعی

مقطع کارشناسی

مدرس: دکتر پدرام اقدسی

P-Aghdasi.ir Pa_phd@live.com

فصل اول

Chapter One

مقدمه ای بر هوش مصنوعی

تعاریف هوش و هوش مصنوعی

هوش: هوش از یکسری اطلاعات اولیه به یک سری نتایج بر اساس یک سری وقایع بدست می آوریم.

بعنوان مثال: ساختن یک کلبه چوبی از چوب کبریت.

هوش: استفاده دانش به نحوی که در محیط مورد نظر بتوانیم بهترین کارائی را نمایش دهیم.

هوش مصنوعی:[1] همانطور که میدانیم هوش انسانها یک هوش ذاتی و طبیعی است. اگر بتوانیم ربات یا نرم افزاری تولید کنیم که عملکردی هوشمندانه از خود نشان دهند و مطمئن باشیم که دارای هوش طبیعی و ذاتی نیست، هوش مصنوعی می گویند.

مانند: ربات های آتش خاموش کن، ربات های فوتبالیست

عامل های هوشمند بر اساس ٤ عامل زیر طراحی و آماده میشوند:

- **عاقلانه فکر کردن:** احتمال خطا صفر، درست فکر کردن
- **مانند انسان فکرکردن:** احتمال خطا وجود دارد (دزدی)
- **عاقلانه عمل کردن:** احتمال خطا صفر (بر روی زمین پوشیده از برف ترمز نمیکند.)
- **مانند انسان عمل کردن:** احتمال خطا وجود دارد (روی زمین برفی ترمز میکند.)

[1] Artificial intelligence

مانند انسان عمل کردن: یعنی از روی احساسات تصمیم گرفتن یعنی هرچند که با منطق سازگار نباشد. مانند عصبانیت دانشجو مقابل استاد وقتی نمره کمتری میگیرد.

رباتی که به این شکل ساخته بشود و عمل کند حتماً در عملکرد دچار خطاهایی خواهد شد، مثال بارزی که در این زمینه مطرح گردیده است **نظریه تورینگ**[۲] است.

مانند انسان فکر کردن: رباتی که به این نحو ساخته شود تئوری وار انجام وظیفه میکند و پا به مرحله عمل نمیگذارد. مانند حل مسئله.

عاقلانه فکر کردن: یعنی صد در صد از محاسبات استفاده کنیم و هیچ نیازی به سخت افزار نداشته باشیم. چون فقط فکر کردن نیاز است.

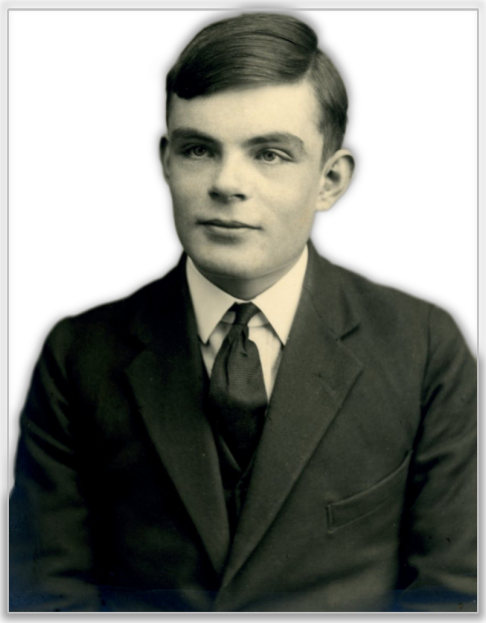
نکته مهم : در این روش هیچگاه خطا وجود نخواهد داشت و همیشه در این مدل انتظار نتیجه ای مطلوب را خواهیم داشت.

عاقلانه عمل کردن: در این نوع ربات ها احساسات کلاً کنار گذاشته شده و بصورت عقلانی و منطقی عمل میکنند، در این روش هم انتظار خطا نخواهیم داشت.

[2] Turing

تست تورینگ

Alan Mathison Turing

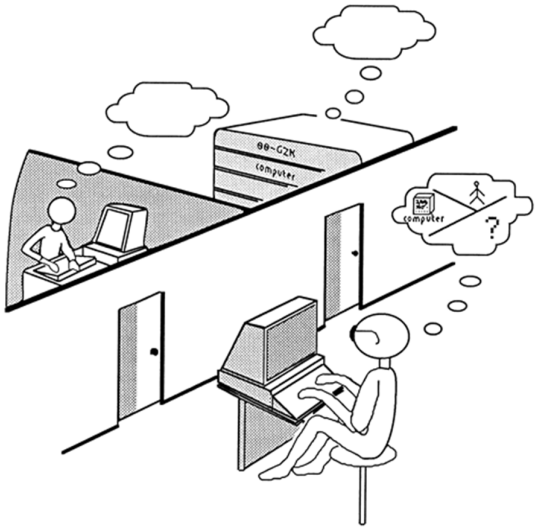


Alan Turing 1912 - 1954

تست تورینگ

شرح تست تورینگ

تست تورینگ[3] آزمایشی است برای انسان گونه عمل کردن. در این تست یک سوالی توسط یک شخص پرسیده میشود و ۲ نفر به این سوال پاسخ میدهند که یک نفر انسان و یک نفر ماشین پاسخ دهنده خواهد بود.



تست تورینگ زمانی به جواب درست یا قطعی خواهد رسید که پاسخ هایی که از این دو دریافت میکند، نتواند تشخیص دهد کدام پاسخ از طرف ربات بوده و کدامیک از طرف انسان.

نیازمندی تست تورینگ

۱. پردازش زبان طبیعی را بمنظور برقراری ارتباط را داشته باشد.
۲. بازنمائی دانش بمنظور ذخیره سازی آن چیز که میبیند و میشنود.
۳. ماشین باید بتواند با شرایط جدید خود را وفق دهد. برای مثال در اتاقی که دمای آن بالاتر از حد مجاز است بتواند تشخیص داده و از اتاق بیرون رود.
۴. باید قابلیت بینایی داشته باشد. مانند تشخیص رنگ دیوار اتاق.
۵. باید علم رباتیک را برای کنترل کردن اشیا در محیط را داشته باشد.
۶. باید قدرت استدلال داشته باشد.

[3] Turing Test

مروری بر مبانی هوش مصنوعی

Fundamentals of Artificial Intelligence

۱ – فلسفه

منظور اینکه بتوانیم از یک سری قوانین یک سری نتایج معتبر را بدست آوریم. به عبارت دیگر توسط فلسفه بتوانیم به چه شکل از فکر کردن انسان گونه بتوانیم نتایج درست را استخراج کنیم. به عبارت دیگر در رابطه با فلسفه چطور بتوانیم یک سری دانش را به عملیات و عمل تبدیل نماییم .

۲ – ریاضیات

توسط علم ریاضی میتوانیم بفهمیم چه چیزی هایی قابل محاسبه می باشد و چه چیزهائی محاسبه شدنی است.

۳ – اقتصاد

ماشینی که طراحی میکنیم چقدر از لحاظ اقتصادی مقرون به صرفه است و بازده اقتصادی آن چقدر خواهد بود. اگر بخواهیم از ماشینی که طراحی کرده ایم سودهی بلند مدت نتیجه بگیریم چه کارهایی را باید انجام دهیم.

٤ – عصب شناسی

باعث میشود بدانیم که مغز انسان به چه روشی اطلاعات را پردازش می کند تا بتوانی آنرا به رباتی که طراحی میکنیم انتقال دهیم.

٥ – زبان شناسی

توسط این علم می توانیم بفهمیم چگونه **زبان** با **تفکر** در ارتباط است، زبان شناسی شامل گرامر می باشد.

۶ – روان شناسی

توسط این علم می توانیم نحوه برخورد با احساسات را به آن ربات وارد کنیم و همچنین تاثیر رنگها بر محیط را به ربات بفهمانیم.

۷ – مهندسی کامپیوتر

توسط این علم برنامه نویسی ماشین های هوشمند را انجام می دهیم، همچنین مهندس می تواند بیان کند ماشین به چه شکلی طراحی شود که حداکثر سرعت را داشته باشد.

۸– نظریه کنترل و سیبرنتیک[۴]

این علم بیان میکند که اشیا چگونه می توانند کنترل خودکار داشته باشند.

4 Cybernetic: Science concerned with the study of systems of any nature which are capable of receiving, storing and processing information so as to use it for control.

تست های فصل اول

۱ – کدامیک از موارد زیر اهداف هوش مصنوعی می باشد.

الف) ساختن سیستمی که در تست تورینگ پذیرفته شود.　　ب) ساختن موجودیت های هوشمند

ج) ساختن عامل های خردمند　　د) پی بردن به نحوه تفکر انسانها

۲– نویسندگان برنامه ی GPS چه رهیافتی را از هوش مصنوعی دریافت کردند؟

الف) تفکر منطقی　　ب) تفکر انسان گونه　　ج) عملکرد منطقی　　د) عملکرد انسانی

۳– تست تورینگ برای ارزیابی سیستم های کامپیوتری در چه رهیافتی از هوش مصنوعی ارائه شد؟

الف) قوانین تفکر　　ب) مدل سازی شناختی　　ج) عملکرد عقلانی　　د) عملکرد انسانی

٤– تعریف زیر از هوش مصنوعی مربوط به چه رهیافتی از هوش مصنوعی است؟

– خودکار سازی فعالیت هایی که با تفکر انسان مرتبط هستند مانند یادگیری و حل مسئله

الف) تفکر منطقی　　ب) تفکر انسانی　　ج) عملکرد منطقی　　د) عملکرد انسانی

٥– در رباتیک در کدامیک از موارد زیر کمتر از تکنیک های هوش مصنوعی استفاده شده است؟

الف) کنترل　　ب) تخصیص وظیفه　　ج) طراحی سنسورها　　د) برنامه ریزی حرکت

٦– یک عامل توانسته است تمام انسانها را در یک بازی شکست دهد، کدام جمله بنظر شما صحیح است؟

الف) این عامل قطعاً در تست تورینگ برنده خواهد شد

ب) این عامل می تواند بدون توجه به نحوه تفکر انسان طراحی شده باشد.

ج) عامل صرفاً بر تکرار و شبیه سازی روشهای بازیگران حرفه ای استوار است.

د) عامل می تواند صرفاً بر سرعت بالا و حافظه ی حجیم سوپر کامپیوترها تکیه کند.

فصل دوم

Chapter Two

عامل های هوشمند

Intelligent agents

عامل های هوشمند

شرح برخی از تعاریف

عامل: موجودیتی است که بتواند توسط حسگر محیط را درک نماید و سپس توسط اثر کننده ها روی محیط تاثیر بگذارد. **مثال:** انسان توسط گوش چشم و بینی، محیط اطراف را درک می نماید سپس توسط دست و پا اقدام به یک عمل می نماید.

عامل هوشمند: سخت افزار ها یا نرم افزارهایی هستند که توسط حسگرها و دوربین ها محیط شان را درک می نمایند. و توسط مفاصلی که روی این عامل های هوشمند قرار گرفته روی محیط تاثیر میگذارند.

کاربردهای هوش مصنوعی

- **بازی کردن** مانند: ربات فوتبالیست (هوش مصنوعی)
- **زمانبندی و برنامه ریزی خودکار** مانند: برنامه دانشگاه
- **کنترل خودکار** مانند: هواپیمای بدون سرنشین
- **تشخیص** مانند: تشخیص بیماری مانند: پوست، مغز و اعصاب
- **رباتیک** مانند: ربات سازی در زمینه های مختلف

اصطلاحات عامل های هوشمند

- **حسگر** Sensor
- **عملگر** Actuator
- **عمل** Action
- **ادراک** Percept
- **دنباله ادراک** Percept Sequence

حسگر: یک قطعه فیزیکی است که محیط خود را درک میکند. مانند: سنسور حساس به دما، جعبه سیاه ماشین، چراغ Check موتور

عملگر: یک قطعه فیزیکی است که روی محیط تاثیر میگذارد، مانند: مفصل و دست پای یک ربات.

عمل: خروجی عامل را **عمل** میگویند. مانند گل زدن ربات، زنده یاب.

ادراک: درک محیط بشکل تئوری را ادراک میگویند.

دنباله ادراک: به مجموعه ادراک، دنباله ادراک می گویند. مانند: سابقه کامل هر چیزی است که عامل تاکنون درک کرده است.

تابع عامل[1]

اگر بتوانیم به ازای هر دنباله ادراک یک خروجی داشته باشیم تابع عامل می گویند.

$$F: P^* \rightarrow A$$

فعالیت ⟵ دنباله ادراک : تابع عامل

برنامه عامل[2]:

برنامه ای که به ازای هر دنباله ادراک، به ما یک خروجی بدهد.

نکته مهم : برنامه عامل، نقش مهمی در هوش مصنوعی دارد. برای مثال اگر برنامه ای برای یک ربات تعریف نشده باشد، چگونه میتواند وظایف خود را انجام دهد؟

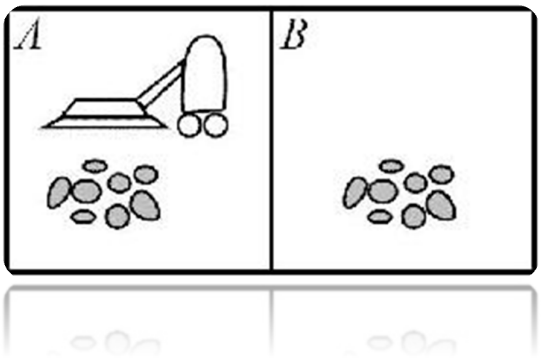
[1] Agent Function
[2] Agent Program

عامل های جارو برقی هوشمند

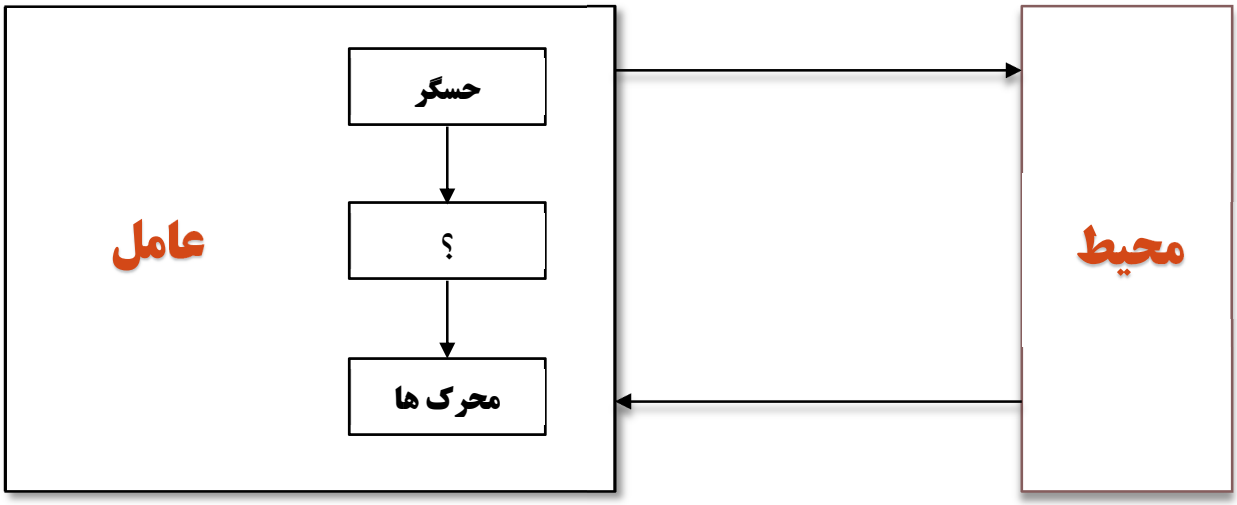
مثالی از عامل واکنشی ساده در دنیای جاروبرقی

- **تصمیم گیری آن بر اساس مکان فعلی و کثیف بودن آن مکان صورت میگیرد.**
- **انتخاب فعالیت بر اساس موقعیت شرطی :** If dirty then SUCK

فعالیت یا Action	دنباله ادراک یا Percept Sequence
Right	[A, Clean]
Suck	[A, Dirty]
Left	[B, Clean]
Suck	[A, Dirty]
Right	[A, Clean], [A, Clean]
Suck	[A, Clean], [A, Dirty]
Now	[A, Clean], [B, Clean]
…Right	… [A, Clean], [A, Clean], [A, Clean]
Suck…	[A, Clean], [A, Clean], [A, Dirty]…



ساختار عامل های هوشمند



عامل : هر چیزی که قادر است محیط خود را از طریق حسگرها درک کند و از طریق محرک ها عمل نماید.

سیستم خودکار هدایت خودرو:

- ادراک[۳]
- اعمال[۴]
- اهداف[۵]
- محیط[۶]

معیارهای کارایی برای عامل های هوشمند:

سازنده ربات ها بر اساس عملکرد آن ربات یکسری کارایی های خاصی را مطرح میکنند. برای مثال کارایی ربات آتش نشان با ربات فوتبالیست متفاوت است.

[3] Percept
[4] Actions
[5] Goals
[6] Environment

تعریف محیط وظیفه عامل های هوشمند:

عبارت است از محیطی که عامل هوشمند در آنجا فعالیت هایی را انجام میدهد. مانند: محیط شطرنج برای شطرنج باز.

