« به نام او »

آموزش شبیه سازی دو بعدی فوتبال

نویسندگان : محمدعلی میرزایی - علی یعقوبی

مرجع روبوکاپ ایران

www.iranrcss.com

جلسه نهم

مباحث این جلسه :

۱ – آموزش بیس یو وی ای

۲ – آموزش هوش مصنوعی، جست و جوی آگاهانه و اکتشاف

امیدواریم 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-     {
```

```
25-       if( WM->getAgentStamina().getStamina() >     // if stamina high
26-                                SS->getRecoverDecThr()*SS->getStaminaMax()+800 )
27-       {
28-         soc = moveToPos(WM->getStrategicPosition(),
29-                              PS->getPlayerWhenToTurnAngle());
30-         ACT->putCommandInQueue( soc );            // move to strategic pos
31-         ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
32-       }
33-       else                                         // else watch ball
34-       {
35-         ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT_BALL ) );
36-         ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
37-       }
38-     }
39-     else if( fabs( WM->getRelativeAngle( OBJECT_BALL ) ) > 10 ) // watch ball
40-     {
41-       ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT_BALL ) );
42-       ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
43-     }
44-     else                                          // nothing to do
45-       ACT->putCommandInQueue( SoccerCommand(CMD_TURNNECK,0.0) );
46-   }
```

در جلسه قبل به بحث در مورد Stamina Manager بیس پرداختیم . در این جلسه به بحث در مورد دفاع خواهیم پرداخت .

هر برنامه نویس برای این منظور الگوریتم و شیوه ی مخصوص به خود را دارد ، اما توابعی که بدین منظور در بیس تعریف شده، در تمامی این شیوه ها یکسان است . در کلاس basicPlayer این توابع تعریف شده اند . در زیر تعریف این توابع را آورده ایم :

```
/*! This method returns the command to tackle the ball. */

SoccerCommand BasicPlayer::tackle( )
{
  return SoccerCommand( CMD_TACKLE, \...* );
}

/*************************************************************************/

/*! This skill enables an agent to mark an opponent, i.e. to guard him
    one-on-one with the purpose to minimize his usefulness for the opponent
    team. It can be used, for example, to block the path from the ball to
    an opponent or from an opponent to the goal. In this way the agent can
    prevent this opponent from receiving a pass or from moving closer to the
    goal while also obstructing a possible shot. This skill amounts to
    calculating the desired marking position based on the given arguments
    and then moving to this position. It receives three arguments: an
    object o (usually an opponent) that the agent wants to mark, a distance
    'dDist' representing the desired distance between o and the marking
    position and a type indicator that denotes the type of marking that is
    required. We distinguish three types of marking:
    - MARK BALL: marking the opponent by standing at a distance 'dDist' away
                 from him on the line between him and the ball. This type of
                 marking will make it difficult for the opponent to receive a
                 pass.
    - MARK GOAL: marking the opponent by standing at a distance 'dDist' away
                 from him on the line between him and the center point of the
                 goal he attacks. This type of marking will make it difficult
                 for the opponent to score a goal.
    - MARK BISECTOR: marking the opponent by standing at a distance 'dDist'
                 away from him on the bisector of the ball-opponent-goal
                 angle. This type of marking enables the agent to intercept
                 both a direct and a leading pass to the opponent.

    After determining the marking position, the agent uses the moveToPos
skill
    to move to this position. Note that the decision whether to turn or dash
```

```
    in the current situation depends on the angle of the marking position
    relative to the agent's body direction and on the distance to this
position
    if this point lies behind the agent. In this case the moveToPos skill
uses
    the threshold parameters MarkTurnAngle (=۳۰) and MarkDistanceBack (=۳) to
    make this decision. The values for these parameters are such that the
    condition which must hold for allowing a dash is fairly flexible. This is
    done because the marking position will be different in consecutive cycles
    due to the fact that the opponent and the ball move around from each
    cycle to the next. As a result, the agent will be able to actually
    progress towards a point that lies close to the marking position instead
    of constantly turning towards the newly calculated marking position in
    each cycle.
    \param o object that has to be marked
    \param dDist distance marking position is located from object position
    \param mark marking technique that should be used
    \return SoccerCommand to mark object 'o'. */
SoccerCommand BasicPlayer::mark( ObjectT o, double dDist, MarkT mark )
{
  VecPosition posMark  = getMarkingPosition( o, dDist, mark );
  VecPosition posAgent = WM->getAgentGlobalPosition();
  VecPosition posBall  = WM->getGlobalPosition( OBJECT_BALL );
//  AngDeg      angBody  = WM->getAgentGlobalBodyAngle();

  if( o == OBJECT_BALL )
  {
    if( posMark.getDistanceTo( posAgent ) < ۱.۵ )
      return turnBodyToObject( OBJECT_BALL );
    else
     return moveToPos( posMark, ۳۰.۰, ۳.۰, false );
  }

  if( posAgent.getDistanceTo( posMark ) < ۲.۰ )
  {
    AngDeg angOpp = (WM->getGlobalPosition( o ) - posAgent).getDirection();
```

