

بنام خداوند جان و دین



# هوش مصنوعی پیشرفته

Advanced Artificial Intelligence

سهیلا اشکذری طوسی

مروری بر حل مسئله از طریق جستجو  
و جستجوی ناآگاهانه

# فهرست مطالب

- عامل‌های حل مسئله
- مسئله
- اندازه‌گیری کارایی حل مسئله
- جستجوی ناآگاهانه

# عوامل های حل مسئله

4

□ عامل حل مسئله، یک نوع عامل هدف گراست.

□ مسئله، در واقع مجموعه‌ای از اطلاعات است که عامل از آنها برای تصمیم‌گیری در مورد اینکه چه کاری انجام دهد، استفاده می‌کند.

# عوامل های حل مسئله...

□ چهار گام اساسی برای حل مسائل

□ فرموله کردن هدف: وضعیتهای مطلوب نهایی کدامند؟

□ فرموله کردن مسئله: چه اقدامات و وضعیتهایی برای رسیدن به هدف موجود است؟

□ جستجو: انتخاب بهترین دنباله از اقداماتی که منجر به حالاتی با مقدار شناخته شده میشود (راه حل).

□ اجرا: وقتی دنباله اقدامات مطلوب پیدا شد، فعالیتهای پیشنهادی آن می تواند اجرا شود.

□ چهار نوع اساسی از مسائل وجود دارند:

■ مسائل تک حالت (Single-state)

■ مسائل چند حالت (Multiple-state)

■ مسائل احتمالی (Contingency)

■ مسائل اکتشافی (Exploration)

# انواع مسائل...

## □ مدل تک حالت :

□ حس‌گرهای عامل به آن اطلاعات کافی می‌دهند تا وضعیت دقیق مشخص شود. (دنيا رؤیت پذیر و قطعی است). عامل می‌تواند محاسبه کند که کدام وضعیت پس از هر دنباله از عملیات قرار خواهد گرفت.

## □ مدل چند حالت:

□ عامل تمام اثرهای عملیاتش را می‌داند اما دسترسی به حالت دنیا را محدود کرده است. یعنی ممکن است نداند در چه حالتی قرار دارد. زمانی که دنیا تماماً رؤیت پذیر نیست عامل باید در مورد مجموعه حالت‌هایی که ممکن است به آن برسد استدلال کند.

# انواع مسائل ...

## □ مدل احتمالی:

□ با این مدل حل مسئله، حس‌گرهایی را در طول فاز اجرایی نیاز داریم. عامل اکنون باید تمام درخت عملیاتی را بر خلاف دنباله عملیاتی منفرد، محاسبه کند. که به طور کلی هر شاخه درخت، با یک امکان احتمالی که از آن ناشی می‌شود، بررسی می‌شود. محیط نیمه رویت پذیر و/یا غیر قطعی است.

## □ مدل اکتشافی:

□ عاملی که هیچ اطلاعاتی در مورد اثرات عملیاتی ندارد. فضای حالت ناشناخته است. در این حالت، عامل باید تجربه کند و به تدریج کشف کند که چه عملیاتی باید انجام شود و چه وضعیت‌هایی وجود دارند. این روش یک نوع جستجو است. اگر عامل نجات یابد، «نقشه‌ای» از محیط را یاد می‌گیرد که می‌تواند مسائل بعدی را حل کند.



# انواع مسائل...

- برای تعریف یک مسئله موارد زیر نیاز داریم:
- وضعیت آغازین/حالت ابتدایی (initial state) که عامل خودش از بودن در آن آگاه است.
- تابع پسین (Successor-Function)، مجموعه‌ای از عملیات ممکن، که برای عامل قابل دسترسی باشد.
- آزمون هدف (goal test)، که عامل می‌تواند برای ارزیابی حالتی که در آن قرار گرفته، استفاده نماید تا مشخص شود حالت مطلوب است یا خیر.
- تابع هزینه مسیر (path cost function)، تابعی است که برای هر مسیر، هزینه‌ای را در نظر می‌گیرد.
- مثلاً هزینه یک سفر = مجموع هزینه‌های عملیات اختصاصی در طول مسیر

# انواع مسائل / اندازه گیری کارایی حل مسئله

10

□ کارایی یک جستجو، حداقل از سه طریق می تواند اندازه گیری شود:

□ آیا این جستجو راه حلی پیدا می کند؟

□ آیا راه حلی مناسبی است؟

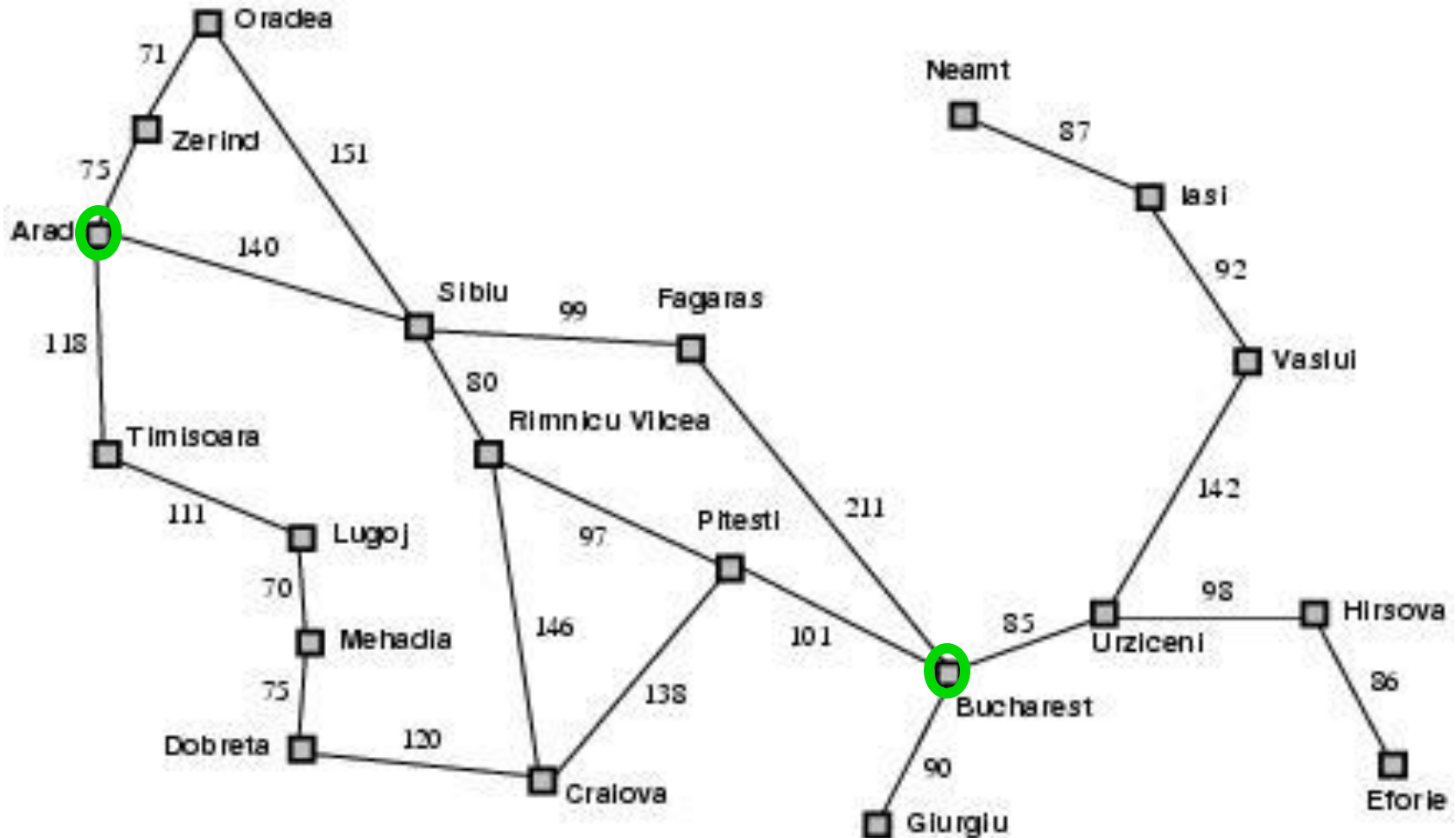
□ هزینه جستجو از نظر زمانی و حافظه مورد نیاز برای یافتن راه حل چیست؟

□  $\text{مجموع هزینه جستجو} = \text{هزینه مسیر} + \text{هزینه جستجو}$

□ عامل باید تصمیم بگیرد که چه منابعی را فدای جستجو و چه منابعی را صرف اجرا کند.

# حل مسئله با جستجو / نقشه رومانی

۱۱



## حل مسئله با جستجو...

- حالت اولیه: حالتی که عامل از آن شروع میکند.
- در مثال رومانی: شهر آراد  $In(Arad)$
- تابع پسین: توصیفی از فعالیتهای ممکن که برای عامل مهیا است.
- در مثال رومانی:  $S(Arad) = \{Zerind, Sibui, Timisoara\}$
- فضای حالت: مجموعه ای از حالتها که از حالت اولیه میتوان به آنها رسید.
- در مثال رومانی: کلیه شهرها که با شروع از آراد میتوان به آنها رسید.
- تابع پسین + حالت اولیه = فضای حالت

# حل مسئله با جستجو / نقشه رومانی...

- صورت مسأله: رفتن از آراد به بخارست
- فرموله کردن هدف: رسیدن به بخارست
- فرموله کردن مسئله:
- وضعیتها: شهرهای مختلف
- فعالیتها: حرکت بین شهرها
- جستجو: دنباله ای از شهرها مثل: آراد، سیبیو، فاگارس، بخارست
- این جستجو با توجه به کم هزینه ترین مسیر انتخاب میشود

# حل مسئله با جستجو / نقشه رومانی...

- آزمون هدف: تعیین میکند که آیا حالت خاصی، حالت هدف است یا خیر
- هدف صریح: در مثال رومانی، رسیدن به بخارست
- هدف انتزاعی: در مثال شطرنج، رسیدن به حالت کیش و مات
- مسیر: دنباله ای از حالتها که دنباله ای از فعالیتهای را به هم متصل میکند.
- در مثال رومانی: Arad, Sibiu, Fagaras یک مسیر است
- هزینه مسیر: برای هر مسیر یک هزینه عددی در نظر میگیرد.
- در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر
- راه حل مسئله مسیری از حالت اولیه به حالت هدف است
- راه حل بهینه کمترین هزینه مسیر را دارد

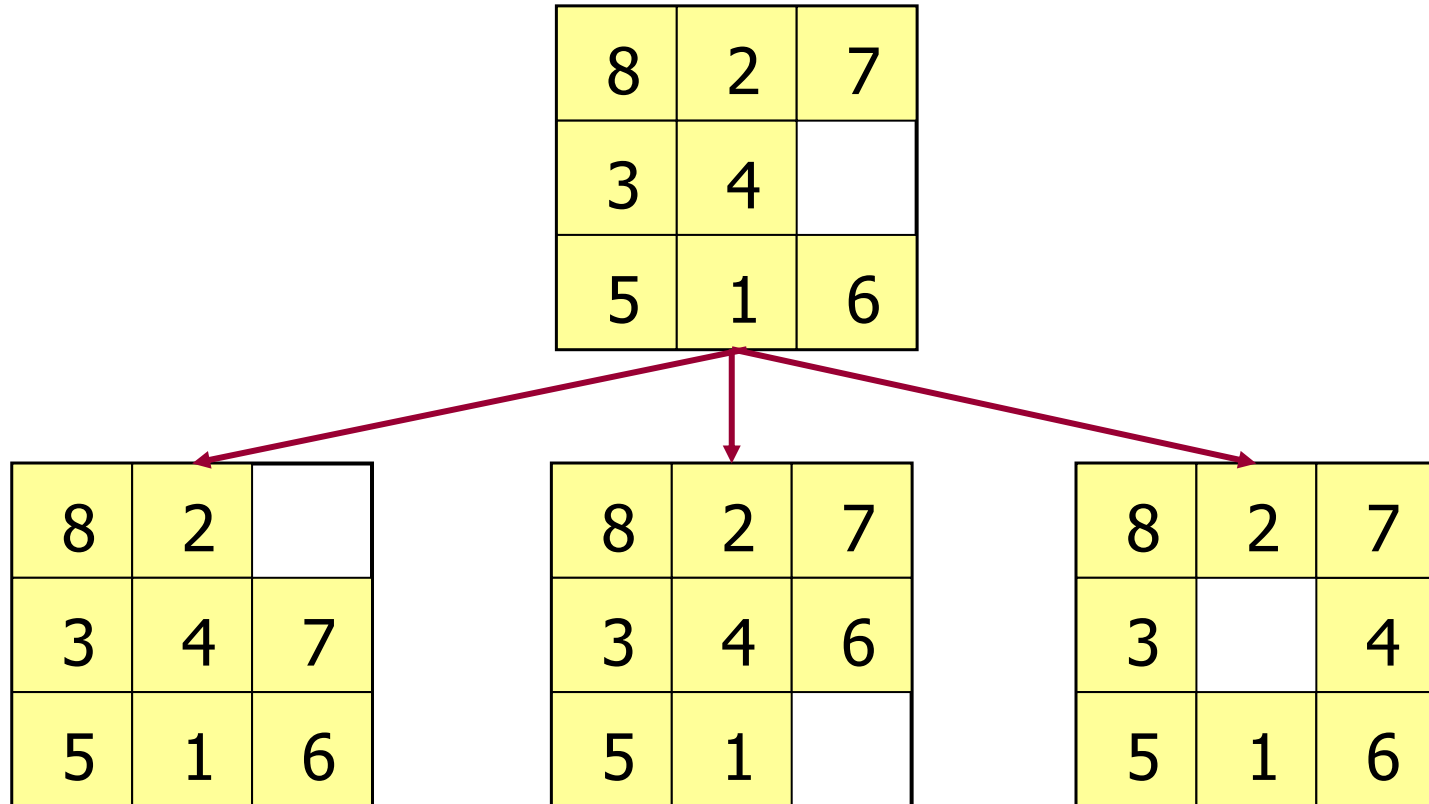
# حل مسئله با جستجو...

15

- با انجام جستجو در فضای حالت
- در این قسمت از درخت جستجو استفاده می نمایم که با استفاده از حالت ابتدایی و تابع پسین ایجاد می شود
- می توان به جای درخت از گراف جستجو استفاده نمود
- هنگامی که یک حالت از چند مسیر قابل دسترسی باشد