محیط وظیفه ی عامل هوشمند از چهار مؤلفه به شرح زیر تشکیل شده است:

۱. **کارایی** Performance
۲. **محیط** Environment
۳. **حسگرها** Sensors
۴. **محرک ها** Actuators

جدول مثالی برای توصیف PEAS

نوع عامل	مقیاس کارایی	محیط	محرک ها	حسگرها
راننده تاکسی	امن سریع، قانونی	جاده،عابرین پیاده، مشتریان	فرمان، گاز، ترمز، چراغ ها، بوق و صفحه نمایش	دوربین ها، سرعت سنج کیلومتر شمار
سیستم تشخیص پزشکی	بیمار سالم، مسائل حقوقی	بیماران، کارکنان بیمارستان	آزمایشگاه ها، تجویز ها، نمایش پرسشها	وارد کردن علائم بیماری
کنترل گر پالایشگاه	بازده واقعیت، بیشینه کردن خلوص	پالایشگاه؛ کارکنان	شیرها، پمپ ها، گرما سازها	حسگرهای دما، فشار

خواص محیط های وظیفه:

۱ – **محیط های کاملاً قابل مشاهده در مقابل قابل مشاهده جزئی.**

- **محیط های کاملاً قابل مشاهده** :یعنی تمامی وضعیت ها در آن محیط در ابتدا مشخص باشد مانند محیط شطرنج.
- **قابل مشاهده جزئی** : یعنی وضعیت محیط از قبل قابل پیش بینی نباشد. مانند محیطی که خودرو هوشمند در آن تردد می کند.

۲ – **قطعی در مقابل غیر قطعی:**

- **محیط قطعی:** حالت اولیه معلوم، هدف معلوم و حالت نهایی هم معلوم باشد. مانند: بازی شطرنج.
- **محیط غیر قطعی:** حالت اولیه معلوم، هدف معلوم، ولی حالت نهایی نا معلوم باشد. مانند: ربات زنده یاب، اتومبیل هوشمند.

نکته مهم: محیط های راهبردی اگر در محیطی به جز تاثیر فعالیت های عامل های دیگر، محیط قطعی باشد به چنین محیط هایی محیط های راهبردی می گویند. مانند: بازی شطرنج زمان دار.

۳ – **رویدادی در مقابل ترتیبی:**

- **محیط رویدادی:** به محیطی گفته میشود که حالت های عامل هوشمند پشت سر هم باشد. بدون توقف و بدون فاز بندی. مانند خودروی هوشمند.
- **محیط ترتیبی:** در این محیط عامل هوشمند گام به گام عمل می کند. مانند بازی شطرنج.

۴ – **ایستا در مقابل پویا:** ایستا یعنی ثابت مانند بازی شطرنج و پویا یعنی در حال تغییر مانند چراغ .

- **محیط نیمه پویا:** اگر محیطی پویا نباشد ولی عاملیت زمان باعث تغییر در کارایی آن شود محیط پویا می گویند. مانند: بازی شطرنج زمان دار.

۵ – **گسسته در مقابل پیوسته:**

- **محیط گسسته:** به محیطی گفته میشود که تعداد حالت آن قابل شمارش باشد.
- **محیط پیوسته:** در این محیط تعداد حالت های آن غیر قابل شمارش باشد.

۶ – **تک عاملی در مقابل چند عاملی:**

- **محیط تک عاملی:** رباتی که جدول حل کند، رباتی که قوطی را جابجا کند.
- **محیط چند عاملی:** مانند شطرنج که دو نفره بازی میشود.

۷ – **چند عاملی رقابتی در مقابل چند عاملی همیاری:**

- **محیط تک عاملی:** رباتی که جدول حل کند، رباتی که قوطی را جابجا کند.
- **محیط چند عاملی:** مانند شطرنج که دو نفره بازی میشود.

چند نکته مهم در مورد عامل های هوشمند

- اگر محیط کاملاً قابل مشاهده نباشد غیر قطعی است.
- بهتر است قطعی یا غیر قطعی بودن را از دیدگاه عامل در نظر بگیریم.
- در محیط های رویدادی، انتخاب فعالیت در هر رویداد به خود رویداد بستگی داد و به فعالیت های رویداد قبلی بستگی ندارد.
- در محیط های ترتیبی، تصمیم فعلی می تواند در تصمیمات بعدی موثر باشد.
- اگر محیط در طول عمر عامل تغییر کند پویا است در غیر اینصورت ایستا است.
- سخت ترین محیط، قابل مشاهده جزئی، غیر قطعی، ترتیبی، پویا، پیوسته و چند عاملی است.

ساختار عامل

برنامه + معماری = عامل **مثال: برنامه + عضلات ربات**

محیط های کاری و ویژگیهای آن. (چند مثال):

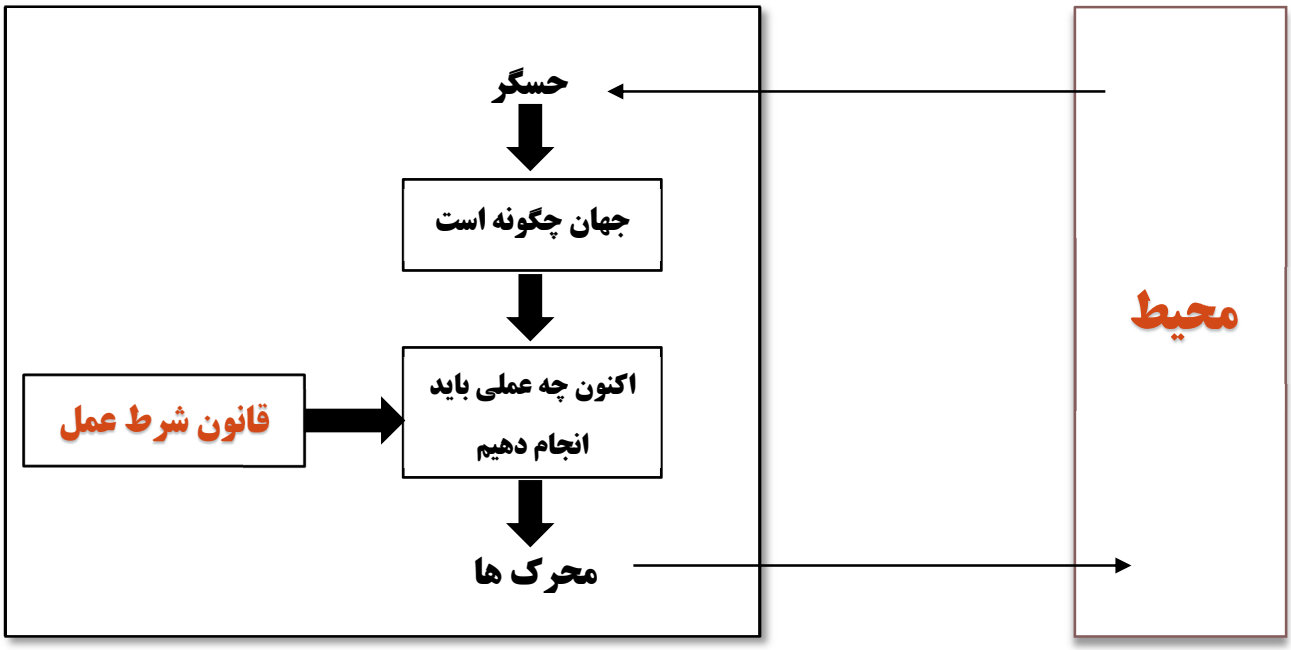
محیط کار	مشاهده پذیر	قطعی	مرحله ای	ایستا	گسسته	عامل ها
جدول کلمات متقاطع	کاملاً	قطعی	ترتیبی	ایستا	گسسته	تک عاملی
شطرنج زمان دار	کاملاً	راهبردی	ترتیبی	نیمه پویا	گسسته	چند عاملی
رانندگی تاکسی	بعضاً	اتفاقی	ترتیبی	پویا	پیوسته	چند عاملی
تشخیص پزشکی	بعضاً	اتفاقی	ترتیبی	پویا	پیوسته	تک عاملی
تحلیل تصاویر	کاملاً	قطعی	مرحله ای	نیمه پویا	پیوسته	تک عاملی
ربات قطعه بردار	بعضاً	اتفاقی	مرحله ای	پویا	پیوسته	تک عاملی
کنترل گر پالایشگاه	بعضاً	اتفاقی	ترتیبی	پویا	پیوسته	تک عاملی
معلم انگلیسی	بعضاً	اتفاقی	ترتیبی	پویا	گسسته	چند عاملی

کار هوش مصنوعی طراحی برنامه عامل است که تابع عامل را پیاده سازی میکند. این برنامه بر روی یک دستگاه محاسباتی با حسگرها و محرک های فیزیکی، یعنی معماری اجرا میشود.

انواع عامل ها:

- عامل های واکنشی ساده
- عامل های هدف گرا
- عامل های واکنشی مدل گرا
- عامل های سودمند
- عامل های یاد گیرنده

ساختار عامل های واکنشی ساده



شرح عامل های واکنشی ساده

این نوع عامل ها ساده ترین نوع عامل می باشند که برنامه نویسی ساده ای دارند. نحوه عملکرد عامل های واکنشی ساده به شرح زیر میباشد:

این نوع عامل ها فقط بر اساس درک فعلی از محیط و بدون ذخیره کردن و بدون استفاده از دنباله ادراکی، وضعیت جاری را تشخیص میدهند. بعد از تشخیص وضعیت جاری با توجه به قانون شرط و عمل بهترین عمل را انجام میدهند. مانند ترموستات، ترموستات وضعیت فعلی محیط را سنجیده و بشکل زیر عمل میکند.

نکته: اگر دما ۹۰ درجه باشد ترموستات روشن میشود، در غیر اینصورت ترموستات خاموش شود.

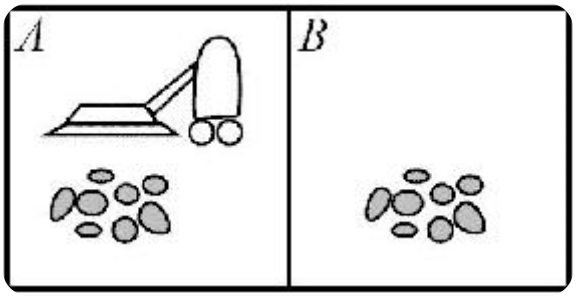
نکته بسیار مهم: در ساختار عامل ها شکل های بیضی شکل، دانش و اطلاعاتی هستند که عامل آنها را داد. شکل های مستطیل شکل، وظیفه یا کارهایی هستند که طی تصمیم گیری ها عامل آنها را انجام خواهد داد.

مثالی از قانون شرط و عمل

برای دانشگاه: یک اگر نمره کمتر از ۱۰ شد، آنگاه مردود است.

برای ماشین: اگر چراغ قرمز است، آنگاه ایست کن.

مثالی عامل های واکنشی ساده (جارو برقی)

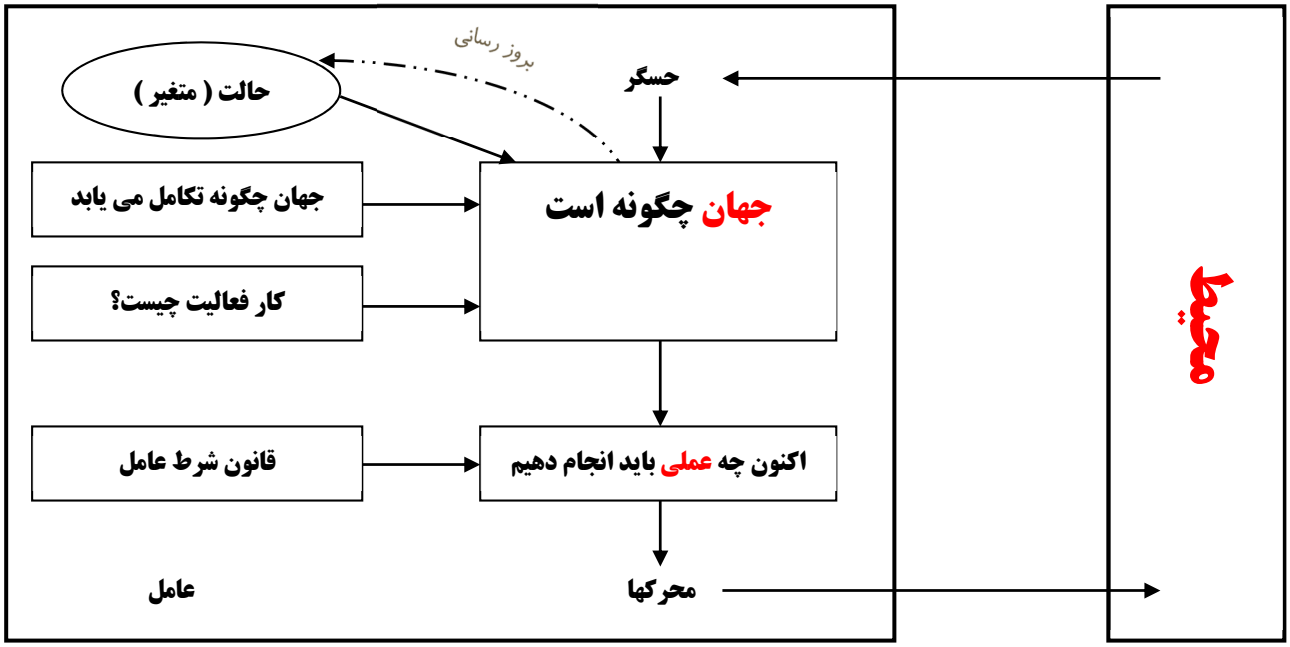



```
Function REFLEX_VACUUM_AGENT
([Location, Status])
Return an action;
If Status == Dirty then Return Suck
Elseif Location == A then Return Right
Elseif Location == B then Return Left
```

نکات لازم در مورد عامل های واکنشی ساده:

- اگر عامل واکنش ساده بخواهد به درستی عامل کند، باید محیط کاملاً قابل مشاهده باشد. چون در این روش عامل باید بتواند از روی درک فعلی وضعیت فعلی محیط را تعیین کند.
- ساده ترین برنامه عامل برای عامل های هوشمند برنامه عامل واکنشی ساده می باشد، که از قوانین شرط و عامل استفاده میکند.

ساختار عامل های واکنشی مدل گرا



شرح عامل های واکنشی مدل گرا:

عامل های واکنشی مدل گرا در این مدل برنامه عامل، وضعیت قبلی محیط را در یک متغیری بنام متغیر حالت ذخیره میکند در هر لحظه عامل از روی ادراک فعلی با توجه به حالت، وضعیت فعلی را تشخیص میدهد. همچنین در این حالت متغیر حالت داخلی بروز رسانی میشود. در این مدل حالت قبلی میتواند نقشی در حالت های بعدی ایفا کند.

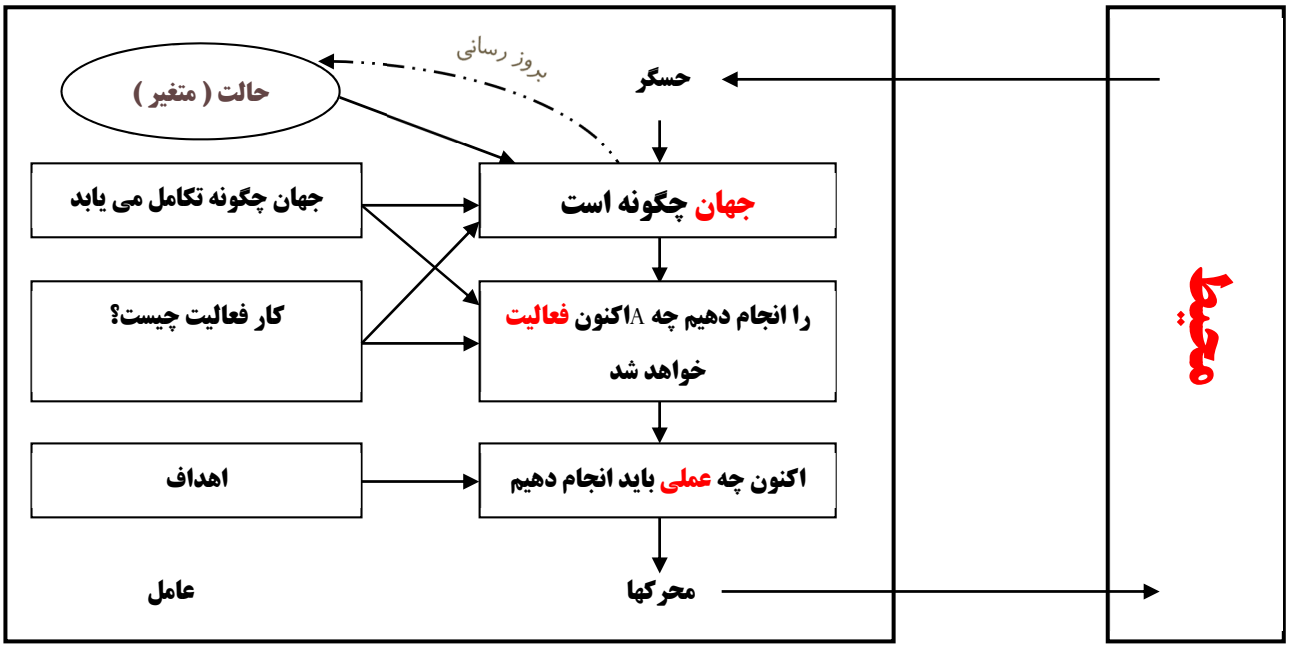
نکات مهم:

- برای اینکه عامل مدل گرا با موفقیت عمل کند، باید محیط قطعی باشد. یعنی عامل باید از حالت قبلی و عملی که در مرحله در مرحله قبل انجام داده وضعیت فعلی را تشخیص دهد.
- اگر محیط کاملاً قابل مشاهده باشد در این حالت مدل واکنشی ساده برای ما کافی است. یعنی در محیط های کاملاً قابل مشاهده نیازی به ذخیره حالت های قبلی نداریم.
- عامل ها برای اینکه بتوانند متغیر حالت بروز رسانی کنند نیاز به دو دانش زیر دارند:

۱ – محیط به مرو زمان بدون دخالت عامل به چه شکلی تغییر پیدا میکند

۲ – کار فعالیت چیست؟ یعنی هر علمی که عامل انجام می دهد منجر به چه تغییر در محیط میشود.