```
    AngDeg angBall = (posBall - posAgent).getDirection();
    if( isAngInInterval( angBall, angOpp,
                         VecPosition::normalizeAngle( angOpp + ۱۸۰ ) ) )
      angOpp += ۸۰;
    else
      angOpp -= ۸۰;
    angOpp  = VecPosition::normalizeAngle( angOpp );
    Log.log( ۵۱۳, "mark: turn body to ang %f", angOpp );
    return turnBodyToPoint( posAgent + VecPosition( ۱۰, angOpp, POLAR )  );
  }
  Log.log( ۵۱۳, "move to marking position" );

  return moveToPos( posMark, ۲۵, ۳۰, false );
}
```

این دو تابع از اصلی ترین توابع مربوط به دفاع می باشند . البته این توابع مختص اینکار هستند اما ما با توابعی به جز این توابع نیز می توانیم از تیممان دفاع کنیم . ما با روشی به نام positioning یا به عبارتی سد کردن راه حریف با چینش حرفه ای بازیکنان در جلسه بعد بیشتر به مقوله دفاع و توضیحات توابع بالا به صورت خط به خط و ادامه توضیحات تابع deMeer۵ خواهیم پرداخت .

آموزش هوش مصنوعی، جست و جوی آگاهانه و اکتشاف

فهرست

متدهای جستجوی آگاهانه

بهترین جستجو

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

جستجوی محلی و بهینه سازی

- تپه نوردی
- شبیه سازی حرارت
- پرتو محلی
- الگوریتمهای ژنتیک

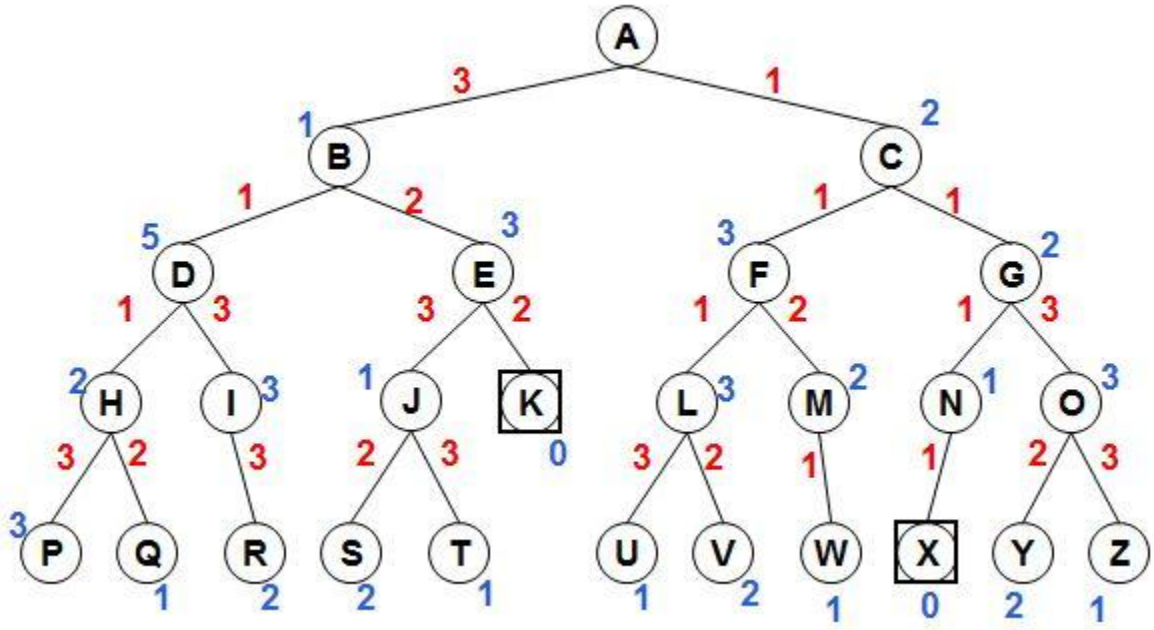
تعاریف

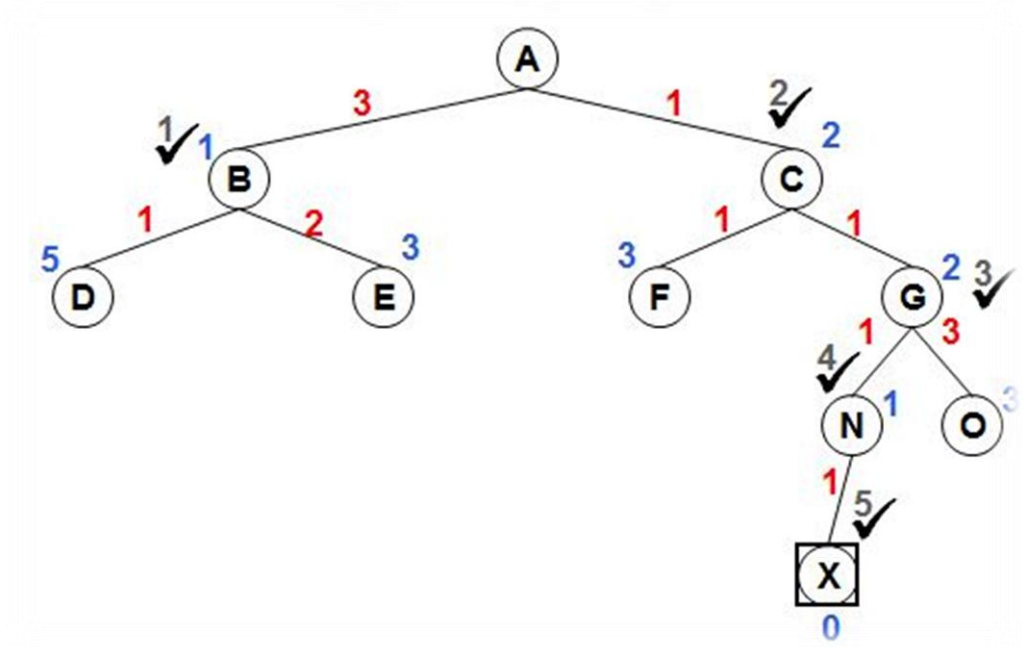
- **تابع هزینه مسیر، g(n) : هزینه مسیر از گره اولیه تا گره n**
- **تابع اکتشافی، h(n) : هزینه تخمینی ارزان ترین مسیر از گره n به گره هدف**