# حل مسئله با جستجو/پازل هشت تایی

16

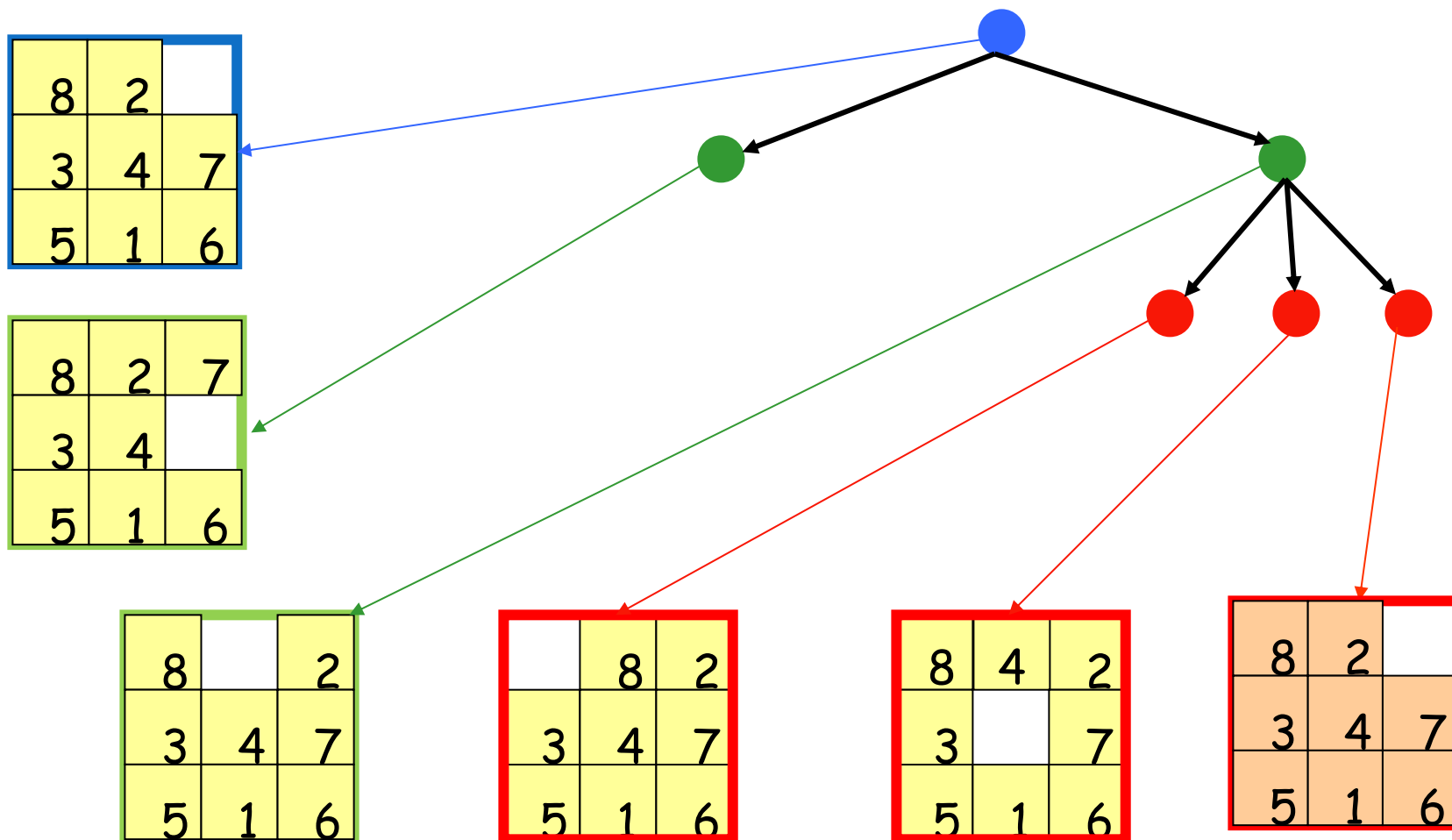


هر گره در درخت جستجو معادل یک حالت از «فضای حالت» می باشد



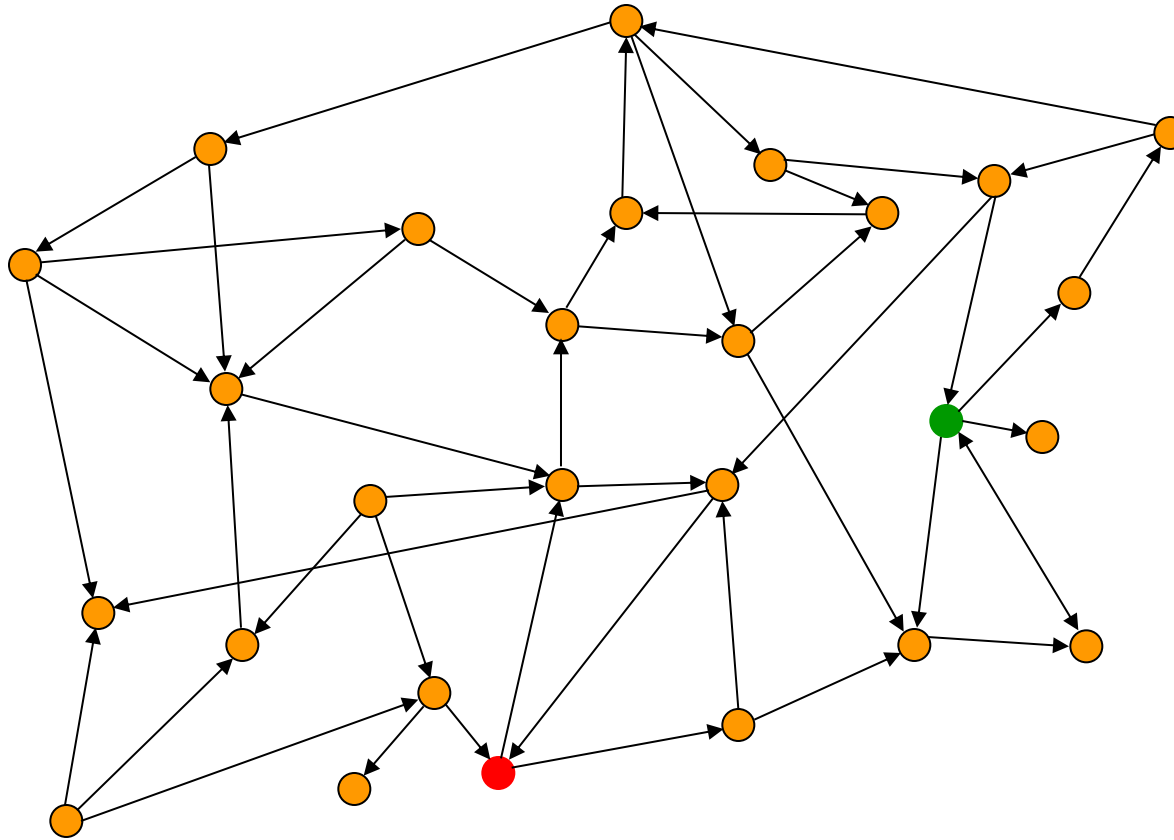
# حل مسئله با جستجو/پازل هشت تایی...

17



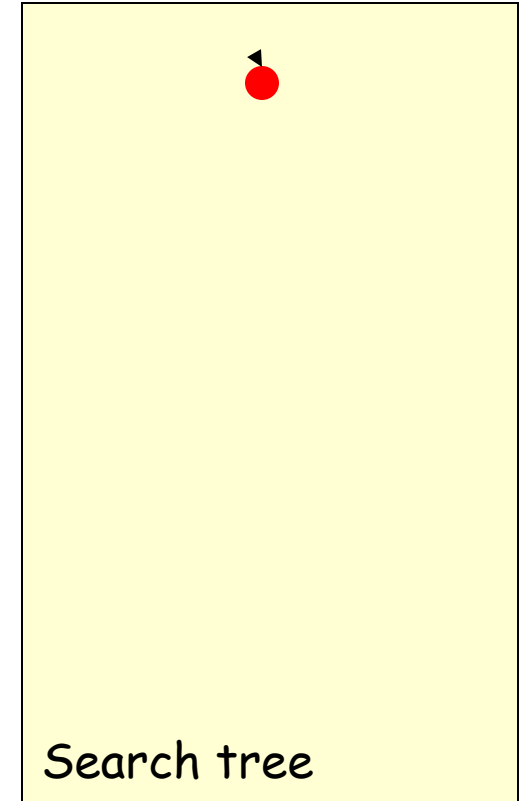
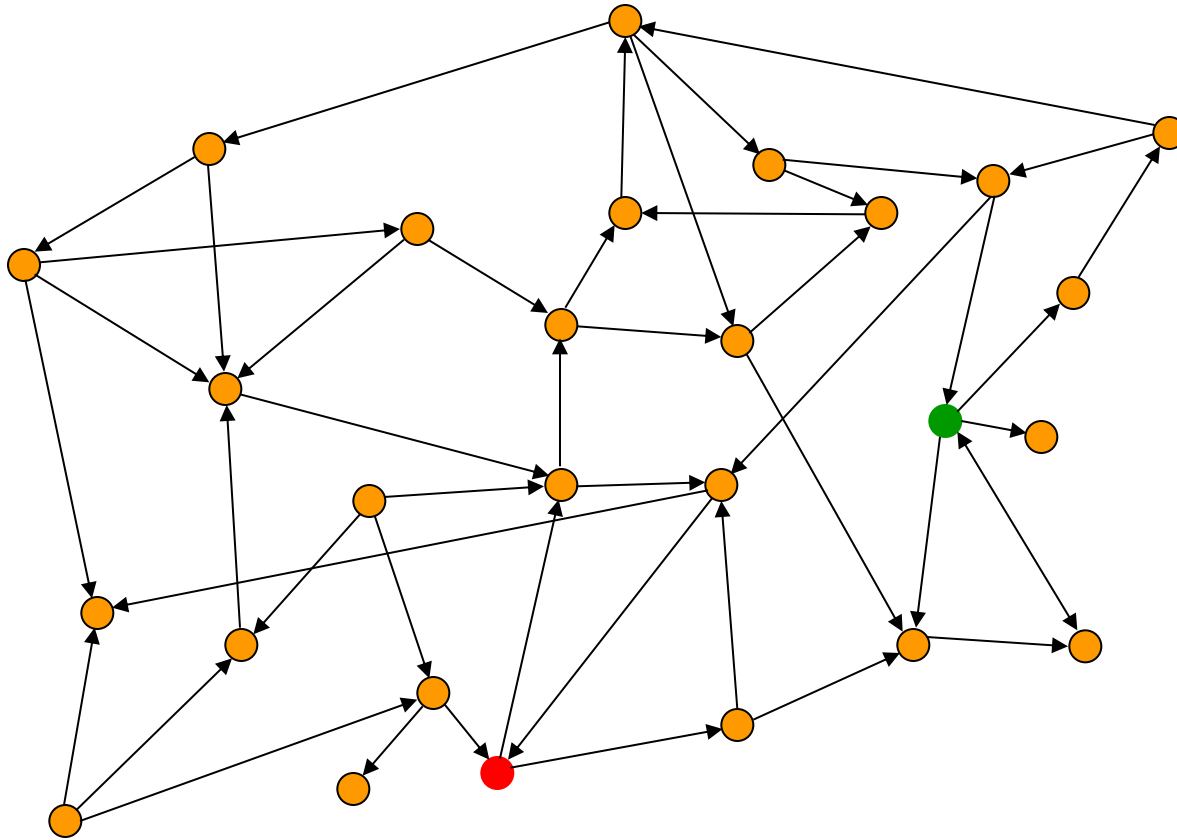
# حل مسئله با جستجو / جستجوی فضای حالت

18



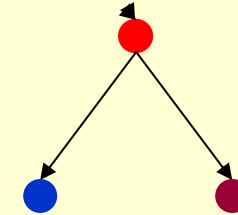
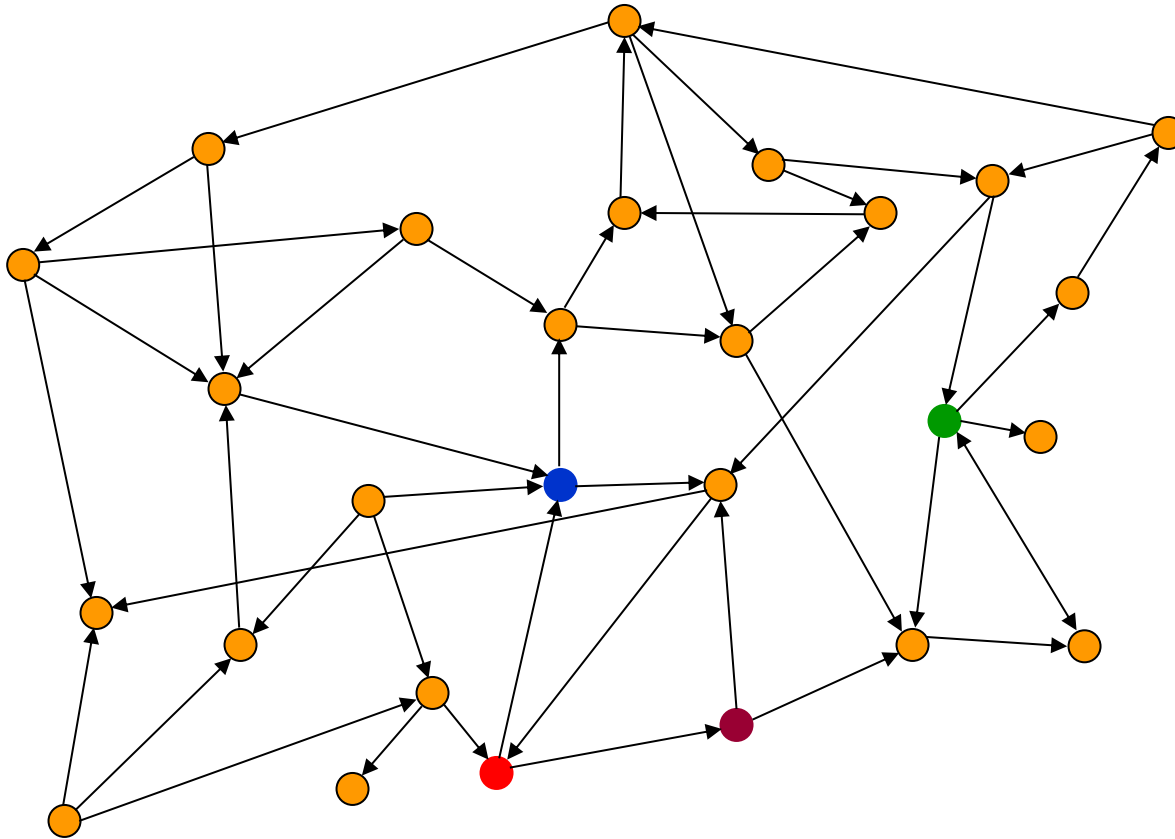
□ اغلب ساخت یک بازنمایی کامل از گراف حالت، شدنی نیست یا بسیار گران و پرهزینه تمام می شود.

□ حل کننده مسئله باید با مکاشفه بخش کوچکی از گراف جواب را بیابد.



# حل مسئله با جستجو / جستجوی فضای حالت...

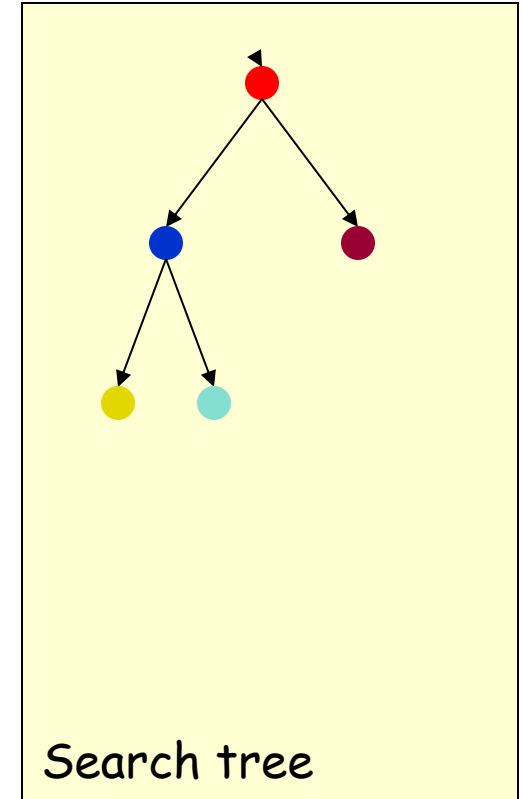
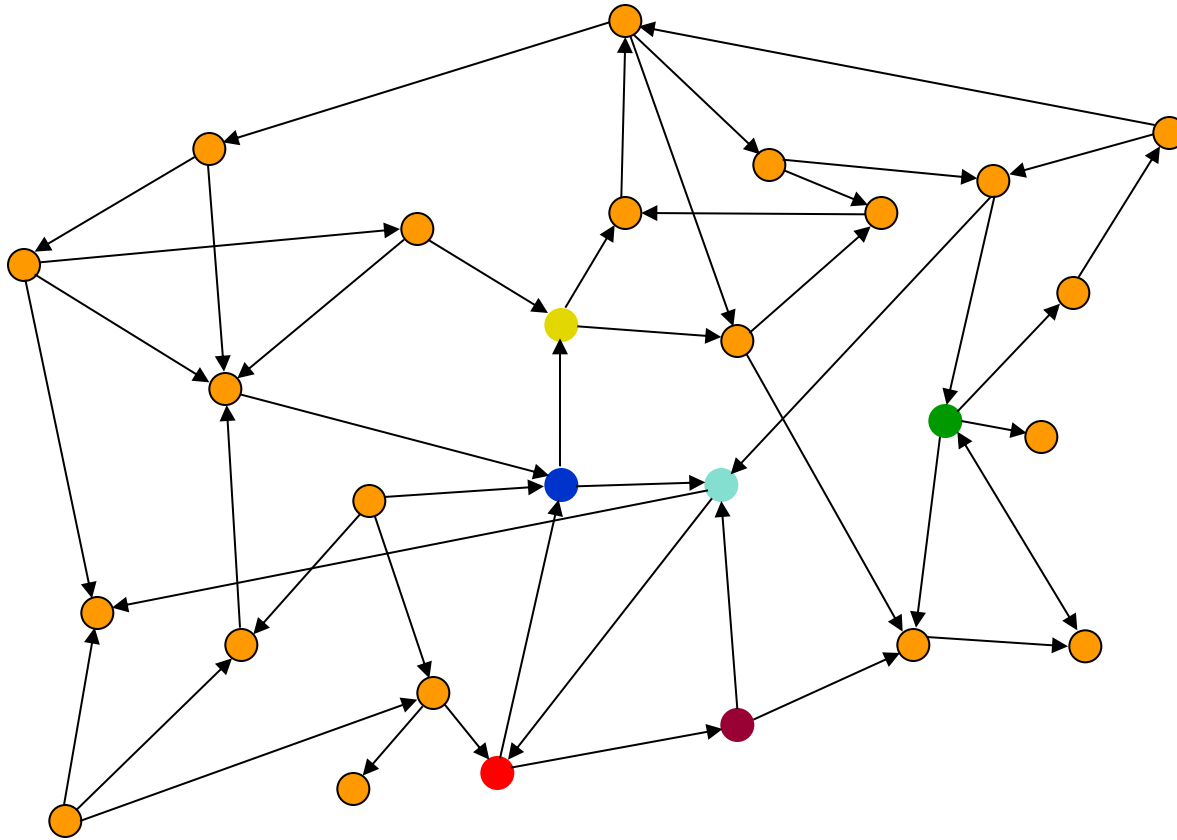
20



