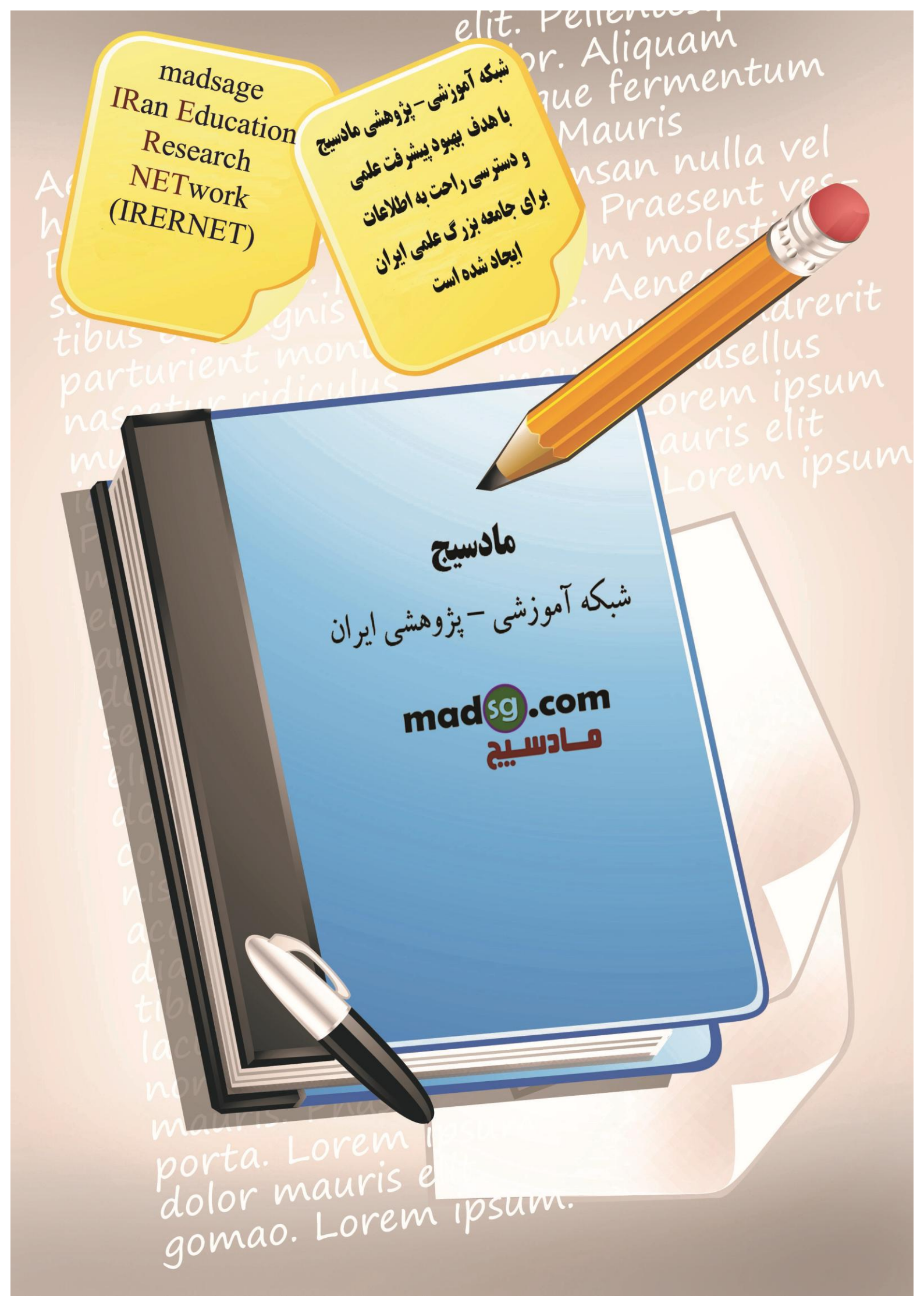

madsage
IRan Education
Research
NETwork
(IRERNET)
شبکه آموزشی-پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
مادسیج
شبکه آموزشی - پژوهشی ایران
madsg.com
مادسیج

حل مسائل توسط جستجو

فصل سوم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

مقدمه

- عامل های حل مسأله
- انواع مسأله
- فرموله سازی مسأله
- مسائل نمونه
- الگوریتم های ابتدایی جستجو

عامل های حل مسأله

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

فرضیات در مورد محیط: ایستا، قابل مشاهده، گسسته و قطعی

مثال: رومانی

- یک روز تعطیل در رومانی؛ مکان فعلی شهرآراد
- پرواز فردا، بخارست را ترک می کند.

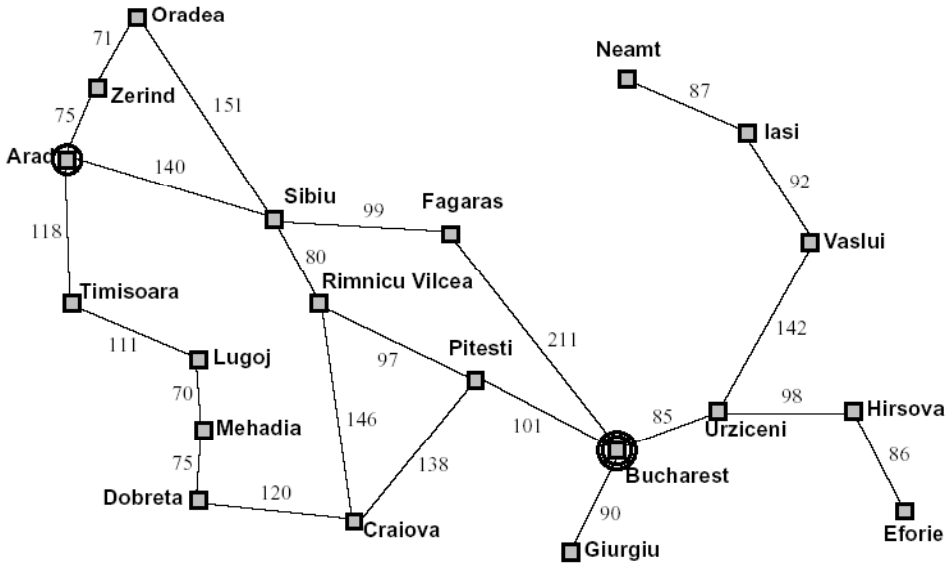
- فرموله سازی هدف:
  - بودن در بخارست
- فرموله سازی مسأله:
  - حالت ها: شهرهای مختلف
  - عملیات: رفتن از شهری به شهر دیگر
- یافتن پاسخ:
  - دنباله ای از شهرها، مانند:

