« به نام او »

آموزش شبیه سازی دو بعدی فوتبال

نویسندگان : محمدعلی میرزایی - علی یعقوبی

مرجع روبوکاپ ایران

www.iranrcss.com

جلسه هشتم

مباحث این جلسه :

۱- آموزش بیس **UVA_Trilearn Base**

۲- **هوش مصنوعی، حل مسأله به روش جستجو**

امیدوارم تا به اینجای آموزش برای شما مفید بوده باشد . به ادامه بحث جلسه قبل می پردازیم :

```
۱ - else if( WM->getFastestInSetTo( OBJECT_SET_TEAMMATES, OBJECT_BALL, &iTmp )
۲ -              == WM->getAgentObjectType()  && !WM->isDeadBallThem() )
۳ -   {                                                  // if fastest to ball
۴ -     Log.log( ۱۰۰, "I am fastest to ball; can get there in %d cycles", iTmp );
۵ -     soc = intercept( false );                        // intercept the ball
۶ -
۷ -     if( soc.commandType == CMD_DASH &&               // if stamina low
۸ -         WM->getAgentStamina().getStamina() <
۹ -            SS->getRecoverDecThr()*SS->getStaminaMax()+۲۰۰ )
۱۰-      {
۱۱-        soc.dPower = ۳۰.۰ * WM->getAgentStamina().getRecovery(); // dash slow
۱۲-        ACT->putCommandInQueue( soc );
۱۳-        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
۱۴-      }
۱۵-      else                                             // if stamina high
۱۶-      {
۱۷-        ACT->putCommandInQueue( soc );                 // dash as intended
۱۸-        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
۱۹-      }
۲۰-     }
۲۱-     else if( posAgent.getDistanceTo(WM->getStrategicPosition()) >
۲۲-                  ۱.۵ + fabs(posAgent.getX()-posBall.getX())/۱۰.۰)
۲۳-                                                  // if not near strategic pos
۲۴-     {
۲۵-       if( WM->getAgentStamina().getStamina() >     // if stamina high
۲۶-                              SS->getRecoverDecThr()*SS->getStaminaMax()+۸۰۰ )
```

```
۲۷-        {
۲۸-          soc = moveToPos(WM->getStrategicPosition(),
۲۹-                          PS->getPlayerWhenToTurnAngle());
۳۰-          ACT->putCommandInQueue( soc );            // move to strategic pos
۳۱-          ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
۳۲-        }
۳۳-        else                                        // else watch ball
۳۴-        {
۳۵-          ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT_BALL ) );
۳۶-          ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
۳۷-        }
۳۸-      }
۳۹-      else if( fabs( WM->getRelativeAngle( OBJECT_BALL ) ) > ۱۰ ) // watch ball
۴۰-      {
۴۱-        ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT_BALL ) );
۴۲-        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
۴۳-      }
۴۴-      else                                          // nothing to do
۴۵-        ACT->putCommandInQueue( SoccerCommand(CMD_TURNNECK,۰۰) );
۴۶-    }
```

بعد از بررسی درستی شرط برنامه وارد مرحله اجرای دستورات داخل شرط می شود . اگر این شرط صحیح باشد ، بدین معناست که بازیکن stamina کمی دارد و درحال dash کردن می باشد .

```
soc.dPower = ۳۰۰ * WM->getAgentStamina().getRecovery(); // dash slow
```

کاری که این دستور انجام می دهد این است که سرعت dash کردن بازیکن را کاهش می دهد . با این کار بازیکن فرصت پیدا می کند تا stamina از دست رفته ی خود را ریکاور کند .

```
ACT->putCommandInQueue( soc );
ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
```

همانطور که قبلا هم اشاره کرده بودم؛ دستور بالا برای فرستادن دستور به سرور می باشد . ما در هر سایکل تنها یک دستور می توانیم به سرور ارسال کنیم اما همراه این دستور ما می توانیم یک چرخش گردن نیز داشته باشیم. حال به بررسی تابع های مربوط به چرخش گردن که یکی از مهمترین توابع بیس می باشد ،می پردازیم : در تابع basicPlayer.h ، ۳ تابع مربوط به چرخش گردن موجود است :

```
۱-  SoccerCommand   turnBackToPoint         ( VecPosition   pos,
                                              int           iPos = ۱        );

۲-  SoccerCommand   turnNeckToPoint         ( VecPosition   pos,
                                              SoccerCommand com             );

۳-  SoccerCommand   turnNeckToObject        ( ObjectT       o,
                                              SoccerCommand com             );
```

تابع شماره ۱ و ۲ کاری مشابه هم انجام می دهند و تفاوتشان در آرگومان های ورودی شان می باشد . این دو تابع گردن را به سمت نقطه ای که به تابع می دهیم ، می چرخاند و تابع شماره ۳ به سمت objectT داده شده است . آرگومان دوم تابع های ۲ و ۳ تنها **SoccerCommand** است که باید به تابع بدهید تا همراه آن اجرا می شود . این دستور باید دستوری باشد که قبل از اجرای **turnNeck** به سرور ارسال شده است .

```
else                                            // if stamina high
{
ACT->putCommandInQueue( soc );                  // dash as intended
ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
}
```

حال اگر شرط درست نبود ، دستور را بدون تغییر به سرور می فرستیم .