Search tree

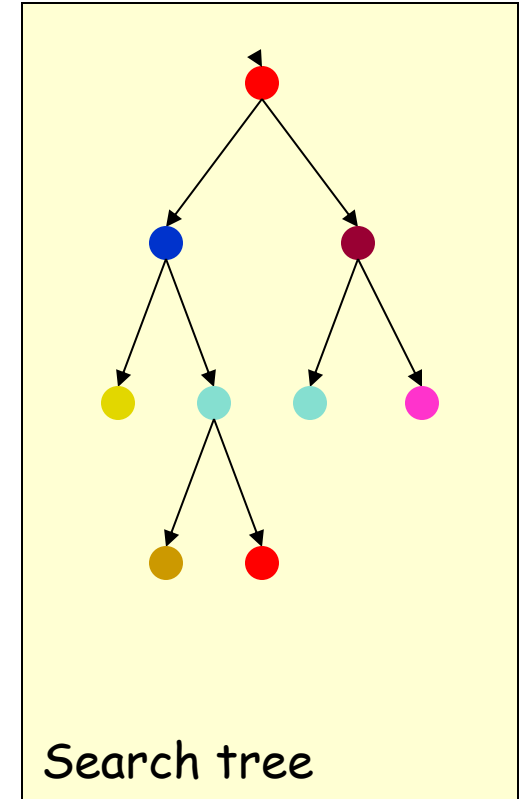
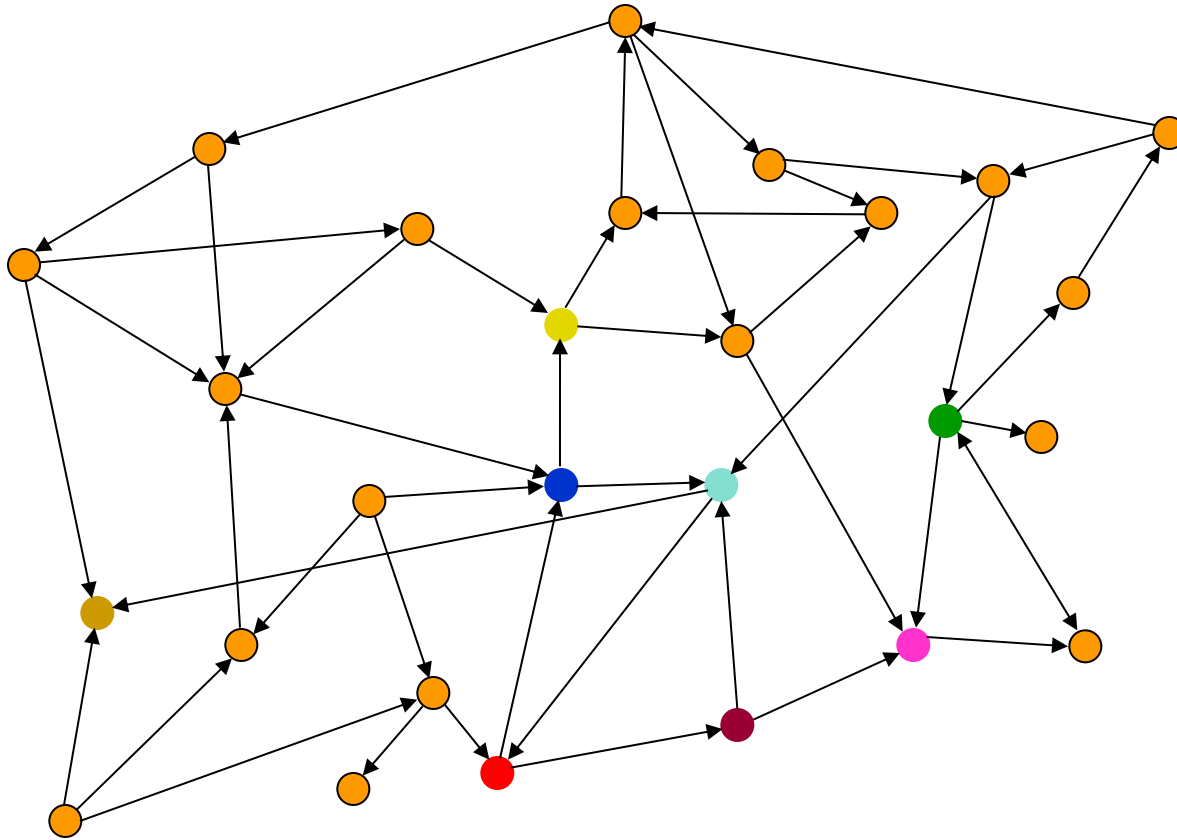
# حل مسئله با جستجو / جستجوی فضای حالت...

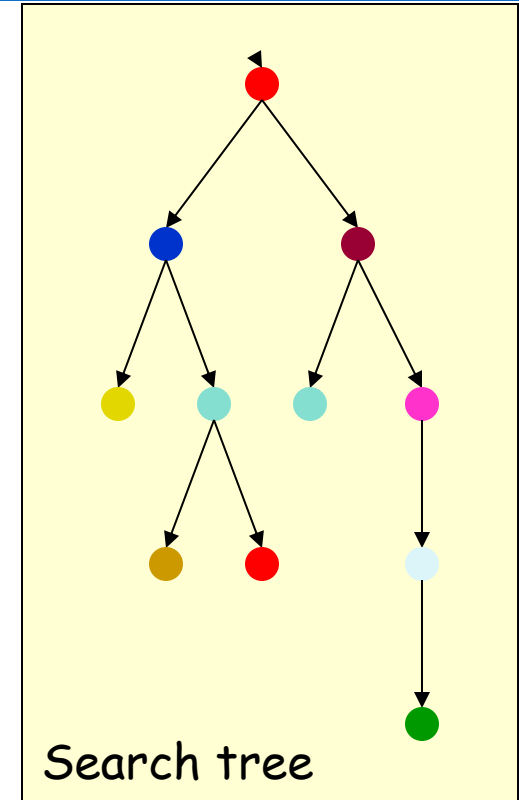
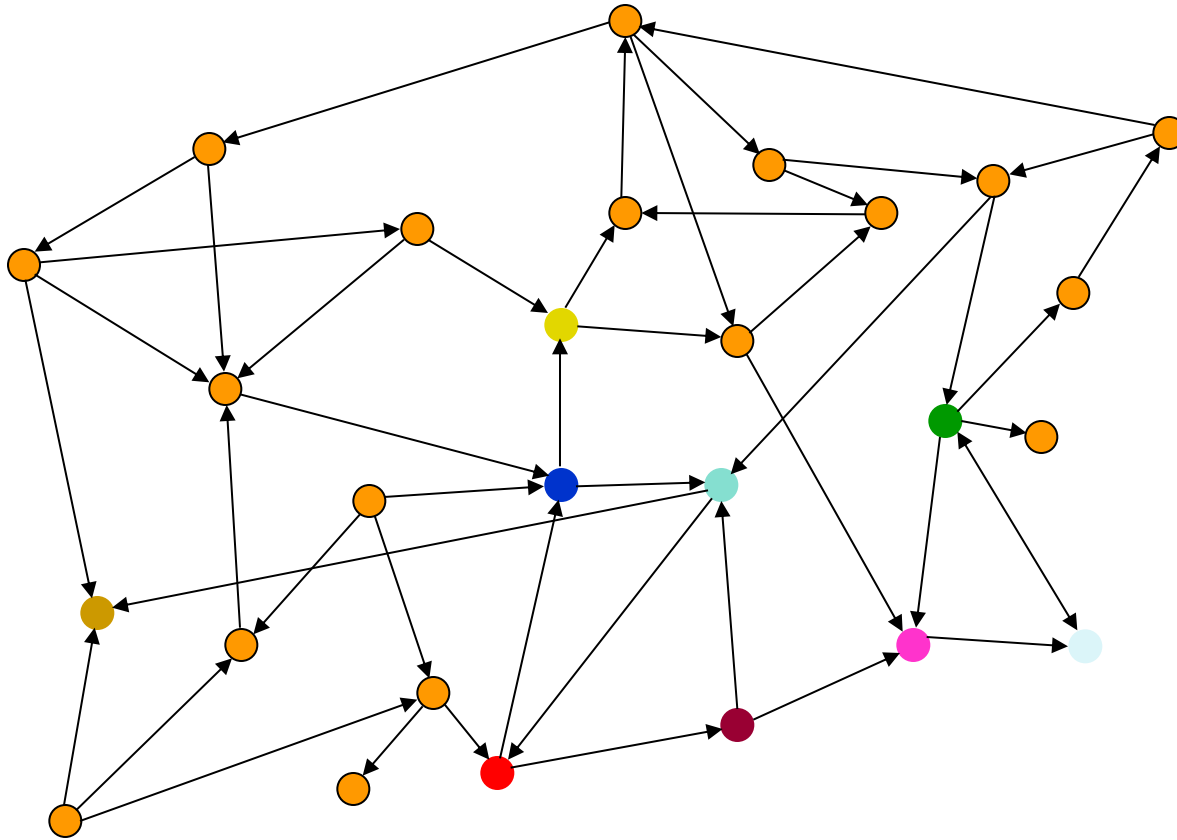
21

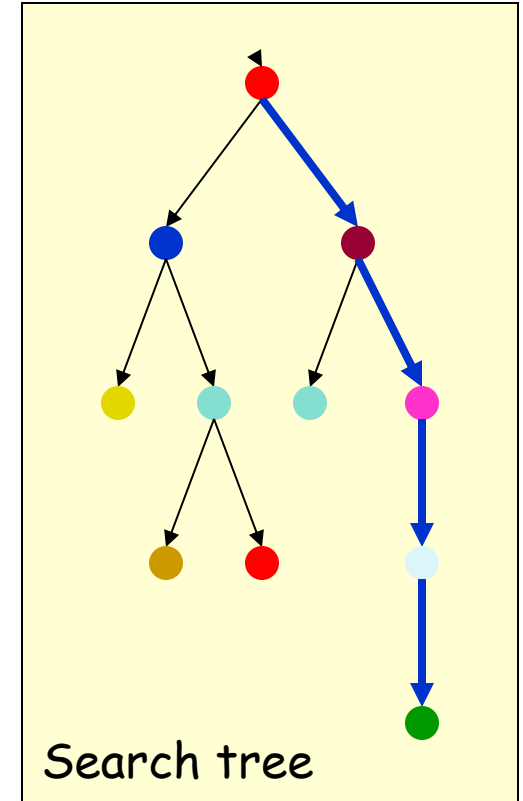
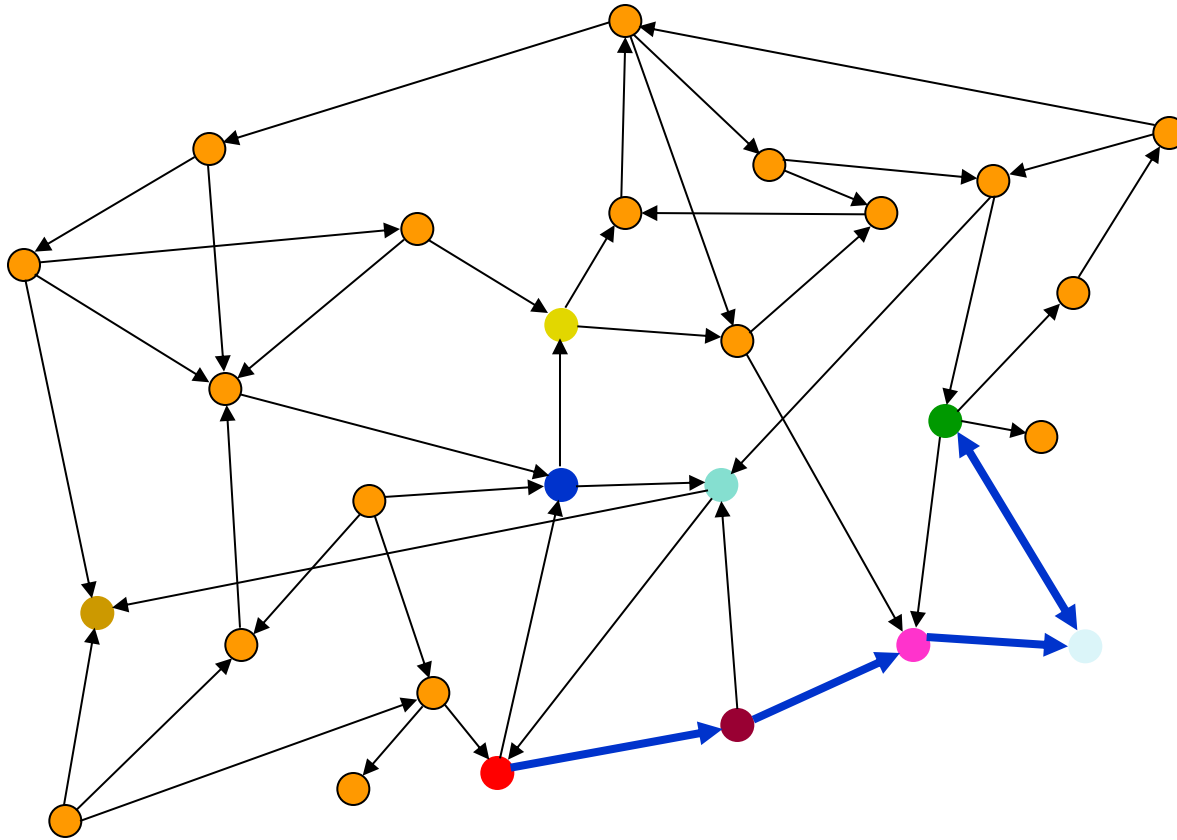


# حل مسئله با جستجو / جستجوی فضای حالت...

22







تفاوت فضای حالت و درخت جستجو چیست؟





# حل مسئله با جستجو/ جستجوی فضای حالت ...

25

8-puzzle  $\rightarrow$  362,880 states

0.036 sec

15-puzzle  $\rightarrow 2.09 \times 10^{13}$  states

$\sim$  55 hours

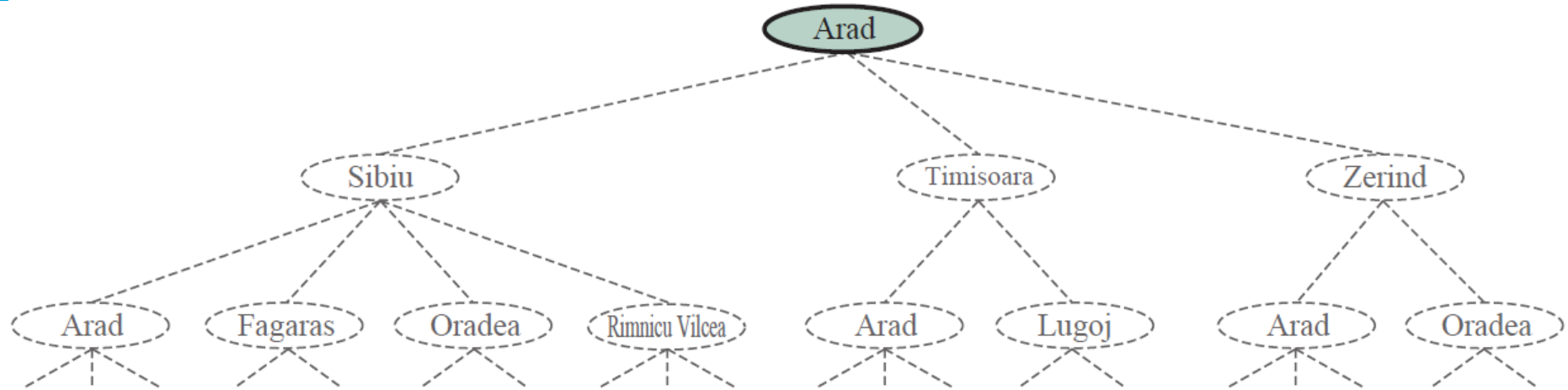
24-puzzle  $\rightarrow 10^{25}$  states