Arad → Sibiu → Fagaras → Bucharest

انواع مسأله

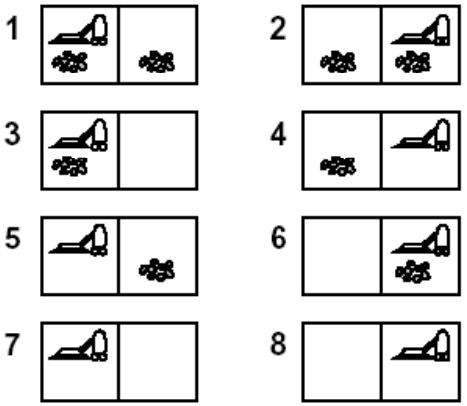
- قطعی، کاملا مشاهده پذیر ← مسائل تک – حالته
  - عامل دقیقا می داند در چه حالتی خواهد بود؛ راه حل یک دنباله می باشد.
- قطعی، مشاهده پذیر جزئی ← مسائل چند–حالته
  - ممکن است عامل ایده ای درباره اینکه کجاست نداشته باشد؛ راه حل یک دنباله است.
- غیر قطعی و/یا مشاهده پذیر جزئی ← مسائل احتمالی
  - ادراک اطلاعات جدیدی درباره حالت فعلی فراهم می کند.
  - در حین اجرا باید از حسگرها استفاده کند.
  - راه حل به صورت یک درخت
  - اغلب جستجو و اجرا به صورت یک در میان (interleave)
- فضای حالت ناشناخته ← مسائل اکتشافی (online)

مثال: رومانی



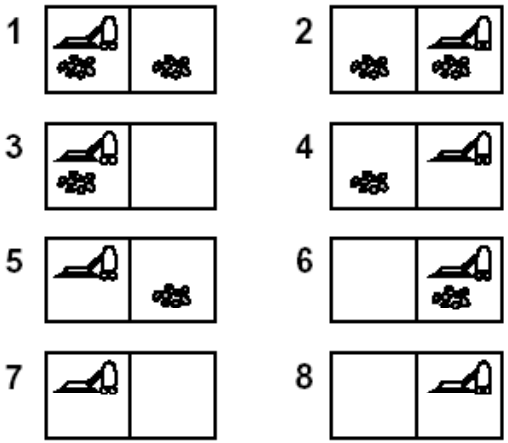
مثال: دنیای مکش

- تک–حالته، شروع در #5.
  راه حل؟ [*Right, Suck*]
- چند–حالته، شروع در {1, 2, 3, 4, 5, 6, 7, 8}مثال عمل *Right* به {2, 4, 6, 8}.
  راه حل؟

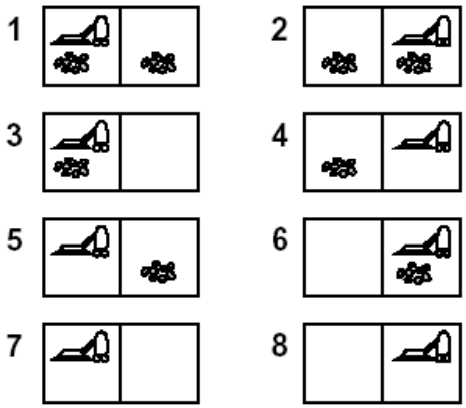


مثال: دنیای مکش

- تک–حالته، شروع در #5.
  راه حل ؟



مثال: دنیای مکش



- چند-حالته، شروع در {1, 2, 3, 4, 5, 6, 7, 8}مثال عمل *Right* به {2, 4, 6, 8}.

راه حل؟

[*Right, Suck, Left, Suck*]

- احتمالی

- غیر قطعی: مکش می تواند یک فرش تمیز را کثیف کند.
- درک محلی: گرد و خاک در محل فعلی
- ادراک: [L, Clean] یعنی شروع در #5 یا #7

راه حل؟

[*Right,* ***if*** *dirt* ***then*** *Suck*]

فرموله سازی مسائل تک-حالته

یک مسأله بوسیله چهار مورد تعریف می شود:

۱. حالت اولیه مثلاً بودن در شهر Arad

۲. عمل ها یا تابع حالت بعدی

S(x)= مجموعه ای از زوج های عمل-حالت

- S(Arad) = {<Arad → Zerind, Zerind>, …}

۳. تابع تست هدف

- صریح: x = “at Bucharest”
- ضمنی: NoDirt(x)

۴. تابع هزینه مسیر :

- مثال: مجموع فواصل، تعداد عمل های انجام شده و ...
- هزینه گام (Step cost): $c(x, a, y) \geq 0$

- راه حل: دنباله ای از عملیات که از حالت اولیه شروع و به حالت هدف ختم می شود.

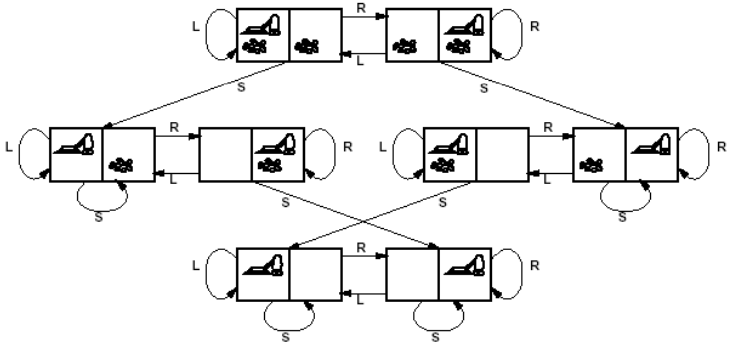
انتخاب یک فضای حالت

- دنیای واقعی به شدت پیچیده می باشد

← بنابراین، برای حل مسأله باید فضای حالت انتزاعی باشد.