تا به اینجای کار، ما تمامی دستورات زیر را با هم بررسی کردیم :

```
else if( WM->getFastestInSetTo( OBJECT_SET_TEAMMATES, OBJECT_BALL, &iTmp )
              == WM->getAgentObjectType()  && !WM->isDeadBallThem() )
   {                                                  // if fastest to ball
     Log.log( ۱۰۰, "I am fastest to ball; can get there in %d cycles", iTmp );
     soc = intercept( false );                        // intercept the ball

     if( soc.commandType == CMD_DASH &&               // if stamina low
         WM->getAgentStamina().getStamina() <
            SS->getRecoverDecThr()*SS->getStaminaMax()+۲۰۰ )
      {
        soc.dPower = ۳۰.۰ * WM->getAgentStamina().getRecovery(); // dash slow
        ACT->putCommandInQueue( soc );
        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
      }
      else                                             // if stamina high
      {
        ACT->putCommandInQueue( soc );                 // dash as intended
        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
      }
     }
```

همانطور که قبلا نیز اشاره شده بود ، دستورات بالا یک stamina manager ساده می باشد که شما بنا بر نیازتان باید آنرا تقویت کنید .

در جلسه بعد به طور مفصل به بحث دفاع خواهیم پرداخت .

موضوع : حل مسئله با روش جستجو

در این جلسه، برخی روش های حل مسأله را با اندازه گیری کارایی حل مسأله بررسی میکنیم.

اندازه گیری کارایی حل مسئله

- ❖ **کامل بودن: آیا الگوریتم تضمین میکند که در صورت وجود راه حل، آن را بیابد؟**
- ❖ **بهینگی: آیا این راهبرد، راه حل بهینه ای را ارائه میکند؟**
- ❖ **پیچیدگی زمانی: چقدر طول میکشد تا راه حل را پیدا کند؟**
  - ▪ **تعداد گره های تولید شده در اثنای جستجو**
- ❖ **پیچیدگی فضا: برای جستجو چقدر حافظه نیاز دارد؟**
  - ▪ **حداکثر تعداد گره های ذخیره شده در حافظه**

***

جستجوی ناآگاهانه

- ✓ **ناآگاهی این است که الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد.**
- ✓ **این الگوریتمها فقط میتواند جانشینهایی را تولید و هدف را از غیر هدف تشخیص دهند.**
- ✓ **راهبردهایی که تشخیص میدهد یک حالت غیر هدف نسبت به گره غیر هدف دیگر، امید بخش تر است، جست و جوی آگاهانه یا جست و جوی اکتشافی نامیده میشود.**

راهبردها

- **جست و جوی عرضی**
- **جست و جوی عمقی**
- **جست و جوی عمیق کننده تکراری**
- **جست و جوی هزینه یکنواخت**
- **جست و جوی عمقی محدود**
- **جست و جوی دو طرفه**