$> 10^9$  years

100 millions states/sec

# حل مسئله با جستجو / درخت جستجو ساده

26



**function** TREE-SEARCH(*problem*, *strategy*) **return** a solution or failure

Initialize search tree to the *initial state of the problem*

**do**

if no candidates for expansion **then return** failure

choose leaf node for expansion according to *strategy*

if node contains goal state **then return** solution

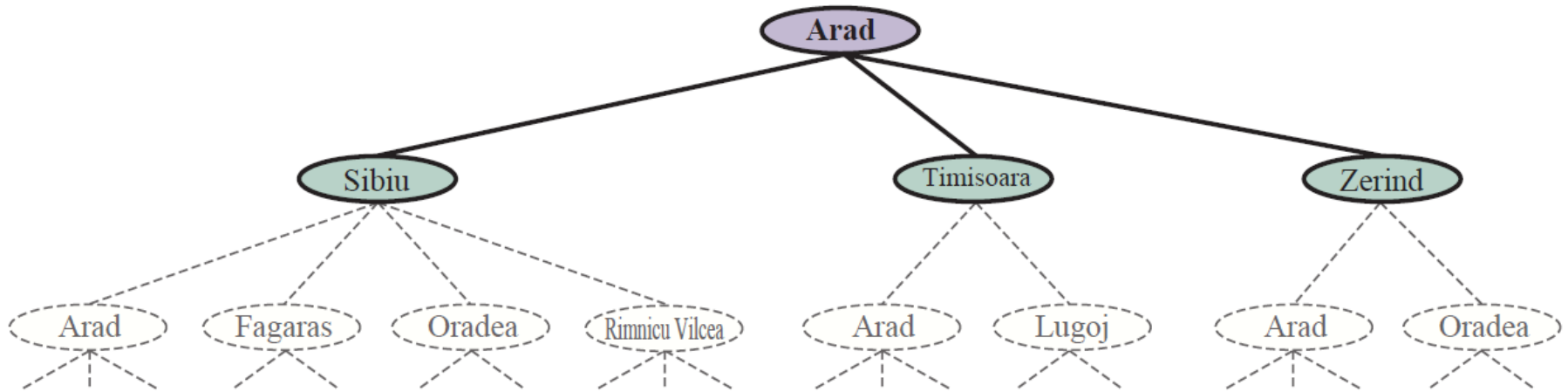
**else** expand the node and add resulting nodes to the search tree

**enddo**



# حل مسئله با جستجو / درخت جستجو ساده...

27



**function** TREE-SEARCH(*problem, strategy*) **return** a solution or failure

Initialize search tree to the *initial state* of the *problem*

**do**

if no candidates for expansion **then return** failure

choose leaf node for expansion according to *strategy*

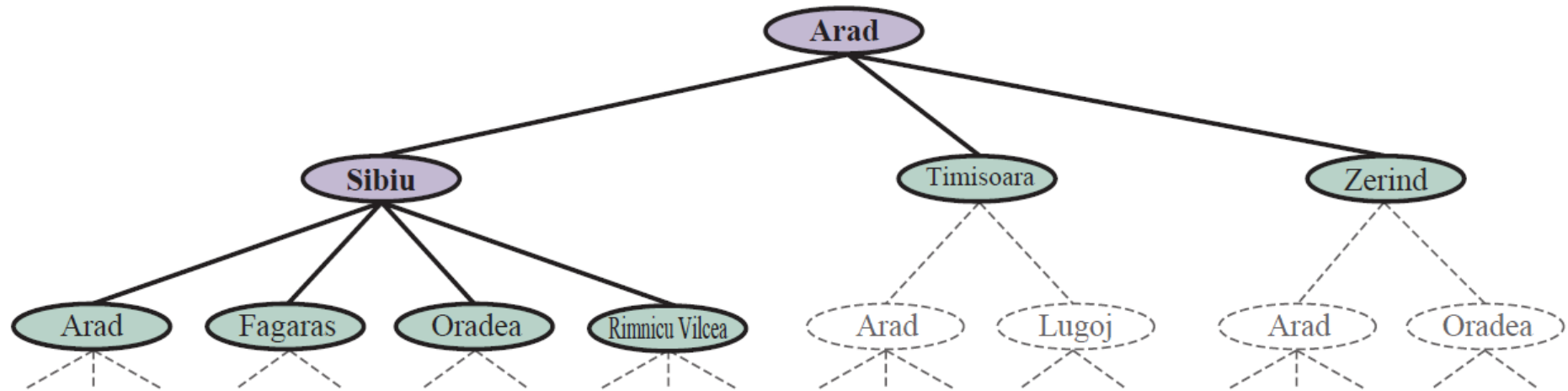
if node contains goal state **then return** solution

**else** expand the node and add resulting nodes to the search tree

**enddo**

# حل مسئله با جستجو / درخت جستجو ساده...

28



```

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

# حل مسئله با جستجو/ درخت جستجو ساده...

29

□ ساختار داده گره:

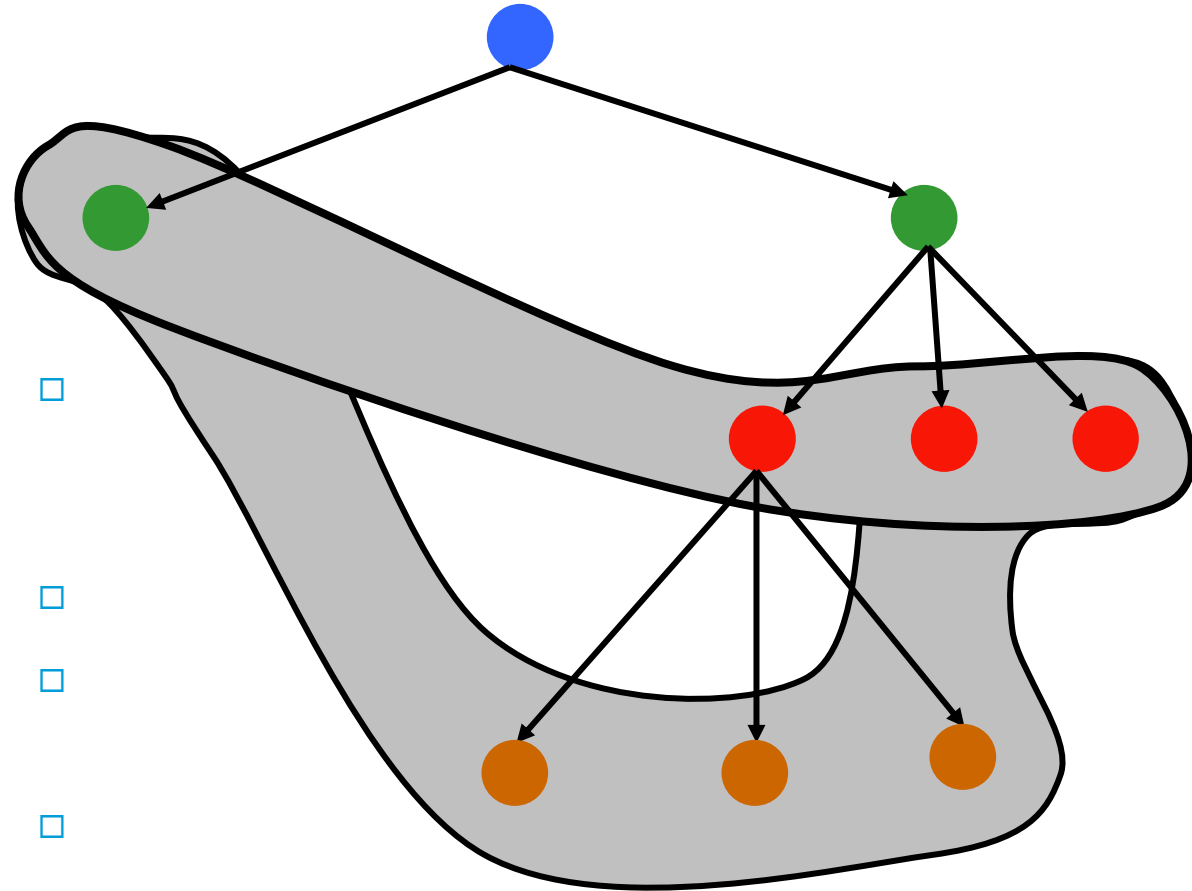
- حالت: حالتی از فضای حالت که گره متناظر با آن است
- گره والد: گره ای از درخت جستجو که این گره را تولید کرده است
- اقدام: عملی که روی گره والد انجام شده تا این گره تولید شود
- هزینه مسیر: هزینه مسیر از حالت اولیه تا رسیدن به این گره ( $g(n)$ )
- عمق: تعداد گام های مسیر از حالت اولیه



# حل مسئله با جستجو / الگوریتم درخت جستجو

30

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



- **Fringe** (لبه) مجموعه ای از نودهای جستجوست که تولید شده اند اما تا به حال توسعه نیافته اند.
- هر عنصر لبه یک «برگ» است.
- معمولاً از صف برای پیاده سازی آن استفاده می شود.
- ترتیب اولویت نودهای لبه، استراتژی جستجو را مشخص می نماید.



# حل مسئله با جستجو / الگوریتم درخت جستجو

32

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

لبه = *Fringe*



# حل مسئله با جستجو...

- خروجی یک الگوریتم حل مسئله، یا شکست است یا یک راه حل.
- در بعضی موارد ممکن است الگوریتم در حلقه بینهایت گیر کند.
- چهار معیار برای بررسی کارایی یک الگوریتم استفاده می شود.



# حل مسئله با جستجو / اندازه گیری کارایی حل مسئله

۳۴

□ کامل بودن:

□ بهینگی:

□ پیچیدگی زمانی:

□

□ پیچیدگی فضایی:

□

- ❑ **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 می کند.
- ❑ **پیچیدگی زمانی:** چقدر طول میکشد تا راه حل را پیدا کند؟
  - ❑ تعداد گره های تولید شده در اثنای جستجو
- ❑ **پیچیدگی فضایی:** برای جستجو چقدر حافظه نیاز دارد؟
  - ❑ حداکثر تعداد گره های ذخیره شده در حافظه
- ❑ **پیچیدگی با سه کمیت نشان داده می شود:**
  - ❑ ضریب انشعاب (b) : حداکثر تعداد پسین های یک گره
  - ❑ عمق کم عمق ترین گره هدف (d)
  - ❑ طولانی ترین مسیر در فضای حالت (m)

## □ جستجوی ناآگاهانه: (Blind / uninformed)

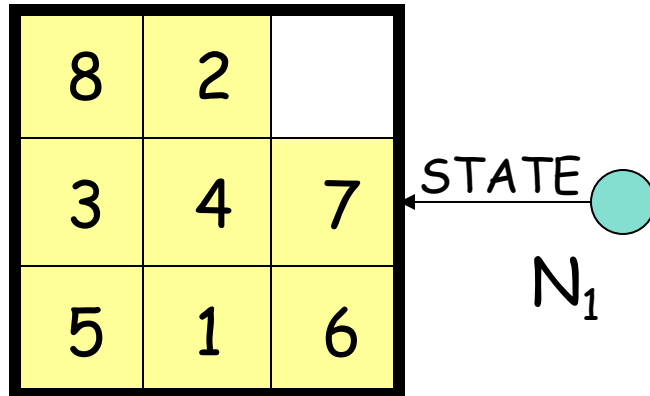
□ هیچ اطلاعات اضافه ای درباره حالت بجز آنچه در تعریف مسئله آمده، ندارند.

□ تنها قادر به تولید پسین ها و تشخیص حالت هدف از غیر هدف می باشند.

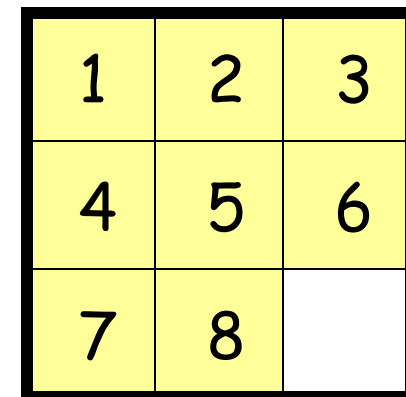
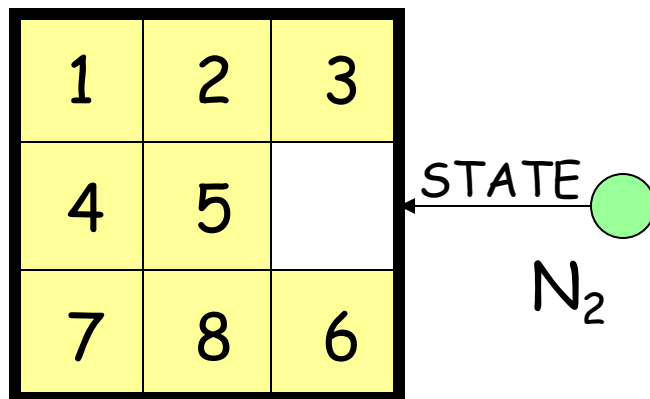
## □ جستجوی آگاهانه یا مکاشفه ای: (informed / heuristic)

□ گسترش گره ها دارای اولویت است.

□ می توانند مشخص نمایند یک حالت غیرهدف، بهتر از بقیه است.




For a **blind strategy**,  $N_1$  and  $N_2$  are just two nodes (at some position in the search tree)




Goal state

8	2	
3	4	7
5	1	6

STATE   $N_1$

For a **heuristic strategy** counting the number of misplaced tiles,  $N_2$  is more promising than  $N_1$

1	2	3
4	5	
7	8	6

STATE   $N_2$

1	2	3
4	5	6
7	8	

Goal state

# حل مسئله با جستجو / جستجوی ناآگاهانه

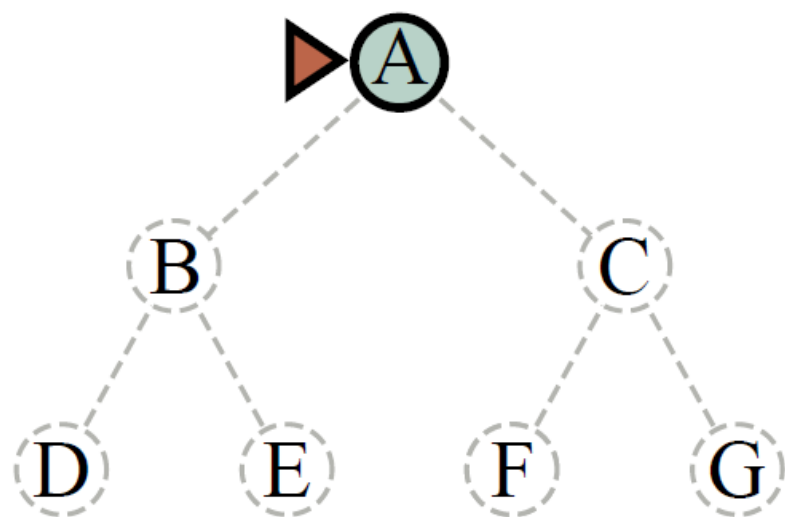
39

- ناآگاهی یعنی الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد.
- این الگوریتم‌ها فقط می‌توانند جانشین‌هایی (پسین) را تولید و هدف را از غیر هدف تشخیص دهند
- روش‌ها:
  - جست و جوی اول سطح (Breadth-first search)
  - جست و جوی هزینه یکنواخت (Uniform-cost search)
  - جست و جوی اول عمق (Depth First Search)
  - جست و جوی عمقی محدود (Depth-limited search)
  - جست و جوی عمیق شونده تکراری (Iterative deepening search)
  - جست و جوی دو طرفه (Bidirectional search)

# جستجوی ناآگاهانه / جستجوی اول سطح

40

- ابتدا ریشه گسترش می یابد.
- سپس، تمامی پسین های گره ریشه گسترش می یابند، بعد پسین های آنها و ....
- ابتدا باید همه گره های یک سطح از درخت جستجو گسترش یابند تا گرهی در سطح بعدی بتواند گسترش یابد.
- اجرا: لبه خالی یک صف FIFO است

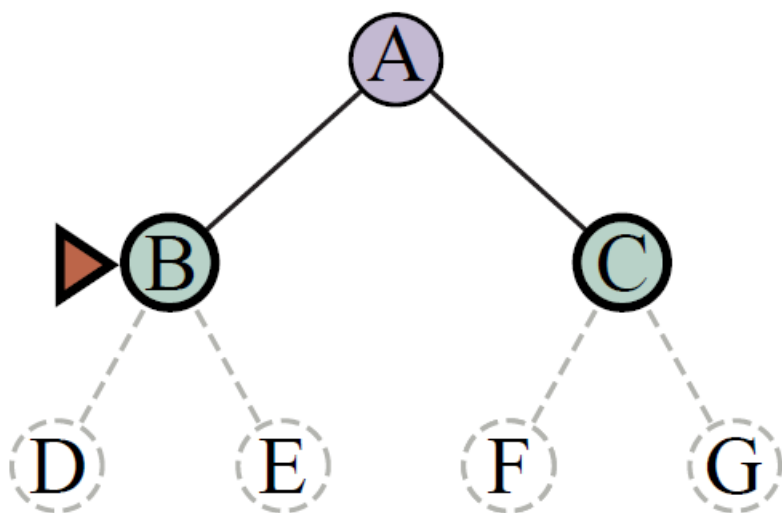




# جستجوی ناآگاهانه / جستجوی اول سطح

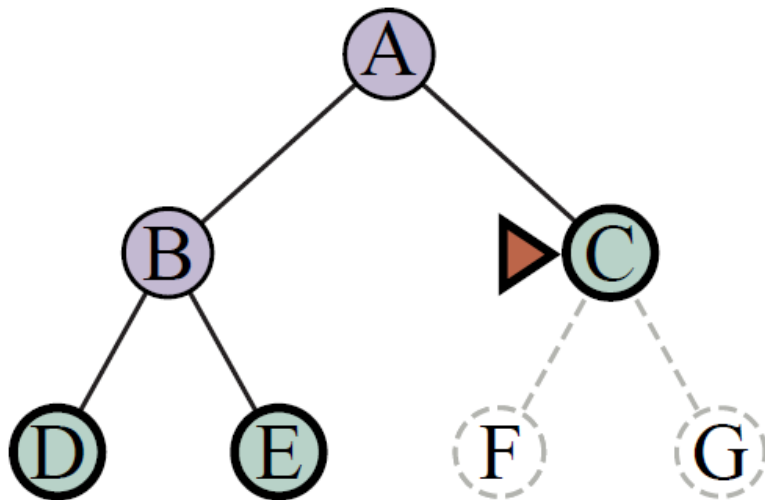
41

- ابتدا ریشه گسترش می یابد.
- سپس، تمامی پسین های گره ریشه گسترش می یابند، بعد پسین های آنها و ....
- ابتدا باید همه گره های یک سطح از درخت جستجو گسترش یابند تا گرهی در سطح بعدی بتواند گسترش یابد.
- اجرا: لبه خالی یک صف FIFO است



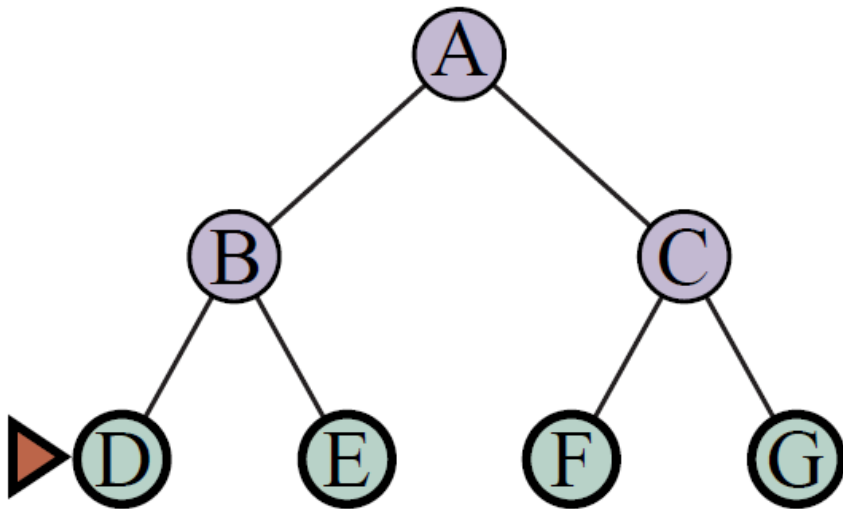
# جستجوی ناآگاهانه / جستجوی اول سطح

- ابتدا ریشه گسترش می یابد.
- سپس، تمامی پسین های گره ریشه گسترش می یابند، بعد پسین های آنها و ....
- ابتدا باید همه گره های یک سطح از درخت جستجو گسترش یابند تا گرهی در سطح بعدی بتواند گسترش یابد.
- اجرا: لبه خالی یک صف FIFO است

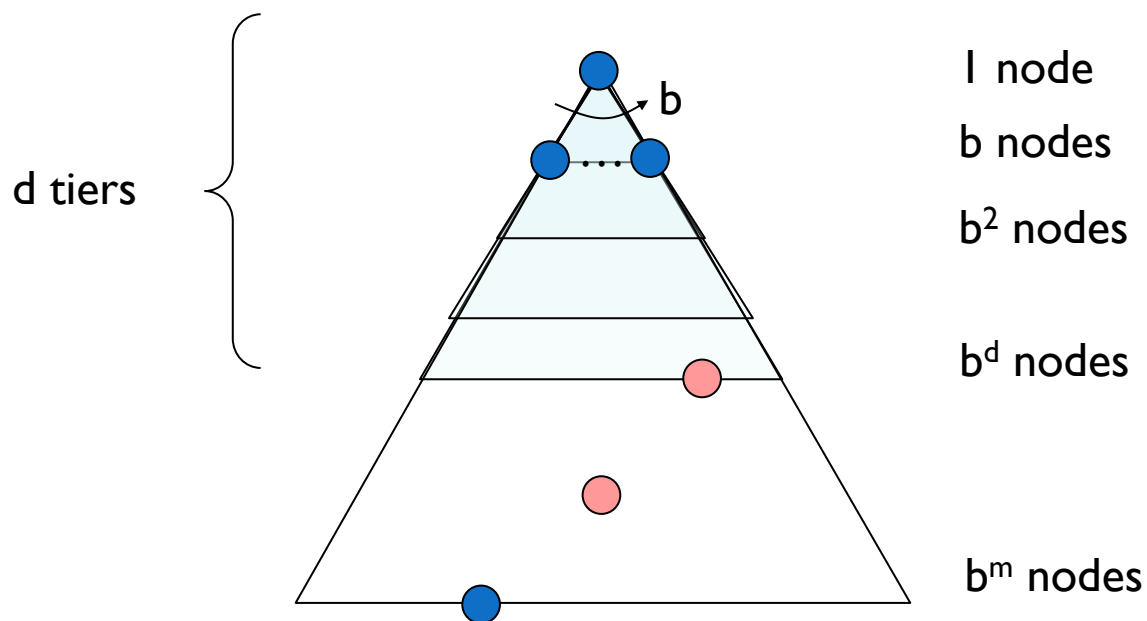


# جستجوی ناآگاهانه / جستجوی اول سطح

- ابتدا ریشه گسترش می یابد.
- سپس، تمامی پسین های گره ریشه گسترش می یابند، بعد پسین های آنها و ....
- ابتدا باید همه گره های یک سطح از درخت جستجو گسترش یابند تا گرهی در سطح بعدی بتواند گسترش یابد.
- اجرا: لبه خالی یک صف FIFO است



# جستجوی ناآگاهانه / جستجوی اول سطح



# جستجوی ناآگاهانه / جستجوی اول سطح...

45

□ کامل بودن:

□ بله، اگر کم عمق ترین گره هدف در عمق  $d$  باشد، جستجو بعد از گسترش گره های کم عمق تر نهایتاً آن را خواهد یافت. (مشروط به محدود بودن  $b$ )

□ بهینگی:

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

# جستجوی ناآگاهانه / جستجوی اول سطح...

46

□ پیچیدگی زمانی:

- فرض می کنیم هر حالت  $b$  پسین دارد. ریشه درخت جستجو در سطح اول  $b$  گره و هر گره در سطح بعد  $b$  گره دیگر تولید می کند (سطح ۲،  $b^2$  گره و ...
- فرض کنیم جواب در سطح  $d$  قرار دارد.
- در بدترین حالت همه گره های سطح  $d$  بجز یک گره (گره هدف) را گسترش داده ایم.
- بنابراین تعداد کل گره هایی که ایجاد شده :

$$b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

□ پیچیدگی فضایی:

- اگر همه گره ها در حافظه باقی بماند مشابه پیچیدگی زمانی خواهد بود.



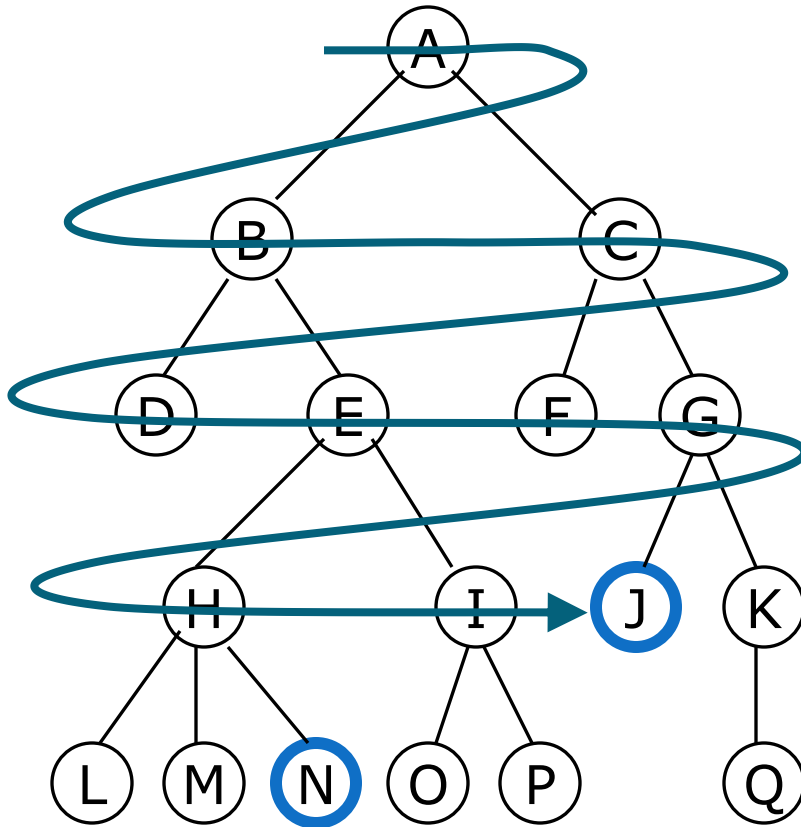
# جستجوی ناآگاهانه / جستجوی اول سطح...

47

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure  
  node  $\leftarrow$  NODE(problem.INITIAL)  
  if problem.IS-GOAL(node.STATE) then return node  
  frontier  $\leftarrow$  a FIFO queue, with node as an element  
  reached  $\leftarrow$  {problem.INITIAL}  
  while not IS-EMPTY(frontier) do  
    node  $\leftarrow$  POP(frontier)  
    for each child in EXPAND(problem, node) do  
      s  $\leftarrow$  child.STATE  
      if problem.IS-GOAL(s) then return child  
      if s is not in reached then  
        add s to reached  
        add child to frontier  
  return failure
```