ساختار عامل های هدف گرا



شرح عامل های هدف گرا:

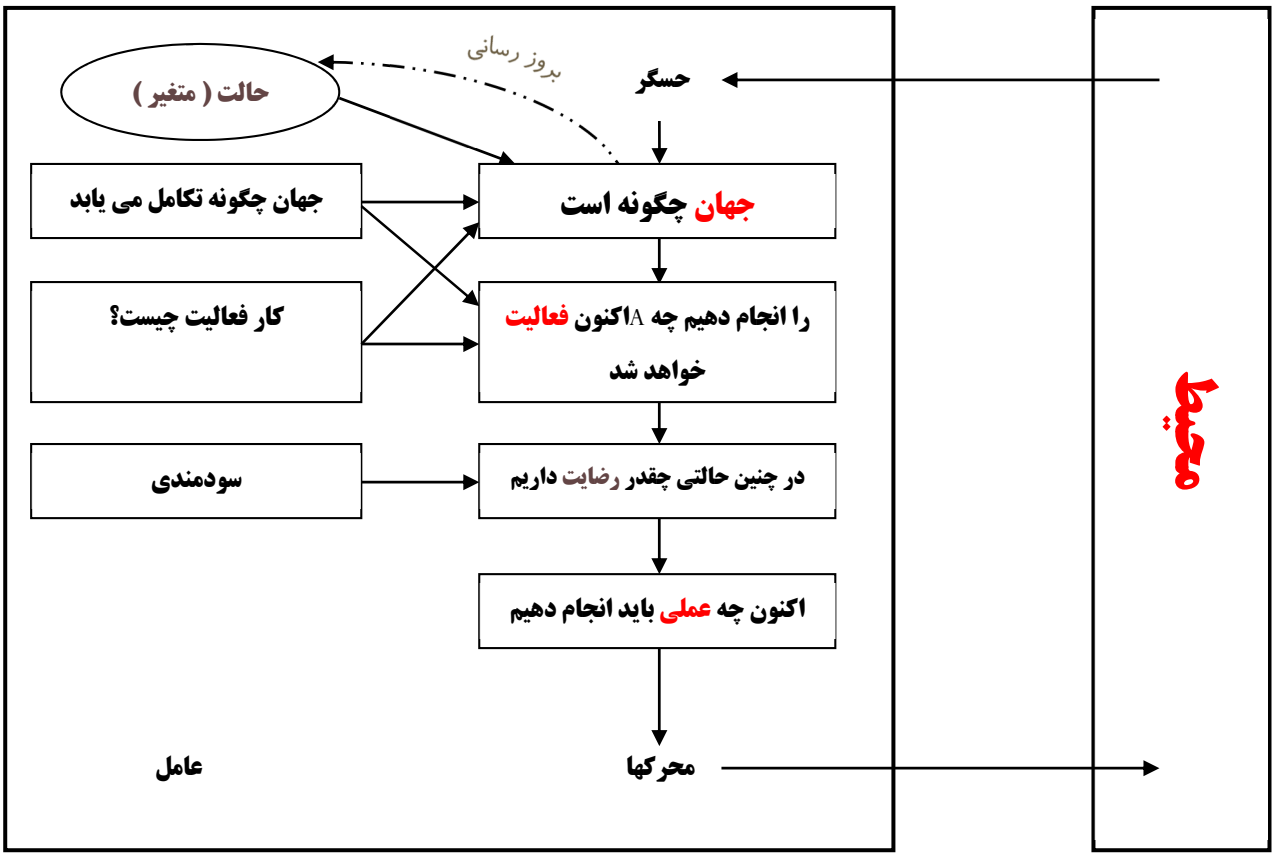
در این عامل ها دانستن وضعیت فعلی محیط برای موفق شدن عامل در انجام عمل کافی نیست.

مثال: اگر خودرویی به چهار راهی برسد، باید تشخیص دهد به کدام سمت چهار راه حرکت کند. در این حالت: دانستن وضعیت فعلی کافی نخواهد بود بلکه هدف باید مشخص شود. چون در این عامل ها همیشه هدف حالت مطلوب میباشد.(کمترین مسیر).

نکات مهم:

- همانطور که در شکل مشاهده میکنیم، عامل های هدف گرا هم برای بروز رسانی متغیر، نیاز به ۲ دانش گفته شده را دارد.
- در عامل های هدف گرا خبری از قوانین شرط و عمل نیست.
- عامل های واکنشی هدف گرا، انعطاف بیشتری نسبت به عامل های قبلی دارند.

ساختار عامل های سودمند:



شرح عامل های سودمند:

همیشه عامل های هدف گرا نمی توانند کارایی بهینه را به مان نشان دهند. بعنوان مثال اگر چندین مسیر برای رسیدن داشته باشیم عاملی که مسیر بهینه را از چندین مسیر انتخاب کند، میگویند حالت عامل سودمند گرا است. به عبارت ساده تر اگر یک حالت محیط را به چندین حالت محیط ترجیح دهیم، می گوییم عامل سودمند گرا است.

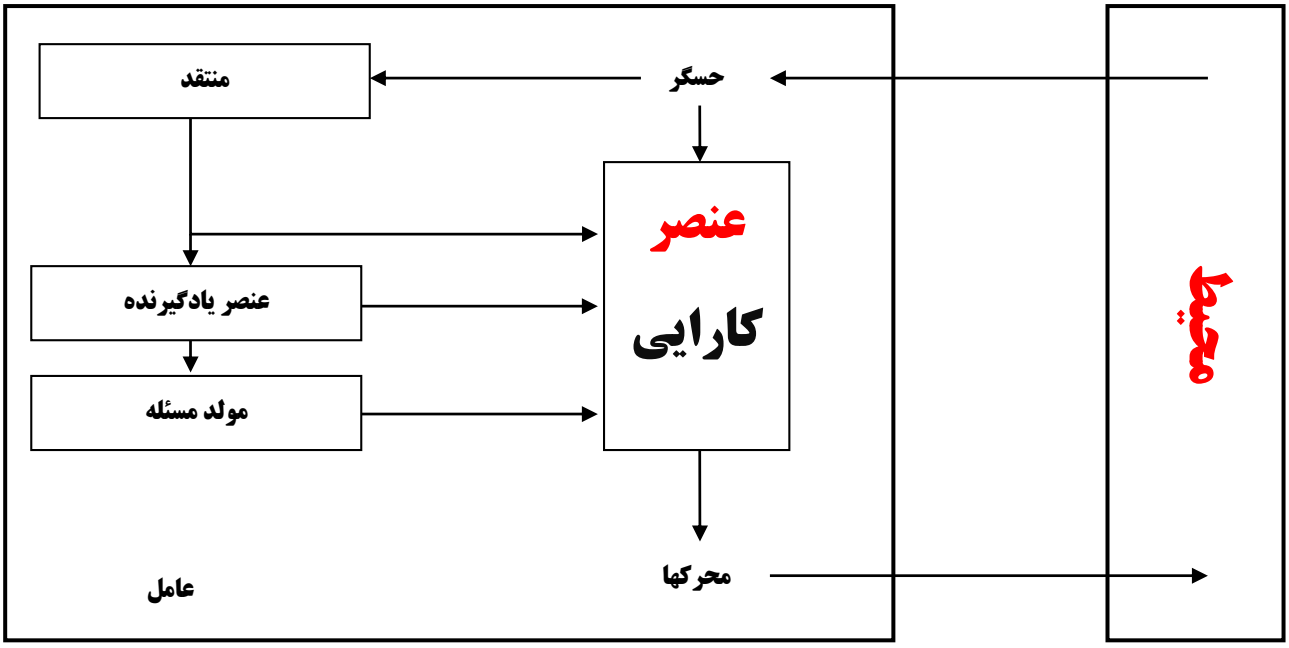
منظور از تابع سودمندی در عامل های سودمندی چیست ؟

منظور این تابع به هر وضعیت محیط یک عددی را بعنوان میزان سودمندی اختصاص میدهد، که عامل با استفاده از این عدد، ارزشمند بودن یا سودمند بودن را تشخیص می دهد.

نکات مهم:

- در عامل های سودمند هم برای بروز رسانی متغیر حالت نیاز به ۲ دانش گفته شده در مدل های قبلی دارد.
- در این عامل هم خبری از قوانین شرط و عمل نیست! پس انعطاف پذیری دارد.

ساختار عامل های یاد گیرنده:



شرح عامل های یاد گیرنده:

تمامی عامل هایی که تا این لحظه توضیح دادیم، هیچ کدام قابلیت یادگیری از محیط را نداشتند و بر اساس دانشی که طراح ربات بعنوان دانش اولیه به آنها تزریق کرده تصمیم گیری را انجام می دادند ولی عامل های یاد گیرنده بر اساس فعالیت های روز مره اشان، اطلاعات خود را افزایش می دهند این عمل را در محیط های ناشناخته به درستی عمل میکنند.

عامل های یاد گیرنده از ٤ مؤلفه تشکیل شده اند :

۱ – عنصر کارائی: همان عاملی است که تا این لحظه در رابطه آن صحبت میکردیم، یعنی با استفاده از درک فعلی محیط، یک عملی را انجام می دهد.

نکته: استاندارد کارائی یک مؤلفه، خارج از عامل می باشد که بر اساس عملیات هایی که عامل انجام می دهد یک ورودی بعنوان پاداش یا مجازات به مؤلفه منتقد میدهد.

پاداش : نشان دهنده آن است که عمل قبلی درست است ؟ یا خیر.

۲ – منتقد: با درک فعلی از محیط خود و همچنین پاداشی و مجازاتی که در قبال عمل قبلی دریافت کرده، یک باز خوردی را به عنصر یاد گیرنده میدهد.

۳ – عنصر یاد گیرنده: با توجه به باز خوردی که از مؤلفه منتقد دریافت میکند، اطلاعات خودش را تغییر را اصلاح میکند.

۴ – مولد مسئله: وظیفه این مؤلفه، کشف تجربیات و دانش جدید تا بتواند دانش خود را افزایش دهد.

تست های فصل دوم

۱ – عاملی که تخته نرد بازی میکند، در چه محیطی قرار دارد؟

الف) ایستا، قطعی، مشاهده پذیر، ترتیبی

ب) پویا، قطعی، مشاهده پذیر، رویدادی

ج) ایستا، غیر قطعی، مشاهده پذیر، ترتیبی

د) ایستا، قطعی، مشاهده پذیر، گسسته، رویدادی

۲ – کدامیک از جملات زیر صحیح است؟

الف) ممکن است تابع عاملی وجود داشته باشد که نتوان آنرا با هیچ تابع عاملی پیاده سازی نمود.

ب) یک عامل مبتنی بر دانش را نمی توان با کمک معماری انعکاسی ساده ساخت.

ج) عامل مبتنی بر مدل برای محیط های با حالات پیوسته مناسب نیست.

د) در محیط های کاملاً قابل مشاهده، دلیلی برای دانستن حالات داخلی نیست.

۳ – کدامیک از عبارات زیر صحیح است؟

الف) همه محیط های نیمه مشاهده غیر قطعی هستند.

ب) عاملی که به زبان طبیعی محاوره میکند، در یک محیط نیمه مشاهده عمل میکند.

ج) هر عاملی که فقط بخش ار محیط را حس میکند، نمیتواند عقلانی باشد.

د) عاملی که کاملاً در محیط مشاهده عمل میکند، نیاز به حالات درونی ندارد.

٤ – نسبت بین عامل های هوشمند، در کدام گزینه درست بیان شده است؟

الف) عامل = ساختار + الگوریتم

ب) عامل = ساختار + برنامه

ج) عامل = الگوریتم + برنامه

د) عامل = الگوریتم + ساختار + برنامه

٥ – محیط کدامیک از موارد زیر پویا است؟

الف) تخته نرد ب) شطرنج ج) راننده تاکسی د) پوکر

٦ – کدامیک از برنامه های عامل زیر از قوانین شرط و عمل استفاده نمی کنند؟

الف) عامل مدل گرا ب) عامل هدف گرا ج) عامل سودمند گرا د) **ب و ج**

٧ – کدامیک از عامل های زیر، وضعیت دنیا را ذخیره نمی کنند؟

الف) واکنشی ساده ب) مدل گرا ج) مبتنی بر سودمندی د) هدف گرا

٨– در یک عامل یاد گیرنده، کدامیک از مؤلفه های زیر مسئول تصمیم گیری در مورد عملی است که باید انجام شود؟

الف) عنصر کارایی ب) استاندارد کارایی ج) عنصر یاد گیرنده د) مولد مسئله

فصل سوم

Chapter Three

حل مسئله با جستجو

Solving Problems by Searching

حل مسئله با جستجو

شرح برخی از تعاریف

عامل های حل مسئله عامل های مبتنی بر هدف یا مبتنی بر سودمندی می باشند.

نکته مهم:

- ما مسائلی را با روش جستجو حل می نماییم، که به دنبال هدف یا سودمندی باشیم.
- عامل های حل مسئله با یافتن دنباله ای از فعالیت هایی که منجر به حالت های مطلوب میشوند، تصمیم می گیرند که چه کاری باید انجام دهند.
- الگوریتم هایی که می توانیم استفاده کنیم : الگوریتم آگاهانه و نا آگاهانه.

الگوریتم آگاهانه و ناآگاهانه

الگوریتم های ناآگاهانه:

- به الگوریتم هایی گفته میشود که فقط صورت مسئله میدانیم و خیری از مسیر ها و نتایج آنها نداریم.

الگوریتم های آگاهانه:

- الگوریتم هایی هستند، که علاوه بر صورت مسئله از مسیر ها و وضعیت آنها خیر داریم که مسیرهای مسئله و هدف را می دانیم.

چهار گام اساسی برای حل مسئله:

۱. **فرموله کردن هدف:** یعنی هدف را مشخص کنیم.

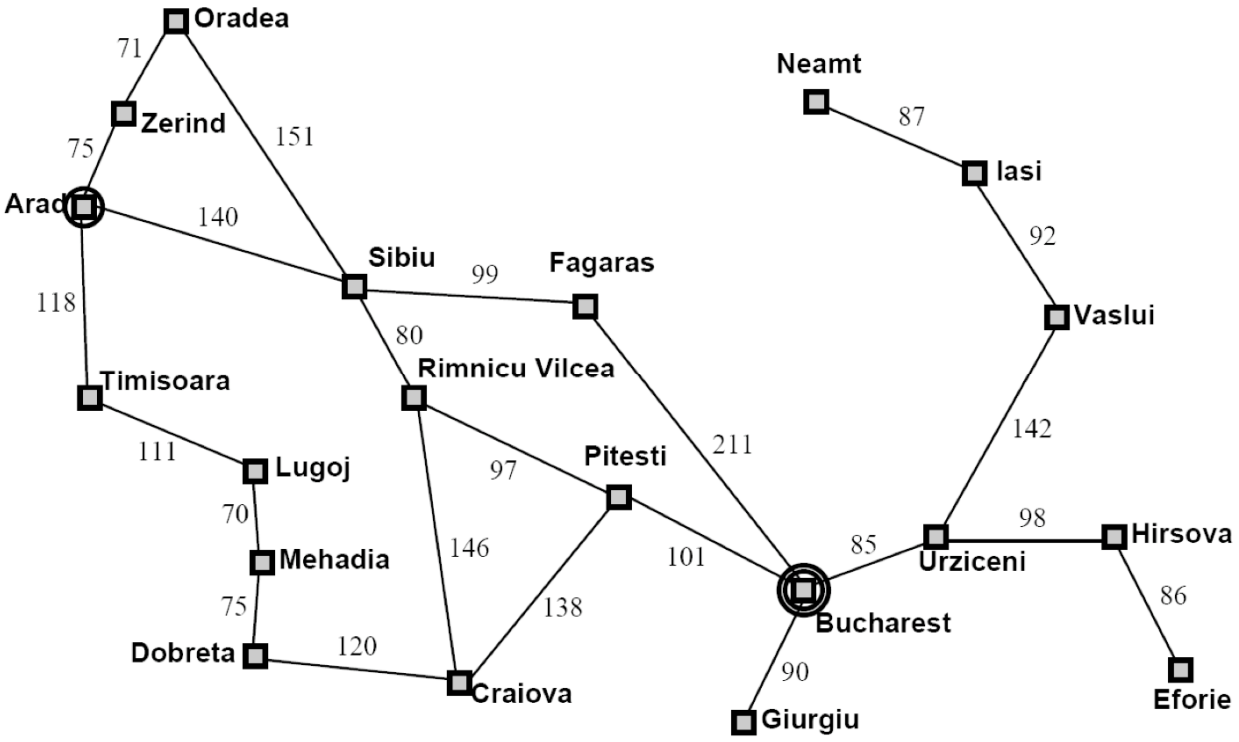
۲. **فرموله کردن مسئله:** باید برای رسیدن به هدف راه حل ها را بدانیم چیست!