جستجوی عرضی

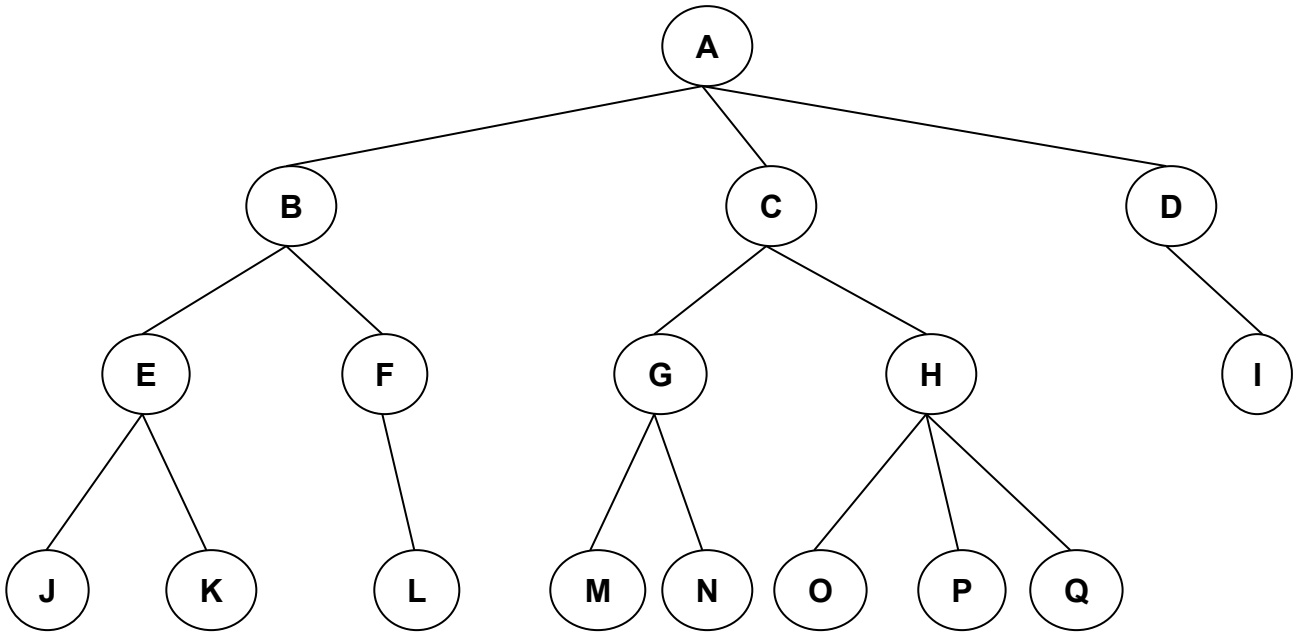
کامل بودن: بله

بهینگی: بله (مشروط)

در صورتی بهینه است که هزینه مسیر، تابعی غیر نزولی از عمق گره باشد.(مثل وقتی که فعالیتها هزینه یکسانی دارند)

پیچیدگی زمانی: $O(b^{d+1})$

پیچیدگی فضا: $O(b^{d+1})$



جستجوی عمقی

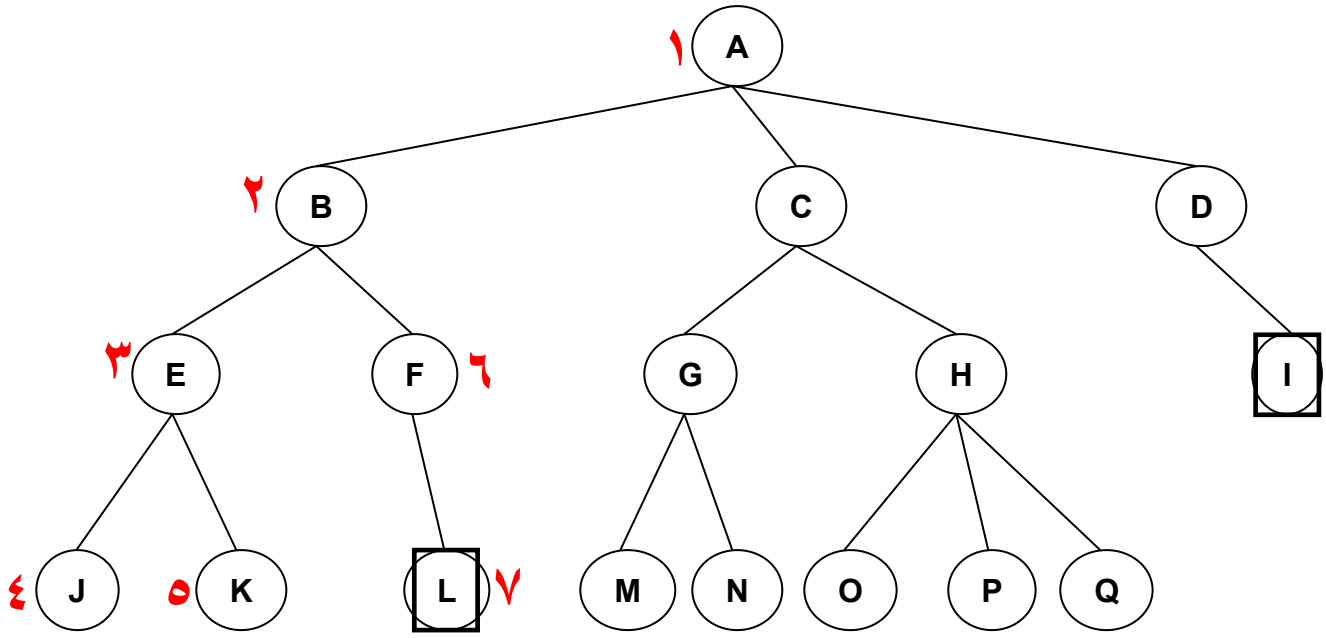
کامل بودن: خیر

اگر زیر درخت چپ عمق نامحدود داشت و فاقد هر گونه راه حل باشد، جستجو هرگز خاتمه نمی یابد.

