1 -سیستمی که مانند انسان رفتار می کند را با ذکر مثال تشریح کنید؟

تئور تورینگ: فردی که ازطریق سیستم با افراد دیگر درارتباط است بتواند تشخیص دهد که فردمقابل واقعا یک انسان است یا یک ربات.

2- هدف از تفکر عاقلانه چیست و چه آورده ای در پی خواهد داشت؟

منجر به برنامه نویسی منطقی میگردد،تفکر عاقلانه منجر به رفتارعقلانه میشود،رفتار عاقلانه باعث انجام عمل درست میشود و عمل درست یعنی دستیابی به بهترین هدف

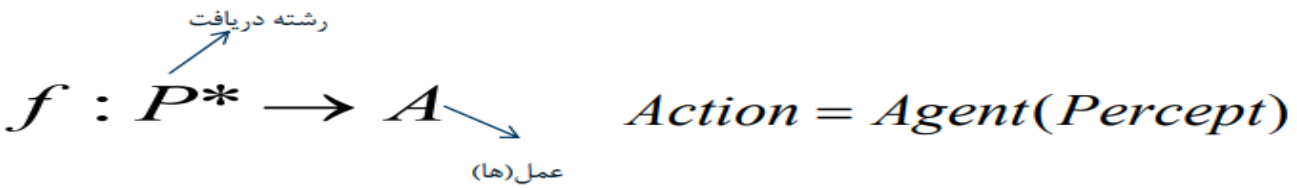
دو مزیت :عمومیت بیشتر نسبت به تفکر – عدم نیاز به تئوریهای پیشرفته علمی

3- اجزای عامل و وظیفه عامل را با رسم شکل و تابع نویسی بررسی کنید؟

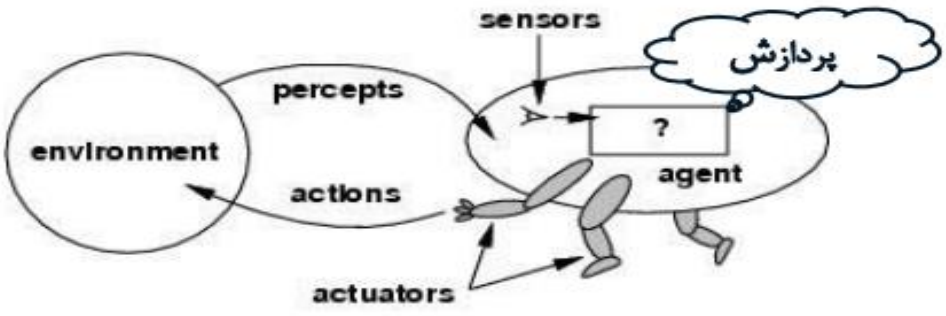
سنسور : وظیفه دریافت مشخصه هایی از محیط

عملگر : وظیفه انجام اعمال بر روی محیط

عامل وظیفه دارد رشته دریافتهای ورودی را به دنبالهایی از اعمال نگاشت نماید.



عامل میتواند اعمال خود در محیط را درک کند، اما تاثیر آنها بر روی محیط همیشه قابل پیش بینی نیست.



4- PEAS را برای ربات فضانورد و فوتبالیست تشریح کنید؟

ربات فضانورد:

P:تداخل کنترل-هزینه پایین-عملکرد درست-پاسخ درست به عوامل محیطی

E:محیط بدون جاذبه-محیط باجاذبه های مختلف-شرایط جوی مختلف

A:حرکت-چرخش-ایستادن-بلندگو-صفحه نمایش

S:دوربین-فاصله یاب-موقعیت یاب

ربات فوتبالیست:

P:رعایت قوانین-هزینه پایین-حرکت درست-مقصد درست

E:چمن مصنوعی-چمن طبیعی-آب وهوا-

A:دویدن-ایستادن-پاس دادن

S:دوربین-فاصله یاب-تشخیص خطوط

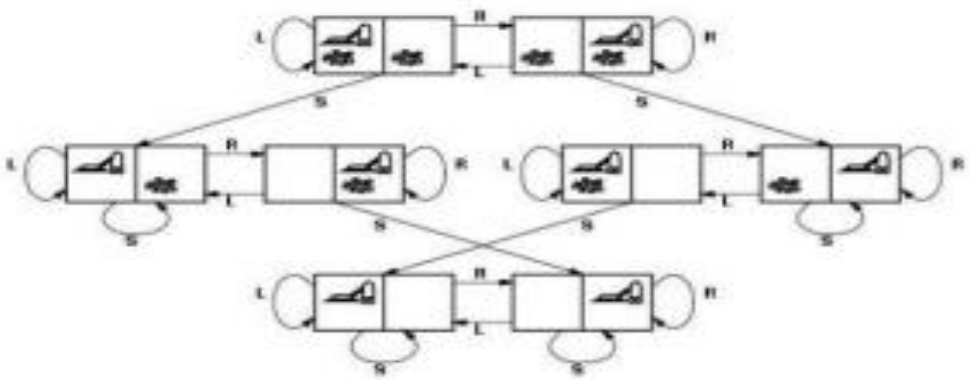
5- طبق شبکه کد زیر چرا عامل مبتنی بر جدول به شکست مواجه می شود؟ راهکارهای پیشنهادی خود را نام برده و مختصری در خصوص هر کدام توضیح دهید؟

Function TABLE-DRIVEN_AGENT(*percept*) **returns** an action

static: *percepts*, a sequence initially empty
table, a table of actions, indexed by percept sequence

append *percept* to the end of *percepts*
action ← LOOKUP(*percepts*, *table*)
return *action*

6- دنیای جارو برقی را با توجه به فرموله سازی مساله تشریح کنید؟



حالات: 8 حالت مختلف

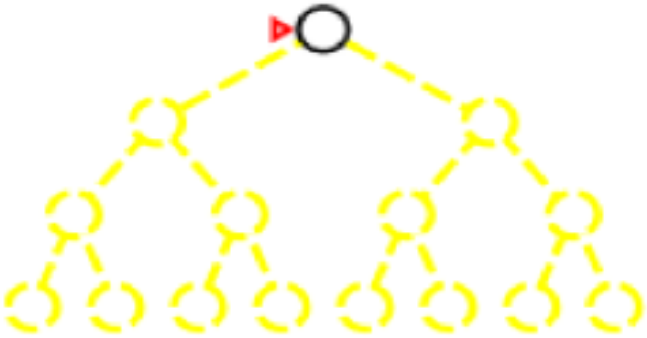
حالت شروع: هریک از حاالت

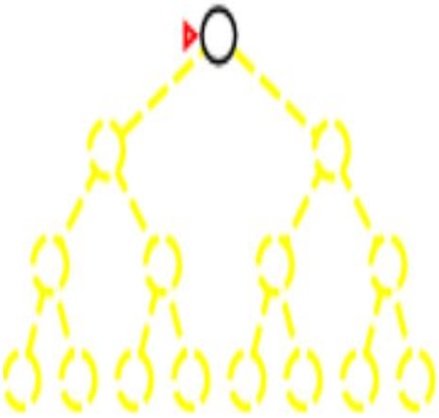
اعمال:{چپ، راست، مکش یا هیچ کار}

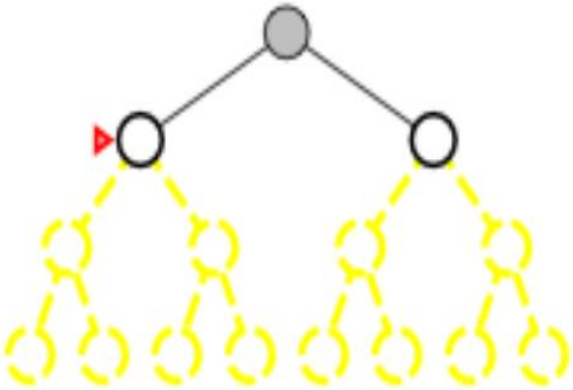
آزمون هدف: حالات{ 7 و 8}

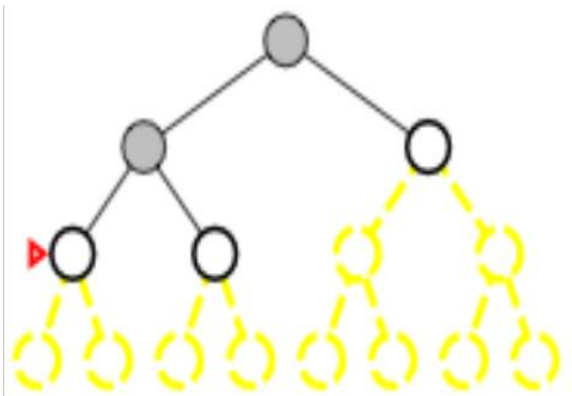
هزینه مسیر: تعداد اعمال انجام شده تا رسیدن به هدف