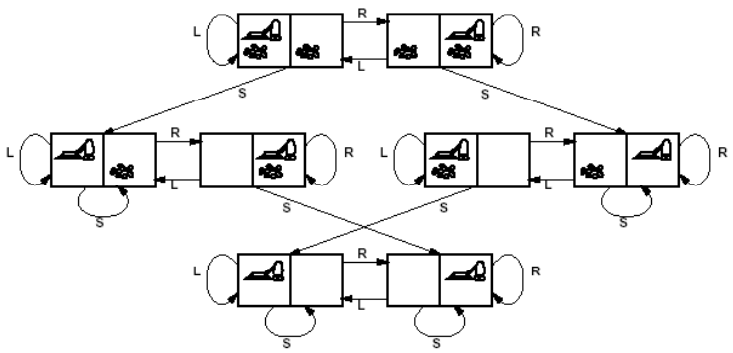
- حالت (انتزاعی) = مجموعه ای از حالت های واقعی
- عمل (انتزاعی) = ترکیبی پیچیده از عمل های واقعی
  - مثلا عمل Arad → Zerind می تواند مجموعه ای پیچیده از اعمال باشد.
- راه حل (انتزاعی) =
  - مجموعه ای از مسیرهای واقعی که در دنیای واقعی راه حل می باشند.
- هر عمل انتزاعی باید از مسأله اصلی ساده تر باشد!

مثال: گراف فضای حالت دنیای مکش



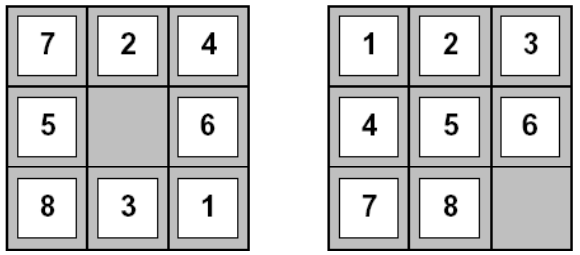
- حالات؟
- اعمال؟
- تست هدف؟
- هزینه مسیر؟

مثال: گراف فضای حالت دنیای مکش



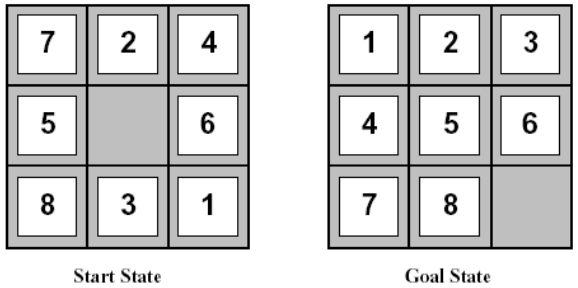
- حالات؟ وجود گرد و خاک و مکان های عامل (بدون در نظر گرفتن مقدار گرد و خاک)
- اعمال؟ *Left, Right, Suck*
- تست هدف؟ نبودن گرد و خاک
- هزینه مسیر؟ بازاء هر عمل ۱

مثال: معمای هشت



- حالات؟
- اعمال؟
- تست هدف؟
- هزینه مسیر؟

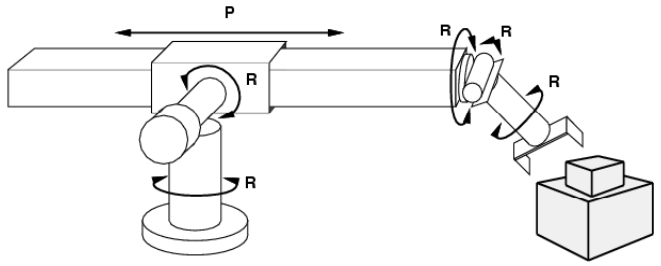
مثال: معمای هشت



- حالات؟ اعداد صحیح بیانگر محل کاشی ها
- اعمال؟ حرکت خانه خالی به چپ، بالا، راست و پایین
- تست هدف؟ حالت هدف (داده شده)
- هزینه مسیر؟ بازاء هر حرکت ۱

[توجه: راه حل بهینه خانواده معمای n یک مسأله NP-hard می باشد]

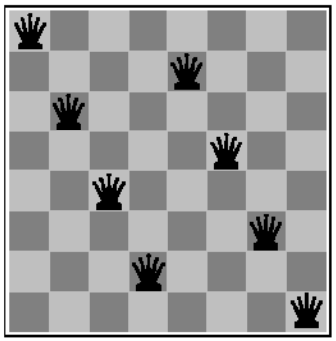
مثال: روبات اسمبل کننده



- حالات؟ زاویه مفاصل روبات، مختصات قطعات
- اعمال؟ حرکت پیوسته مفاصل روبات
- تست هدف؟ سرهم بندی کامل
- هزینه مسیر؟ زمان اجرا

مسأله هشت وزیر

- قراردادن هشت وزیر در صفحه شطرنج به طوریکه هیچ وزیری نتواند به وزیر دیگری حمله کند.