- **تابع بهترین مسیر، h*(n) : ارزان ترین مسیر از گره n تا گره هدف**
- **تابع ارزیابی، f(n) : هزینه تخمینی ارزان ترین مسیر از طریق n**

f(n): g(n) + h(n)

- **f*(n) : هزینه ارزان ترین مسیر از طریقn** **f*(n): g(n) + h*(n)**

✓ **جستجوی حریصانه**





ارزیابی :

کامل بودن: خیر

- **اما اگر $h = h^*$ آنگاه جستجو کامل میشود**

بهینگی: خیر

- **اما اگر $h = h^*$ آنگاه جستجو کامل میشود**

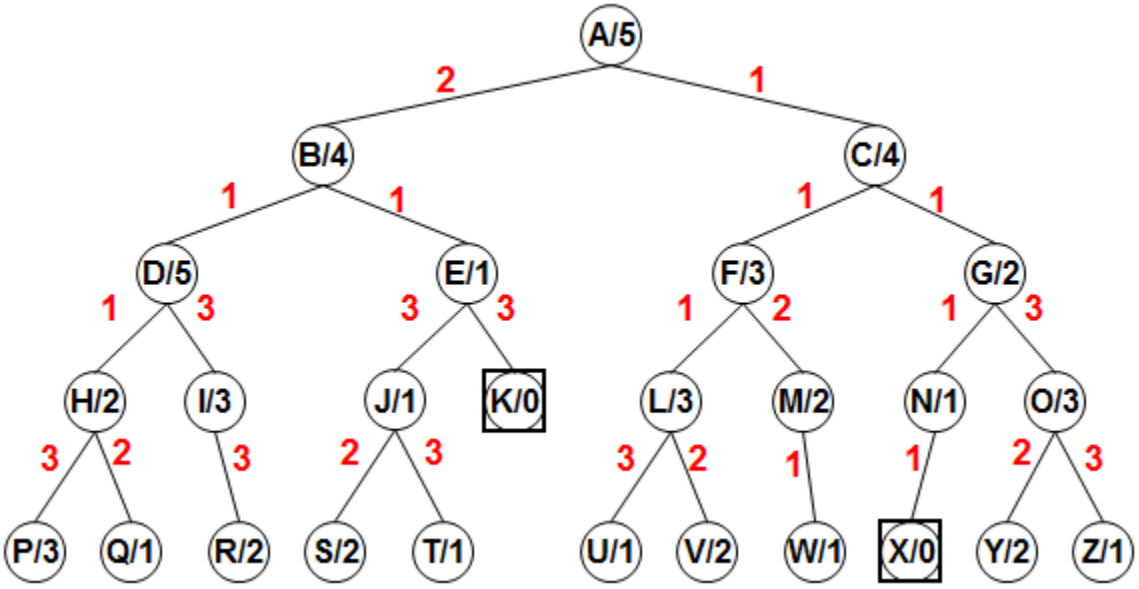
پیچیدگی زمانی: $O(b^m)$

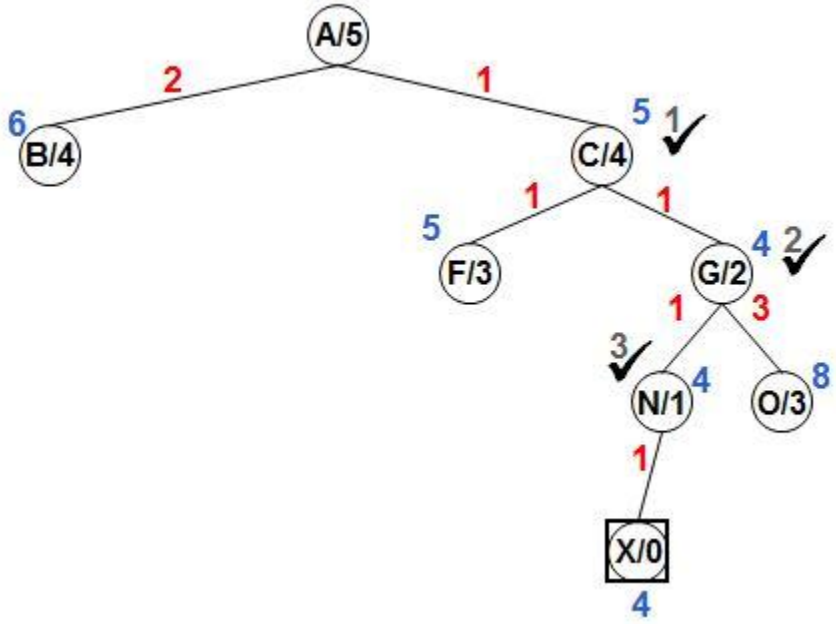
- **اما اگر $h = h^*$ آنگاه** $O(bd)$

پیچیدگی فضا: $O(b^m)$

- **اما اگر $h = h^*$ آنگاه** $O(bd)$

✓ جستجوی **A***


A/5
2
1
B/4
C/4
1
1
1
1
D/5
E/1
F/3
G/2
1
3
3
3
1