# Breadth-first searching

48



- A **breadth-first** search (**BFS**) explores nodes nearest the root before exploring nodes further away
- For example, after searching **A**, then **B**, then **C**, the search proceeds with **D**, **E**, **F**, **G**
- Node are explored in the order **A B C D E F G H I J K L M N O P Q**
- **J** will be found before **N**

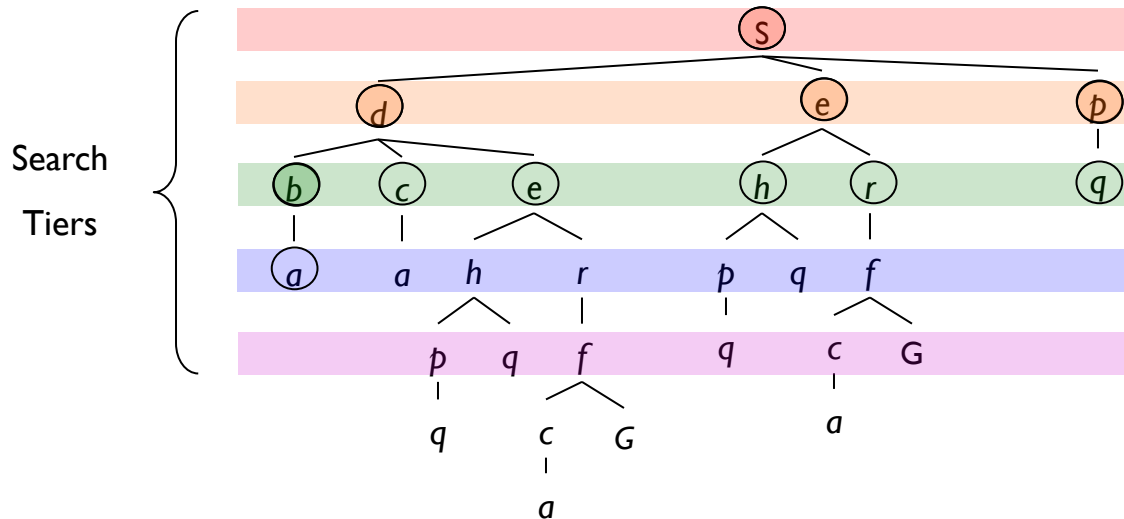
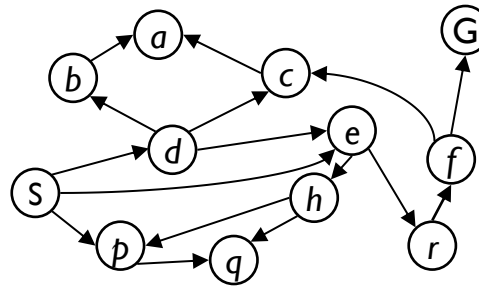


# Breadth-First Search

49

*Strategy: expand a shallowest node first*

*Implementation: Fringe is a FIFO queue*



# جستجوی ناآگاهانه / جستجوی هزینه یکنواخت

50

- این روش توسعه ای از روش اول سطح است.
  - به جای گسترش کم عمق ترین گره، گره با کمترین هزینه مسیر را بسط می دهد.
  - اگر هزینه مراحل مختلف یکسان باشد، معادل اول سطح خواهد بود.
  - به تعداد مراحل اهمیت نمی دهد، بلکه هزینه کل آنها را در نظر می گیرد.
- $$g(n) = \sum \text{costs of arcs}$$
- آیا این جستجو در حلقه تکرار بینهایت گیر می افتد؟  
□
  - پس، چه زمانی این روش کامل است؟  
□

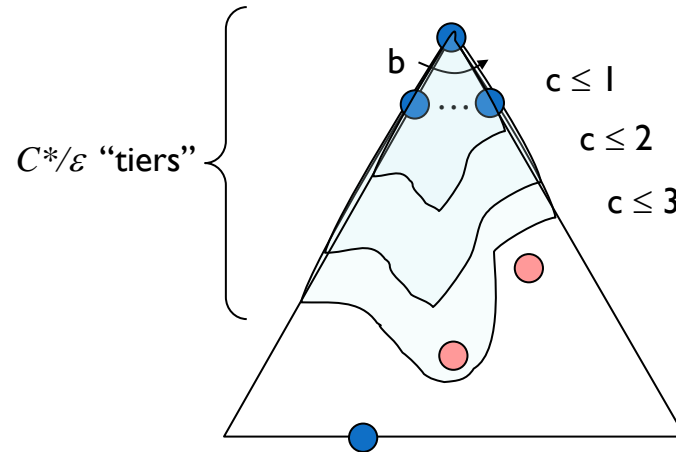
# جستجوی ناآگاهانه / جستجوی هزینه یکنواخت

51

- این روش توسعه ای از روش اول سطح است.
  - به جای گسترش کم عمق ترین گره، گره با کمترین هزینه مسیر را بسط می دهد.
  - اگر هزینه مراحل مختلف یکسان باشد، معادل اول سطح خواهد بود.
  - به تعداد مراحل اهمیت نمی دهد، بلکه هزینه کل آنها را در نظر می گیرد.
- $$g(n) = \sum \text{costs of arcs}$$
- آیا این جستجو در حلقه تکرار بینهایت گیر می افتد؟
  - اگر گرهی را گسترش دهد که هزینه مسیر آن صفر باشد و به همان حالت برگردد. (NoOp ←)
  - پس، چه زمانی این روش کامل است؟
  - هزینه هر مرحله بزرگ تر یا مساوی یک مقدار ثابت کوچک مانند  $\epsilon$  باشد.