۳. **جستجو:** از بین حالت های مختلف، حالت بهینه را انتخاب نماییم.

- *نکته مهم : برای حل مسائل در جستجو باید برای مسئله، درخت جستجو را تشکیل دهیم.*

۴. **اجرا:** بعد از اینکه حالت مطلوب را انتخاب کردیم آن حالت را برای رسیدن به هدف اجرائی کنیم.

مثال: حل مسئله نقشه رومانی، رفتن از Arad به Bucharest .



- **فرموله کردن هدف**: حضور در Bucharest .
- **فرموله کردن مسئله**: رفتن شهر به شهر موجود.
- **جستجو**: از بین شهر های موجود، بهترین مسیر را رفتن.
- **اجرا**: Bucharest و Fagaras و Sibiu و Arad.

" این جستجو با توجه به کم هزینه ترین مسیر انتخاب میشود. "

صورت مسئله: رفتن از Arad به Bucharest

- **فرموله کردن هدف**: رسیدن به Bucharest
- **فرموله کردن مسئله**: وضعیت ها: شهرهای مختلف
- **فعالیت ها :** حرکت بین شهرها
- **جستجو:** دنباله ای از شهرهای مانند: Bucharest , Fagaras , Sibiu , Arad

هر مسئله از ٤ مؤلفه زیر تشکیل میشود:

۱ – **حالت اولیه یا Start State :** مکان جاری را حالت اولیه میگویند. در مثال رومانی: شهر آراد (in Arad)

۲– **تابع جانشین**: یعنی از حالت جاری به چه حالت هایی میتوان رفت.

Successor (in_Arad) = {<Go(Zerind), in_Zerind>, <Go(Sibiu), in_Sibiu>, <Go(Timisoara), in_Timisoara>}

و یا

S (Arad)={Zerind, Timisoara, Sibiu}

فضای حالت: فضائی که از حالت اولیه Start State بتوانیم به هر نقطه برسیم، شامل فضای حالت خواهد بود.

فضای حالت = تابع جانشین + حالت اولیه

۳ – **آزمون هدف:** تعیین میکند که حالت جاری جواب مسئله هدف است یا خیر!

- **هدف صریح (آشکار):** در مثال نقشه رمانی، هدف رسیدن به Bucharest است، در اینجا هدف آشکار است.
- **هدف انتزاعی:** در مثال شطرنج، رسیدن به حالت کیش و مات!

مسیر: به مجموعه حالت ها و فعالیت ها، مسیر میگوییم.

۴ – **تابع هزینه مسیر:** برای هر مسیر، یک هزینه عددی در نظر میگیرم. (برای بهینه کردن)

در مثال نقشه رومانی: طول مسیر بین شهرها برحسب کیلومتر.

بررسی نقشه رومانی با استفاده از ٤ مولف فوق:

۱ – **حالت اولیه:** بودن در Arad (In_Arad)

۲ – **تابع جانشین:** همیشه از حالت اولیه نشات میگیرد S(Arad) = {Zerind, Timosoara, Sibiu}

۳ – **تابع آزمون هدف:** حالت جاری را بررسی میکند، آیا به هدف رسیده است یا خیر!

۴ – **تابع هزینه مسیر:** هزینه رفتن به هر شهر 140 +80 + 97 + 101 = 418

بررسی حل مسئله جستجو معمای هشت

۷	۲	۴
۵		۶
۸	۳	۱

START STATE

	۱	۲
۳	۴	۵
۶	۷	۸

GOAL STATE

حالت ها : مکان هر ۸ خانه شماره دار، و خانه خالی در یکی از ۹ خانه.

حالت اولیه: هر حالتی را میتوان، بعنوان حالت اولیه در نظر گرفت.

تابع جانشین: حالت های معتبر از ۴ عمل انتقال خانه خالی به چپ، راست، بالا، پایین

آزمون هدف: بررسی میکند آیا اعداد به ترتیب چیده شده اند؟ یا خیر ؟

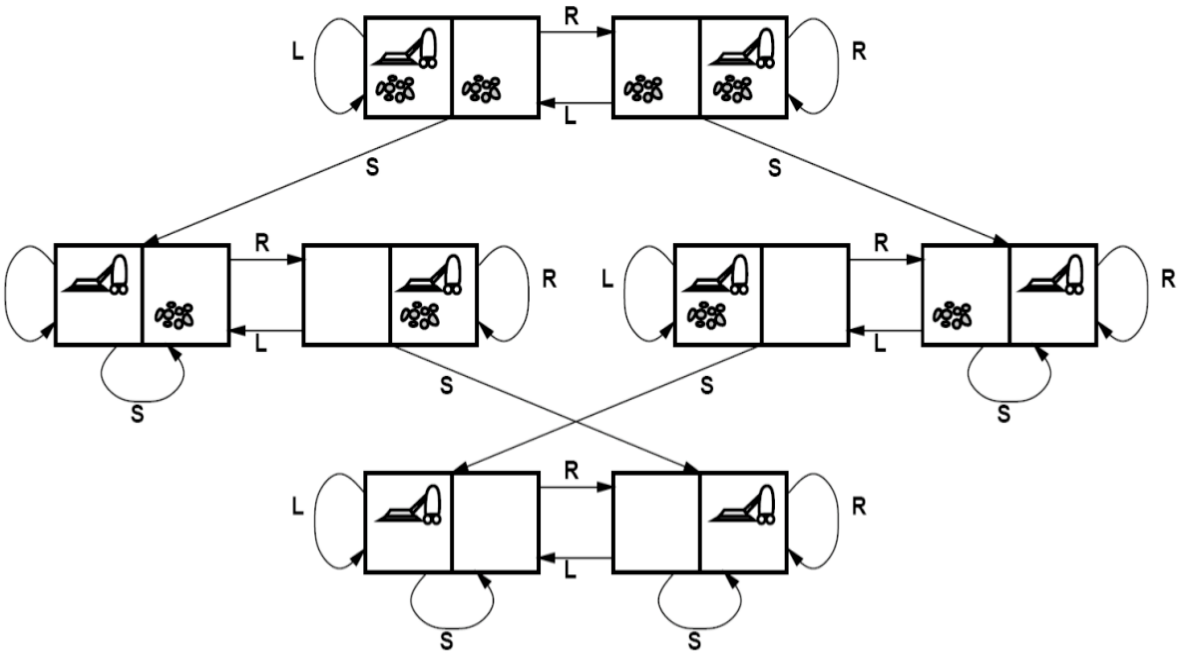
هزینه مسیر : برابر با تعداد مراحل در مسیر. (هزینه هر مرحله ۱ است)

بررسی مسئله جستجو جارو برقی

L = Left R = Right S = Suction

تعداد کل حالت : ۸ حالت

۱ – هر دو کثیف ۲ – هر دو تمیز ۳ – یکی کثیف، یکی تمیز ۴ – یکی کثیف، یکی تمیز



حالت ها : دو مکان که هر یک ممکن است کثیف یا تمیز باشند، لذا $n^2 \times 2$

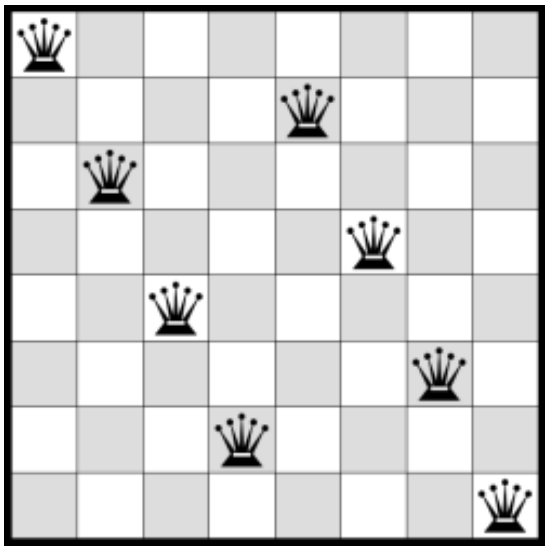
$2 \times 2^8 = 8$ حالت در این جهان وجود دارد.

حالت اولیه : هر حالتی می تواند بعنوان حالت اولیه طراحی شود.

تابع جانشین: حالت های معتبر از ۳ عملیات راست، چپ، مکش

آزمون هدف: بررسی می کند که آیا تمام مربع ها تمیز هستند یا خیر!

هزینه مسیر: تعداد مراحل در مسیر. (در واقع هزینه هر مرحله یک است.)



بررسی حل مسئله با جستجو ۸ وزیر

حالت ها : هر ترتیبی از صفر تا ۸ وزیر در صفحه یک حالت است.

حالت اولیه: هیچ وزیری در صفحه نیست.

تابع جانشین: وزیری را به خانه خالی اضافه میکند.

آزمون هدف: ۸ وزیر در صفحه وجود دارد و هیچ کدام به یکدیگر گارد نمی گیرند.

تعداد کل حالات : ۶۴ × ۶۳ × ۶۲ × ۶۱ × ۶۰ × ۵۹ × ۵۸ × ۵۷

اندازه گیری کارایی حل مسئله

تمام مسائلی که تا این لحظه حل کردیم یک روش برای حل آنها انتخاب نمودیم. باید الگوریتم انتخاب شده را از لحاظ کامل بودن، بهینگی، پیچیدگی زمانی، پیچیدگی فضا بررسی نماییم. یعنی:

کامل بودن: آیا الگوریتم، جستجوی تضمینی برای یافتن پاسخ دارد یا خیر!

بهینگی : آیا استراتژی پاسخ در مسیر بهینه انجام گرفته یا خیر!

پیچیدگی زمانی: زمان اجرای الگوریتم، چقدر طول میکشد.

پیچیدگی فضایی: مقدار حافظه ای که برای اجرای الگوریتم نیاز است.

انواع الگوریتم های مورد استفاده در جستجوی نا آگاهانه یا کورکورانه:

۱ – جستجوی عرضی Breadth-first Search (BFS)

۲ – جستجوی عمقی Depth-First Search (DFS)

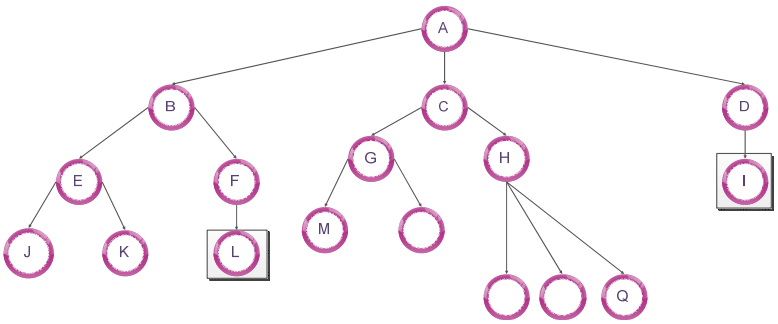
۲ – جستجوی عمقی کننده تکراری Iterative depth-first Search

۴ – جستجوی هزینه یکنواخت Uniform cost Search

۵ – جستجوی عمقی محدود Depth-limited Search

۶ – جستجوی دوطرفه

جستجوی عرضی Breadth-first Search (BFS)



کامل بودن: بلی

بهینگی: بهینه است در صورتیکه، هزینه مسیر، تابعی غیر نزولی از عمق گره باشد. (مانند وقتی که فعالیت ها هزینه یکسانی دارند).

پیچیدگی زمانی: $O(b^{d+1})$

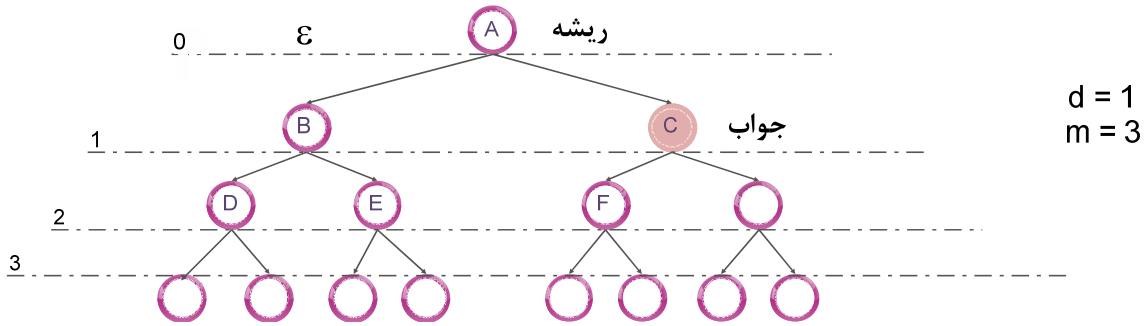
پیچیدگی فضا: $O(b^{d+1})$

در محاسبه پیچیدگی زمانی یا فضایی باید مؤلفه های زیر را بدانیم :

فاکتور انشعاب : b - حداکثر تعداد فرزند یک گروه در درخت.

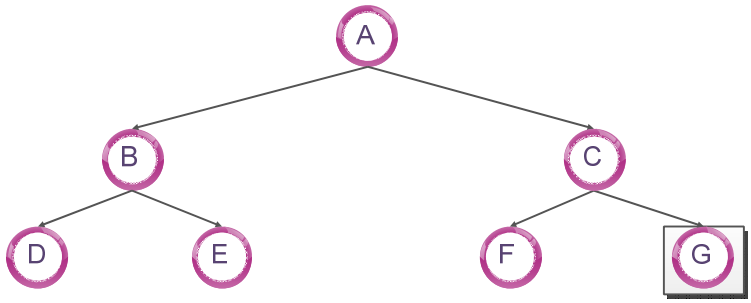
عمق جواب: d عمق درخت: m هزینه راه حل بهینه: c* هزینه از ریشه تا نود جاری: ε

در مسائلی که با استفاده از الگوریتم های ناآگاهانه حل خواهیم کرد، باید مسئله را بشکل درخت در بیاوریم.



جستجوی عرضی:

این الگوریتم گره های درخت را گره سطح به سطح بسط داده، تا به جواب برسد.



A B C D E F G

نکات مهم:

- مهم فرزندان هر گره در الگوریتم جستجوی عرضی با توجه به قرارداد یا استاندارد از چپ ترین گره بسط داده میشود.
- ترتیب چیدمان فرزندان هر گره از چپ به راست، به ترتیب حروف الفبا یا ترتیب شماره (صعودی) از چپ به راست می باشد.
- الگوریتم جستجوی عرضی برای نگهداری گره ها از صف استفاده میکند.
- در این الگوریتم جستجوی عرضی، تکراری بودن گره ها کنترل نمیشود.

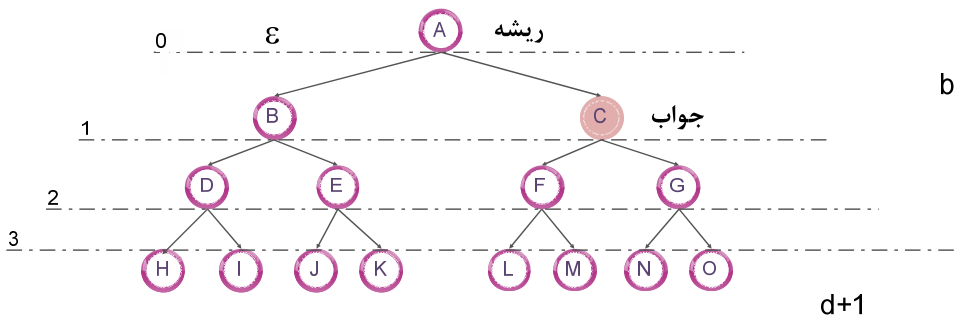
چه زمانی الگوریتم جستجوی عرضی کامل نخواهد بود؟ زمانی کامل نخواهد بود که گرهی بی نهایت فرزند داشته باشد.

نکته :

- این الگوریتم بهینه است، فقط به شرطی که هزینه عملیات یکسان باشد، اگر هزینه عملیات یکسان نباشد بهینه نخواهد بود .
- در الگوریتم جستجوی عرضی موقع ارزیابی هدف، فرزندانش شکل نمیگیرد .

پیچیدگی زمانی دقیق $\theta = b^{d+1} - d$

پیچیدگی زمانی $O = b^{d+1} - d$



b = 2 d = 3

$(2^{3+1} - 2) = 16 - 2 = 14$

پیچیدگی زمانی و مکانی در الگوریتم جستجوی عرضی با توجه به نمایه بودن جالب نیست و این برای الگوریتم فوق خوب نیست. چون عدد خیلی بزرگی خواهد بود ولی نکته مهمی که برای این الگوریتم حائز اهمیت است، زمان نیست! بلکه پیچیدگی فضایی است. یعنی به محض اجرای الگوریتم مقدار حافظه لازم برای اجرای الگوریتم از حافظه موجود آن بیشتر است.