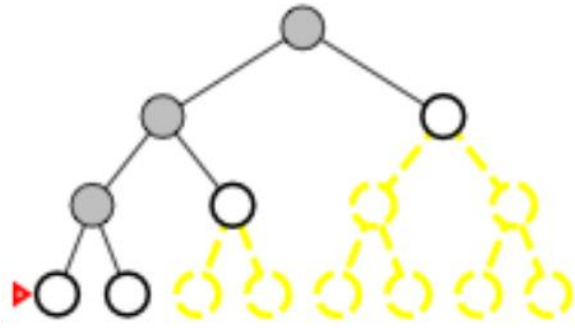
7- جستجوی عمقی را با رسم مرحله به مرحله شرح دهید و در نهایت کارایی الگوریتم را با چهار معیاراندازی گیری بیان کنید؟

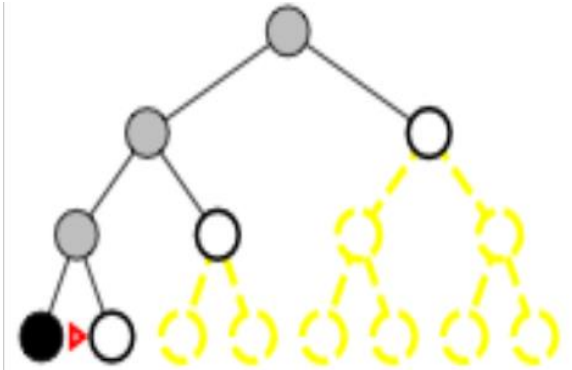


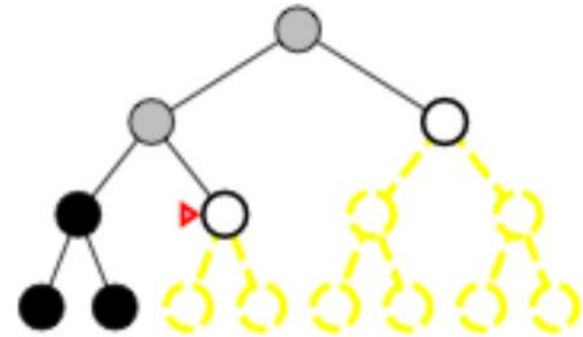


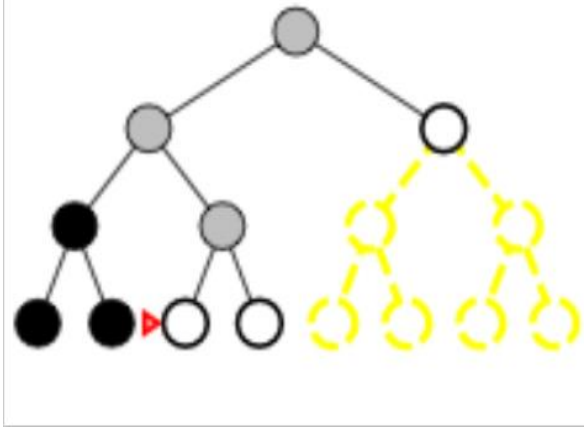


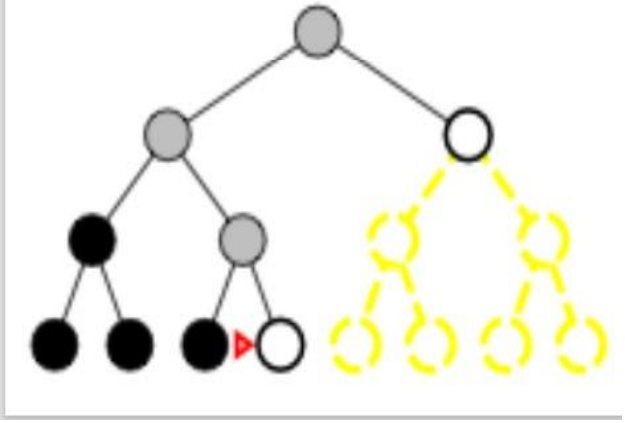


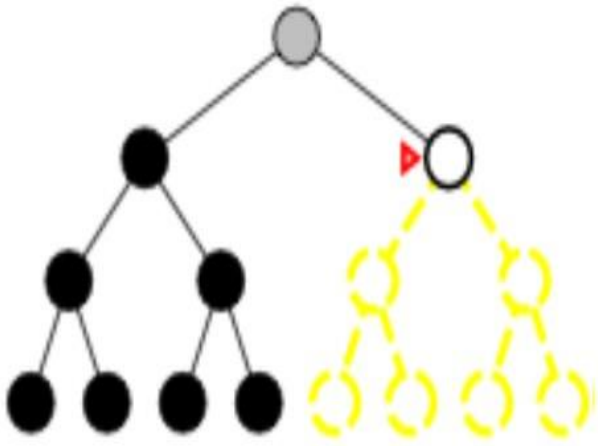


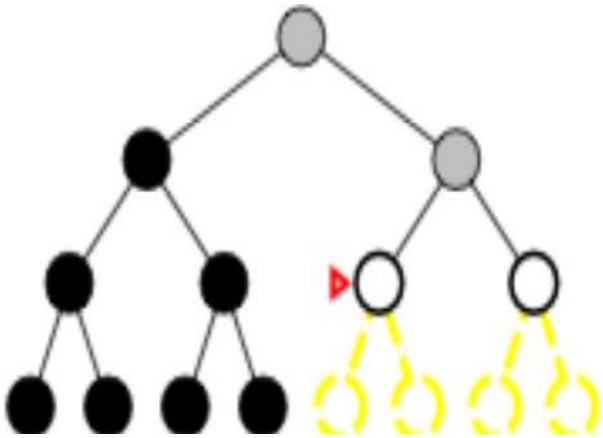