# Uniform Cost Search (UCS)

52



# جستجوی ناآگاهانه / جستجوی هزینه یکنواخت...

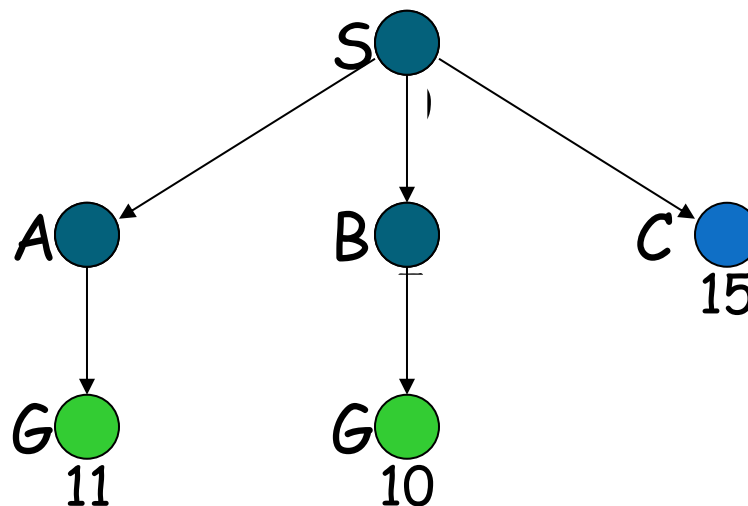
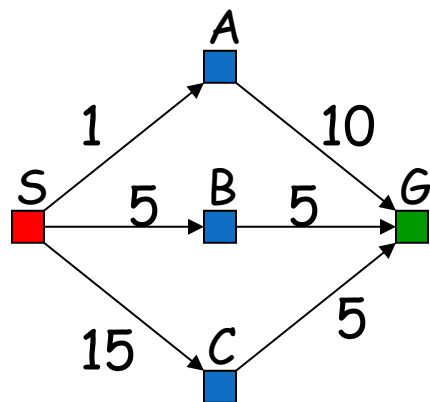
53

- لبه، صفی است که بر اساس افزایش هزینه مسیر مرتب شده است.
- کامل بودن:
- اگر هزینه هر مرحله بزرگ تر یا مساوی یک مقدار ثابت کوچک مانند  $\epsilon$  باشد.
- بهینه بودن:
- اگر کامل باشد.
- پیچیدگی زمانی و فضایی :
- فرض کنیم  $C^*$  هزینه راه حل بهینه باشد
- فرض کنیم هر اقدامی حداقل هزینه  $\epsilon$  دارد.
- در بدترین حالت  $O(b^{C^*/\epsilon})$

# جستجوی ناآگاهانه / جستجوی هزینه یکنواخت

54

مثال:

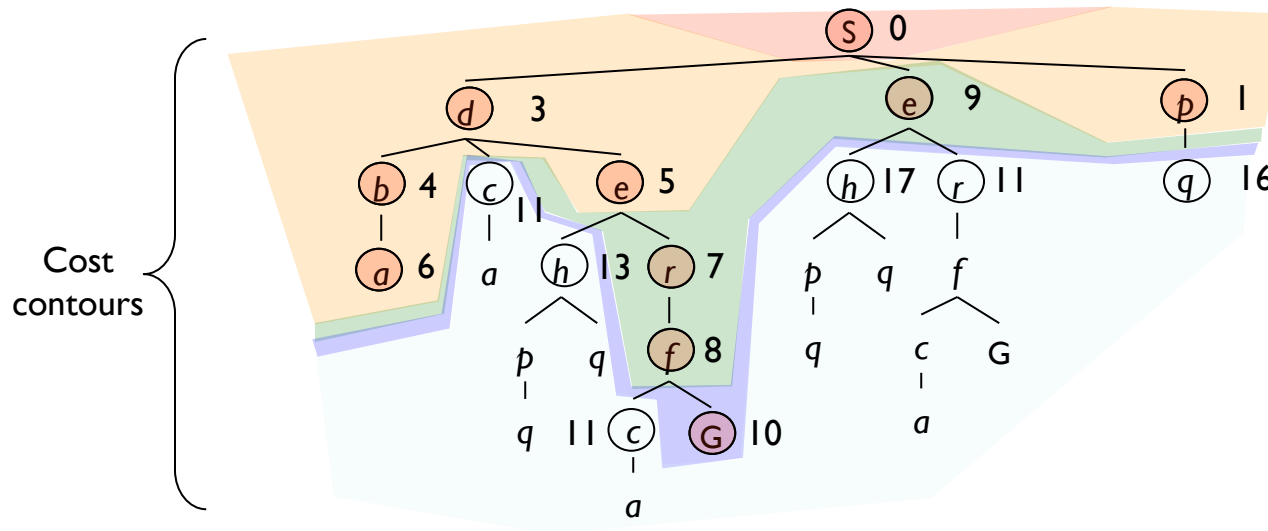
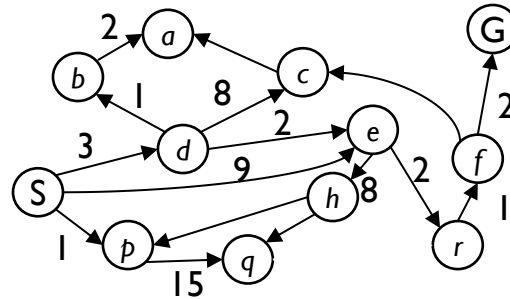


# Uniform Cost Search

55

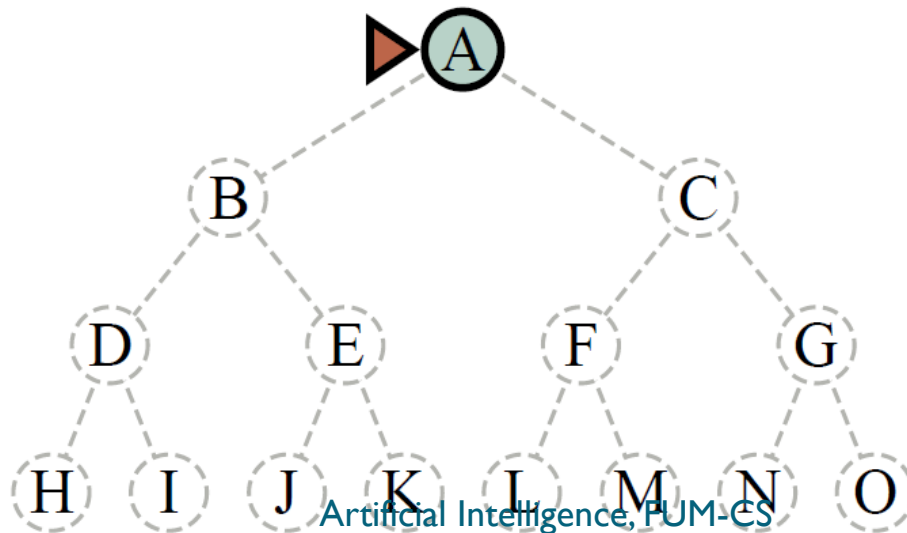
Strategy: expand a cheapest node first:

Fringe is a priority queue  
(priority: cumulative cost)



# جستجوی ناآگاهانه / جستجوی اول عمق

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.

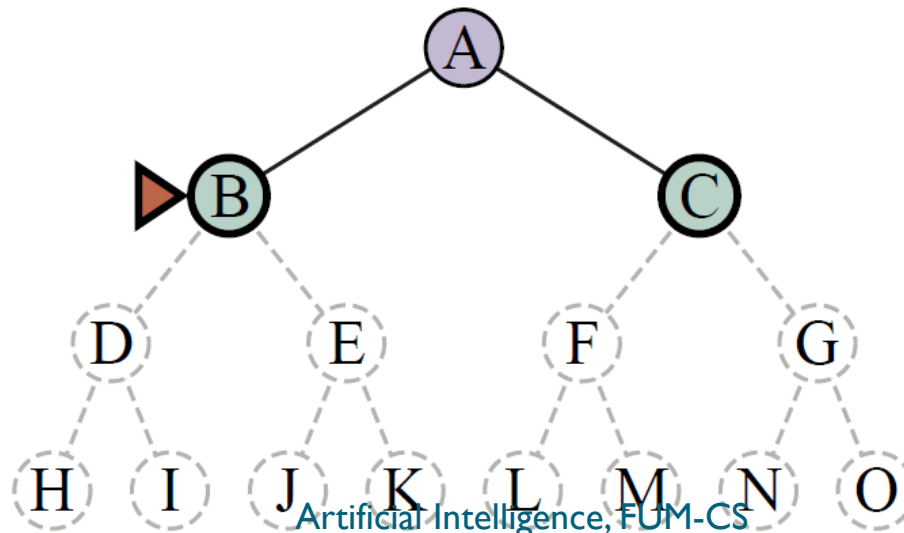




# جستجوی ناآگاهانه / جستجوی اول عمق

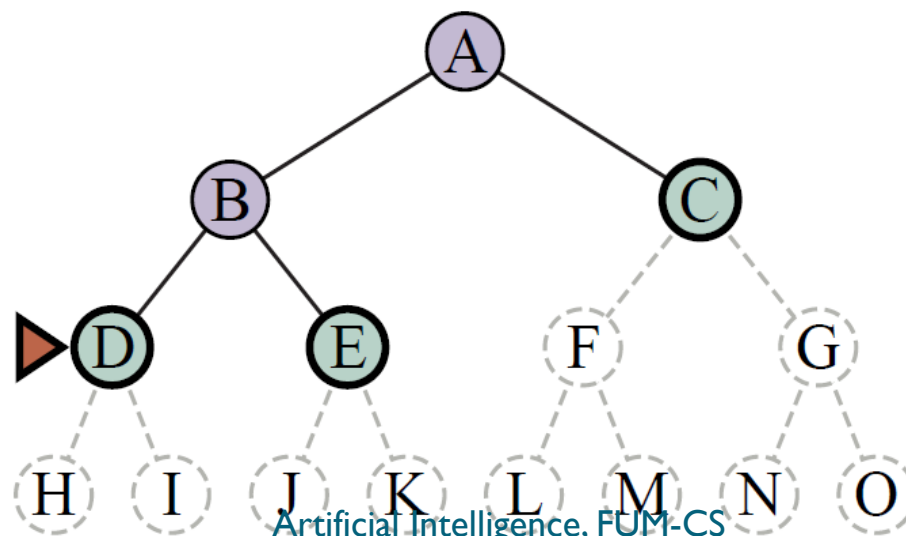
57

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



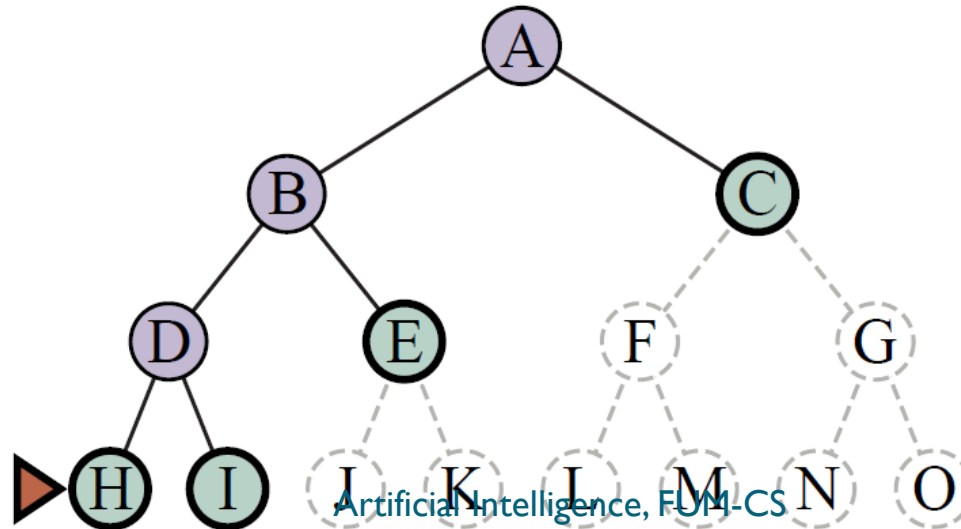
# جستجوی ناآگاهانه / جستجوی اول عمق

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



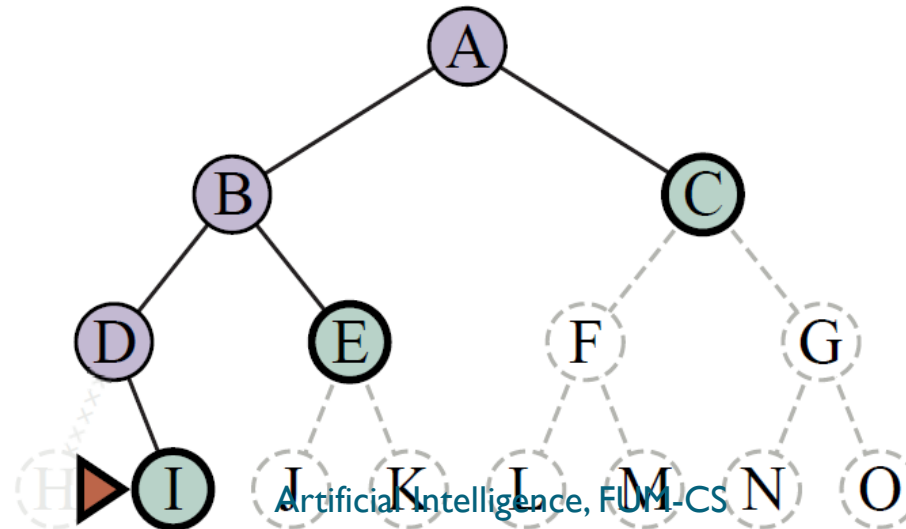
# جستجوی ناآگاهانه / جستجوی اول عمق

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق

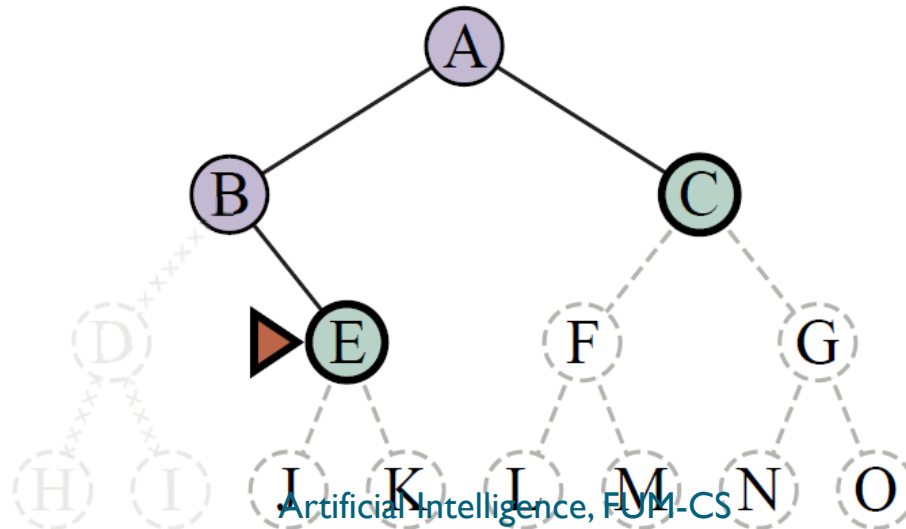
- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق

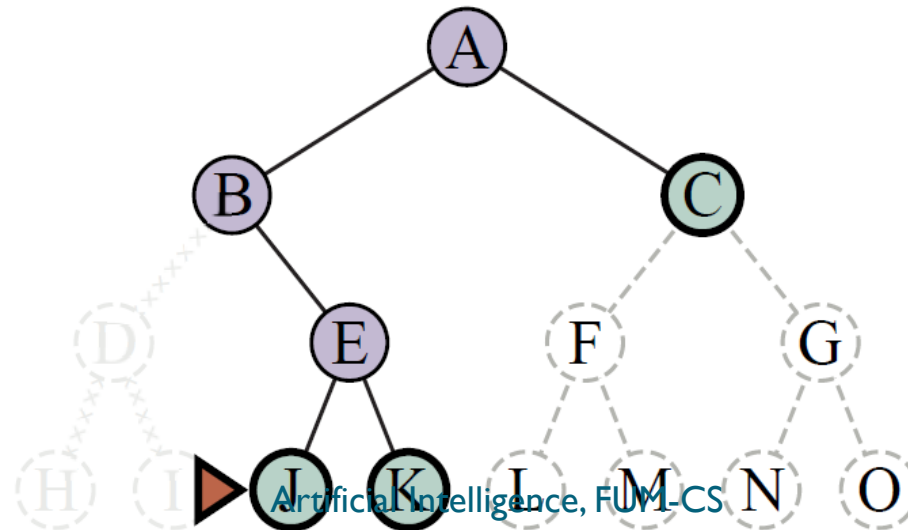
61

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



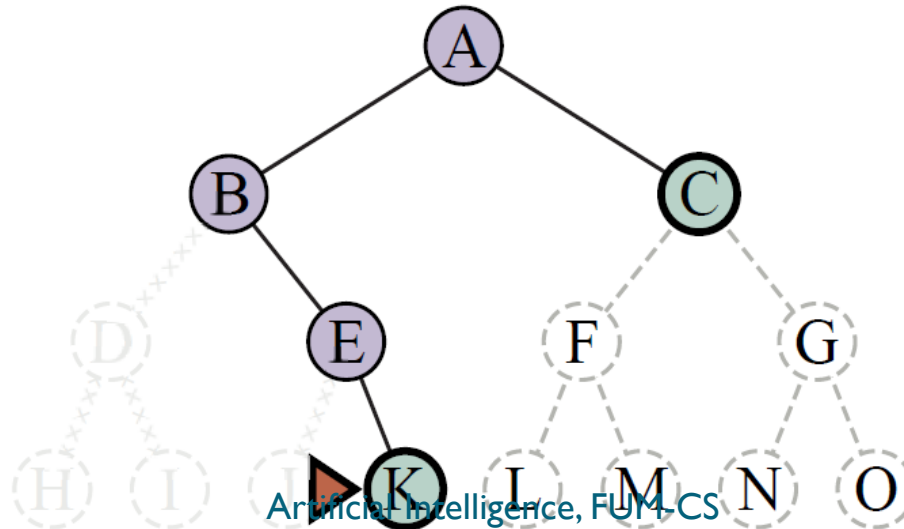
# جستجوی ناآگاهانه / جستجوی اول عمق

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق

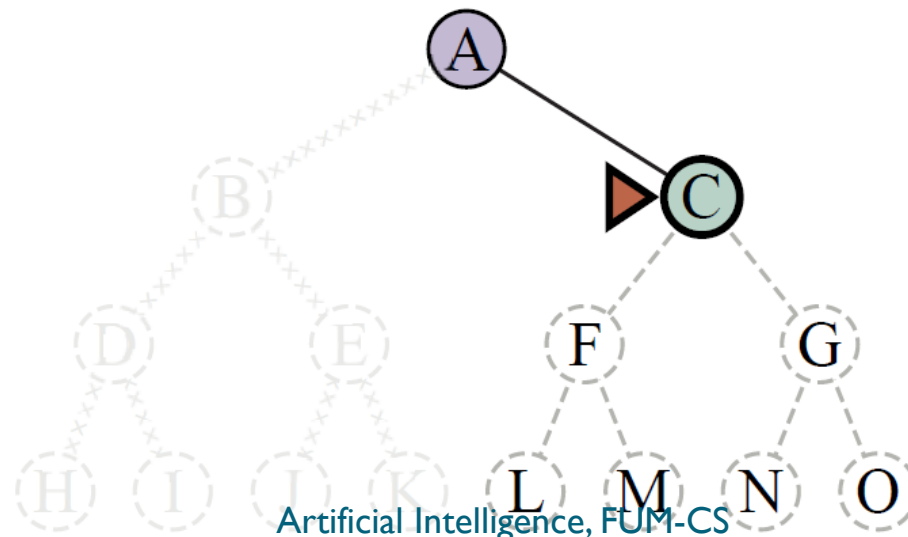
- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق

64

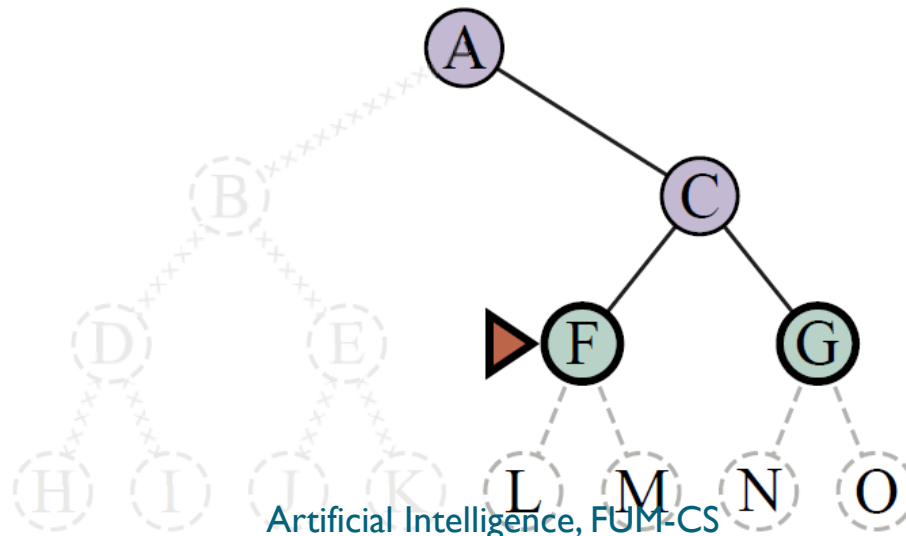
- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.