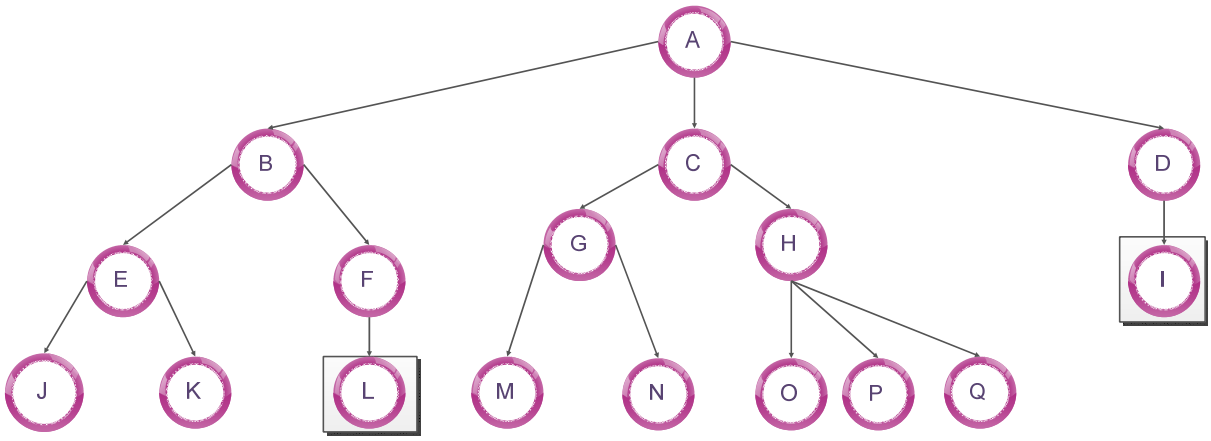
جستجوی هزینه یکنواخت Uniform Cost Search (UCS)

هزینه از ریشه تا گره: $g(n) = n$

کامل بودن: وقتی تضمین میشود که هزینه هر مرحله بزرگتر یا مساوی یک مقدار ثابت و مثبت ε باشد این شرط برای بهینگی نیز کافی است.

پیچیدگی زمانی : $o(b^{[c*/E]})$

پیچیدگی فضا: $o(b^{[c*/E]})$



الگوریتم جستجو با هزینه یکنواخت، گرهی را که هزینه کمتری را دارد، بسط میدهد.

نکات مهم:

- این الگوریتم برای نگهداری گره ها از صف اولویت استفاده میکند .
- در این الگوریتم هزینه از ریشه تا گره n ام را با g(n) نشان می دهد.

کامل بودن: اگر هزینه فرزندان یک عدد مثبت نسبت به پدر یا ریشه اش باشد این الگوریتم کامل است.

بهینگی: اگر هزینه گره ها مقدار مثبت نسبت به پدر یا ریشه اش باشد آنگاه بهینه است.

پیچیدگی زمانی و مکانی : $o(b^{[c*/E]})$

چه زمانی هزینه الگوریتم یکنواخت به هزینه الگوریتم سطحی تبدیل میشود؟

اگر هزینه تمام مراحل در این الگوریتم یکسان باشد.

برابری پیچیدگی زمانی و مکانی :

چون همان تعداد گره که عبور میکند تا به جواب برسد از لحاظ مدت زمان برابر است با همان تعداد گرهی که در حافظه ذخیره میشود.

همانطوری که در مرتبه زمانی و مکانی مشاهده میشود، وضعیت زمانی و مکانی آن نمایه است. (یعنی توان دار)

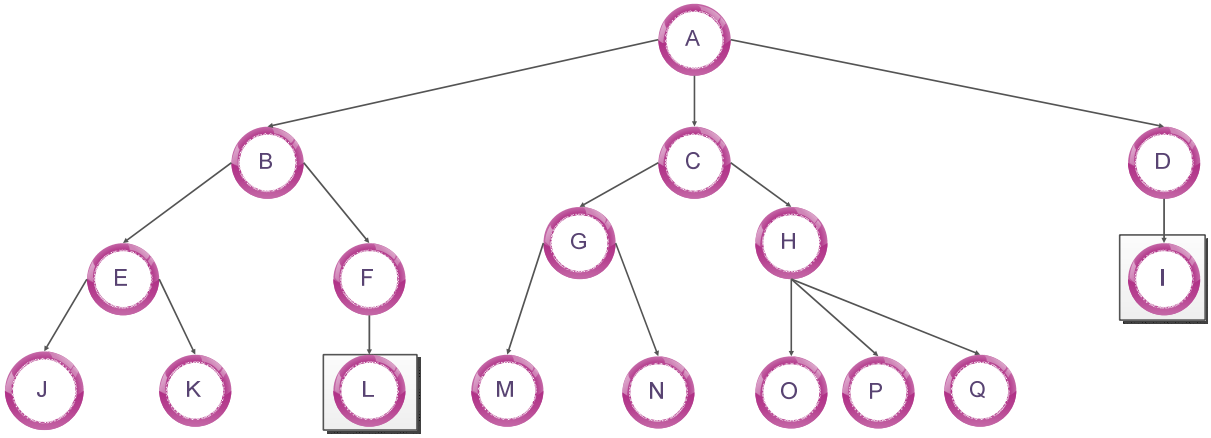
کاربرد الگوریتم هزینه یکنواخت:

این الگوریتم در مسائلی که بخوبی عمل میکند، هزینه عملیات به شکل متفاوت و مقدار مثبت باشد.

جستجوی عمقی Depth-First Search (DFS)

این الگوریتم عمیق ترین گره را برای بسط دادن انتخاب میکند، اگر در این الگوریتم چند گره با عمق یکسان در یک سطح وجود داشته باشد، چپ ترین گره را بسط میدهد.

این الگوریتم برای نگهداری جواب از پشته استفاده میکند.



کامل بودن : خیر، کامل نیست چون جوابی که در عمق است اگر ادامه یابد به بی نهایت میرسد و کامل نخواهد بود.

بهینگی : بهینه نیست چون به جواب L میرسد در صورتیکه جواب بهینه I است.

پیچیدگی زمانی: $o(b^m)$ **پیچیدگی فضایی :** $o(b\ m)$

M نشان دهنده حداکثر عمق است.

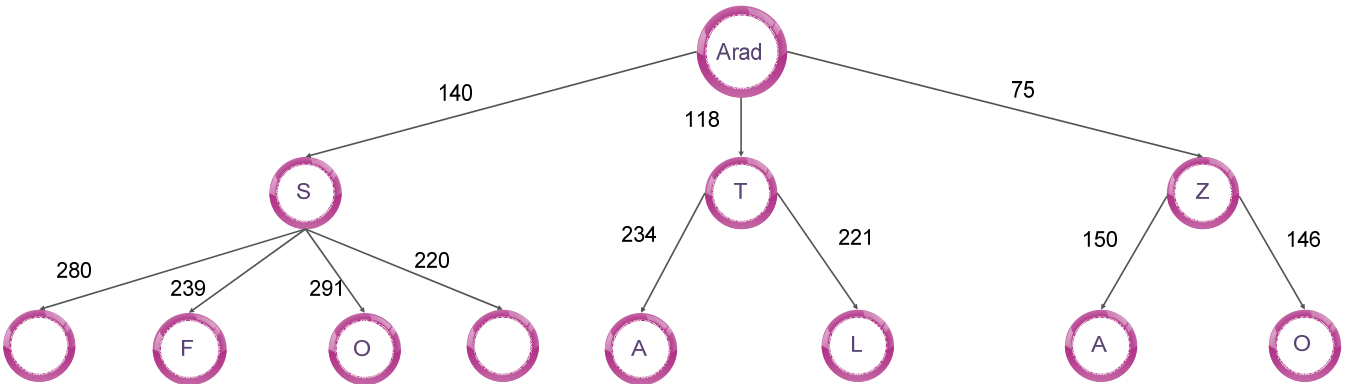
نکته:

با توجه به اینکه پیچیدگی مکانی یا فضایی این الگوریتم خطی است، وضعیت خوبی را از لحاظ پیچیدگی فضایی دارد.

چه زمانی طی میشود؟ هر وقت پیمایش عمقی داشته باشیم پیچیدگی فضایی یا مکانی خطی میشود.

چه زمانی DFS سریع تر از BFS عمل میکند؟ زمانیکه تعداد راه حل های زیادی داشته باشد.

الگوریتم عمقی در نقشه رومانی :



جواب نمی دهد چون ایجاد حلقه میکند.

نکته مهم :

الگوریتم جستجوی عمقی را می توانیم، بشکل بازگشتی حل کنیم، ولی الگوریتم های قبلی را نمی توانیم بشکل بازگشتی حل کنیم.

جستجوی عمقی محدود Depth-limited Search (DLS)

الگوریتم جستجوی عمقی محدود، برای حل کردن مشکل کامل نبودن جستجوی عمقی مطرح شد.

به این شکل که یک حداکثر عمق جستجو، با نام L در مسئله تعریف میکند. بعنوان مثال در نقشه رومانی با توجه به اینکه ۲۰ شهر وجود دارد میدانیم که حداکثر در $L = 20$ به جواب خواهیم رسید، ولی این جواب دقیق نیست. جواب دقیق این است که از رابطه زیر باید محاسبه شود:

حداکثر تعداد یال بین ۲ گره (نه حداکثر مسیر ها یعنی با حداقل پیمودن آن تعداد یال ۲ گره را به یکدیگر وصل کنیم.)

به چه دلیل فرموله کردن مسئله، بعد از فرموله کردن هدف مطرح میشود؟

اگر هدف فرموله نشود ما باشد راه حل متفاوتی پیدا کنیم، اگر هدف فرموله شود. ما با یک راه حل انجام میدهیم. برای کاهش راه حل ها فرموله کردن مسئله بعد از فرموله کردن هدف مطرح میشود.

جستجوی عمقی محدود:

کامل بودن: اگر L کوچکتر از D باشد ($L < D$) ، کامل نیست! (L عمق جستجو است).

بهینه بودن: اگر L بزرگتر از D باشد، بهینه نیست.

پیچیدگی زمانی: $o(b^L)$

پیچیدگی مکانی: $o(b\ L)$

"چون پیمایش عمق دارد، خطی است. هر جا پیمایش عمقی باشد پیچیدگی مکانی خطی است."

جستجوی عمقی کننده تکراری Iterative depth-first Search (IDS)

با توجه به اینکه در الگوریتم جستجوی عمقی محدود پیدا کردن L مشکل است، الگوریتم جستجوی عمقی تکراری معرفی گردید. در این الگوریتم L از صفر تا بی نهایت تکرار میشود تا L برابر شود با D یعنی L = D در این موقع یعنی به جواب رسیدیم.

L = 0 L = 1 L = 2 L = 3 L = 4

در هر بار محاسبه و شکل گیری L ، درخت ما تکرار میشود.

نکته مهم:

- با توجه به اینکه L از صفر شروع میشود به ترتیب تا بی نهایت در هر بار دوباره از صفر شروع میکنیم. با توجه به اینکه شکل گیری دوباره است.
- با توجه به اینکه درخت جستجو با هر L از اول شکل میگیرد، باید مشکل حافظه داشته باشیم در حالیکه این الگوریتم از لحاظ حافظه هیچ مشکلی ندارد.

کاربرد این الگوریتم در چیست ؟

در مسائلی که فضای جستجو بزرگ باشد، عمق راه حل مشخص نباشد جستجوی عمیق کننده تکراری ناآگاهانه روش مناسبی است.

کامل بودن: بلی، چون بالاخره L =D خواهد شد در صورتیکه فاکتور انشعاب محدود باشد.

بهینگی: بلی، وقتی که هزینه عملیات یکسان باشد.

پیچیدگی زمانی: $o(b^d)$

پیچیدگی مکانی: $o(b\ d)$

جستجوی دوطرفه

انجام دو جستجو همزمان یک از حالات اولیه به هدف و دیگری از هدف به حالت اولیه تا زمانیکه دو جستجو بهم برسند.

کامل بودن: زمانی کامل است که هر دو طرف سطحی باشد.

بهینگی: زمانی بهینه است که هر دو طرف سطحی باشد.

الگوریتم ILS یا طولانی تکرار شونده

این الگوریتم برای حل مشکل نمایی بودن پیچیدگی فضایی یا مکانی الگوریتم UCS مطرح گردید. نحوه عملکرد الگوریتم فوق به شرح زیر می باشد.

الگوریتم ILS، جستجوی UCS را بصورت تکراری اجرا میکند و در هر بار تکرار سقفی برای هزینه مسیر در نظر میگیرد . (سقف هزینه مسیر را مقدار برش با Cut of Value میگویند. که با L نمایش می دهیم.

الگوریتم فوق به ازای هر مقدار L درخت جستجو را از ریشه به صورت عمقی ساخته و پیمایش میکند.

این الگوریتم تا چه زمانی به پیمایش خود ادامه میدهد؟

تا جایی که هزینه مسیر های گره های آن از L **بیشتر** نباشد.

کامل بودن: بله،چون L زمانی هزینه مسیر هدف را پیدا میکند .

بهینه بودن : بلی، چون دقیقاً مشابه UCS است .

پیچیدگی زمانی: نمایی

پیچیدگی مکانی: خطی

نحوه بدست آوردن سقف هزینه مسیر (L) ؟

سقف هزینه مسیر برابر است با کمترین هزینه گرهی که از سقف تکرار قبلی بیشتر باشد.

L= سقف هزینه میسر

جستجوی دو طرفه

جستجوی دو طرفه یک الگوریتم از مست چپ حالت اولیه به سمت هدف می رود که به آن Forward یا رو به جلو میگویند و الگوریتم دوم از سمت هدف به سمت حالت اولیه یا Start State حرکت میکند که به آن Backward یا رو به عقب میگویند.

نکات مهم :

- ما قبل های گره n، گره هایی هستند که n ما بعد آنها باشند.
- جستجو به سمت عقب به این معناست که تولید ماقبل ها، از گره هدف آغاز شود .
- در جستجویی که حرکت از سمت هدف به سمت حالت اولیه می باشدباید وضعیت قبلی هر گره را پیدا کرد که اسن می تواند مشکل باشد.
- مسئله دیگر در جستجوی دو طرفه این است که هدف باید مشخص باشد.
- بررسی اینکه دو جستجو به یکدیگر رسیده اند یا خیر باید با سرعت بالایی انجام شود تا به مرتبه زمانی جستجوتاثیر منفی نگذارد. برای این کار معمولاً از جدول پراکندگی(Hash Table) استفاده میشود.

کامل بودن: بله، اگر هر دو جستجو عرضی و هزینه تمام مراحل یکسان باشد.

بهینه بودن: بله، اگر هر دو جستجو عرضی باشد و هزینه تمامراحل یکسان باشد.

پیچیدگی زمانی: $o(b^{d/2})$

پیچیدگی فضایی : $o(b^{d/2})$ که حداقل یکی از درخت ها باید در حافظه نگهداری شود.

نکته مهم :

- در الگوریتم جستجوی دو طرفه زمانیکه هر دو الگوریتم به یک نقطه مشترک رسیده اند یعنی یک مسیر از ریشه به هدف پیدا شده است.
- در الگوریتم جستوجوی دوطرفه میتوانیم از هر الگوریتمی برای آن استفاده کنیم.

الگوریتم های آگاهانه

پارمترهایی که در جستجوی آگاهانه استفاده میشود عبارتند از:

۱- **تابع هزینه مسیر :** $g(n)$ هزینه مسیر از گره اولیه تا گره n

۲- **تابع اکتشافی :** $h(n)$ هزینه تخمینی ارزان ترین مسیر از گره n به گره هدف

۳- **تابع بهترین مسیر:** $h^*(n)$ هزینه واقعی ارزان ترین مسیر از گره n تا گره هدف

۴- **تابع ارزیابی:** $F(n)$ هزینه تخمینی ارزان ترین مسیر از طریق n

جستجو محلی و بهینه سازی:

۱- تپه نوردی ۲ - شبیه سازی حرارت ۳ - پرتو محلی ۴ - الگوریتم های ژنتیک

بهترین جستجو:

- حریصانه
- A*
- IDA*
- RBFS
- MA* و SMA*

الگوریتم های حریصانه

الگوریتم فوق سعی میکند نزدیکترین گره به هدف را برای بسط دادن انتخاب کند. نزدیک تر نه از مسافت دقیق بلکه از آن تابع هیورستیک heuristic (تابع اکتشافی).