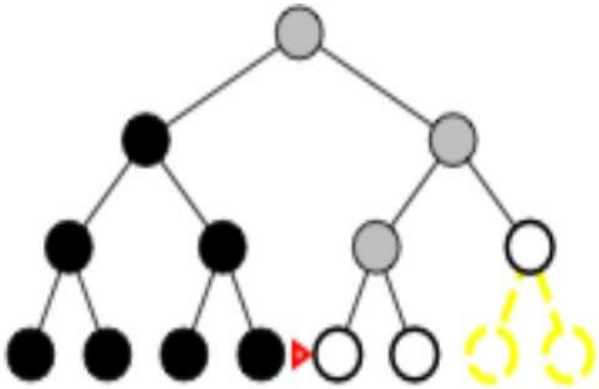


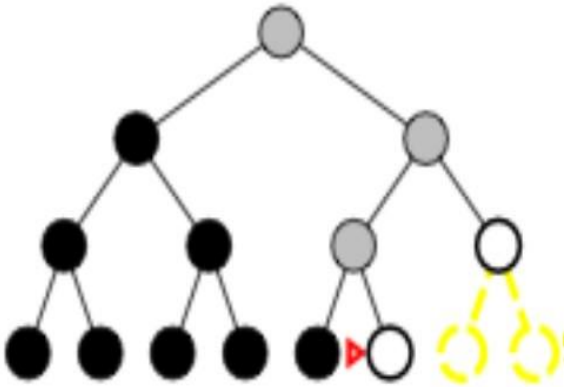












کامل بودن؟ خیر ، مگر اینکه فضای حالت محدود باشد و حلقه تکرار وجود نداشته باشد.

بهینه بودن؟ خیر ،چون کامل نیست.

پیچیدگی زمانی؟(O (b^m

اگر m خیلی بزرگتر از d باشد به مراتب بدتر است.

در بسیاری از مسایل سریعتر از جستجوی BF است.

پیچیدگی حافظه؟ (O(bm+1

در زمان عقبگرد حافظه آزاد میشود.