بهینگی: خیر

پیچیدگی زمانی: $O(b^m)$

پیچیدگی فضا: $O(bm)$



جستجوی عمیق کننده تکراری

کامل بودن: بله

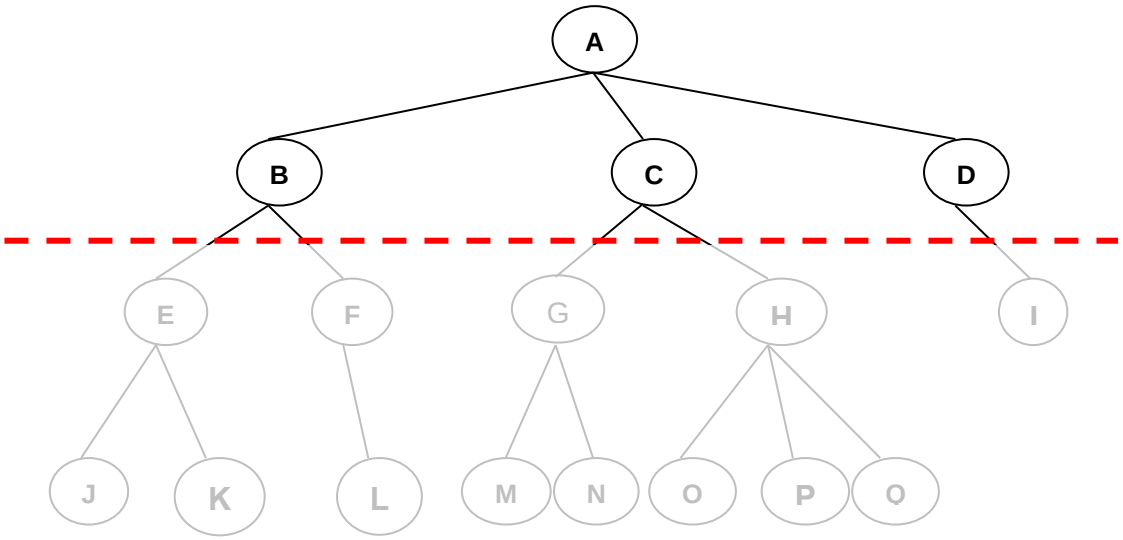
در صورتی که فاکتور انشعاب محدود باشد.

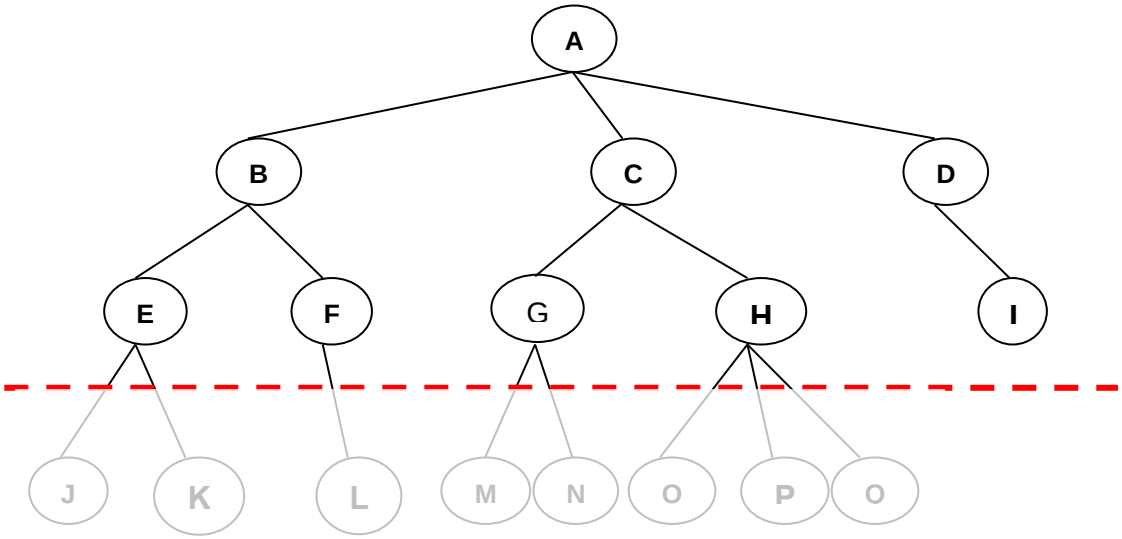
بهینگی: بله

وقتی که هزینه مسیر، تابعی غیر نزولی از عمق گره باشد.