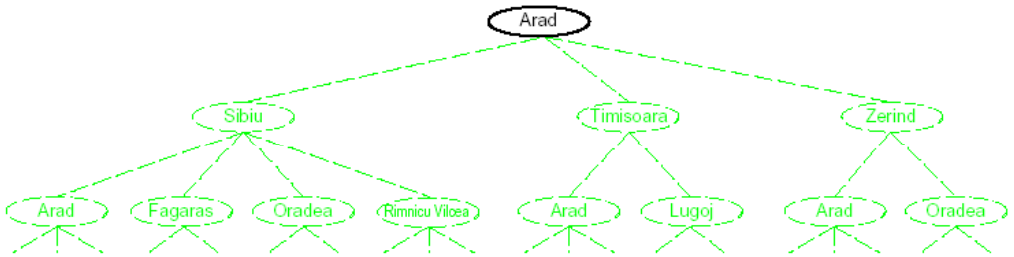
- آزمون هدف: ۸ وزیر روی صفحه شطرنج که با هم برخورد ندارند.
- هزینه مسیر: صفر
- حالات: ترتیب ۸ وزیر هر کدام در یک ستون
  مثال روبرو: {8, 6, 4, 2, 7, 5, 3, 1}
- عملگرها: انتقال یک وزیر دارای برخورد به مربع دیگری در همان ستون

الگوریتم های جستجوی درخت

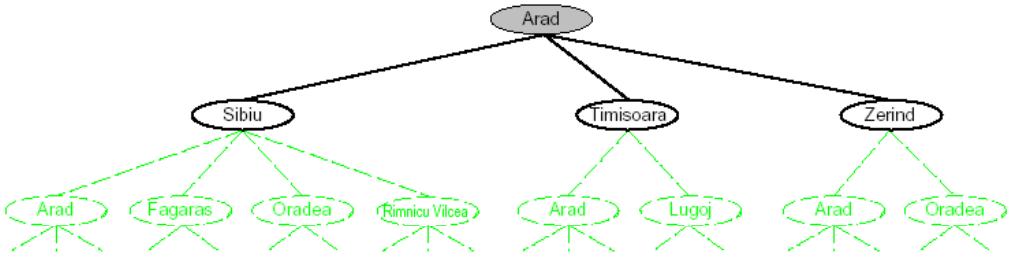
- **ایده اصلی:** کاوش offline و شبیه سازی شده فضای حالت بوسیله تولید حالات بعدی حالت هایی که تا کنون تولید شده اند.