هزینه تخمین ارزان ترین مسیر از گره به n ، به گره هدف h(n)=n

نکته مهم:

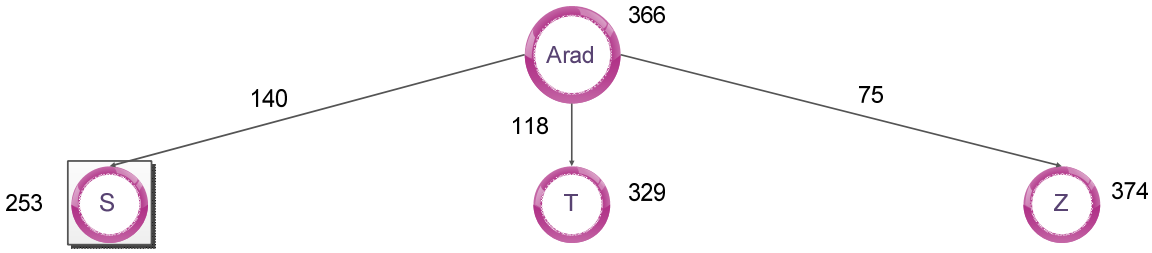
- در واقع تابع ارزیابی f(n) = h(n) می باشدکه h(n) همان تابع اکتشافی یا Heuristic است.
- جستجوی حریصانه هیچ توجهی به هزینه مسیر، تا رسیدن به گره فعلی ندارد.

مثال: حل مسئله نقشه رومانی با استفاده از جستجوی حریصانه (در جدول زیر صفر همان بخارست هدف می باشد)

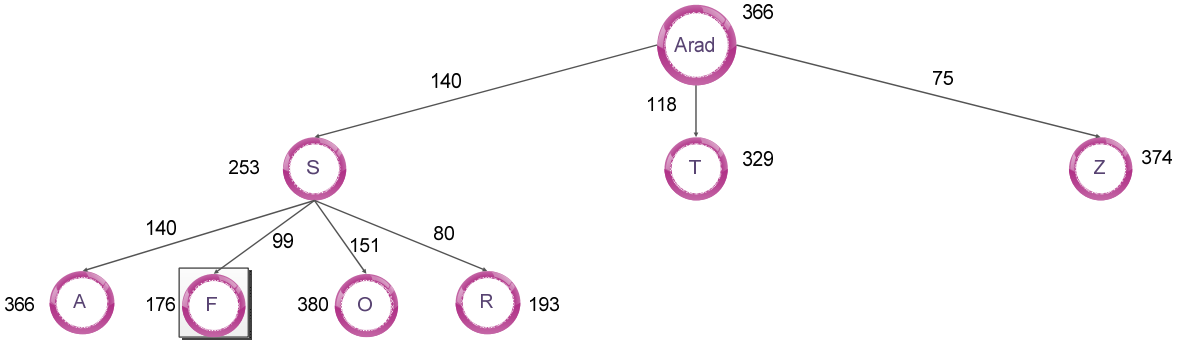
City name	value	City name	value	City name	value
Arad	366	Hirsava	151	Rimicuvilcea	193
Bucharest	0	Lasi	226	Sibiu	253
Craiova	160	Lugoj	244	Timisoara	329
Doberta	242	Mehadia	241	urziceni	80
Eforie	161	Neamt	234	vaslui	199
Fagaras	176	Oradea	380	zerind	374
Giurgiu	177	Pitesti	100		

هدف بخارست است، چون صفر است.

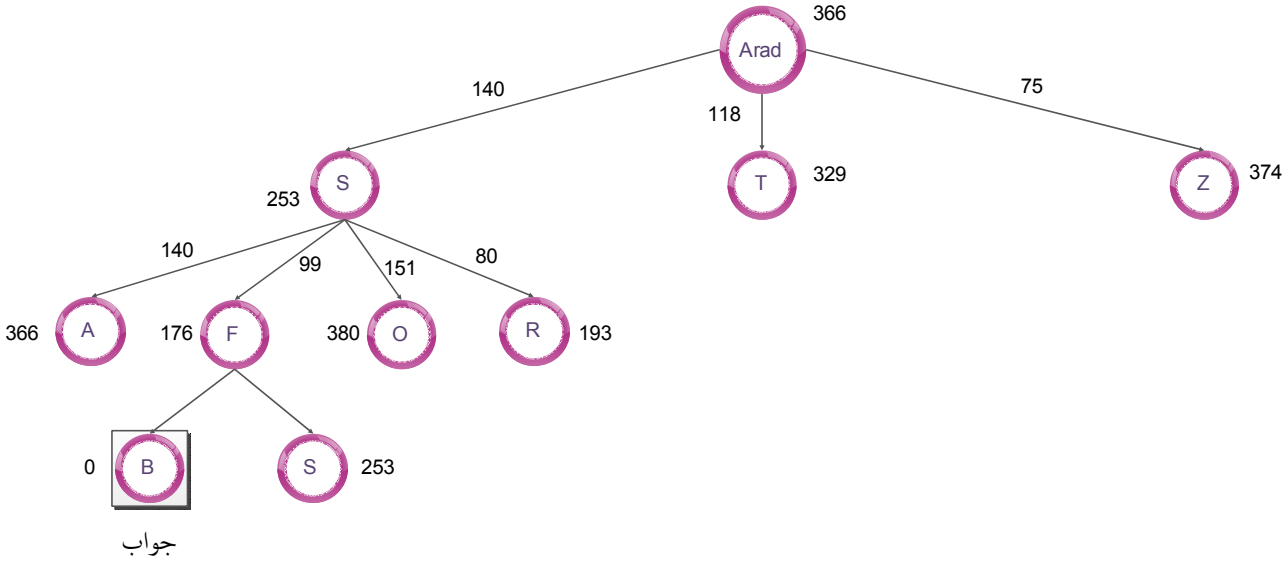
جواب مسئله نقشه رومانی با استفاده از جستجوی حریصانه



حل مسئله نقشه رومانی - شکل شماره ۱

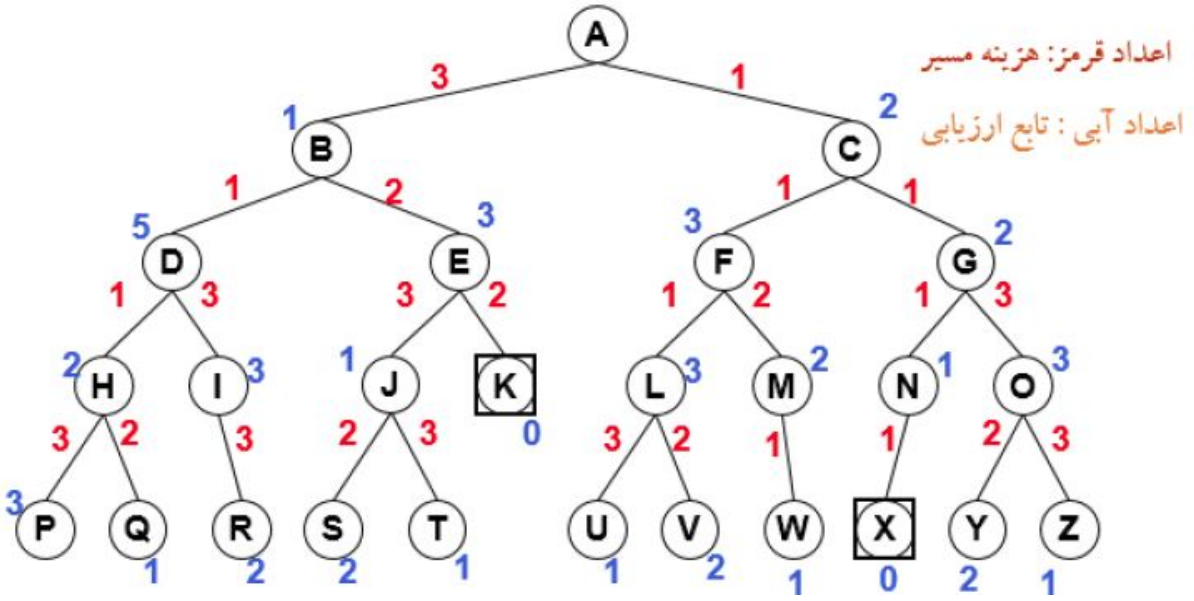


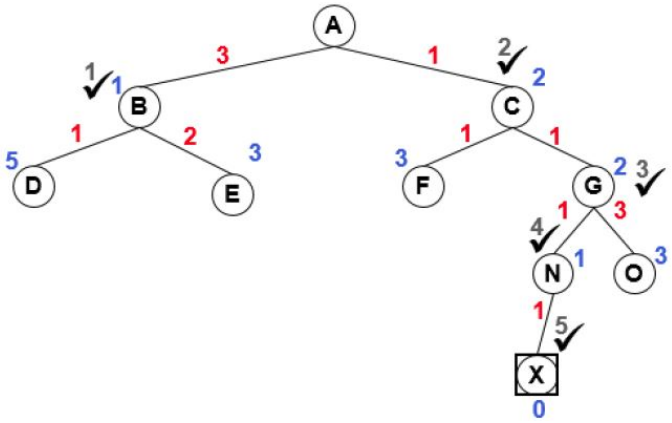
حل مسئله نقشه رومانی - شکل شماره ۲

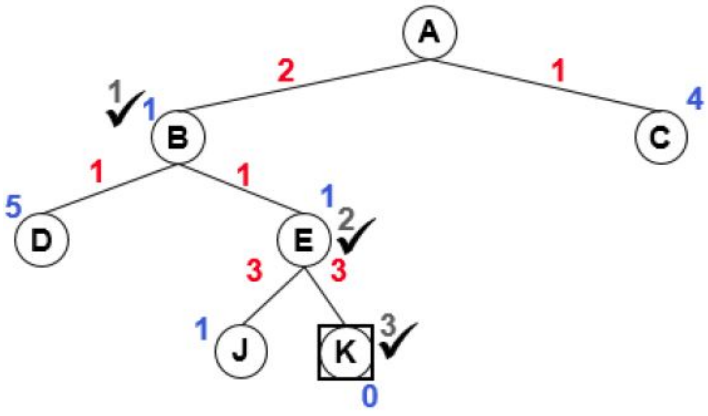


حل مسئله نقشه رومانی - شکل شماره ۳

مثال دوم


اعداد قرمز: هزینه مسیر
اعداد آبی : تابع ارزیابی
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z


A
B
C
D
E
F
G
N
O
X


A
B
C
D
E
J
K

الگوریتم جستجوی حریصانه برای نگهداری گره ها از صف اولویت استفاده میکند.

کامل بودن: کامل نیست! اما اگر h=h* آنگاه جستجو کامل است.

بهینه بودن: خیر! اما اگر h=h* آنگاه جستجو بهینه است.

پیچیدگی زمانی: $o(b^m)$ اما اگر h=h* آنگاه $o(b \quad d)$

پیچیدگی فضایی: $o(b^m)$ اما اگر h=h* آنگاه $o(b \quad d)$

جستجوی A*

معروف ترین شکل جستجوی Best first search، جستجوی A* است.

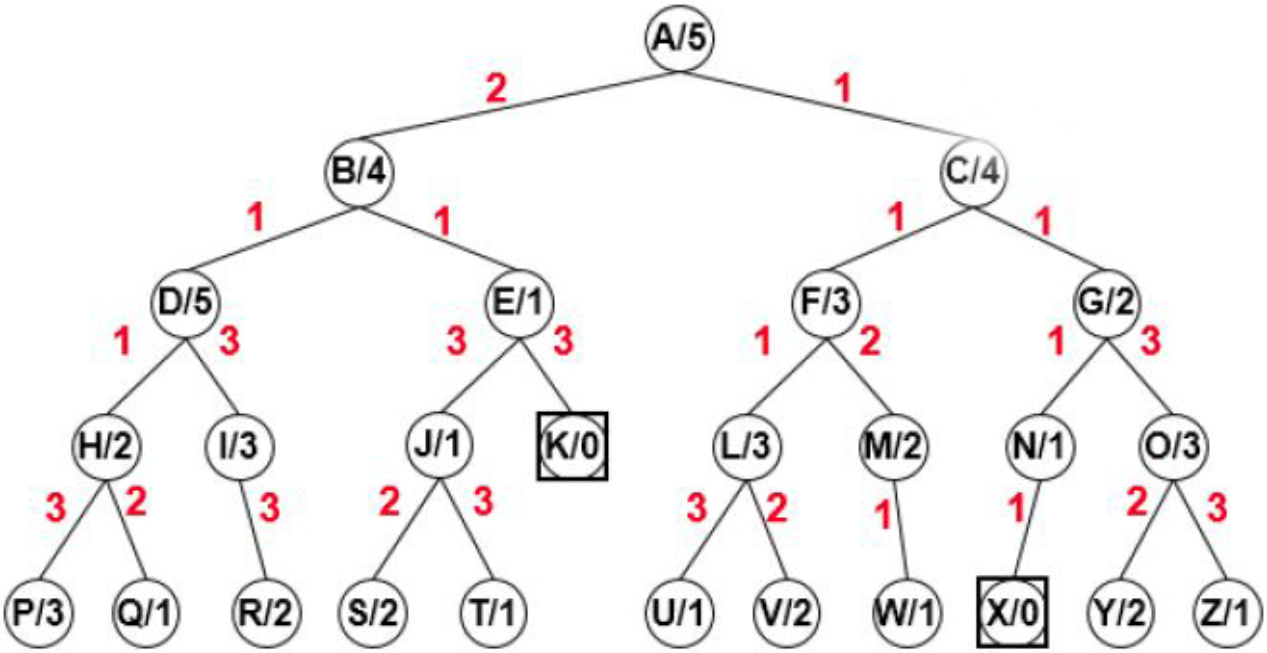
در این جستجو تابع ارزیابی به صورت زیر تعریف میشود.

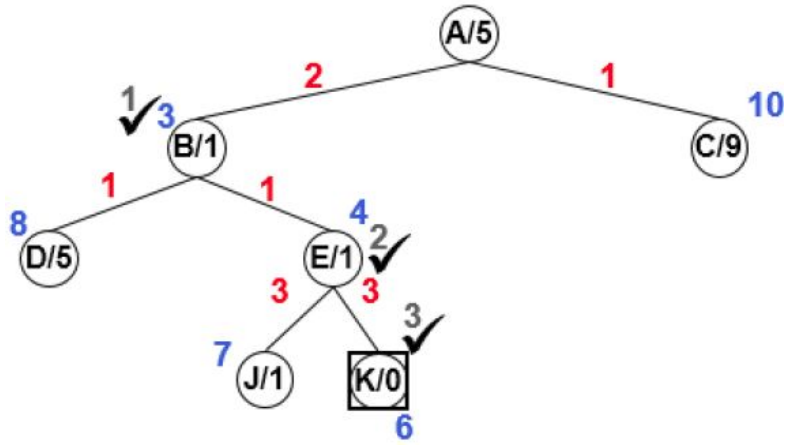
F (n) = g (n) + h (n)

تفاوت این الگوریتم با الگوریتم حریصانه در این است که در تابع ارزیابی علاوه بر تابع اکتشافی به هزینه رسیدن به گره هم توجه دارد.

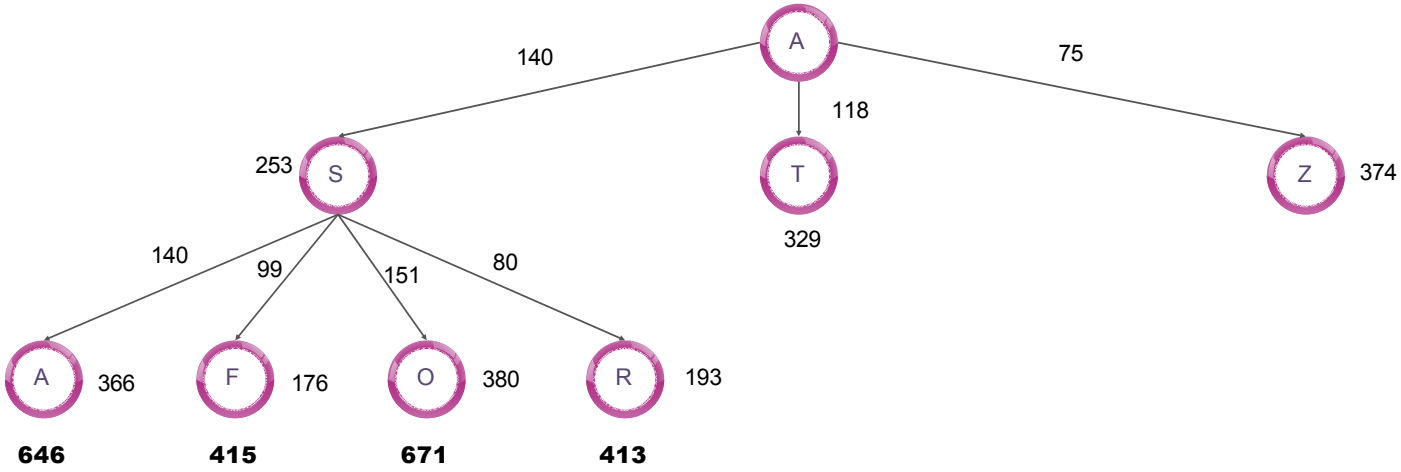
جستجوی A* – مثال ۱

اعداد روی یال هزینه مسیر و اعداد درون گره ها مقادیر تابع اکتشافی را نشان می دهند.





جستجوی A* – مثال ۲ – نقشه رومانی



جستجوی A*

دلیل اینکه در A* حافظه کم می آید، تمامی گره ها را از ریشه تا آخر نگه میدارد.

کامل بودن: کامل است چون بالاخره به جواب میرسیم.