2
1
3
H/2
I/3
J/1
K/0
L/3
M/2
N/1
O/3
3
2
3
2
3
3
2
1
1
2
3
P/3
Q/1
R/2
S/2
T/1
U/1
V/2
W/1
X/0
Y/2
Z/1


A/5
2
1
6
5
1
B/4
C/4
1
1
5
4
2
F/3
G/2
1
3
3
4
8
N/1
O/3
1
X/0
4

ارزیابی :

کامل بودن: بله

بهینگی: بله

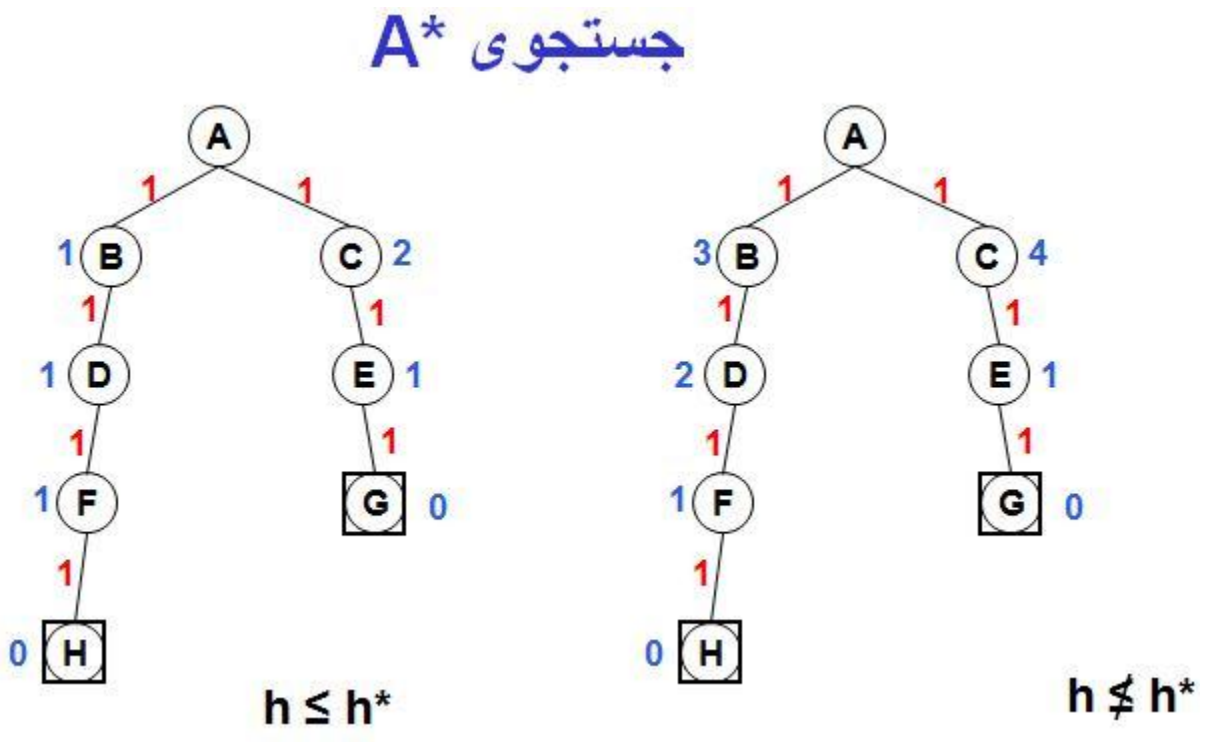
پیچیدگی زمانی: $O(b^m)$

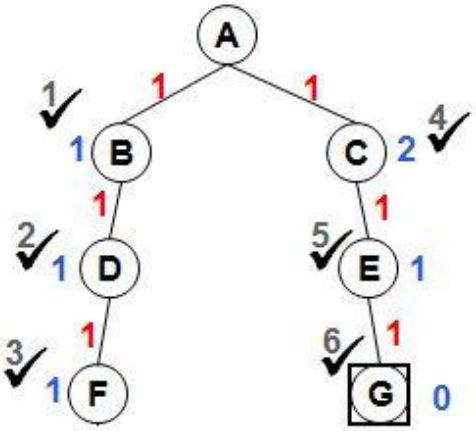
- اما اگر $h = h^*$ آنگاه $O(bd)$

پیچیدگی فضا:
$O(b^m)$

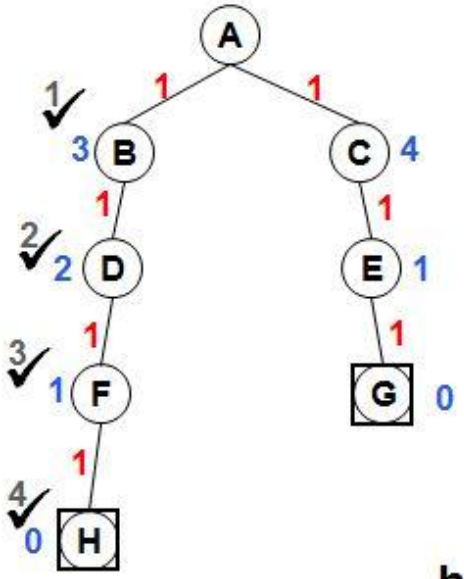
- اما اگر $h = h^*$ آنگاه $O(bd)$

مثال :



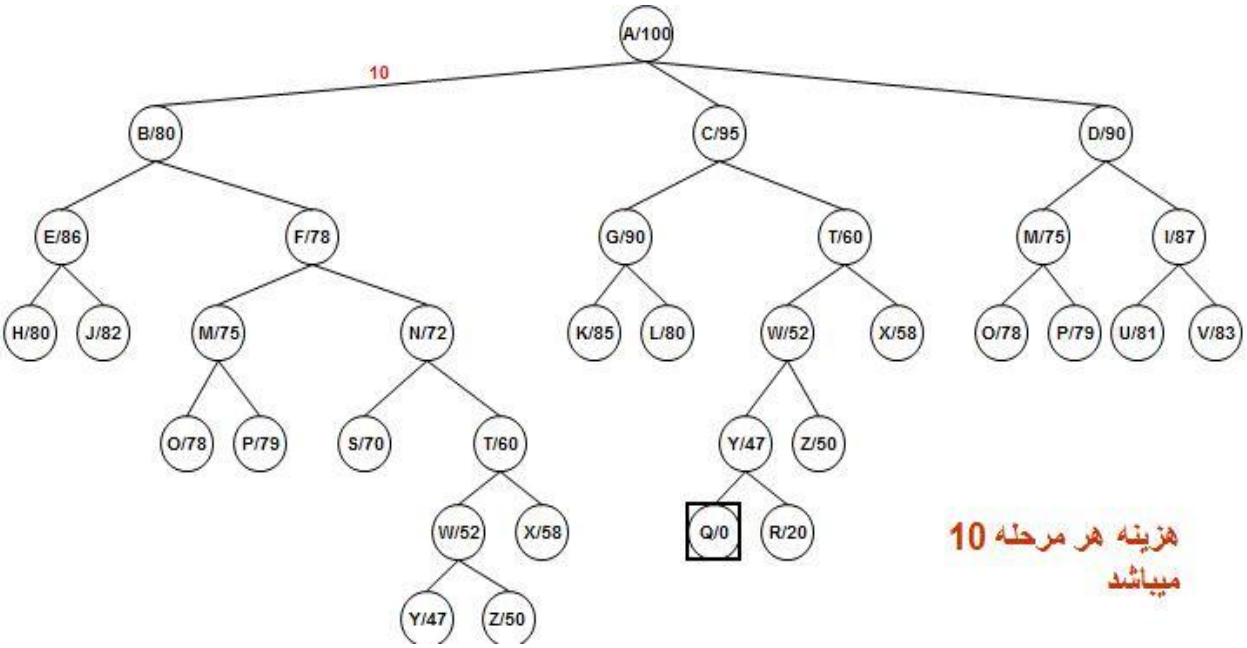


$h \leq h^*$

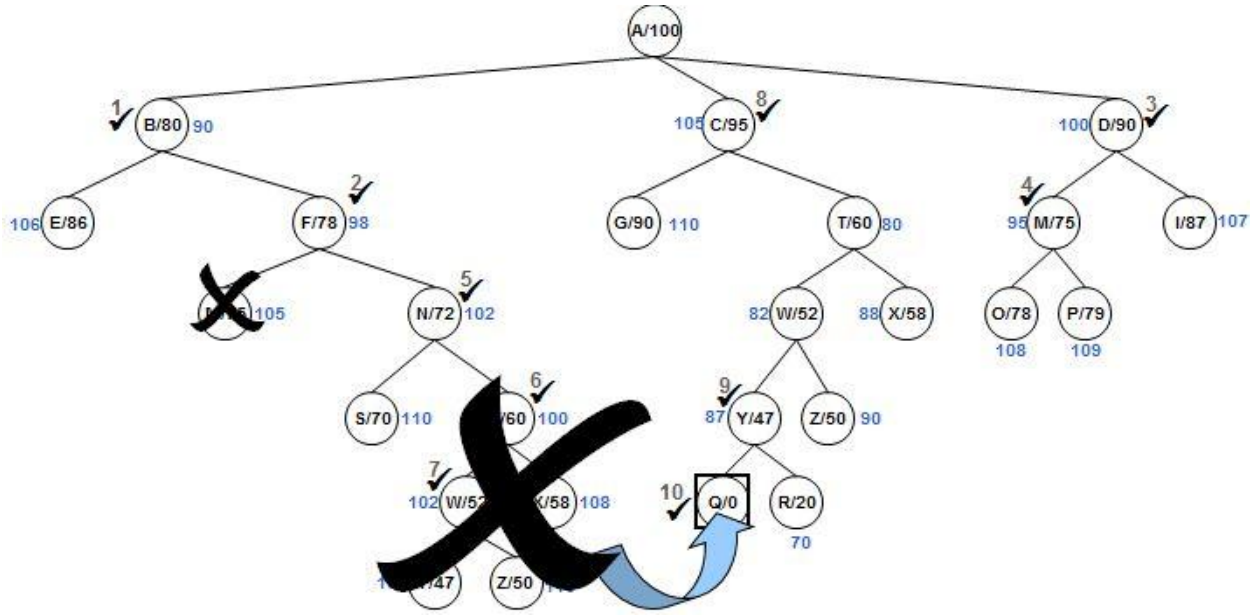


$h \nleq h^*$

جستجوی A* و اجتناب از گره های تکراری

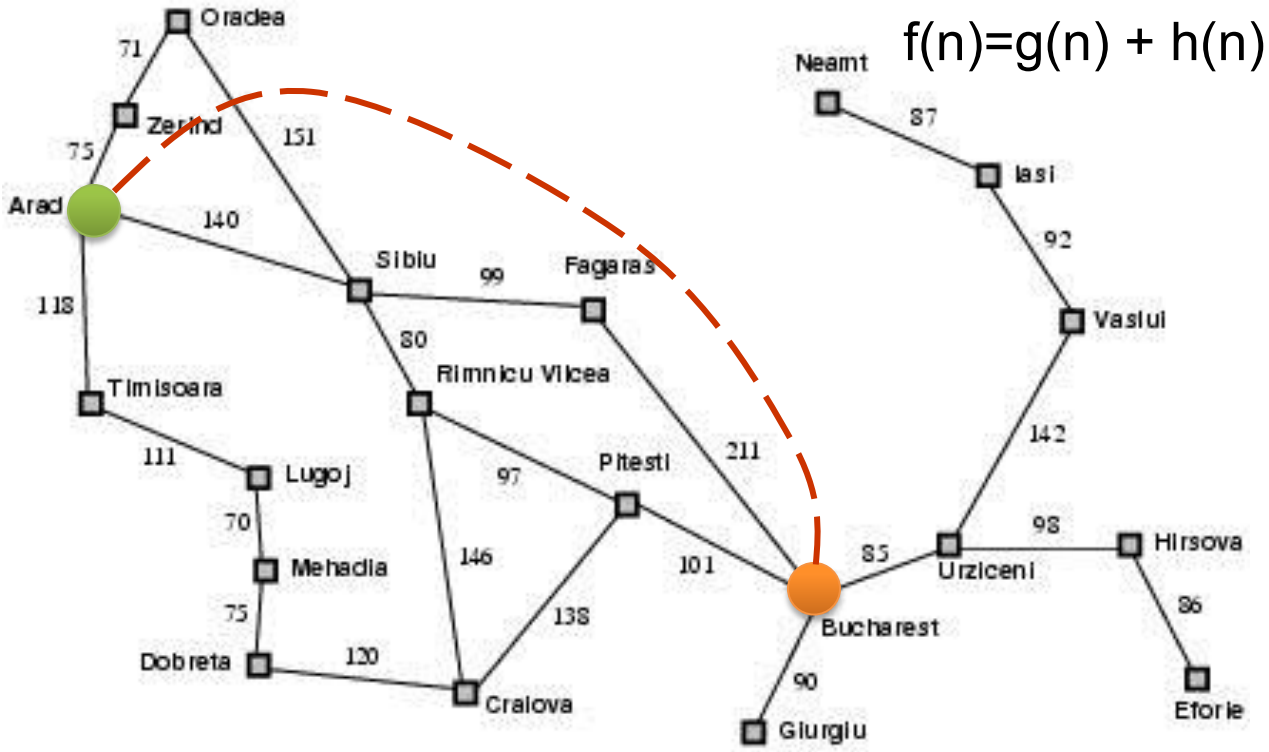