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

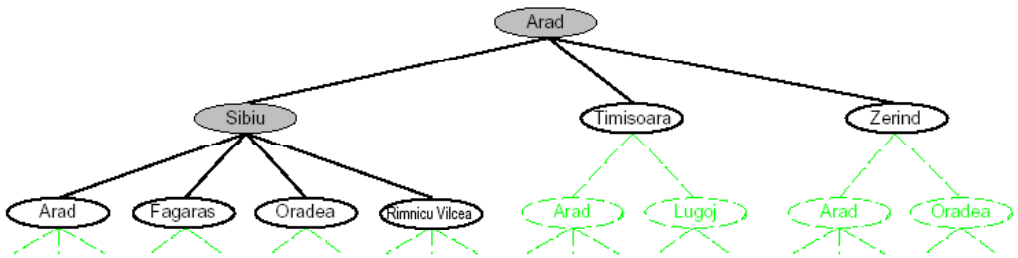

مثال جستجوی درخت



مثال جستجوی درخت



مثال جستجوی درخت



پیاده سازی: جستجوی عمومی درخت

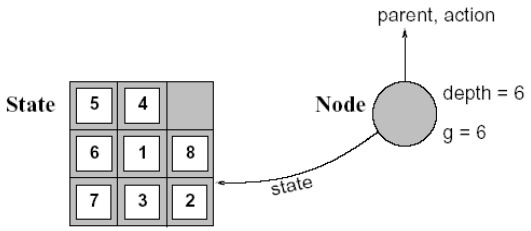
```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```


پیاده سازی: حالت و گره

- یک حالت (بیانگر) یک پیکره بندی فیزیکی می باشد
- یک گره یک ساختار داده ای تشکیل دهنده بخشی از درخت جستجو شامل: پدر، فرزندان، عمق و هزینه مسیر $g(x)$ است.
- حالت ها :

پدر، فرزند، عمق و هزینه مسیر ندارند!



- تابع EXPAND گره های جدید ایجاد می کند، فیلدهای مختلف را مقدار می دهد و با استفاده از تابع SUCCESSORS-FN مسأله، حالت های مربوطه ایجاد می شود.

استراتژی های جستجو

- یک استراتژی بوسیله ترتیب گسترش گره ها تعریف می شود.
- ابعاد ارزیابی استراتژی ها:
  - کامل بودن– آیا در صورت وجود راه حل، همیشه راه حلی پیدا می کند؟
  - پیچیدگی زمانی – تعداد گره های تولید شده/گسترش یافته
  - پیچیدگی حافظه – حداکثر تعداد گره ها در حافظه
  - بهینگی – آیا همیشه کم هزینه ترین راه حل را پیدا می کند؟

- پیچیدگی زمان و فضا برحسب پارامترهای زیر سنجیده می شوند:
  - b : حداکثر فاکتور انشعاب درخت جستجو
  - d : عمق کم هزینه ترین راه حل
  - m : حداکثر عمق فضای حالت (ممکن است ∞ باشد)

استرتژی های جستجوی ناآگاهانه

- استراتژی های **ناآگاهانه** تنها از اطلاعات موجود در تعریف مسأله استفاده می کنند.

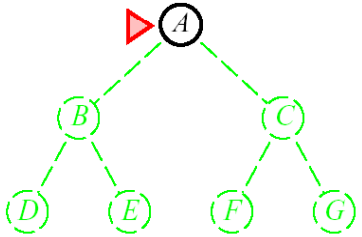
- جستجوی اول-سطح (BFS)
- جستجوی هزینه-یکنواخت (UCS)
- جستجوی اول-عمق (عمقی) (DFS)
- جستجوی با عمق محدود (DLS)
- جستجوی عمیق کننده تکراری (IDS)

جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

0: [A]

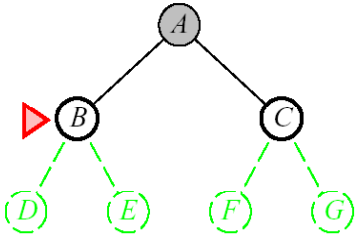


جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

1: [B, C]

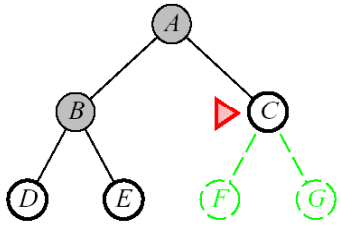


جستجوی سطحی

- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

2: [C, D, E]

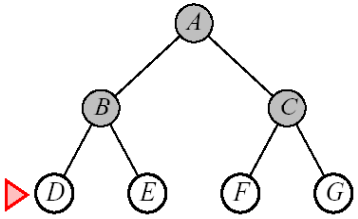


جستجوی سطحی

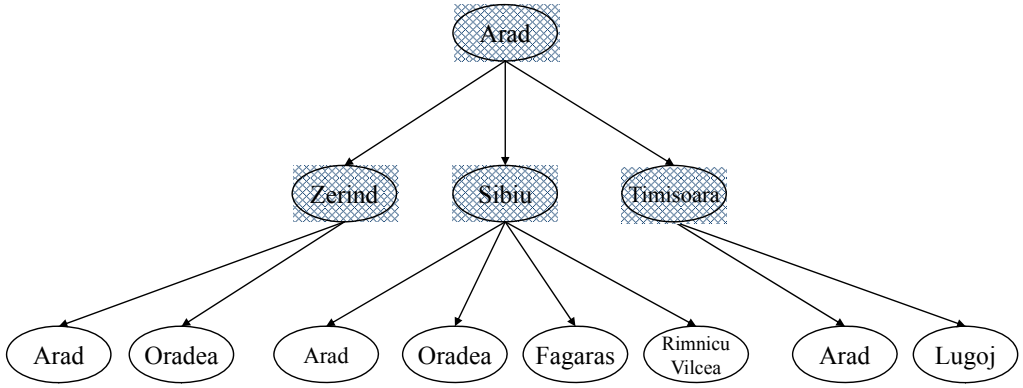
- هربار سطحی ترین گره گسترش نیافته را گسترش می دهد.
- **پیاده سازی:**

– *fringe* یک صف FIFO می باشد. یعنی، فرزندان جدید به انتهای صف اضافه می شوند.

3: [D, E, F, G]



مثال: جستجوی سطحی

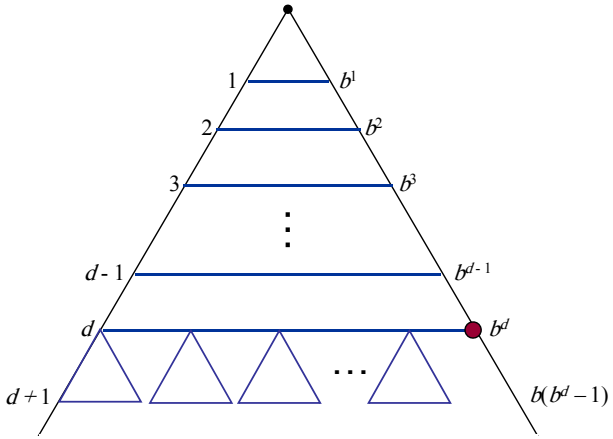


خصوصیات جستجوی سطحی

- کامل؟ بله (به شرط محدود بودن b)
- پیچیدگی زمانی؟ $b + b^2 + \ldots + b^d + b(b^d\text{-}1) = O(b^{d+1})$
- پیچیدگی حافظه؟ $O(b^{d+1})$ چون همه گره ها را در حافظه نگه می دارد
- بهینه؟ بله (مثلا اگر بازاء هر عمل، هزینه = ۱)

- مشکل اصلی **حافظه** می باشد. (نسبت به زمان)

پیچیدگی زمانی و حافظه جستجوی سطحی



$$b + b^2 + \ldots + b^d + b(b^d\text{-}1) = O(b^{d+1})$$

زمان و فضای لازم در جستجوی سطحی

عمق	تعداد گره ها	زمان	حافظه
۲	۱۱۰۰	۱۱/۰ ثانیه	۱ مگا بایت
۴	۱۱۱۱۰۰	۱۱ ثانیه	۱۰۶ مگا بایت
۶	$۱۰^{۷}$	۱۹ دقیقه	۱۰ گیگا بایت
۸	$۱۰^{۹}$	۳۱ ساعت	۱ ترا بایت
۱۰	$۱۰^{۱۱}$	۱۲۹ روز	۱۰۱ ترا بایت
۱۲	$۱۰^{۱۳}$	۳۵ سال	۱۰ پتا بایت
۱۴	$۱۰^{۱۵}$	۳۵۲۳ سال	۱ هگزا بایت

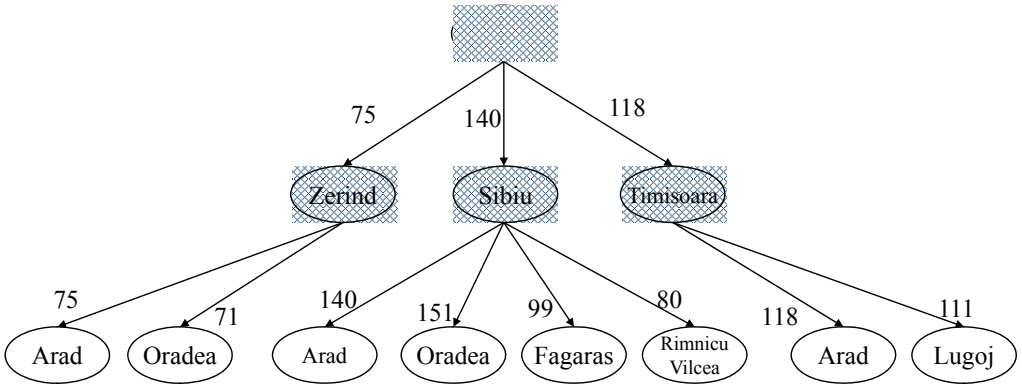
- $b = ۱۰$
- ۱۰۰۰۰ گره در هر ثانیه
- هر گره ۱۰۰۰ بایت

جستجوی هزینه-یکنواخت

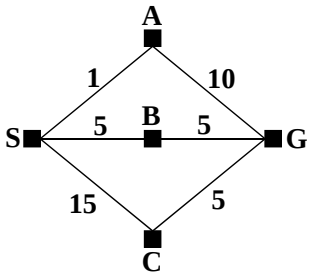
- هربار کم هزینه ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = صفی که براساس هزینه مسیر مرتب شده باشد.
- معادل جستجوی سطحی اگر هزینه گام ها مساوی باشند.

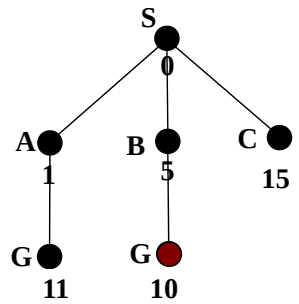
- کامل؟ بله اگر هزینه گامها $\geq \varepsilon$
- پیچیدگی زمانی؟ تعداد گره هایی که هزینه مسیر آنها کوچکتر یا مساوی هزینه راه حل بهینه باشد.
  $O(b^{\lceil C^*/\varepsilon \rceil})$
- پیچیدگی حافظه؟ مانند پیچیدگی زمانی
- بهینه؟ بله (گره ها به ترتیب صعودی $g(n)$ گسترش می یابند).

مثال: جستجوی هزینه یکنواخت