پیچیدگی زمانی: $O(b^d)$

پیچیدگی فضا: $O(bd)$

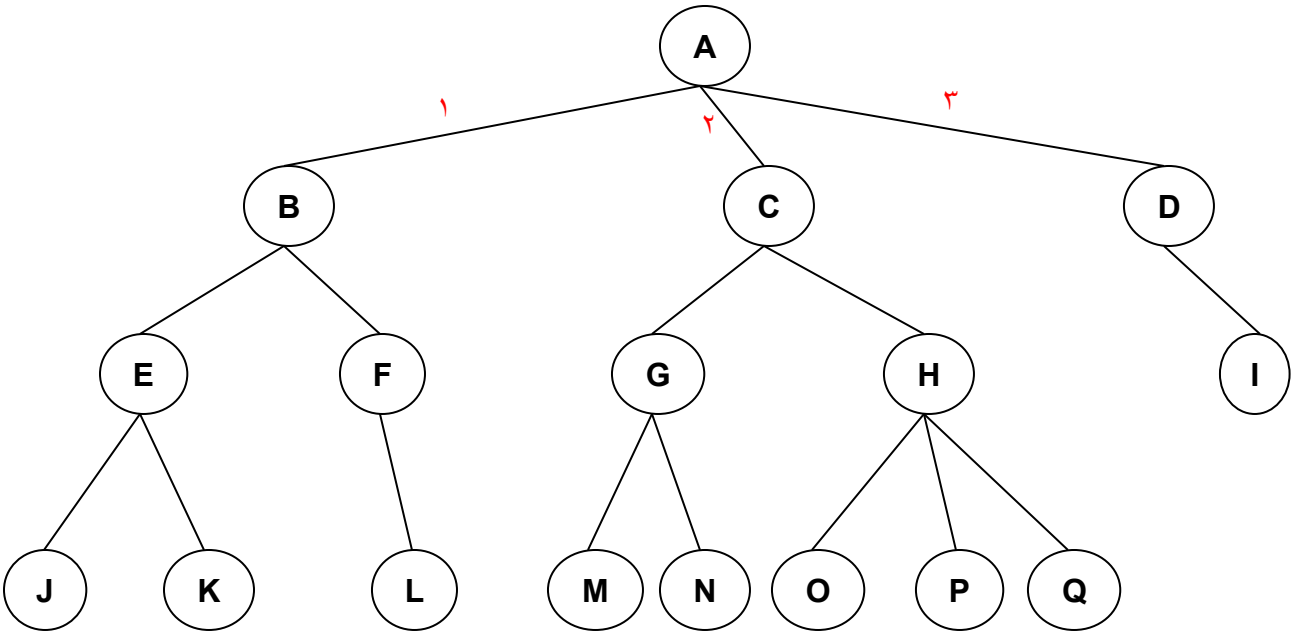




.
.
.

جستجوی هزینه یکنواخت

این جستجو گره n را با کمترین هزینه مسیر بسط میدهد.



کامل بودن: بله

هزینه هر مرحله بزرگتر یا مساوی یک مقدار ثابت و مثبت ε باشد.(هزینه مسیر با حرکت در مسیر افزایش می یابد)

بهینگی: بله

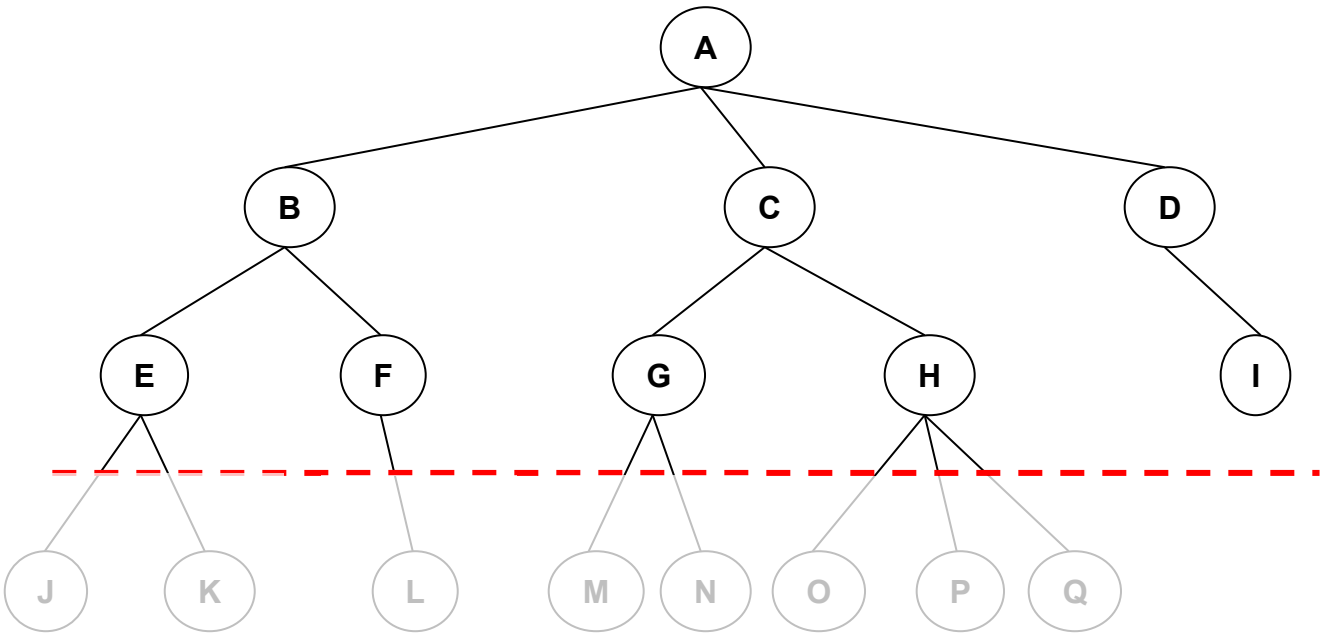
هزینه هر مرحله بزرگتر یا مساوی **ε** باشد.