# جستجوی ناآگاهانه / جستجوی اول عمق

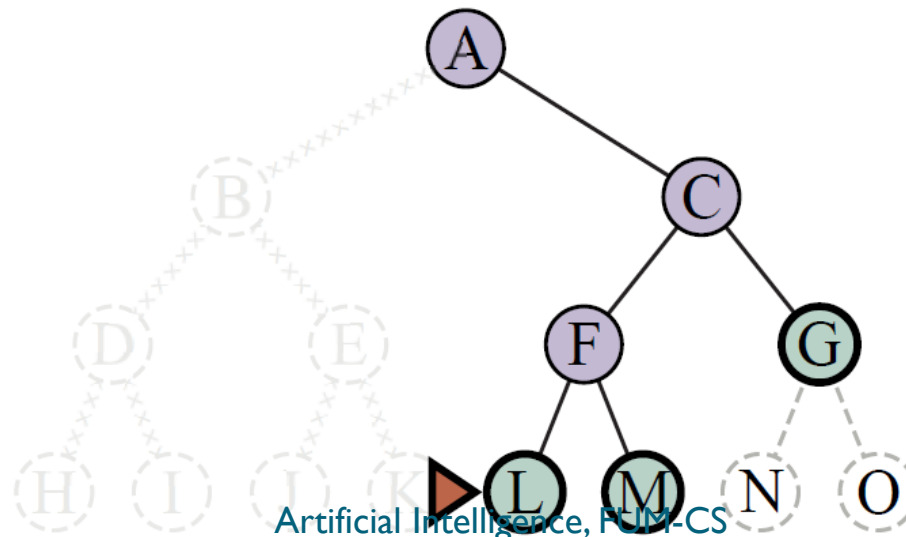
- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق

66

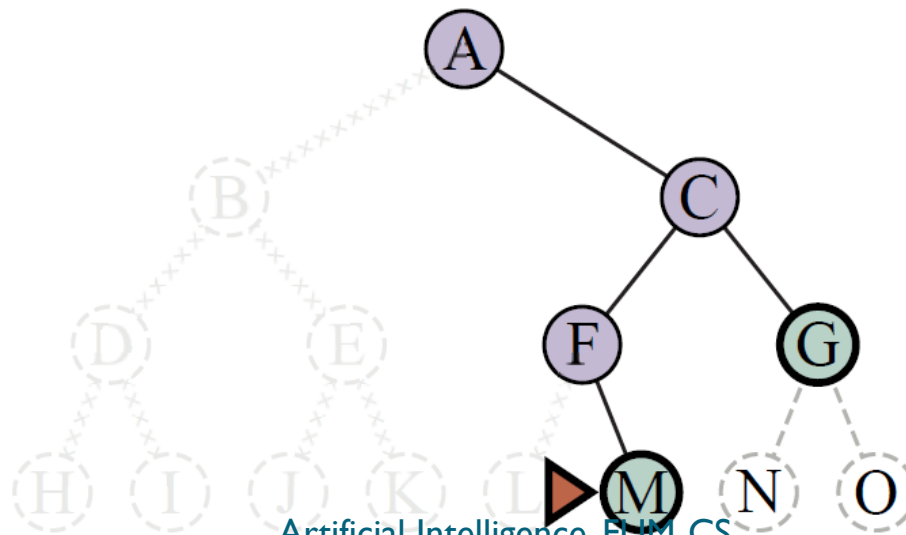
- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



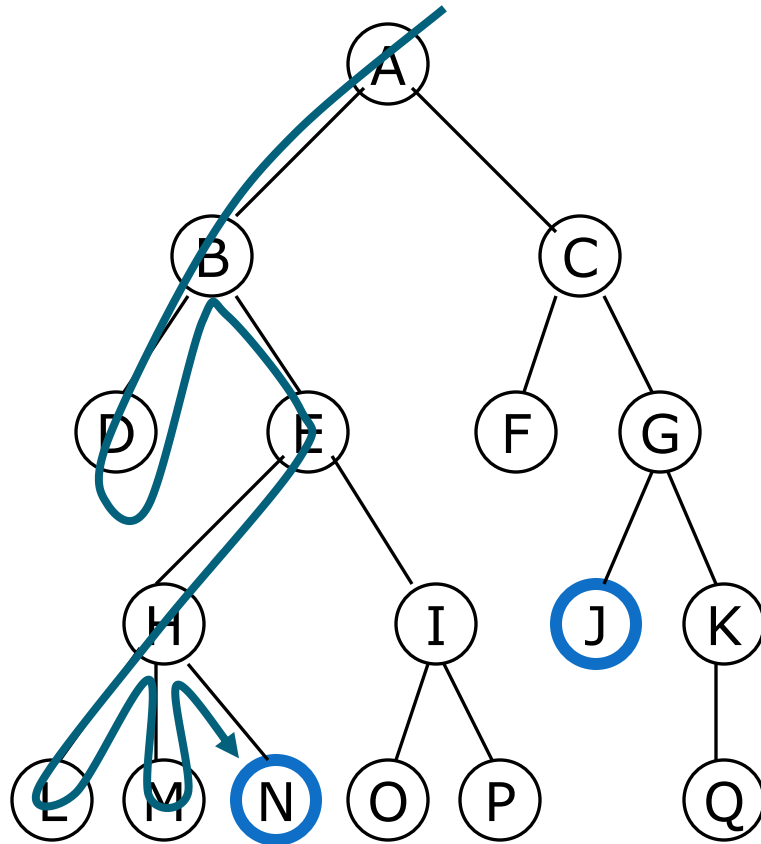
# جستجوی ناآگاهانه / جستجوی اول عمق

67

- عمیق ترین گره در لبه فعلی درخت را گسترش می دهد.
- لبه در اینجا صف آخرین ورودی اولین خروجی (LIFO) یا پشته می باشد.
- می توان به صورت بازگشتی نیز آن را پیاده سازی نمود.



# جستجوی ناآگاهانه / جستجوی اول عمق



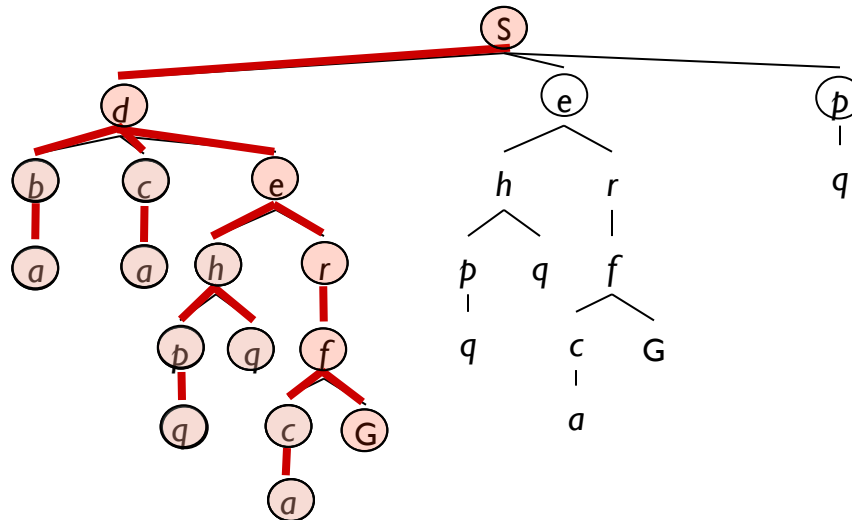
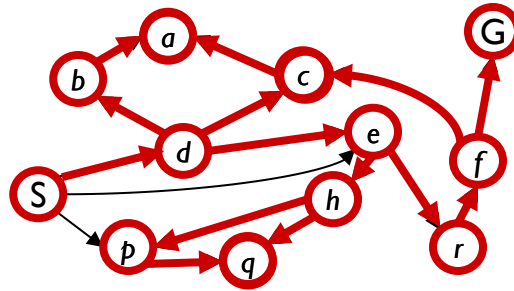
- A **depth-first** search (DFS) explores a path all the way to a leaf before **backtracking** and exploring another path
- For example, after searching **A**, then **B**, then **D**, the search backtracks and tries another path from **B**
- Node are explored in the order **A B D E H L M N I O P C F G J K Q**
- **N** will be found before **J**

# جستجوی ناآگاهانه / جستجوی اول عمق

69

Strategy: expand a  
deepest node first

Implementation:  
Fringe is a LIFO stack



# جستجوی ناآگاهانه / جستجوی اول عمق

70

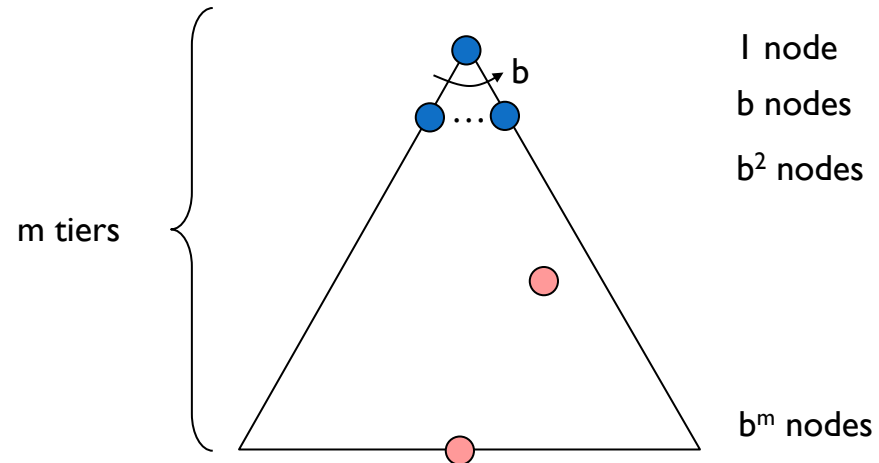
- ❑ Complete: Guaranteed to find a solution if one exists?
- ❑ Optimal: Guaranteed to find the least cost path?
- ❑ Time complexity?
- ❑ Space complexity?

## ❑ Cartoon of search tree:

- $b$  is the branching factor
- $m$  is the maximum depth
- solutions at various depths

## ❑ Number of nodes in entire tree?

- $1 + b + b^2 + \dots + b^m = O(b^{m+1})$



# جستجوی ناآگاهانه / جستجوی اول عمق

71

□ کامل بودن :

□ خیر، مگر اینکه فضای حالت متناهی باشد و حلقه ای وجود نداشته باشد.

□ بهینه بودن :

□ خیر

□ پیچیدگی زمانی:  $O(b^m)$

□ اگر  $m$  خیلی بزرگ تر از  $d$  (عمق راه حل بهینه) باشد، بسیار بد خواهد بود.

□ اگر تعداد جوابها زیاد باشد، از جستجوی اول سطح سریعتر است.

□ پیچیدگی فضایی:  $O(bm)$

# جستجوی ناآگاهانه / جستجوی اول عمق

72

□ معایب:

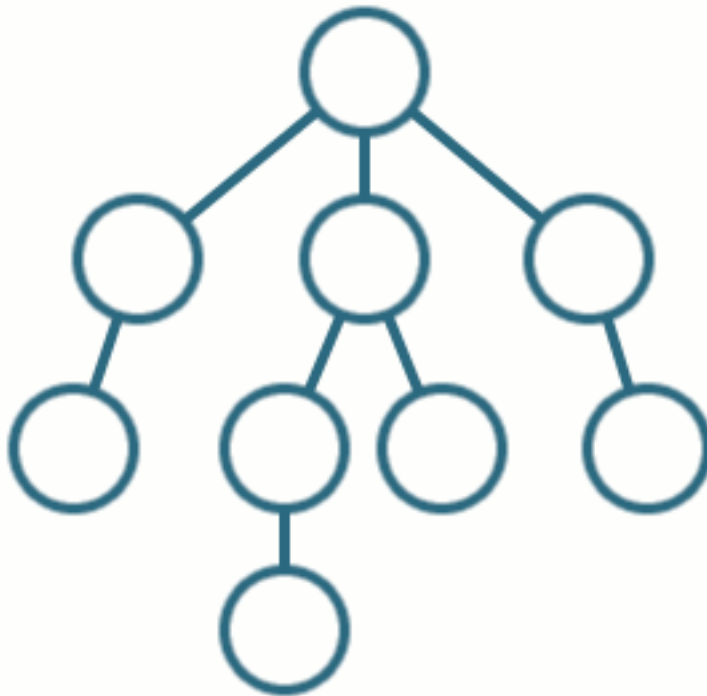
- اگر مسیری را اشتباه طی کند، هنگام پائین رفتن گیر می کند.
- جستجوی عمقی نه کامل و نه بهینه است.
- در درخت‌های با عمق نامحدود و بزرگ این استراتژی کار نمی کند.



# مقایسه جستجوی اول عمق و اول سطح

73

DFS



BFS



# جستجوی ناآگاهانه / جستجوی عمق محدود

74

- مشابه جستجوی اول عمق، با محدودیت عمق  $l$ 
  - یعنی گره هایی که در عمق  $l$  هستند، پسینی ندارند.
- مشکل عمق نامحدود در روش اول عمق را حل می کند.
- جستجوی اول عمق، حالت خاصی از این روش است که در آن  $l = \infty$
- اگر  $l < d$  باشد، باعث ناتمام بودن مسئله میشود. (عمق جواب بیشتر از محدودیت تعریف شده است). روش کامل نیست.
- اگر  $l > d$  باشد، بهینه نخواهد بود.
- پیچیدگی زمانی:  $O(b^l)$
- پیچیدگی فضایی:  $O(bl)$

# جستجوی ناآگاهانه/ جستجوی اول عمق عمیق شونده تکراری

- این روش اغلب همراه با جستجوی اول عمق استفاده می شود و بهترین محدودیت عمق را می یابد.
- این کار با افزایش تدریجی محدودیت (اول ۰، بعد ۱، بعد ۲، ...) تا زمانی که یک هدف پیدا شود ادامه می یابد.
- هدف زمانی پیدا می شود که محدودیت عمق به  $d$  یعنی کم عمق ترین گره هدف برسد.
- این روش مزایای جستجوی اول عمق و اول سطح را باهم دارد.
- مانند اول عمق از حافظه کمی استفاده می کند ( $O(bd)$ ).
- مانند اول سطح اگر ضریب انشعاب محدود باشد، کامل و وقتی هزینه مسیر تابع غیرکاهشی از عمق گره باشد، بهینه است.