مثال: جستجوی هزینه یکنواخت



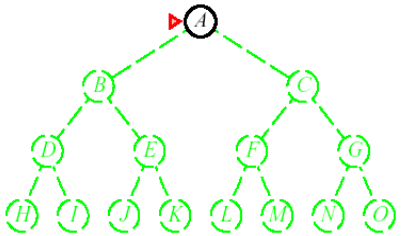


0: [S(0)]
1: [A(1), B(5), C(15)]
2: [B(5), G(11), C(15)]
3: [G(10), G(11), C(15)]
4: [G(11), C(15)]

جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

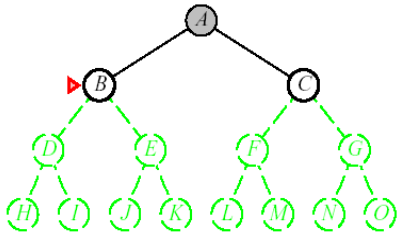
0: [A]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

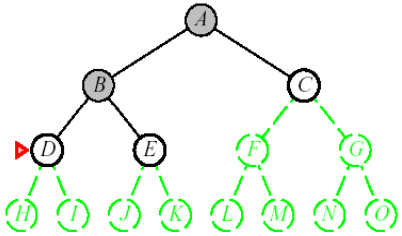
1: [B, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

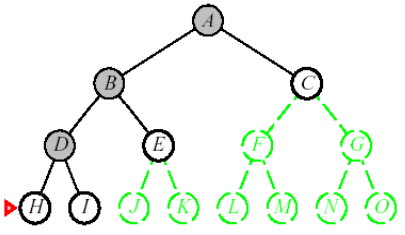
2: [D, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

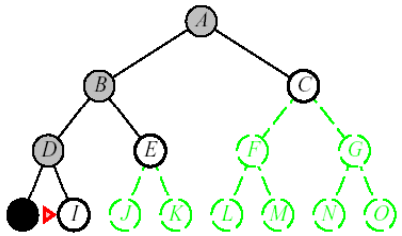
3: [H, I, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

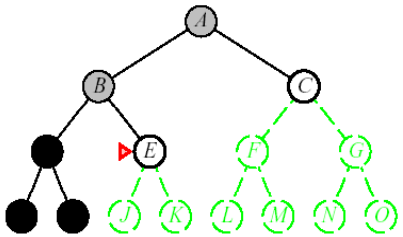
4: [I, E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

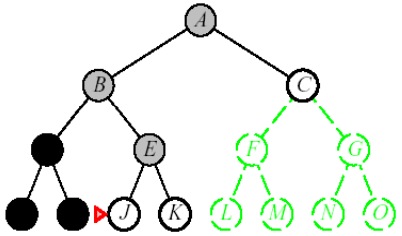
5: [E, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

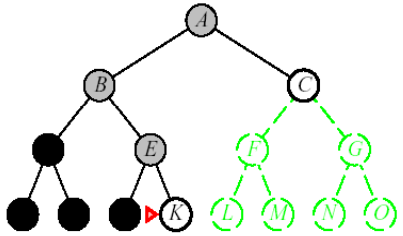
6: [J, K, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

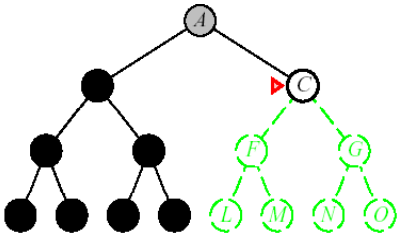
7: [K, C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

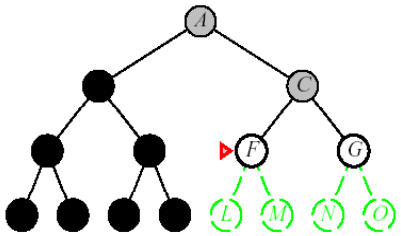
8: [C]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

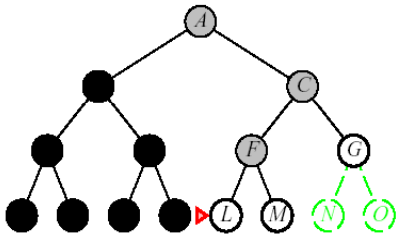
9: [F, G]



جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

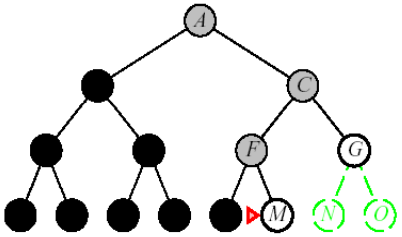
10: [L, M, G]



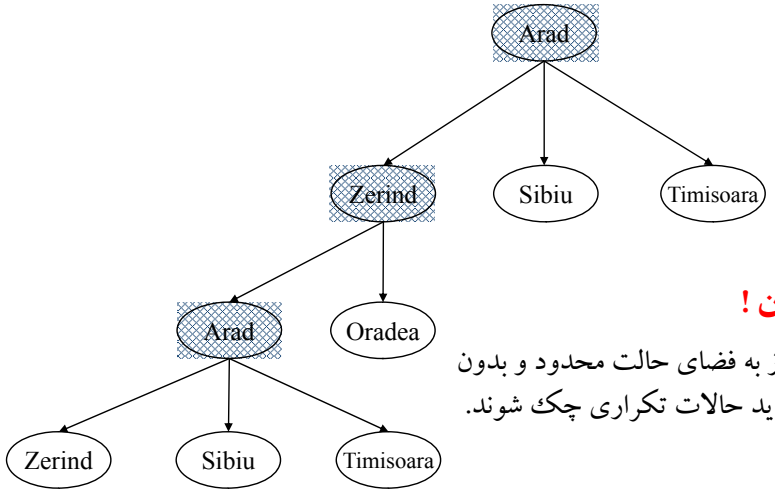
جستجوی عمقی

- هربار عمیق ترین گره گسترش نیافته را گسترش می دهد.
- پیاده سازی:
  - *fringe* = پشته LIFO، فرزندان جدید را در ابتدا درج می کند.

11: [M, G]



مثال: جستجوی عمقی



- **حلقه بی پایان !**

در این جستجو نیاز به فضای حالت محدود و بدون چرخه داریم، یا باید حالات تکراری چک شوند.

خصوصیات جستجوی عمقی

- کامل؟
  - خیر (در فضاهای حالت با عمق نامحدود، دارای حلقه)
  - برای اجتناب از حالات تکرای در طول مسیر، نیاز به اصلاح دارد.
  - بنابراین، در فضای حالت محدود کامل است.
- پیچیدگی زمانی؟ $O(b^m)$
  - در بدترین حالت تمام گره های درخت جستجو تولید می شوند
  - اگر m خیلی بیشتر از d باشد، بسیار زیاد
  - اگر تعداد راه حل ها زیاد باشد، می تواند بسیار سریعتر از جستجوی سطحی باشد
- پیچیدگی حافظه؟
  - $O(bm)$، به صورت خطی!
- بهینه؟ خیر

جستجوی با عمق محدود

= جستجوی عمقی با محدوده عمقی l
یعنی، فرزندان گره های واقع در عمق l تولید نخواهند شد.

- در این استراتژی با در نظر گرفتن یک محدوده عمقی مانند l از به دام افتادن جستجوی عمقی در یک حلقه بی پایان جلوگیری می شود.(برش روی درخت جستجو)
- مثلا در نقشه رومانی چون ۲۰ شهر وجود دارد بنابراین طول راه حل باید حداکثر ۱۹ باشد.
- بنابراین هیچ وقت گره ای با عمق بیش از ۱۹ بررسی نخواهد شد.
- اگر در محدوده عمقی l راه حلی وجود داشته باشد، بالاخره پیدا خواهد شد، اما هیچ تضمینی برای یافتن راه حل بهینه وجود ندارد.

جستجوی با عمق محدود

- کامل؟
  - بله (اگر $l \geq d$)
- پیچیدگی زمانی؟
  - $O(b^l)$
- پیچیدگی حافظه؟
  - $O(bl)$
- بهینه؟
  - خیر

جستجوی با عمق محدود
پیاده سازی بازگشتی

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```


جستجوی عمیق کننده تکراری

- مشکل اصلی در جستجوی عمقی با عمق محدود شده (DLS) انتخاب یک محدوده عمقی مناسب است.
- در نقشه رومانی طول بزرگترین مسیر بین دو شهر ۹ می باشد(قطر)، و این محدوده عمقی مناسب تر از ۱۹ می باشد. اما در بیشتر فضاهای حالت انتخاب محدوده مناسب قبل از حل مسأله میسر نمی باشد.
- جستجوی عمیق کننده تکراری روشی برای تعیین محدوده عمقی مناسب با امتحان کردن تمامی محدوده ها (از صفر به بالا) می باشد. یعنی اول عمق صفر، بعد عمق ۱، بعد عمق ۲ و ...

جستجوی عمیق کننده تکراری

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH( problem, depth)
      if result ≠ cutoff then return result
```