بهینه بودن: بهینه است، در صورتیکه h (n) یک اکتشاف قابل قبول باشد.

پیچیدگی زمانی: $o(b^m)$ اما اگر h* = h آنگاه $o(b\ d)$

پیچیدگی فضایی: $o(b^m)$ اما اگر h* = h آنگاه $o(b\ d)$

الگوریتم A* بصورت مجموعه ای پیش می رود. الگوریتم جستجوی A* برای نگهداری گره ها از صف اولویت دار استفاده میکند.

در چه زمانی جستجوی A* کامل نمی شود ؟

زمانیکه مقادیر یا هزینه بزرگتر از صفر نباشد، و فاکتور انشعاب (b) نا متناهی باشد .

F (n) = h (n)

F (n) = h (n) + g (n)

نکته مهم:

همانطور که الگوریتم A* را بررسی کردیم مشاهده شد حافظه مورد نیاز برای الگوریتم فوق خیلی زیاد بود. (مثال های قبل)

الگوریتم IDA*

الگوریتم فوق برای کاهش حافظه مورد نیاز A* ستاره مطرح گردید. این الگوریتم از روش جستجوی عمیق شونده ی تکراری استفاده کرده و شکل الگوریتم به شکل IDA* کامل شده است.

نکته مهم:

تفاوت عمده الگوریتم فوق با الگوریتم IDS (الگوریتم استاندارد عمقی تکراری) در این است که در الگوریتم IDS برش بر اساس عمق بود (مقدار برش یا Cut of Value) ولی در الگوریتم IDA* مقدار برش برابر بود با f (n) = g (n) + h (n) این الگوریتم از رابطه زیر بدست می آید.

f (n) = g (n) + h (n)

یعنی مقدار برش برابر است با کمترین f گرهی که از مقدار برش مرحله قبل بیشتر باشد.

چون این الگوریتم بشکل عمقی پیاده سازی میشود و درخت جستجو را تشکیل میدهد پس پیچیدگی مکان آن خطی است.

پیچیدگی زمانی مانند A* کامل بودن و بهینه بودن همانند A* .

الگوریتم جستجوی بازگشتی یا Recursive Best-First Search (RBFS)

این الگوریتم بهترین گره را برای بسط انتخاب میکند و شرایط پیچیدگی مکانی خطی را هم حفظ میکند.

نحوه کار بشرح زیر میباشد:

از بین گره های ریشه بهترین گره را برای بسط انتخاب میکند.

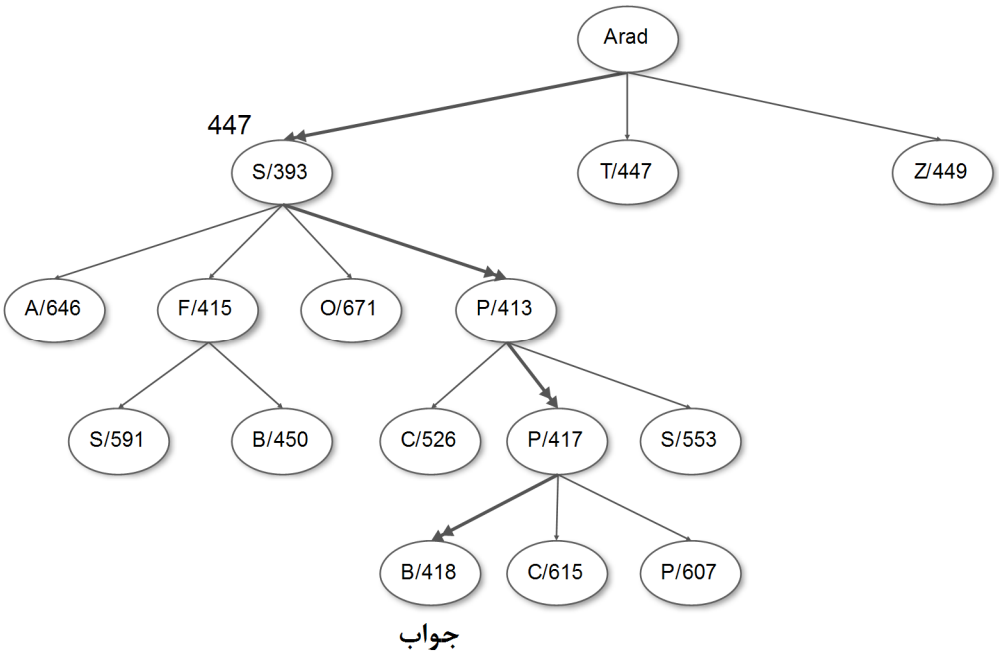
نکته مهم:

پس کمترین f گره های دیگر را در خود بعنوان بهترین دوم ذخیره میکند. سپس از بین فرزندان گره بسط داده شده بهترین را انتخاب کرده اگر این بهترین از بهترین دوم پدر، بهتر نباشد کل فرزندان گره های بسط داده شده حذف می گردد و بهترین فرزند جایگزین پدر خود میشود.

سپس دوباره مراحل را تکرار میکنیم. اگر بهترین فرزند گره بسط داده شده از بهترین دوم بهتر باشد آن گره بسط داده شده و برای آن گره بهترین دوم از گره های موجود را انتخاب و ذخیره مینماییم.

این کار آنقدر تکرار میشود تا به هدف برسیم.

مثال: حل مسئله نقشه رومانی با استفاده از الگوریتم RBFS



جواب

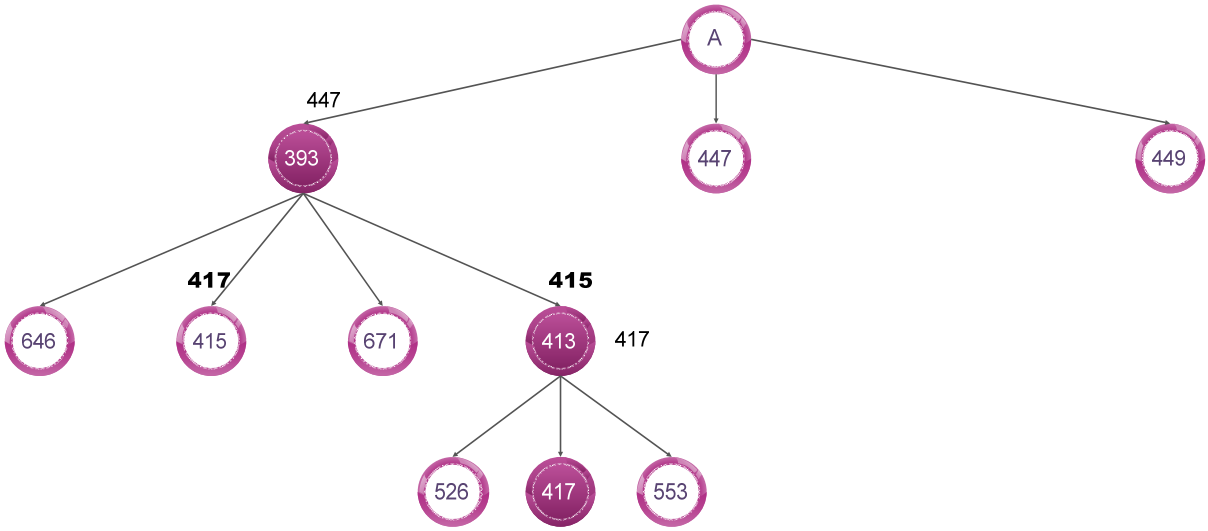
مشکل این الگوریتم این است که پیچیدگی زمانی قابل محاسبه نیست.

پیچیدگی مکانی: خطی است، چون جستجو عمقی است.

پیچیدگی فضایی: $o(b\ \ d)$

بهینه بودن: بهینه است اگر تابع اکتشافی قابل قبولی داشته باشد

کامل بودن: کامل است چون بالاخره به جواب میرسیم.



RBFS تا حدی از IDA* کارآمد تر است! اما گره های زیادی تولید میکند. IDA* و RBFS ممکن است یک گره را بیش از یکبار تولید کنند و بسط دهند که باعث سر باری و زمان اجرایی زیاد می باشد.

IDA* و RBFS به علت تولید تکراری گره ها در معرض افزایش نمایی پیچیدگی زمانی قرار دارند.

جستجوی تپه نوردی

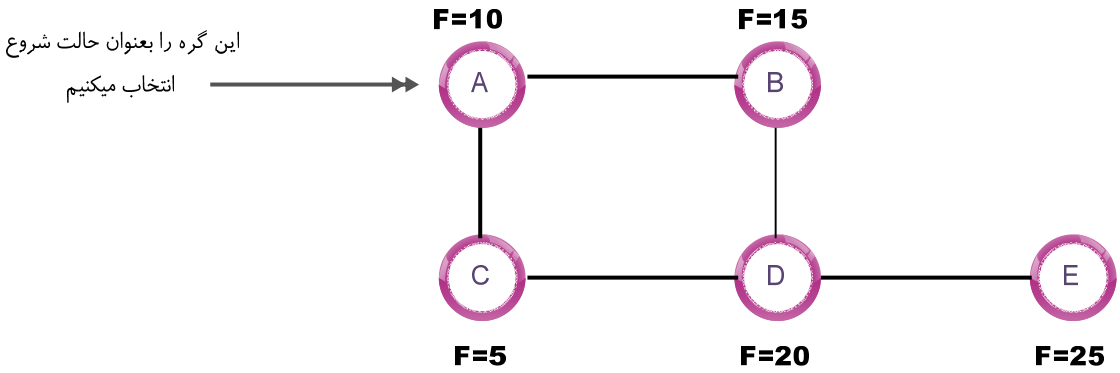
روش کار این الگوریتم به این شکل است که همیشه به سمت بیشترین ارزش حرکت میکند، یعنی این الگوریتم بصورت سعودی عمل کرده و هر جا به قله رسید الگوریتم متوقف میشود.

مراحل حل الگوریتم فوق به شرح زیر می باشد.

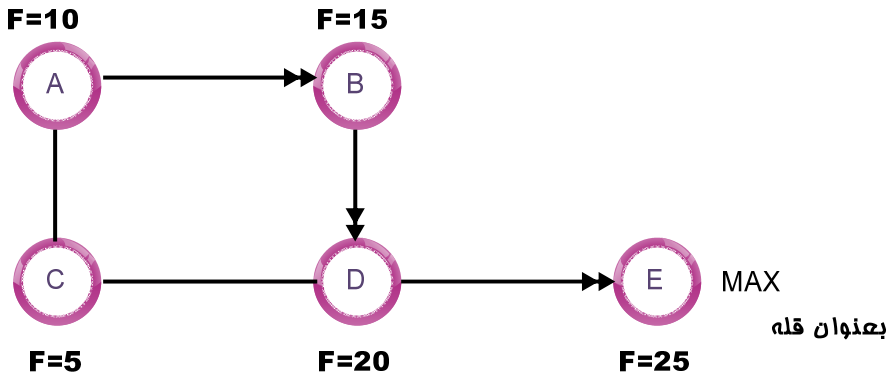
- در ابتدا بصورت پیش فرض یک حالت را بعنوان حالت اولیه در نظر میگیریم.
- بهترین همسایه را با وضعیت فعلی مقایسه میکنیم . اگر بهترین همسایه را که انتخاب کرده ایم از وضعیت فعلی بهتر نباشد، وضعیت فعلی را قله اعلام میکنیم و الگوریتم متوقف میشود.
- اگر بهترین همسایه را که انتخاب کرده ایم از وضعیت فعلی بهتر باشد، بعنوان وضعیت شروع انتخاب میکنیم.

نحوه کار الگوریتم فوق،

- یکی از وضعیت ها را بصورت تصادفی به حالت شروع انتخاب میکنیم.



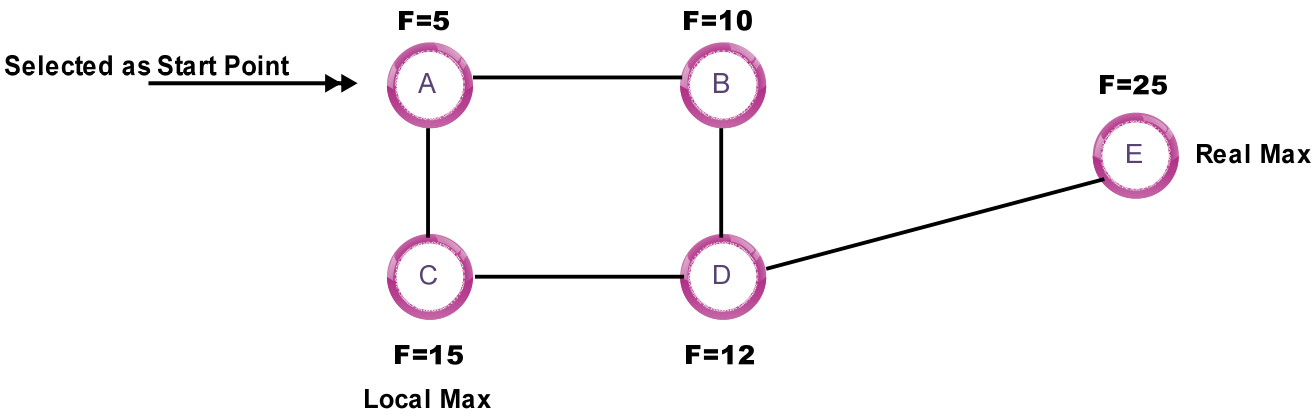
- سپس همسایه های گره انتخاب شده را برسی کرده، بهترین همسایه را انتخاب میکنیم.
- همسایه را با گره آغازین مقایسه کرده، اگر از گره شروع بهتر نباشد، الگوریتم گره شروع را بعنوان قلع معرفی میکند و الگوریتم متوقف میشود.
- در غیر اینصورت یعنی بهترین همسایه بعنوان وضعیت شروع انتخاب میشود. این کار را آنقدر تکرار میکنیم تا Max را بدست آوریم.



مشکل الگوریتم تپه نوردی :

در الگوریتم تپه نوردی اگر بهترین همسایه از وضعیت فعلی بهتر نباشد، الگوریتم وضعیت جاری را بعنوان قله اعلام میکند ولی در حالیکه قله مقدارش چیز دیگری است.

اصطلاحاً به این مشکل، مشکل Max محلی گفته میشود.



Max محلی:

گرهی است که مقدارش از بهترین همسایه بیشتر، ولی از Max سراسری کمتر است.

برای حل این مشکل از عمل انتقال فرعی، استفاده میکنیم. به این صورت که وضعیت فعلی را کمی جابجا کرده سپس دوباره جستجوی تپه نوردی را انجام می دهیم.

نکته مهم:

این جابجایی باید به اندازه ی محدود باشد در غیر این صورت، الگوریتم در حلقه بی نهایت می افتد.

شبه کد مربوط به الگوریتم تپه نوردی:

Function HILL-CLIMBING (problem) **return** a state that is a local maximum

Input: problem, a problem

Local variables: current**, a node.**

Neighbor**, a node.**

Current ← MAKE-NODE (INITIAL-STATE [problem])

Loop do

Neighbor ← a highest valued successor of current

If VALUE [neighbor] ≤ VALUE [current] **then return** STATE [current]

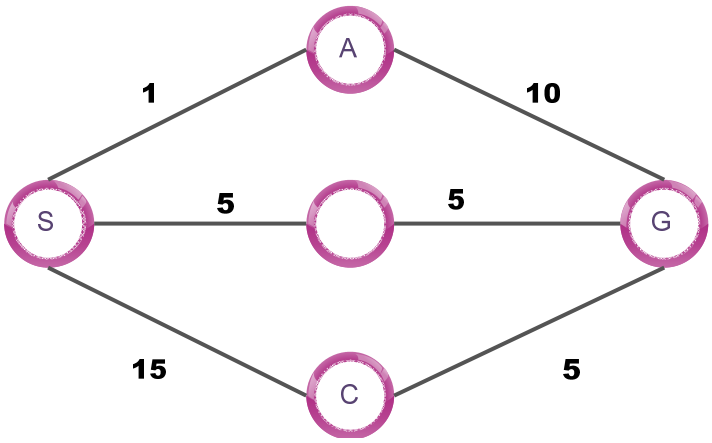
Current ← neighbor

تست های فصل سوم

۱ – کدامیک از الگوریتم های زیر میتواند جواب بهینه برای مسئله ی رومانی را پیدا کند؟

الف) BFS　　ب) UCS　　ج) DLS　　د) IDS

۲ – اگر در گراف زیر از جستجو ی هزینه یکنواخت استفاده کنیم، ترتیب قرار گرفتن گره ها در صف از چپ به راست چه خواهد بود؟



الف) SABCG　　ب) SABG　　ج) SBACG　　د) SBG

۳ – پیچیدگی زمانی در جستجوی تعمیق تکراری به کدامیک از عوامل زیر بستگی دارد.