**function**

**ITERATIVE\_DEEPENING\_SEARCH**(*problem*)

**return** a solution or failure

**inputs:** *problem*

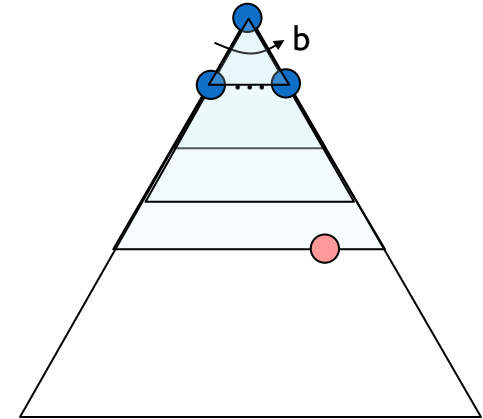
**for** *depth*  $\leftarrow 0$  to  $\infty$  **do**

*result*  $\leftarrow$  DEPTH-LIMITED\_SEARCH(*problem*, *depth*)

**if** *result*  $\neq$  *cutoff* **then return** *result*

*cutoff*, نشان دهنده عدم وجود راه حل در محدوده عمق

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - ▣ Run a DFS with depth limit 1. If no solution...
  - ▣ Run a DFS with depth limit 2. If no solution...
  - ▣ Run a DFS with depth limit 3. ....
  
- Isn't that wastefully redundant?
  - ▣ Generally most work happens in the lowest level searched, so not so bad!



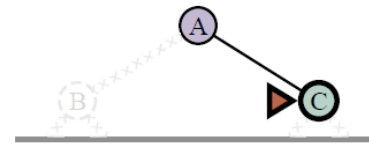
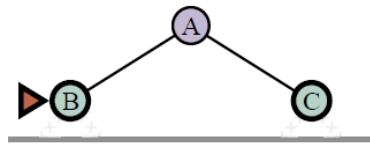
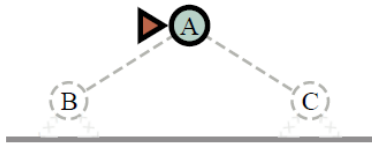
Limit=0 □



# جستجوی ناآگاهانه / جستجوی اول عمق عمیق شونده تکراری

79

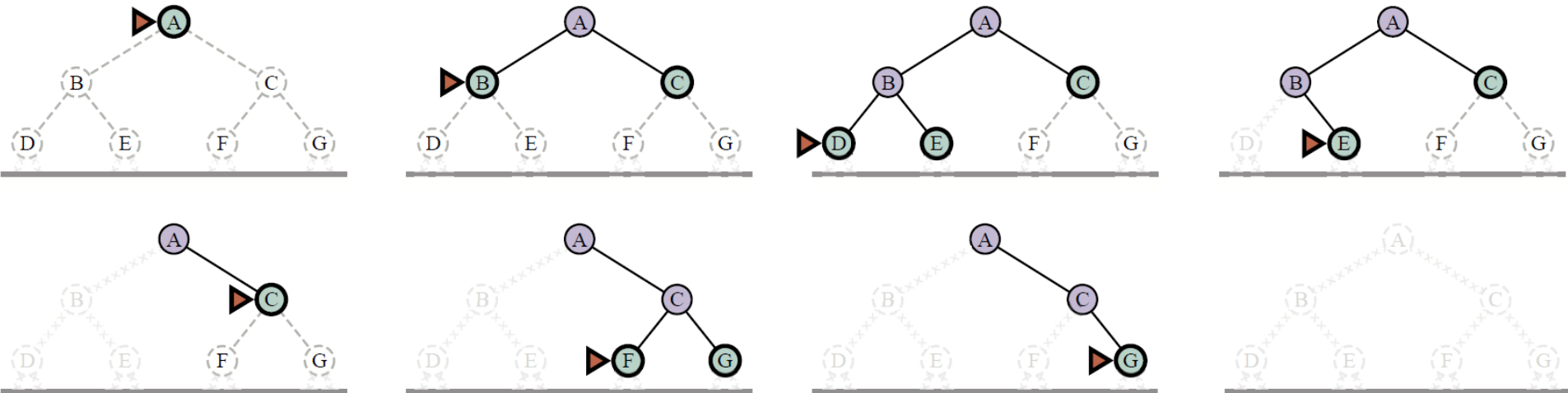
Limit=1 □



# جستجوی ناآگاهانه / جستجوی اول عمیق شونده تکراری

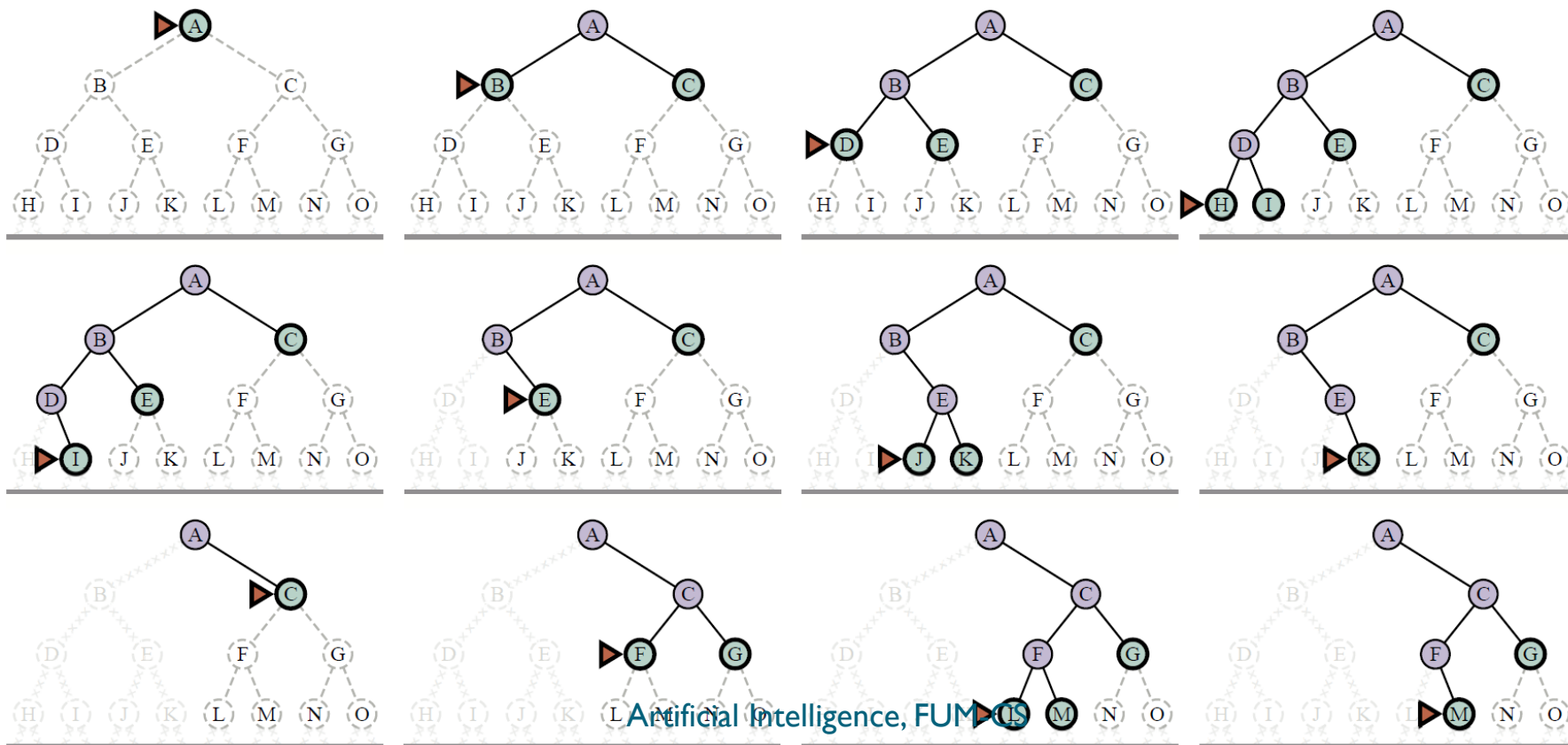
80

Limit=2 □





Limit=3 □





# جستجوی ناآگاهانه / جستجوی اول عمق عمیق شونده تکراری

82

- در این روش، گره‌های سطوح پائینی یک بار بسط داده می‌شوند، آنهایی که یک سطح بالاتر قرار دارند دوبار بسط داده می‌شوند و الی آخر تا به فرزندان ریشه درخت جستجو برسد، که  $d$  بار بسط داده می‌شوند.
- بنابراین، تعداد کل گره‌های تولید شده در مقایسه با روش اول سطح برابر است با

Node generation:  $N(IDS) = (d)b + (d-1)b^2 + \dots + (1)b^d$

$$N(BFS) = b + b^2 + \dots + b^d + (b^{d+1} - b)$$

level d: once

level d-1: 2

level d-2: 3

...

level 2: d-1

level 1: d

$$b=10, d=5$$

$$N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450$$

$$N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$$



# جستجوی ناآگاهانه / جستجوی اول عمق عمیق شونده تکراری

83

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure  
  for depth = 0 to  $\infty$  do  
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)  
    if result  $\neq$  cutoff then return result
```

```
function DEPTH-LIMITED-SEARCH(problem,  $\ell$ ) returns a node or failure or cutoff  
  frontier  $\leftarrow$  a LIFO queue (stack) with NODE(problem.INITIAL) as an element  
  result  $\leftarrow$  failure  
  while not IS-EMPTY(frontier) do  
    node  $\leftarrow$  POP(frontier)  
    if problem.IS-GOAL(node.STATE) then return node  
    if DEPTH(node) >  $\ell$  then  
      result  $\leftarrow$  cutoff  
    else if not IS-CYCLE(node) do  
      for each child in EXPAND(problem, node) do  
        add child to frontier  
  return result
```

□ چرا روش جستجوی عیق شونده تکراری سریع تر از روش جستجوی اول سطح است؟

□ در روش جستجوی اول سطح وقتی جواب در عمق  $d$  است، چند گره در عمق  $d+1$  تولید می شود اما در روش عیق شونده تکراری، این کار را انجام نمی دهد. بنابراین اگرچه حالتها تکرار می شوند اما سریعتر است.

□ هنگامی که فضای جستجو بزرگ و عمق راه حل نامعلوم باشد، این روش بهترین روش جستجوی ناآگاهانه است.

- کامل بودن : اگر مسیر نامحدود نداشته باشد، کامل است
- پیچیدگی زمانی :  $O(b^d)$
- پیچیدگی فضایی :  $O(bd)$
- بهینه بودن : اگر هزینه گام ها ثابت باشد.

□ تمرین : درباره روش جستجوی طولانی کننده تکراری ( Iterative Lengthening Search ) تحقیق نموده و چهار خاصیت بالا را برای آن مشخص کنید.

# جستجوی ناآگاهانه / جستجوی دو طرفه

□ دو جستجو همزمان انجام می شود:

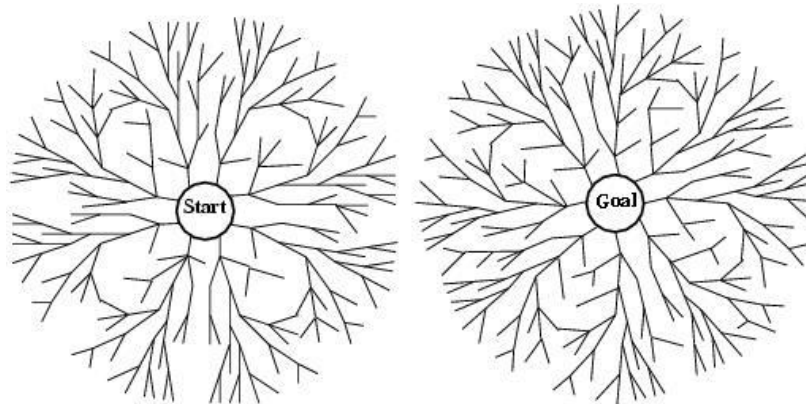
□ از حالت اولیه به سمت جلو

□ از هدف به سمت عقب

□ هنگامی که این دو جستجو به هم برسند، متوقف می شود.

□ چرا این روش پیشنهاد شده است؟

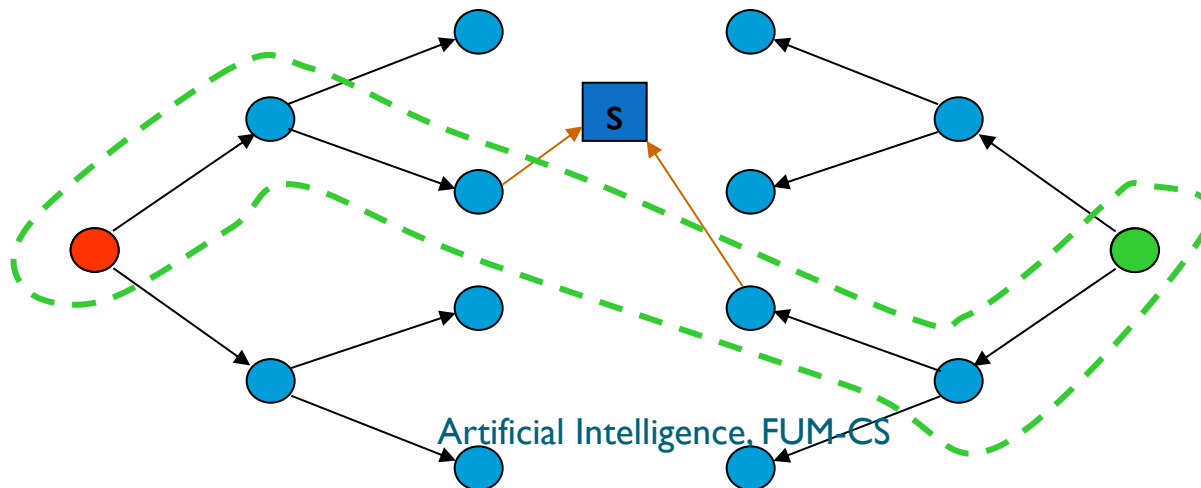
$$b^{d/2} + b^{d/2} < b^d$$



# جستجوی ناآگاهانه / جستجوی دو طرفه

87

- روش کار : یکی یا هر دو جستجو قبل از گسترش هر گره ای از مجموعه لبه، بررسی میکند که آیا آن گره در لبه درخت جستجوی دیگر قرار دارد یا خیر؛ اگر چنین بود، راه حلی پیدا شده است.
- پیچیدگی زمانی و فضایی :  $O(b^{d/2})$
- کامل و بهینه بودن: اگر هر دو جستجو از نوع اول سطح باشد.





# جستجوی ناآگاهانه / جستجوی دو طرفه

88

```
function BIBF-SEARCH( $problem_F, f_F, problem_B, f_B$ ) returns a solution node, or failure  
   $node_F \leftarrow \text{NODE}(problem_F.INITIAL)$  // Node for a start state  
   $node_B \leftarrow \text{NODE}(problem_B.INITIAL)$  // Node for a goal state  
   $frontier_F \leftarrow$  a priority queue ordered by  $f_F$ , with  $node_F$  as an element  
   $frontier_B \leftarrow$  a priority queue ordered by  $f_B$ , with  $node_B$  as an element  
   $reached_F \leftarrow$  a lookup table, with one key  $node_F.STATE$  and value  $node_F$   
   $reached_B \leftarrow$  a lookup table, with one key  $node_B.STATE$  and value  $node_B$   
   $solution \leftarrow failure$   
  while not TERMINATED( $solution, frontier_F, frontier_B$ ) do  
    if  $f_F(\text{TOP}(frontier_F)) < f_B(\text{TOP}(frontier_B))$  then  
       $solution \leftarrow \text{PROCEED}(F, problem_F, frontier_F, reached_F, reached_B, solution)$   
    else  $solution \leftarrow \text{PROCEED}(B, problem_B, frontier_B, reached_B, reached_F, solution)$   
  return  $solution$ 
```



# جستجوی ناآگاهانه / جستجوی دو طرفه

89

```
function PROCEED(dir, problem, frontier, reached, reached2, solution) returns a solution
    // Expand node on frontier; check against the other frontier in reached2.
    // The variable “dir” is the direction: either F for forward or B for backward.
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
        s ← child.STATE
        if s not in reached or PATH-COST(child) < PATH-COST(reached[s]) then
            reached[s] ← child
            add child to frontier
            if s is in reached2 then
                solution2 ← JOIN-NODES(dir, child, reached2[s]))
                if PATH-COST(solution2) < PATH-COST(solution) then
                    solution ← solution2
    return solution
```



# جستجوی ناآگاهانه / مقایسه الگوریتم های مختلف

90

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>1</sup>	Yes <sup>1,2</sup>	No	No	Yes <sup>1</sup>	Yes <sup>1,4</sup>
Optimal cost?	Yes <sup>3</sup>	Yes	No	No	Yes <sup>3</sup>	Yes <sup>3,4</sup>
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

**Figure 3.15** Evaluation of search algorithms.  $b$  is the branching factor;  $m$  is the maximum depth of the search tree;  $d$  is the depth of the shallowest solution, or is  $m$  when there is no solution;  $\ell$  is the depth limit. Superscript caveats are as follows: <sup>1</sup> complete if  $b$  is finite, and the state space either has a solution or is finite. <sup>2</sup> complete if all action costs are  $\geq \epsilon > 0$ ; <sup>3</sup> cost-optimal if action costs are all identical; <sup>4</sup> if both directions are breadth-first or uniform-cost.