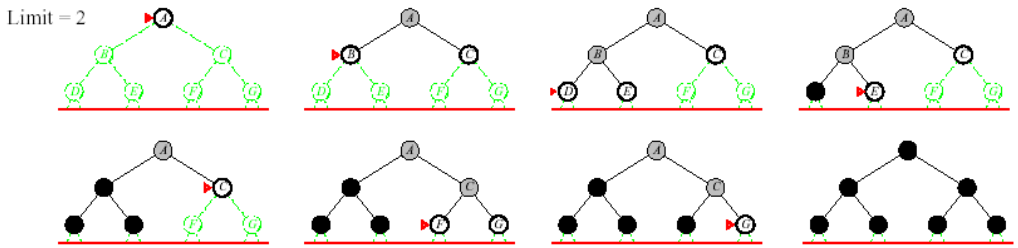

جستجوی عمیق کننده تکراری

- جستجوی عمیق کننده تکراری مزایای جستجوی سطحی و عمقی را با هم ترکیب می کند:
  - مانند جستجوی سطحی کامل (اگر فاکتور انشعاب محدود باشد) و بهینه (اگر هزینه مسیر یک تابع غیر نزولی برحسب عمق باشد)است.
  - جستجوی عمقی دارای مصرف حافظه خطی $O(bd)$ می باشد.
- این جستجو از نظر پیچیدگی زمانی مانند جستجوی محدود شده می باشد، به جز اینکه برخی حالات چند بار بسط داده می شوند.