8-ضمن بررسی الگوریتم جستجوی درختی شبکه کد زیر بررسی کنید که استراتژی در کدام از 4 توابع، پیاده سازی شده است، تواع را نام برده و عملکرد هر یک را بیان کنید؟

```
function TREE-SEARCH(problem, fringe) return a solution or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if      GOAL-TEST[node] then
                return SOLUTION(node)
        else
                fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

9-شبکه کد زیر مربوط به کدام جستجوی ناآگاهانه می باشد و از مزایای کدام جستجوهای دیگر بهره برده است با ترسیم شکل توضیح دهید؟

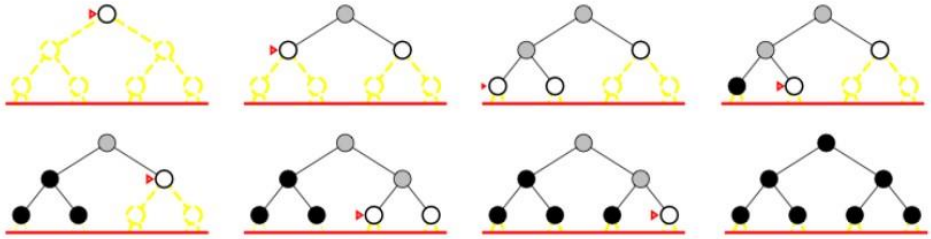
```
function ITERATIVE_DEEPENING_SEARCH(problem) return a solution or failure
    inputs: problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED_SEARCH(problem, depth)
        if result ≠ cuttoff then return result
```

جستجوی عمقی تکراری:

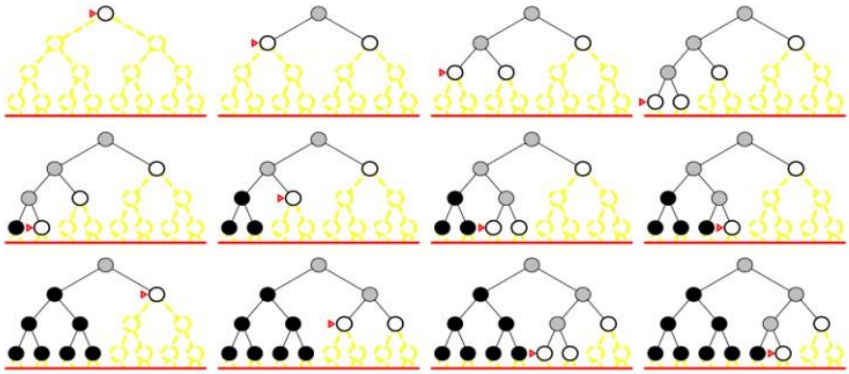
- Limit=0

- Limit=1

- Limit=2



- Limit=3



10- شش نوع جستجوهای ناآگاهنه جدول زیر را به تفکیک، با چهار معیار مربوطه به اختصار شرح دهید ؟

Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	YES*	YES*	NO	YES, if $l \geq d$	YES	YES*
Time	b^{d+1}	$b^{C^*/\varepsilon}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{C^*/\varepsilon}$	bm	bl	bd	$b^{d/2}$
Optimal?	YES*	YES*	NO	NO	YES	YES

جستجوی سطحی: کامل هست به شرطی که جواب بهینه در عمق d قابل دسترس باشد. فاکتور انشعاب b محدود باشد. بهینه هست به شرطی که: مسیرها فاقد هزینه باشند.پیچیدگی زمانی و حافظه O(b^d+1)

جستجو با هزینه یکنواخت :کامل هست به شرطی که: جواب در عمق قابل دسترس باشد. هزینه ها مقدار مثبت داشته باشند. بهینه بودن هست به شرطی که: کامل باشد.پیچیدگی زمانی و حافظه O(b^c/£*)

جستجوی عمقی: کامل نیست مگر اینکه فضای حالت محدود باشد و حلقه تکرار وجود نداشته باشد. بهینه نیست چون کامل نیست. پیچیدگی زمانی O(b^m) و پیچیدگی حافظه O(bm+1)