پیچیدگی زمانی: $O(b^{[C^*/\varepsilon]})$

پیچیدگی فضا: $O(b^{[C^*/\varepsilon]})$

جستجوی عمقی محدود

مسئله درختهای نامحدود میتواند به وسیله جست و جوی عمقی با عمق محدود L بهبود یابد.



کامل بودن: خیر

اگر L<d و سطحی ترین هدف در خارج از عمق محدود قرار داشته باشد، این راهبرد کامل نخواهد بود.

بهینگی: خیر

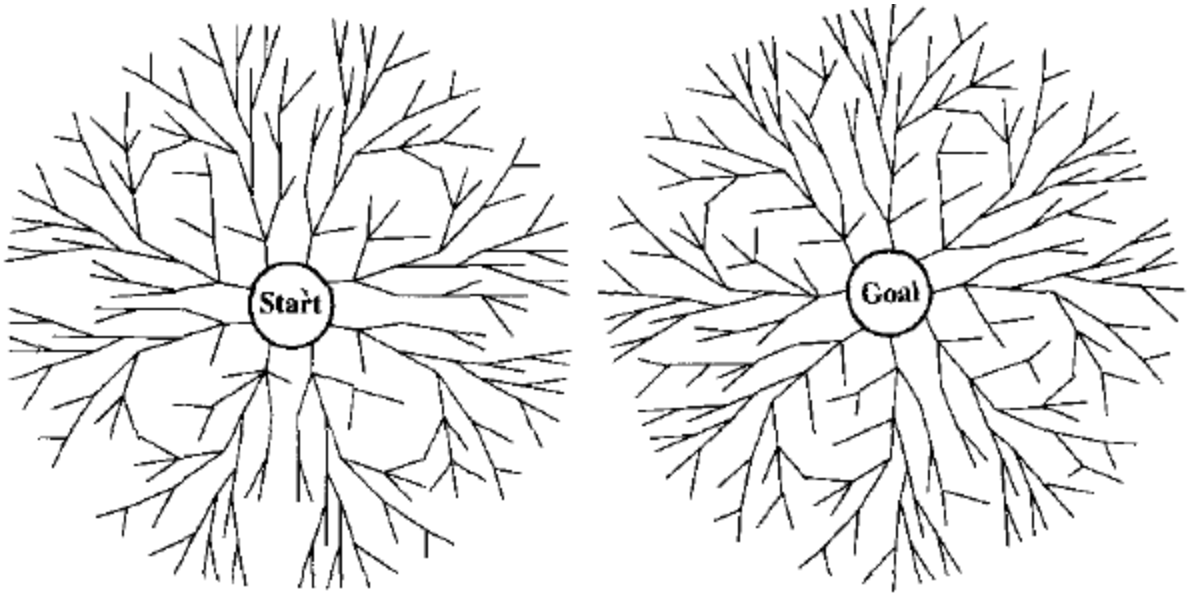
اگر L>d انتخاب شود، این راهبرد بهینه نخواهد بود.

پیچیدگی زمانی: $O(b^L)$

پیچیدگی فضا: $O(bL)$

جستجوی دو طرفه

انجام دو جست و جوی همزمان، یکی از حالت اولیه به هدف و دیگری از هدف به حالت اولیه تا زمانی که دو جست و جو به هم برسند.



کامل بودن: بله

اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد.