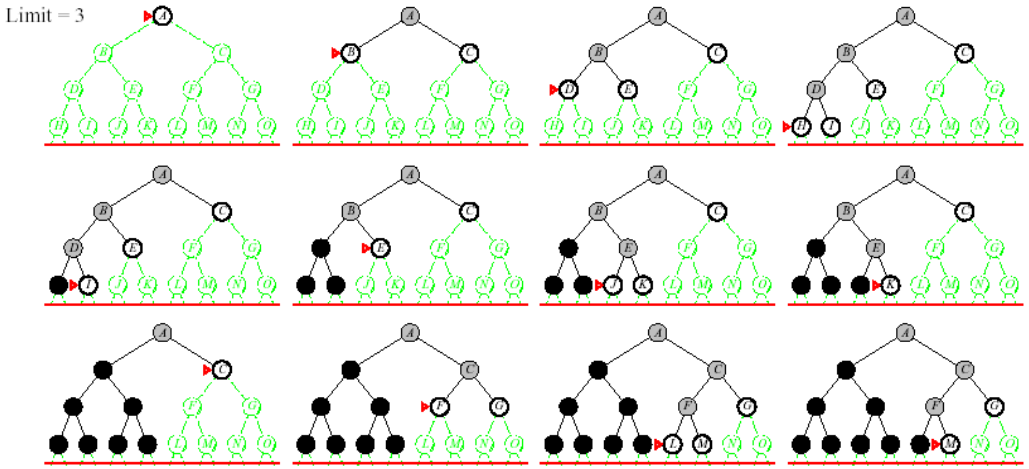
جستجوی عمیق کننده تکراری (Limit = 0)

جستجوی عمیق کننده تکراری (Limit = 1)

جستجوی عمیق کننده تکراری (Limit = 2)



جستجوی عمیق کننده تکراری (Limit = 3)



خواص جستجوی عمیق کننده تکراری

- کامل؟؟ بله (مانند جستجوی سطحی)
- پیچیدگی زمانی ؟؟

$$db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$$

- پیچیدگی حافظه؟؟ $O(bd)$
- بهینه ؟؟ بله، (مانند جستجوی سطحی)

- می تواند برای کاوش درخت جستجوی هزینه یکنواخت اصلاح شود!!!

پیچیدگی زمانی عمیق کننده تکراری

DLS (/= 0)	0
DLS (/= 1)	b^1
DLS (/= 2)	$b^1 + b^2$
.	.
.	.
.	.
DLS (/= d - 1)	$b^1 + b^2 + b^3 + \ldots + b^{d-1}$
DLS (/= d)	$b^1 + b^2 + b^3 + \ldots + b^{d-1} + b^d$

سربار $\leq N_{DLS\ (/=d)}$

$$N_{IDS} = db^1 + (d-1)b^2 + (d-2)b^3 + \ldots + 2b^{d-1} + b^d$$

کارآیی IDS

- تعداد گره های تولید شده توسط DLS در عمق d با فاکتور انشعاب b:

$N_{DLS} = b + b^2 + \ldots + b^{d-1} + b^d$

- تعداد گره های تولید شده توسط IDS در عمق d با فاکتور انشعاب b:

$N_{IDS} = db^1 + (d-1)\,b^2 + \ldots + 2b^{d-1} + 1b^d$

- اگر $b = 10$ و $d = 5$:

$N_{DLS} = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}110$

$N_{IDS} = 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$

- محاسبه میزان سربار:

$((123450 - 111110)/111110) * 100 = 11\%$

  - با افزایش فاکتور انشعاب b میزان سربار کاهش می یابد.
  - در بدترین حالت $b = 2$، سربار 100% است و این جستجو دو برابر زمان می برد.

جستجوی دوطرفه

- ایده: انجام جستجو در دو جهت به طور همزمان
  - رو به جلو: از حالت اولیه به سمت حالت هدف
  - رو به عقب: از حالت هدف به سمت حالت اولیه
- انگیزه: $b^{d/2} + b^{d/2}$ بسیار کمتر از b^d می باشد

- مثال: اگر راه حل یک مسأله در عمق $d = 6$ باشد و $b = 10$ آنگاه
  - جستجوی دو طرفه (در هر دو طرف جستجوی سطحی) ← 22,200 گره
  - جستجوی سطحی ← 11,111,000 گره

جستجوی دوطرفه

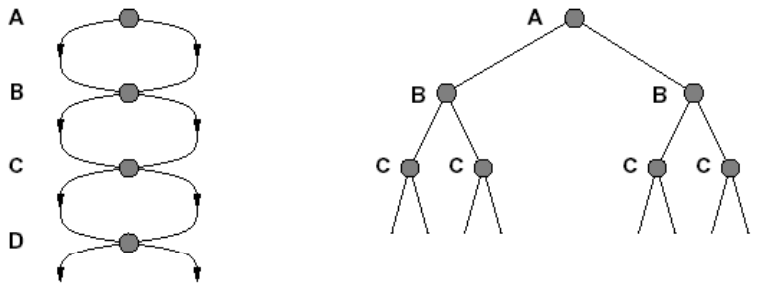
- پیچیدگی زمانی: $O(b^{d/2})$
- پیچیدگی حافظه: $O(b^{d/2})$
  - به منظور بررسی تعلق حداقل یکی از درخت ها باید در حافظه نگهداری شود
  - مصرف حافظه نمایی، بزرگترین ضعف جستجوی دوطرفه می باشد
- کامل بودن و بهینگی (برای هزینه های گام یکسان):
  - اگر در هر دو طرف از جستجوی سطحی استفاده شود

خلاصه الگوریتم ها

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

حالات تکراری

- شکست در تشخیص حالت های تکرای می تواند یک مسأله خطی را به یک مسأله نمایی تبدیل کند!



جستجوی گراف

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

خلاصه

- فرموله سازی مسأله اغلب نیاز به انتزاع جزییات مسأله دارد، تا بتوان فضای حالتی بدست آوردکه به صورت مقرون به صرفه ای قابل کاوش کرد ن و جستجو باشد.

- انواع استراتژی های ناآگاهانه وجود دارد.

- مصرف حافظه جستجوی عمیق کننده تکراری دارای مرتبه خطی می باشد و زمان خیلی بیشتری نسبت به سایر روشهای ناآگاهانه مصرف نمی کند.

شبکه آموزشی- پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است

madsg.com

مادسیج

IRan Education & Research NETwork
(IRERNET)