جستجوی عمقی محدود :

گر d>L آنگاه غیرکامل است.

اگر d<L آنگاه کامل اما غیر بهینه است.

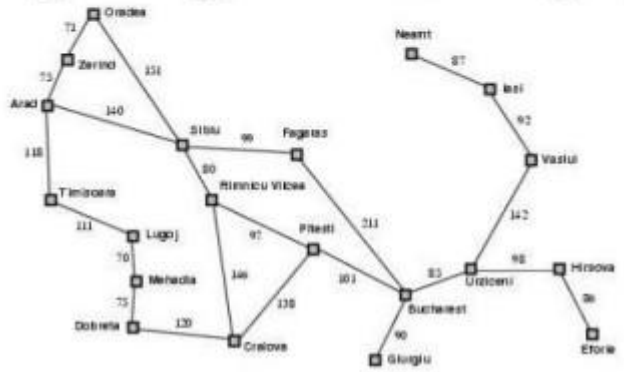
اگر d=L آنگاه کامل و بهینه است.

جستجوی عمقی تکراری :کامل هست به شرطی که حلقه تکرار وجود نداشته باشد. بهینه هست اگر هزینه مسیرها باهم برابر باشد. پیچیدگی زمانی(O(b^d پیچیدگی حافظه(O(bd

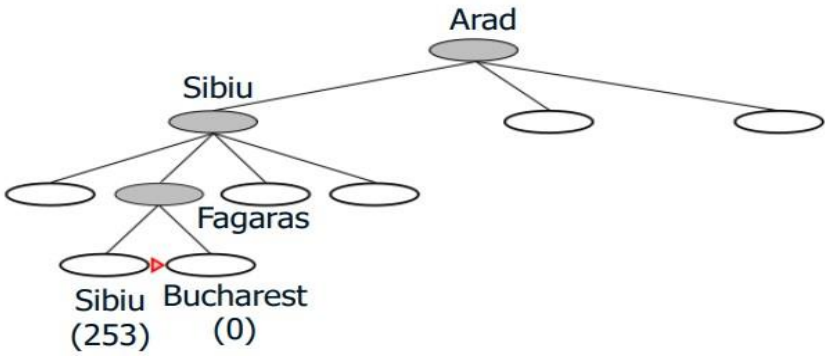
جستجوی دوطرفه :کامل هست به شرطی که از جستجوی سطری استفاده شود. بهینه هستت به شرطی که از جستجوی سطری استفاده شود..پیچیدگی زمانی و حافظه(O/2 (b^d

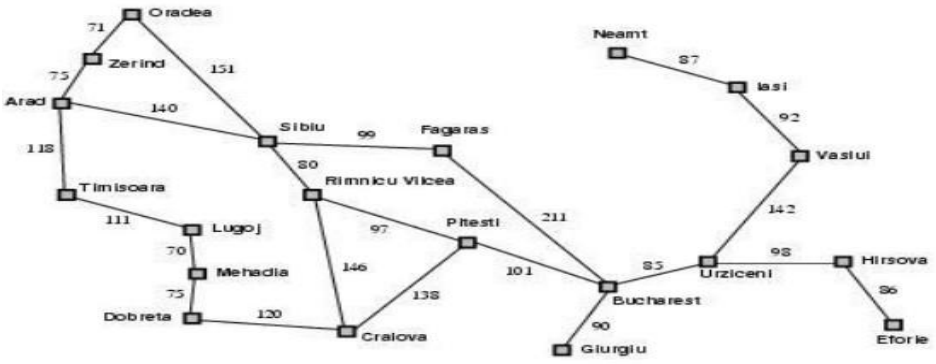
11-جستجوی A* را با توجه به جدول hSLD با جستجوی حریصانه (searc Greedy با رسم درختی به طور کامل توضیح داده و تفاوت ها را با دلیل ذکر کنید؟

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



جستجو حریصانه:کامل نیست (حلقه تکرار) بهینه نیست (کامل نبودن) مرتبه زمانی وحافظه (O(b^m





جستجوی A* : کامل هست. بهینه هست. مرتبه زمانی کماکان از مرتبه نمایی میباشد. مرتبه حافظه هم مرتبه با پیچیدگی زمانی است.

12-الگوریتم زیر را شرح دهید و با توجه به جدول و شکل سوال 11 با رسم درخت جستجو توضیح دهید؟

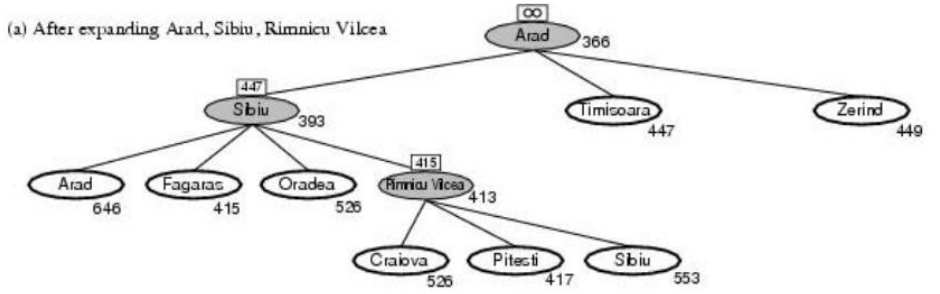
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RFBS( problem, node, f_limit) return a solution or failure and a new f-cost limit
    if GOAL-TEST[problem](STATE[node]) then return node
    successors ← EXPAND(node, problem)
    if successors is empty then return failure, ∞
    for each s in successors do
        f[s] ← max(g(s) + h(s), f[node])
    repeat
        best ← the lowest f-value node in successors
        if f[best] > f_limit then return failure, f[best]
        alternative ← the second lowest f-value among successors
        result, f[best] ← RBFS(problem, best, min(f_limit, alternative))
        if result ≠ failure then return result
```