الف) بیشترین عمق درخت

ب) سایر فضای حالت

ج) تابع مکاشفه ای انتخاب شده

د) عمق کمترین عمق گره هدف

فصل چهارم

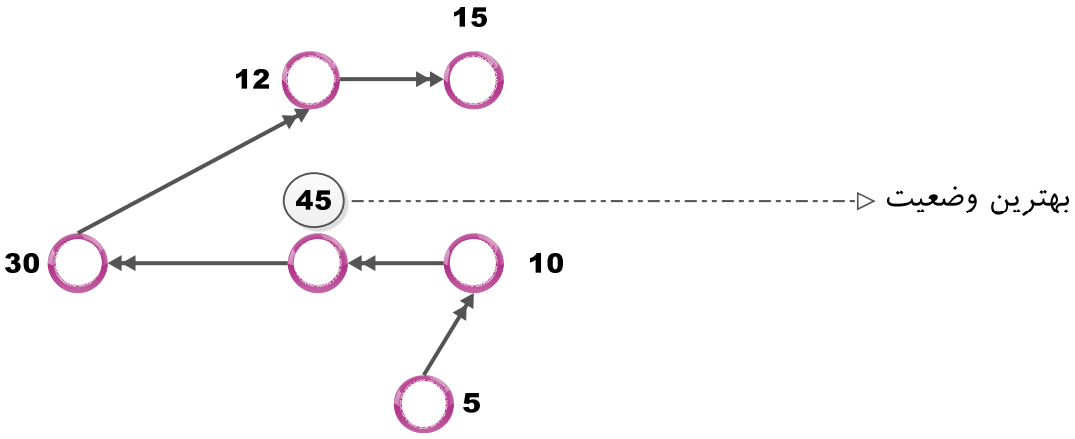
Chapter Four

الگوریتم جستجوی قدم زدن تصادفی

روش کار این الگوریتم به شرح زیر می باشد:

الگوریتم فوق یک نقطه ای را بعنوان وضعیت فعلی در نظر میگیرد، سپس بصورت تصادفی یکی از همسایه های خود را انتخاب میکند. این وضعیت به همین شکل تکرار میشود. احتمال رفتن به همه همسایه ها، چه وضعیتی بهتر نسبت وضعیت فعلی داشته باشد چه نداشته باشد، یکسان است! پس احتمال آنها برابر است.

الگوریتم فوق، بهترین نقطه بازدید شده را در حافظه ی خود نگه داشته سپس آنرا بعنوان جواب بر میگرداند.



الگوریتم فوق **کامل** است چون در نهایت به جواب خواهد رسید. ولی از لحاظ **بهینه** بودن، بهینه نیست چون الگوریتم ممکن است زمان زیادی را صرف کند تا به نقطه مورد نظر برسد. در الگوریتم یاد شده **پیچیدگی زمانی** "نمایی" است.

الگوریتم جستجوی شبیه ساز ذوب فلزات

روش کار این الگوریتم به شرح زیر می باشد:

یک نقطه را بصورت تصادفی بعنوان وضعیت فعلی انتخاب میکند. سپس از بین همسایه ها یک همسایه را بشکل تصادفی انتخاب میکند اگر وضعیت همسایه از وضعیت ابتدایی بهتر باشد، به حالت جدید (همسایه با وضعیت بهتر) تغییر وضعیت میدهد در غیر اینصورت احتمال تغییر وضعیت از فرمول زیر محاسبه شده و به آن وضعیت تغییر پیدا میکند.

$$\text{احتمال تغییر وضعیت} = e^{\Delta E/T}$$

در فرمول فوق حرف T معرف دما و ΔE نشان دهنده اختلاف سطح انرژی می باشد.

ΔE از فرمول زیر محاسبه میشود:

$$\Delta E = \text{مقدار فعلی} - \text{مقدار همسایه}$$

نکته مهم:

اگر مقدار دما (T) خیلی زیاد باشد، مقدار E هم افزایش پیدا میکند که باعث میشود احتمال رفتن به نقاط بدتر از وضعیت فعلی، افزایش یابد. اگر دما کاهش پیدا کند مقدار $e^{\Delta E/T}$ کاهش پیدا خواهد کرد تا زمانیکه مقدار T برابر با صفر شود. در اینجا الگوریتم متوقف شده و نقطه فعلی بعنوان جواب برگشت داده خواهد شد.

الگوریتم فوق کامل است به شرطی که T به سمت کاهش دما و مقدارش نزولی باشد اما بهینه نیست چون عملکرد آن بصورت تصادفی است. در ضمن پیچیدگی زمانی و مکانی آن بشکل نمایی است. کاربرد این الگوریتم در مدارات DSLI و در زمانبندی کارخانه ها مورد استفاده قرار میگیرد.

الگوریتم جستجوی شبیه سازی حرارتی

Function SIMULATED-ANNEALING (*Problem*, *Schedule*) **returns** a solution state

Inputs: *Problem*, a problem

Schedule, a mapping from time to temperature

Local Variables: *Current*, a node

Next, a node

T, a "temperature" controlling the probability of downward steps

Current = MAKE-NODE (INITIAL-STATE [*Problem*])

For t = 1 **to** ∞ **do**

T = *Schedule*[t]

If T = 0 **then return** *Current*

Next = a randomly selected successor of *Current*

ΔE = VALUE [*Next*] – VALUE [*Current*]

If ΔE > 0 **then** *Current* = *Next*

Else *Current* = *Next* only with probability exp(-ΔE/T)

الگوریتم ژنتیک

Genetic Algorithm (GA)

الگوریتم ژنتیک

Genetic Algorithm (GA)

روش کار این الگوریتم به شرح زیر می باشد:

الگوریتم ژنتیک از نظریه داروین استفاده میکند. نظریه داروین میگوید:

در هر جامه ای تعداد افراد جمعیت از یک نسل به نسل دیگر تغییر نمی کند، یعنی تقریباً ثابت است.

چون عوامل محیطی باعث مرگ میر در افراد میشود. پایه ژنتیک از نظریه داروین است!

کاربرد الگوریتم ژنتیک

بهینه سازی، برنامه نویسی خودکار، اقتصاد، تحقیق در عملیات، محیط زیست، مطالعه ی تکامل و یادگیری و سیستم اجتماعی و ....

بهترین کاربرد الگوریتم ژنتیک در بهینه سازی است !

اجزای الگوریتم ژنتیک

- کروموزوم Chromosome
- جمعیت Population
- تابع برا زندگی Fitness Function

عمل گرهای الگوریتم ژنتیک

- عملگر انتخاب Selection
- عمل گر ترکیب Crossover
- عمل گر جهش Mutation

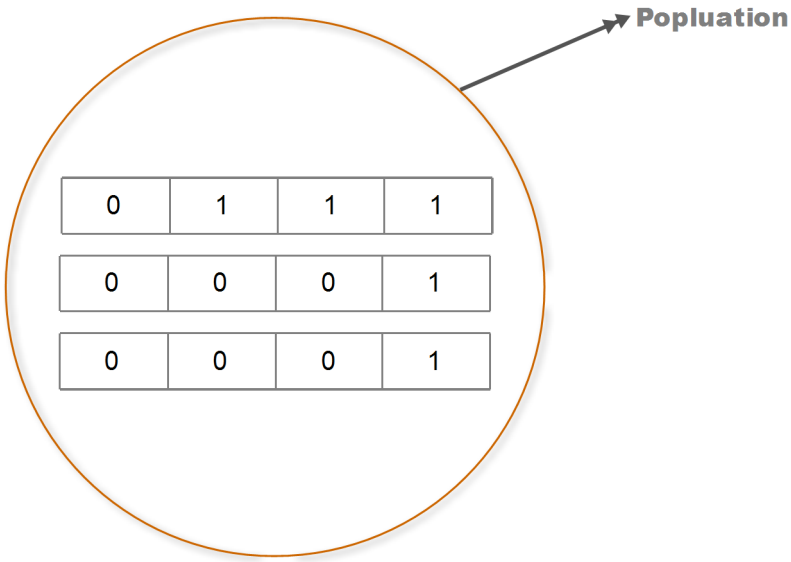
اجزای الگوریتم ژنتیک

کروموزوم Chromosome

کروموزوم در الگوریتم های ژنتیک یک نقطه از فضای جستجو یا یک راه حل برای مسئله مورد نظر می باشد. برای نمایش کروموزوم از کد گذاری دو دویی استفاده میکنند.

جمعیت Population

به مجموعه ای از کروموزوم ها جمعیت یا Population میگویند. که با تاثیر عمل گرها روی جمعیت، جمعیت بعدی شکل میگیرد.



تابع برازندگی Fitness Function

برای حل هر مسئله توسط الگوریتم ژنتیک باید یک تابع شایستگی ابداع کنیم. که این تابع شایستگی عمدی است غیر منفی و میزان شایستگی هر کروموزوم را نشان میدهد.

عمل گرهای الگوریتم ژنتیک

عمل گر انتخاب Selection

توسط این عمل گر از بین کروموزوم های موجود در جمعیت، تعدادی بصورت تصادفی انتخاب می شود.

نکته مهم:

هر کروموزومی که برازندگی بیشتر داشته باشد، احتمال انتخاب آن بیشتر است. وقتی انتخاب کرد عمل ترکیب را انجام میدهد.

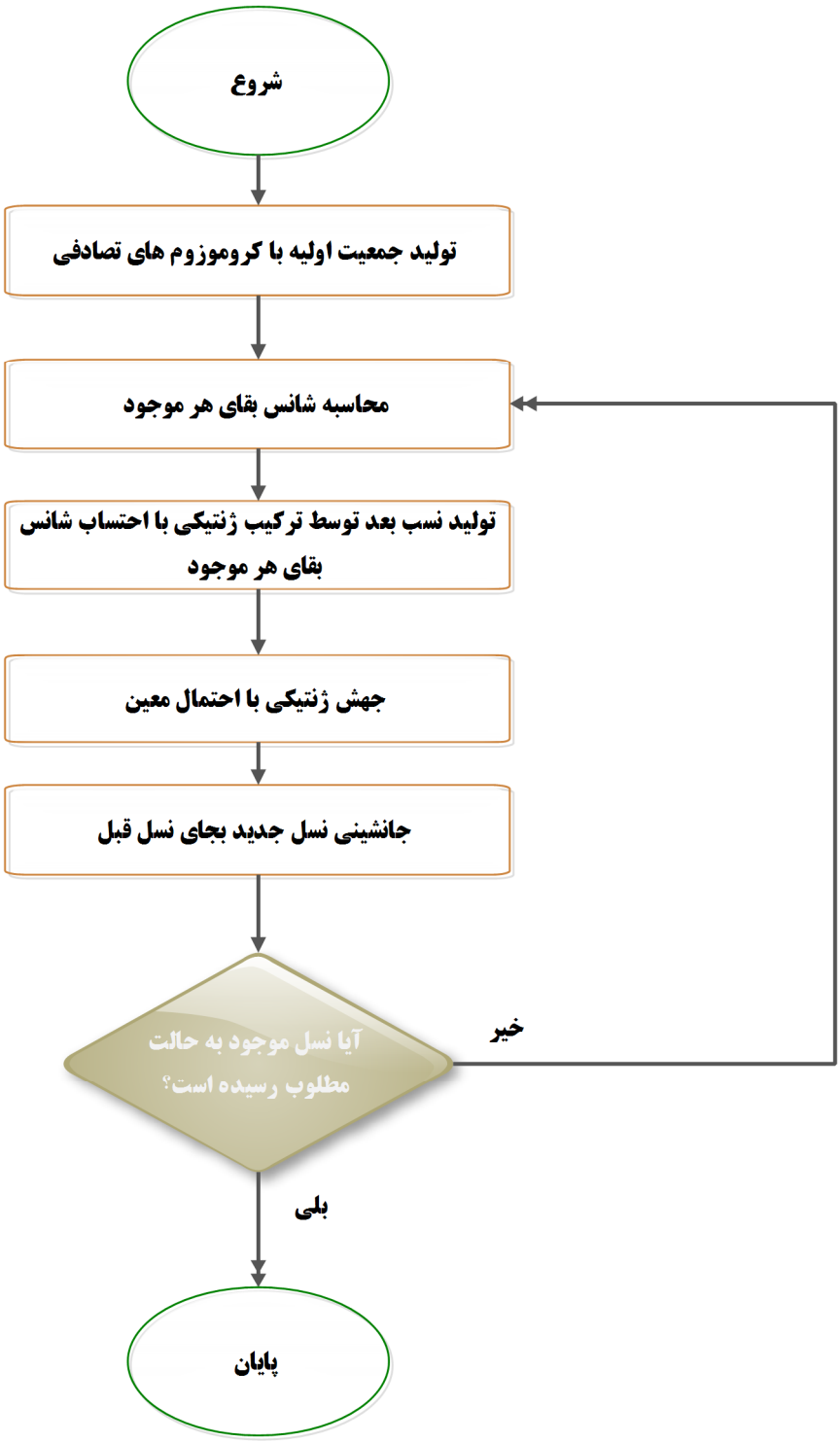
عمل گر ترکیب Crossover

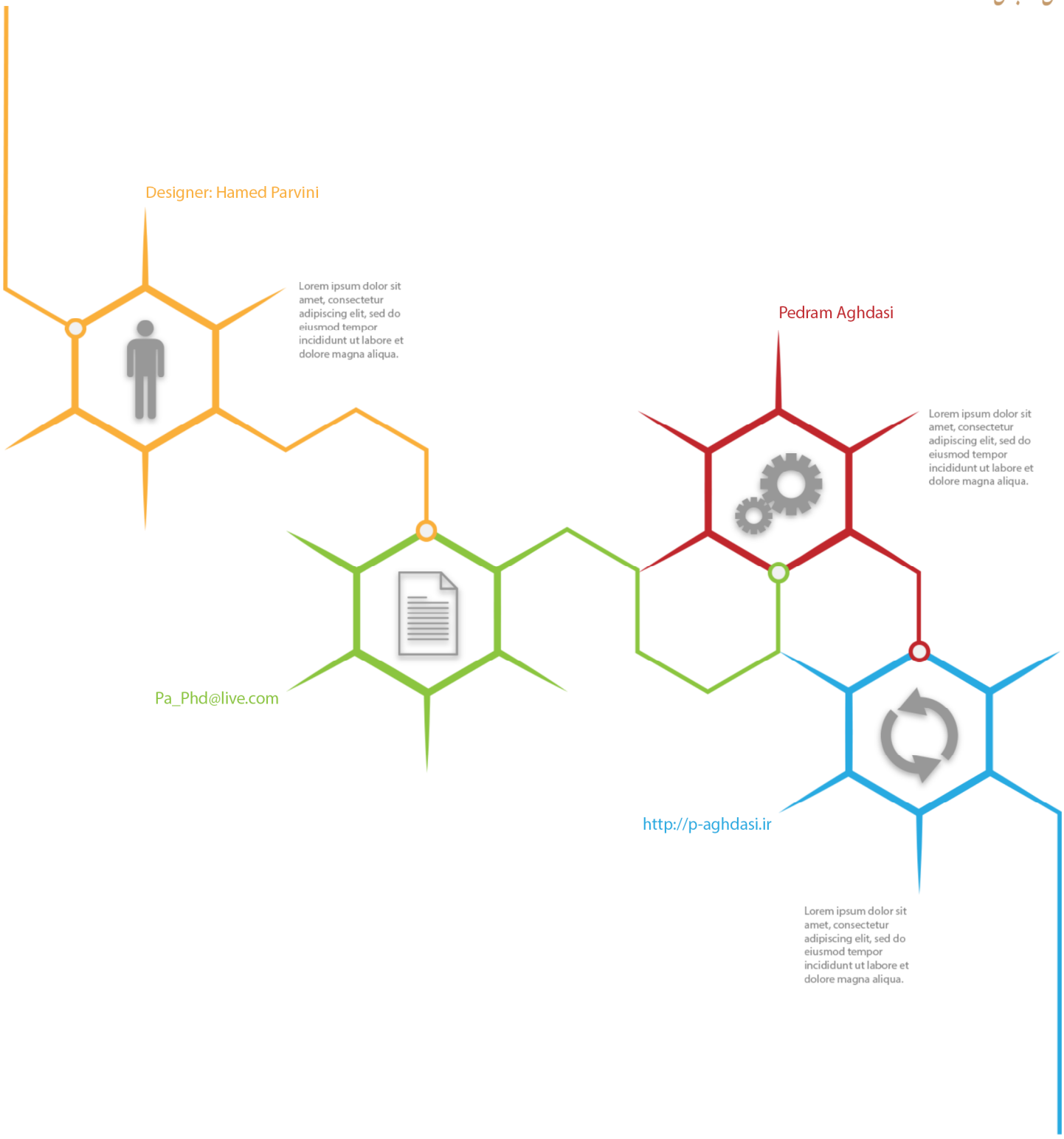
توسط این عمل گر بخشی از کروموزوم های انتخاب شده با هم عوض میشوند که این عمل باعث میشود فرزندان تولید شده خصوصیات والدین خود را داشته ولی دقیقاً مشابه یکی از والدین نشود. هدف اصلی این عمل، تولید فرزند جدید با مشخصات بهینه است.

عمل گر جهش Mutation

توسط این عمل گر یکی از ژن های کروموزوم بصورت تصادفی انتخاب شده و مقدارش تغییر پیدا میکند. کروموزوم های حاصل، نسب بعدی را تشکیل داده و دوباره این عمل تکرار میشود.

فلو چارت الگوریتم ژنتیک





Pedram Aghdasi

Pa_Phd@live.com

www.p-aghdasi.ir

University of Applied Science And Technology, Tabriz, Iran

پایان

دی ماه ۱۳۹۳