بهینگی: بله

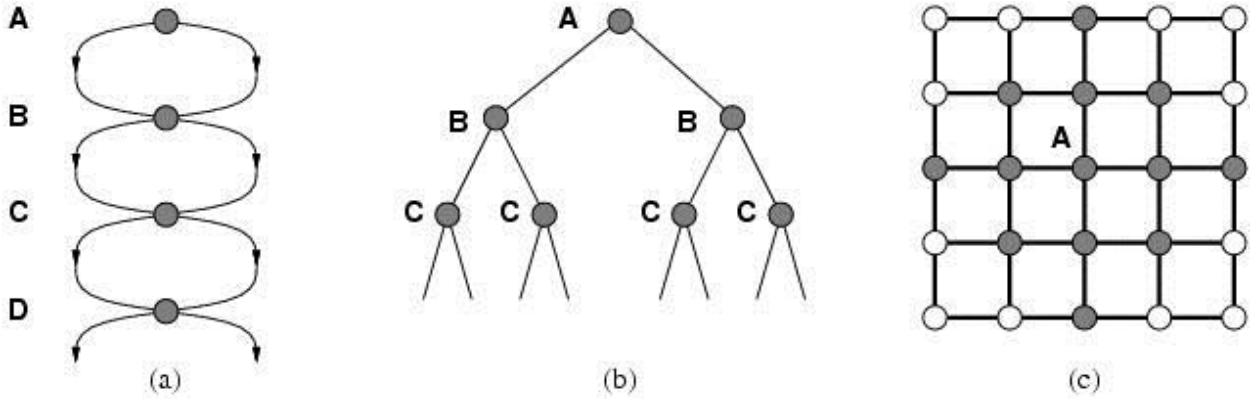
اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد

پیچیدگی زمانی: $O(b^{d/2})$

پیچیدگی فضا: $O(b^{d/2})$

اجتناب از حالتهای تکراری

وجود حالتهای تکراری در یک مسئله قابل حل، میتواند آن را به مسئله غیر قابل حل تبدیل کند.



جستجو با اطلاعات ناقص

- **مسئله های فاقد حسگر: اگر عامل فاقد حسگر باشد، میتواند در یکی از چند حالت اولیه باشد و هر فعالیت میتواند آن را به یکی از چند حالت جانشین ببرد.**
- **مسئله های اقتضایی: اگر محیط به طور جزئی قابل مشاهده باشد یا اگر فعالیتها قطعی نباشد، ادراکات عامل، پس از هر عمل، اطلاعات جدیدی را تهیه میکنند. هر ادراک ممکن، اقتضایی را تعریف میکند که باید برای آن برنامه ریزی شود.**
  - **مسائل خصمانه: اگرعدم قطعیت در اثر فعالیتهای عامل دیگری بوجود آید، مسئله را خصمانه گویند.**
- **مسئله های اکتشافی: وقتی حالتها و فعالیتهای محیط ناشناخته باشند، عامل باید سعی کند آنها را کشف کند. مسئله های اکتشافی را میتوان شکل نهایی مسئله های اقتضایی دانست .**

مثال: دنیای جاروبرقی فاقد حسگر

- عامل جارو تمام اثرات فعالیتهایش را میداند اما فاقد حسگر است.

- حالت اولیه آن یکی از اعضای مجموعه {۱،۲،۳،۴،۵،۶،۷،۸} میباشد.