1-بهترین گره برگ و بهترین جانشین برای آن انتخاب شود.

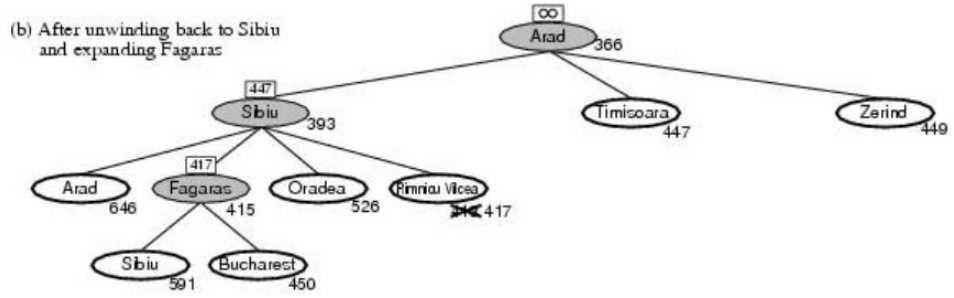
2-اگر مقدار بهترین گره برگ از جانشین آن بیشتر شد، آنگاه به مسیر جانشین عقبگرد شود. .

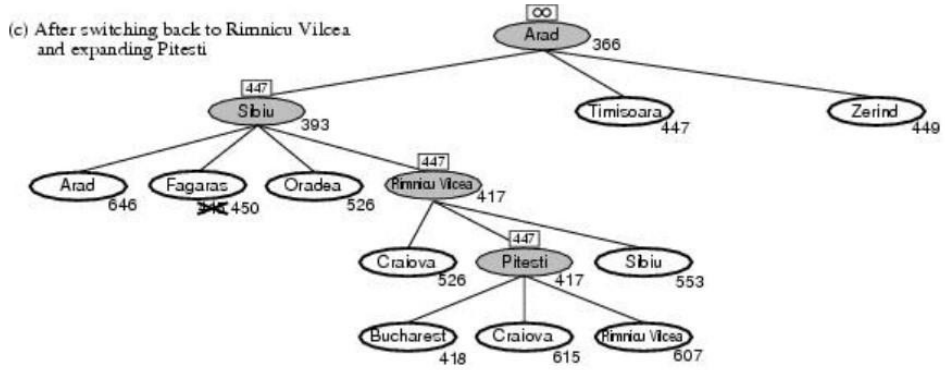
3- در حین عقبگرد، مقدار n(f) بروزرسانی شود.

4 -گره جانشین بسط داده شود.



.





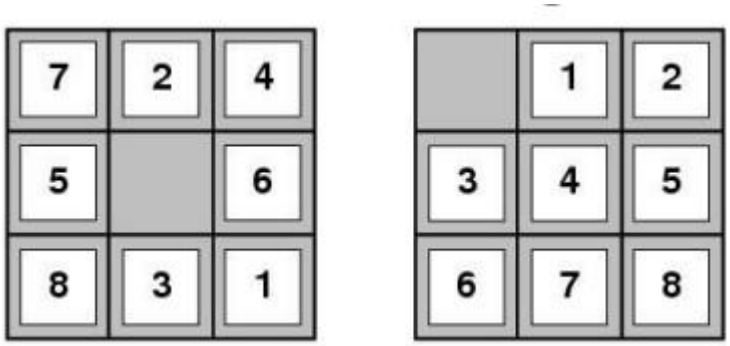
13- چند نوع تابع هیوریستیک را می توان برای پازل اعدا معرفی کرد، با رسم شکل بررسی کنید؟

h1تعداد کاشی هایی که سرجای خود نمیباشند. h1 (s)=8

h2 مجموع فاصله افقی – عمودی هر کاشی تا جای واقعی h2 s)=3+1+2+2+2+3+3+2=18)