هزینه هر مرحله 10 میباشد



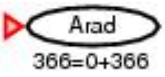
مثال دیگر از جستجوی A*

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



جستجوی **A*** در نقشه رومانی

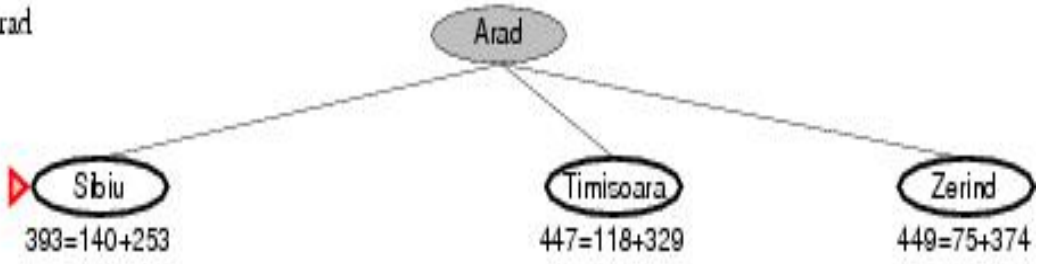
(a) The initial state



جستجوی **Bucharest** با شروع از **Arad**

f(Arad) = g(Arad)+h(Arad)=۰+۳۶۶=۳۶۶

After expanding Arad



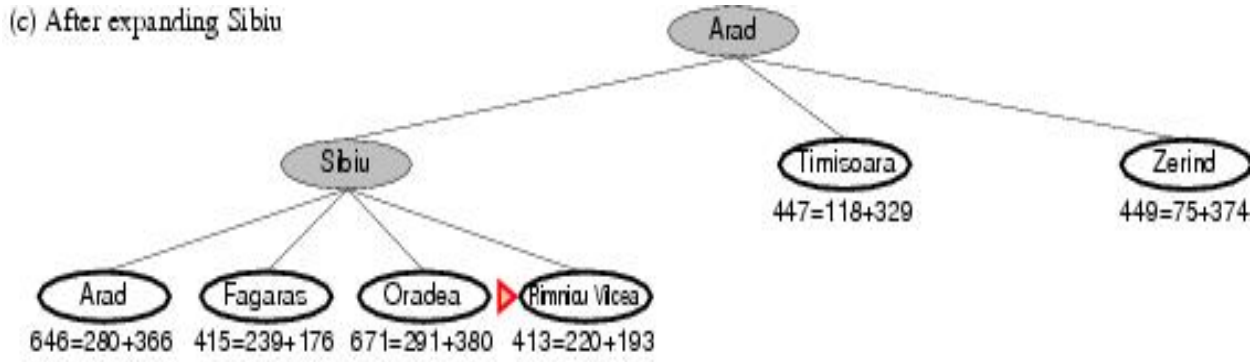
Arad را باز کرده و **f(n)** را برای هر یک از زیربرگها محاسبه میکنیم:

f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=۱۴۰+۲۵۳=۳۹۳

f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=۱۱۸+۳۲۹=۴۴۷

f(Zerind)=c(Arad,Zerind)+h(Zerind)=۷۵+۳۷۴=۴۴۹

بهترین انتخاب شهر **Sibiu** است.