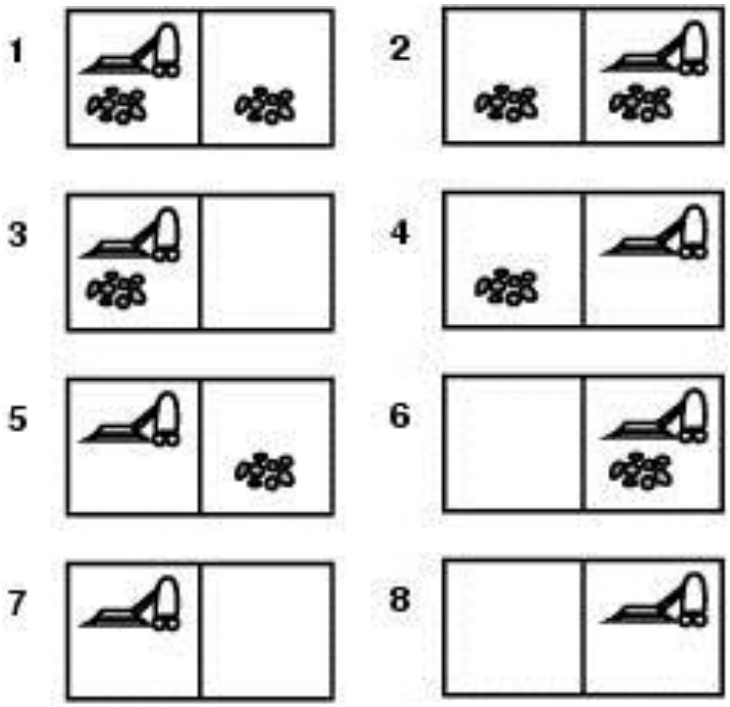
- فعالیت (Right) {۲،۴،۶،۸}

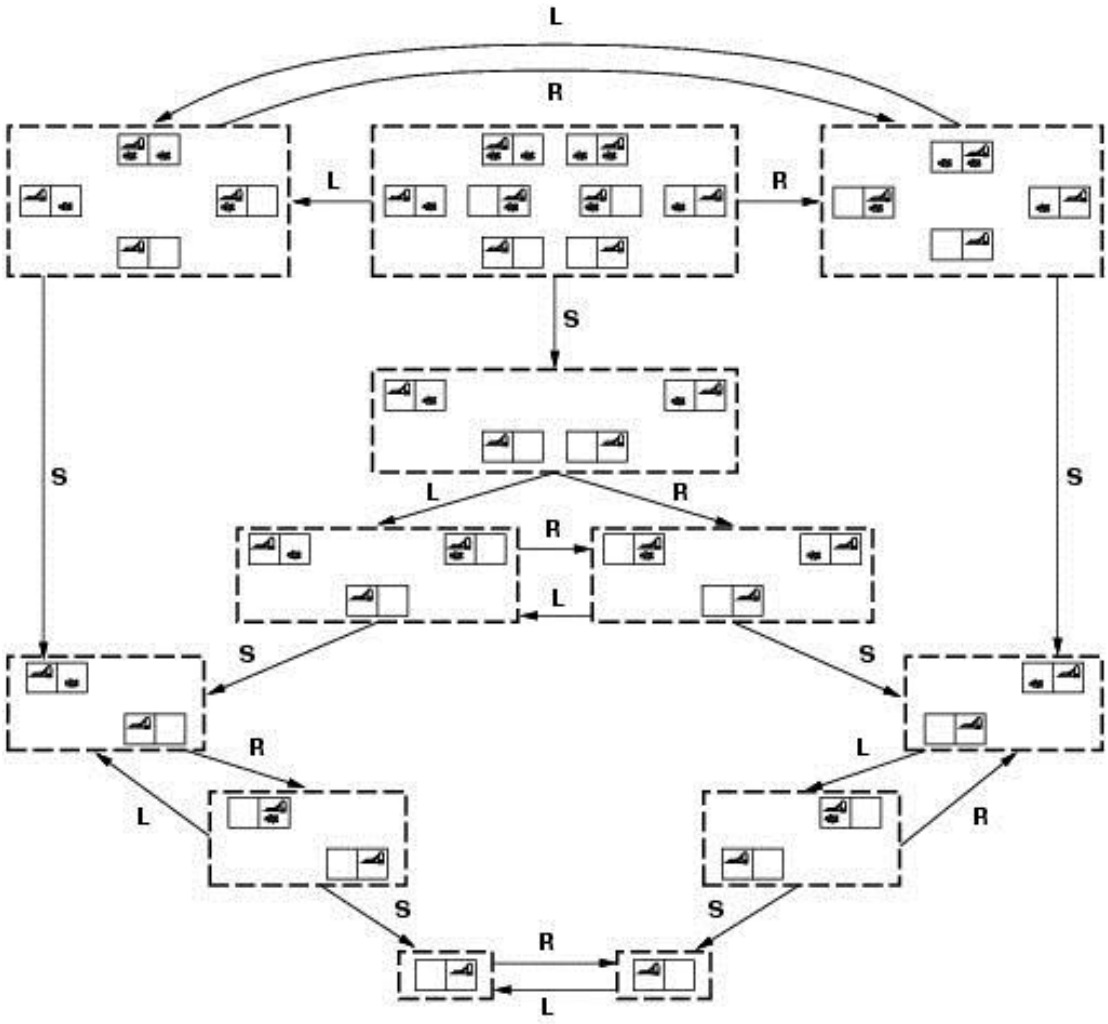
- فعالیت (Right,Suck) {۴،۸}

- فعالیت (Right,Suck,Left,Suck) تضمین میکند که صرف نظر از حالت اولیه، به حالت هدف، یعنی ۷ برسد.

- عامل باید راجع به مجموعه های حالتی که میتواند به آنها برسد استدلال کند. این مجموعه از حالتها را حالت باور گوییم.

- اگر فضای حالت فیزیکی دارای s حالت باشد فضای حالت باور 2^s حالت باور خواهد داشت.





عکس - دنیای جاروبرقی فاقد حسگر

با پایان این بخش، آماده برای فراگیری "جست و جوی آگاهانه و اکتشاف" هستیم. در جلسه بعد به این موضوع خواهیم پرداخت.

پایان