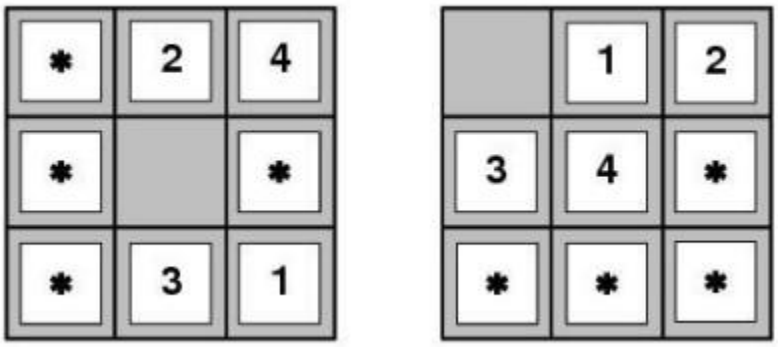
h3 مجموع فاصله افقی – عمودی – قطری هر کاشی تا جای واقعی

h3 (s)=2+1+1+1+2+2+2+2=13



14-سه راه حل جهت ابداع تابع هیوریستیک نام برده و شرح دهید؟

1-از طریق نسخه کوچتر از مساله



2- از طریق نسخه ساده شده از مساله:

h1هر کاشی می تواند به هرجایی منتقل شود.

h2 هر کاشی می تواند به هر خانه همسایه منتقل شود.

ABSolver هزینه راه حل برای مکعب روبیک را تخمین میزند.

3-از طریق یادگیری از تجربه:

حل زیاد مساله

15- انواع جستجوی محلی را نام برده و ایده هر یک را بیان کنید؟

الگوریتم تپه نوردی :

به طور متناوب در جهت بهبود حرکت میکند. زمانی که به قله برسد متوقف میشود.

تپه نوردی به آینده گرههای برگ توجه نمیکند، به همین دلیل به الگوریتم جستجوی محلی حریصانه هم مشهور است.

تپه نوردی در صورتیکه بیشتر از یک گره بهترین وجود داشته باشد، بهترین گره را به صورت تصادفی انتخاب میکند.

پرتوی محلی:

از k حالت شروع به جای یک حالت شروع بهره میبرد.

حالت شروع: k حالت تصادفی

حالت بعدی: انتخاب k تا بهترین حالت از بین تمام برگها

حالت خاتمه: پیدا شدن هدف یا بررسی تمام حاالت

تفاوت با تپه نوردی با شروع مجدد تصادفی این است که اطالعات به اشتراک گذاشته میشود.

SA:

ممکن است الگوریتم از عدم تنوع کافی برخوردار باشد.

اجتناب از گیر کردن در بیشینه های محلی با اجازه دادن به انجام حرکت های فرعی (نامناسب)، که در حین گذشت زمان احتمال و تعداد آن کاهش میابد. پیشینه الگویتم به علوم متالورژی بر میگردد.

16-الگوریتم زیر را شرح داده و انواع آن را نام برده و بررسی کنید؟

```
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
```

به طور متناوب در جهت بهبود حرکت میکند. زمانی که به قله برسد متوقف میشود.

تپه نوردی به آینده گرههای برگ توجه نمیکند، به همین دلیل به الگوریتم جستجوی محلی حریصانه هم مشهور است.

تپه نوردی در صورتیکه بیشتر از یک گره بهترین وجود داشته باشد، بهترین گره را به صورت تصادفی انتخاب میکند.

تپه نوردی غیر قطعی : در بین حرکت های رو به بالا یکی به صورت تصادفی انتخاب شود. البته احتمال انتخاب با شیب متناسب است.

تپه نوردی با انتخاب اولین گزینه : گرهها تا حصول یک گره بهتر بسط داده میشوند.

تپه نوردی تصادفی: از حالت شروع مجدد تصادفی تا حصول جواب مجدداً شروع خواهد نمود.