Sibiu را باز کرده و **f(n)** را برای هر یک از زیربرگها محاسبه میکنیم:

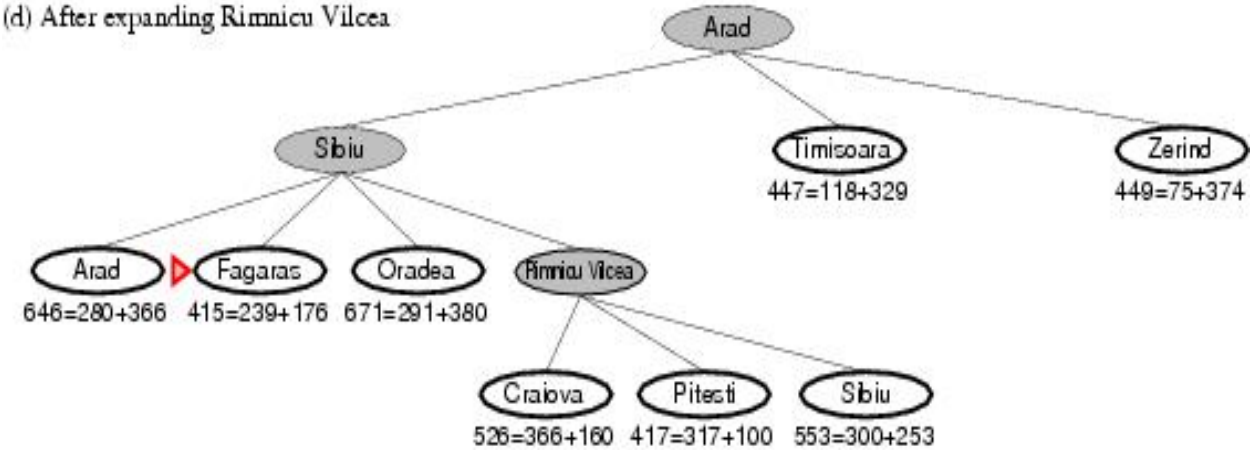
f(Arad)=c(Sibiu,Arad)+h(Arad)=۲۸۰+۳۶۶=۶۴۶

f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=۲۳۹+۱۷۹=۴۱۵

f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=۲۹۱+۳۸۰=۶۷۱

f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=۲۲۰+۱۹۲=۴۱۳

بهترین انتخاب شهر **Rimnicu Vilcea** است.



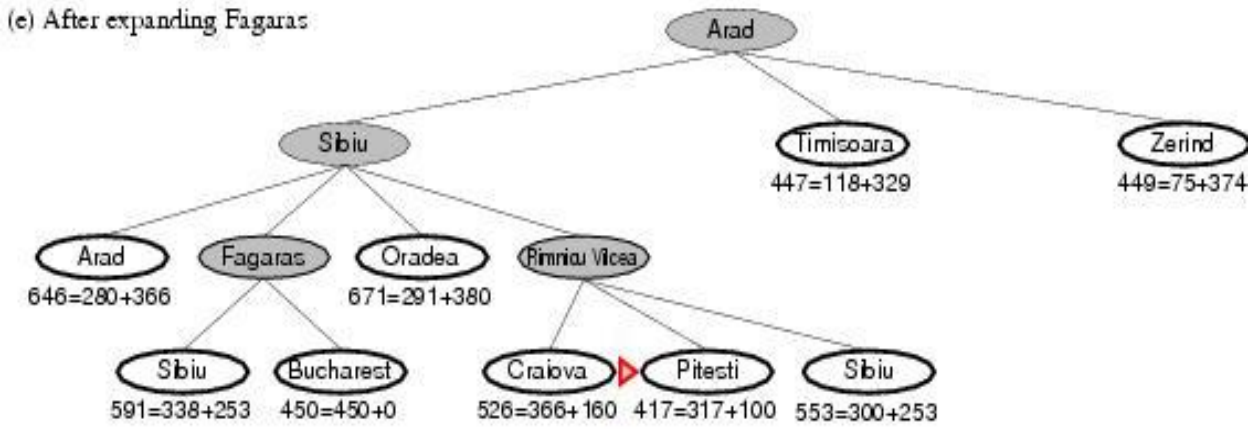
Vilcea را باز کرده و **f(n)** را برای هر یک از زیربرگها محاسبه میکنیم:

f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=۳۶۰+۱۶۰=۵۲۶

f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=۳۱۷+۱۰۰=۴۱۷

f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=۳۰۰+۲۵۳=۵۵۳

بهترین انتخاب شهر **Fagaras** است.

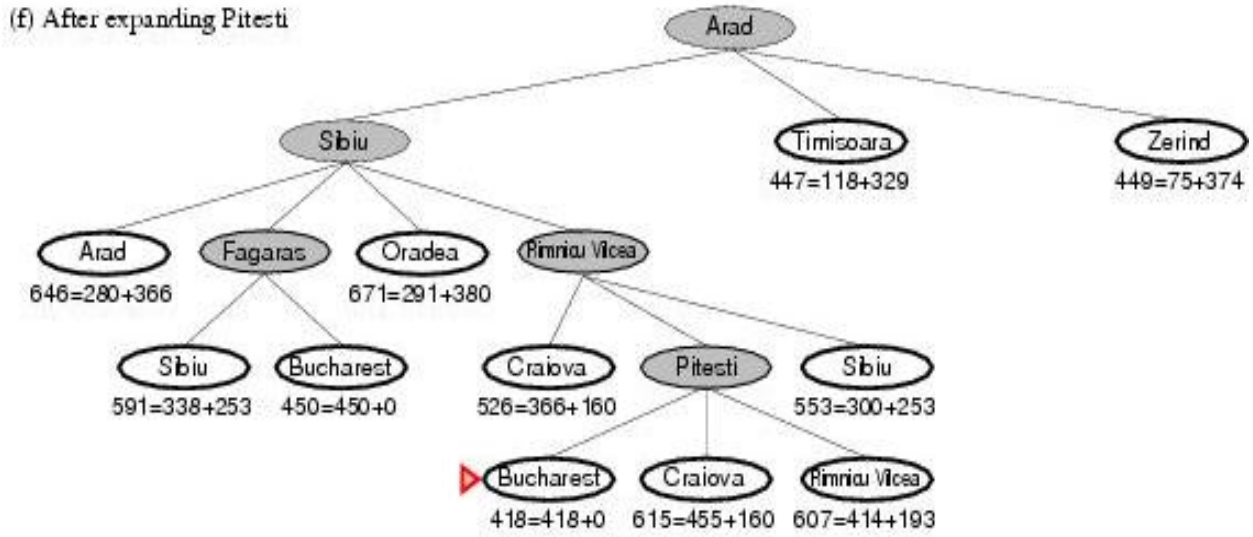


Fagaras را باز کرده و **f(n)** را برای هر یک از زیربرگها محاسبه میکنیم:

f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=۳۳۸+۲۵۳=۵۹۱

f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=۴۵۰+۰=۴۵۰

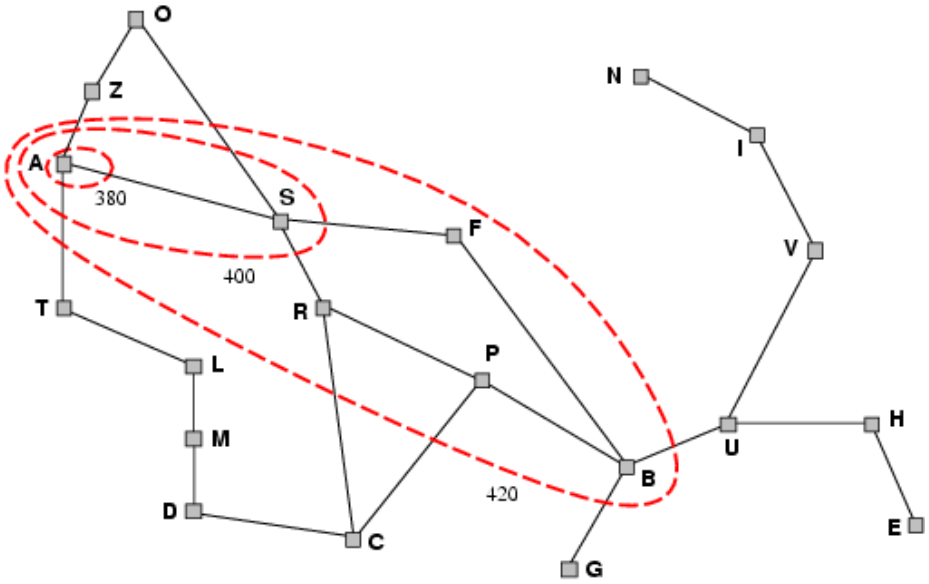
بهترین انتخاب شهر **Pitesti** است.



Pitesti را باز کرده و **f(n)** را برای هر یک از زیربرگها محاسبه میکنیم:

f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=۴۱۸+۰=۴۱۸

بهترین انتخاب شهر **Bucharest** است.



منبع : **Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig**