دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

حل مساله با جستجو

«هوش مصنوعی: رهیافتی نوین»، فصل ۳

مدرس: آرش عبدی هجراندوست

نیم‌سال دوم ۱۴۰۱-۱۴۰۲

- یک جسله عقب افتاده ایم
- نظر مثبتتان در مورد ۵ شنبه ساعت ۹:۳۰ جلسه فوق العاده
- تا بشود پروژه تکاملی را قبل از عید داد

به تماشا سوگند
و به آغاز کلام
و به پرواز کبوتر از ذهن

واژه ای در قفس

است.

"If you don't design your own life plan,
chances are you'll fall into someone else's plan"

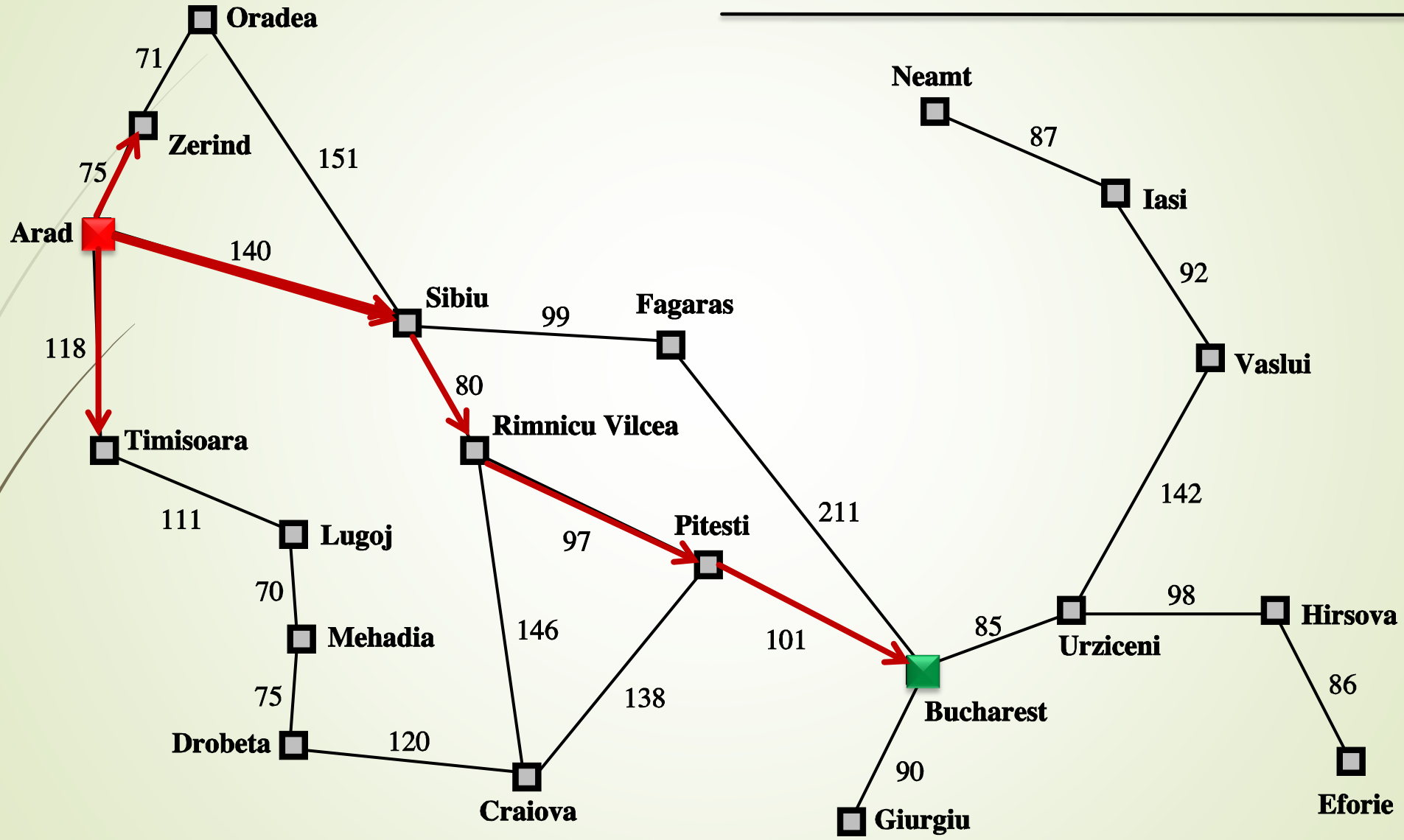
**««اگر خودتان طرح زندگی خود را طراحی نکنید،
احتمالا در طرحی که دیگران کشیده‌اند می‌افتید»**

Jim Rohn

عامل حل مسئله

انتظار می‌رود، عامل‌های هوشمند معیار کارایی خود را به حداکثر برسانند. اگر عامل بتواند یک هدف را انتخاب کند و تصمیم بگیرد که به آن برسد، ماکزیمم کردن معیار کارایی ساده است.

مثال: نقشه جاده رومانی



گام‌های کلی در حل مسئله

- گام اول: فرموله‌سازی هدف
  - فرایند تصمیم‌گیری در مورد انتخاب هدف بعدی براساس وضعیت فعلی و معیار کارایی عامل
- گام دوم: فرموله‌سازی مسئله
  - فرایند تصمیم‌گیری در مورد انتخاب نوع حالات و اقدام‌ها با توجه به هدف تعیین‌شده در مرحله قبل
- گام سوم: جستجو
  - فرایند جستجو برای راه‌حل (دنباله‌ای از اقدام‌ها که عامل را از حالت اولیه به حالت هدف می‌رساند.)
- گام چهارم: اجرا
  - انجام اقدام‌های پیشنهادشده توسط راه‌حل

فرموله‌سازی مسائل

- جهان واقعی کاملاً پیچیده است.
  - فرایند حذف جزئیات از یک بازنمایی برای حل مسئله را تجرید یا انتزاع (abstraction) می‌گویند.
- حالت (انتزاعی) = مجموعه‌ای از حالات واقعی
  - حذف "همراهان در سفر"، "محل‌های استراحت" و ...
- عمل (انتزاعی) = ترکیبی پیچیده از اعمال واقعی
  - حذف "فرمان را به اندازه سه درجه به سمت چپ بچرخان"، "آهسته رفتن به دلیل قوانین اجباری پلیس" و ...
  - فقط میگوییم مثلا از شهر الف برو به شهر ب
- سطح مناسب انتزاع؟
  - معتبر بودن (valid): بتوانیم هر راه‌حل انتزاعی را به یک راه‌حل با جزئیات بیشتر (غیرانتزاعی تر) بسط دهیم.
  - مفید بودن (useful): انجام هر یک از اعمال در راه‌حل آسان‌تر از مسئله‌ی اصلی باشد.

فرضیات

- ویژگی‌هایی که فعلا برای محیط در نظر خواهیم گرفت:
  - شناخته‌شده (Known)
    - داشتن یک نقشه از راه‌ها برای مسائل مسیریابی
  - کاملا قابل مشاهده (Observable)
    - عامل همیشه وضعیت فعلی را بداند.
  - قطعی (Deterministic)
    - حاصل انجام یک عمل در یک وضعیت دقیقا یک وضعیت مشخص باشد.
  - گسسته (Discrete)

مسائل و راه‌حل‌های خوش تعریف

❖ یک مسئله، به‌صورت رسمی با پنج مؤلفه تعریف می‌شود:

۱- حالت اولیه: In(Arad)

۲- اعمال: ACTIONS(s) مجموعه اعمالی را که در وضعیت s می‌تواند انجام گیرد برمی‌گرداند.

❖ برای مثال: $ACTIONS(In(Arad))=\{Go(Sibiu), Go(Timisoara), Go(Zerind)\}$

۳- مدل انتقال: RESULT(s,a) وضعیت حاصل از انجام عمل a در وضعیت s را برمی‌گرداند.

❖ برای مثال: $RESULT(In(Arad),Go(Zerind))=In(Zerind)$

❖ $In(Zerind)$ پسین یا successor نامیده می‌شود.

← فضای حالت: مجموعه‌ی همه‌ی حالت قابل رسیدن از حالت اولیه

← گراف جهت‌دار (نودها: وضعیت‌ها، یال‌ها: اعمال)

مسائل و راه‌حل‌های خوش تعریف

۳- آزمون هدف: $GOALTEST(s)$ تعیین می‌کند که آیا S وضعیت هدف است یا خیر.

- صریح: برای مثال s='at Bucharest'
- ضمنی: برای مثال checkmate(s)

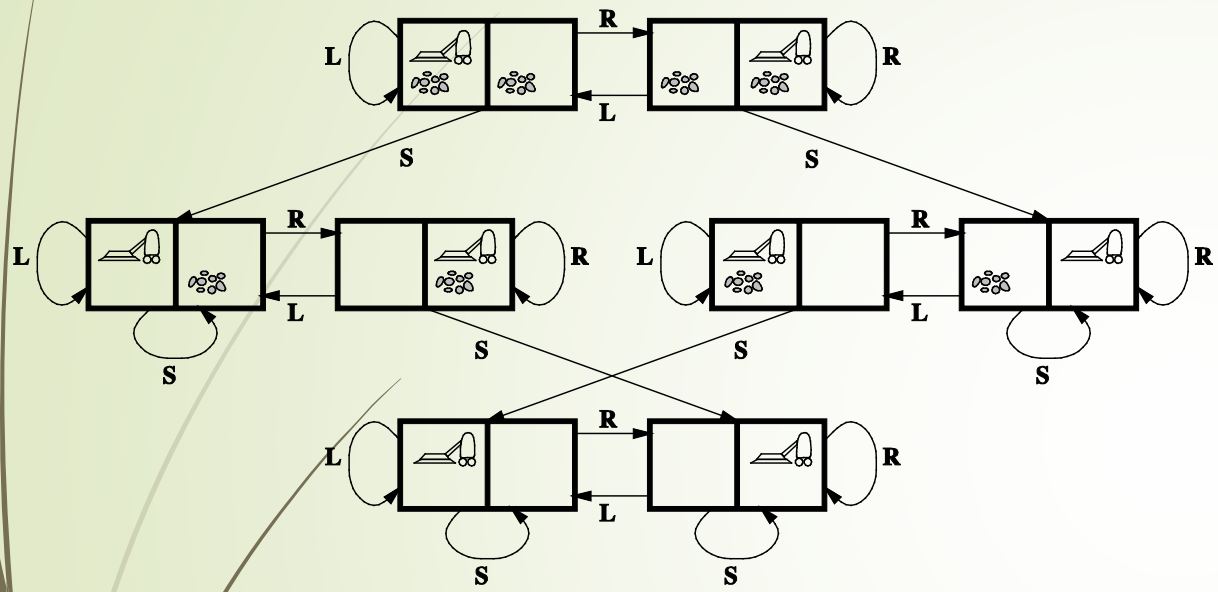
۴- تابع هزینه مسیر: یک هزینه عددی را به هر مسیر انتساب می‌دهد. عامل حل مسئله تابع هزینه‌ای را انتخاب می‌کند که معیار کارایی خودش را منعکس کند.

- برای مثال: مجموع فاصله‌ها، تعداد اعمال اجراشده و ...
- $c(s,a,s')\geq 0$ هزینه‌ی گام است.

← کیفیت راه‌حل با تابع هزینه‌ی مسیر اندازه‌گیری می‌شود.

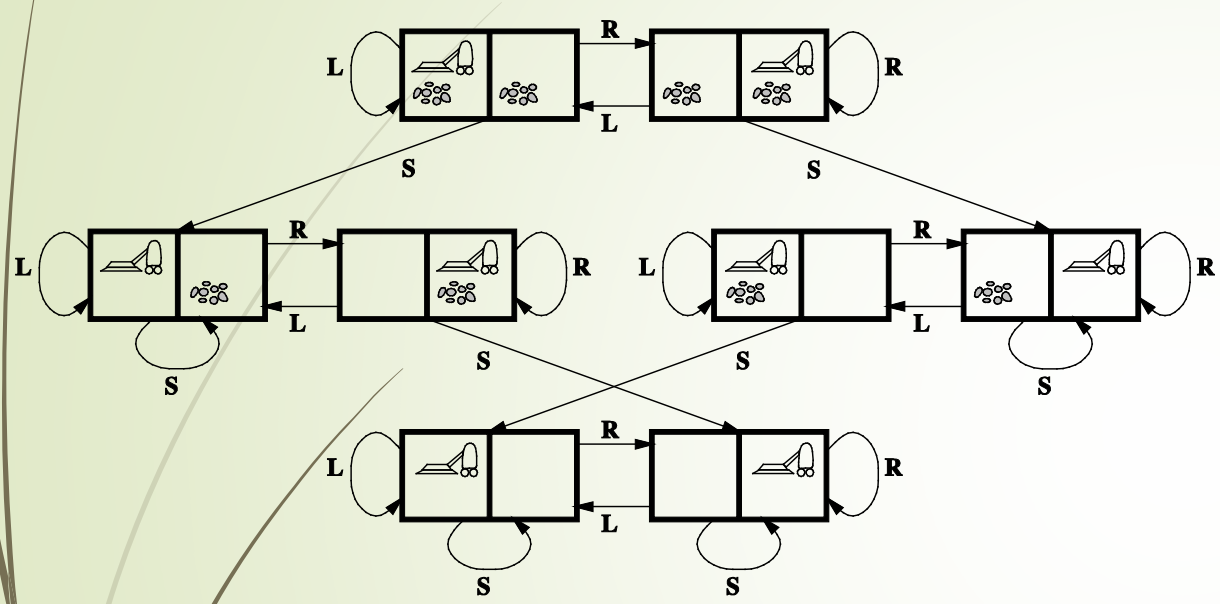
←راه‌حل بهینه راه‌حلی با کم‌ترین هزینه‌ی مسیر در میان تمامی راه‌حل‌ها است.

دنیای جاروبرقی



- حالات؟؟
- حالت اولیه؟؟
- اعمال؟؟
- آزمون هدف؟؟
- هزینه مسیر؟؟

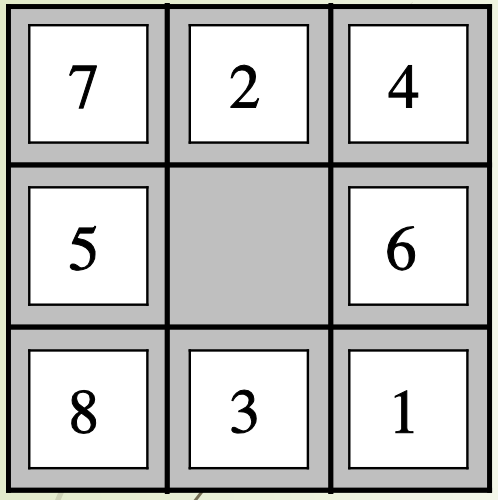
دنیای جاروبرقی



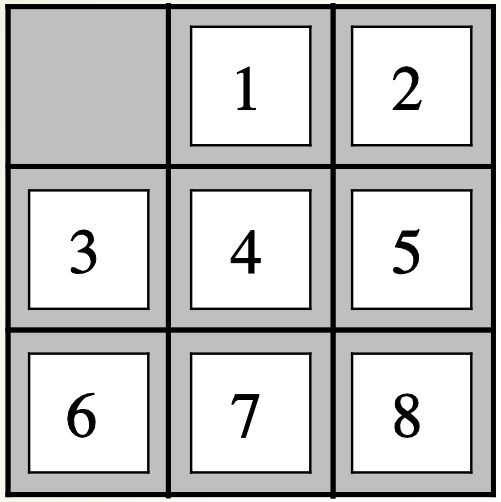
$n \times 2^n$ states

- حالات؟؟ محل عامل و تمیز یا کثیف بودن خانه‌ها
- حالت اولیه؟؟ هر حالت می‌تواند حالت اولیه باشد
- اعمال؟؟ رفتن به راست، رفتن به چپ، مکش
- آزمون هدف؟؟ بررسی آن‌که دو محل تمیز هستند.
- هزینه مسیر؟؟ تعداد اعمال برای رسیدن به هدف

پازل ۸ تایی



Start State

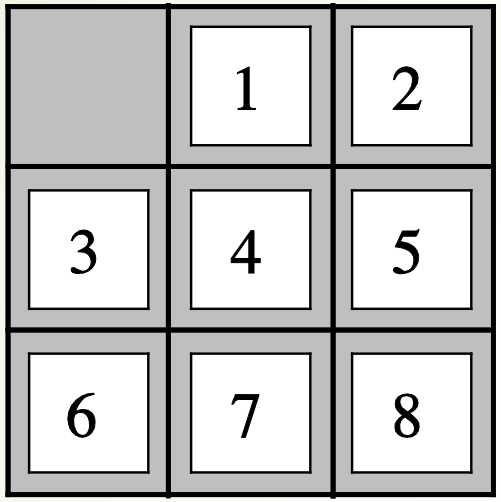


Goal State

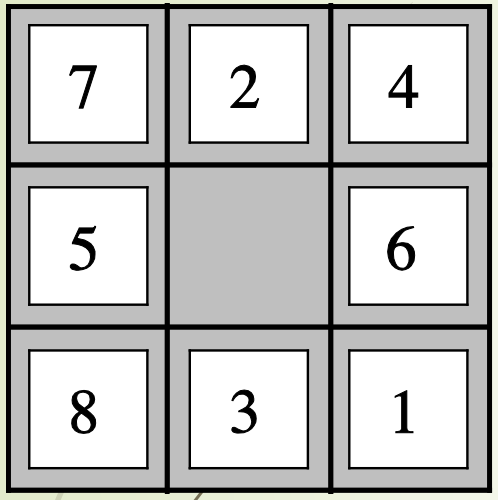
- حالات؟؟
- حالت اولیه؟؟
- اعمال؟؟
- آزمون هدف؟؟
- هزینه مسیر؟؟

پازل ۸ تایی

- حالات؟؟ یک حالت، محل هر یک از ۸ کاشی و فضای خالی در ۹ مربع را مشخص می‌کند.
- حالت اولیه؟؟ هر یک از حالات
- اعمال؟؟ جابه‌جایی فضای خالی به سمت چپ، راست، بالا و پایین
- آزمون هدف؟؟ بررسی آن‌که به یک حالت هدف مشخص رسیده است یا خیر.
- هزینه مسیر؟؟ تعداد اعمال برای رسیدن به هدف



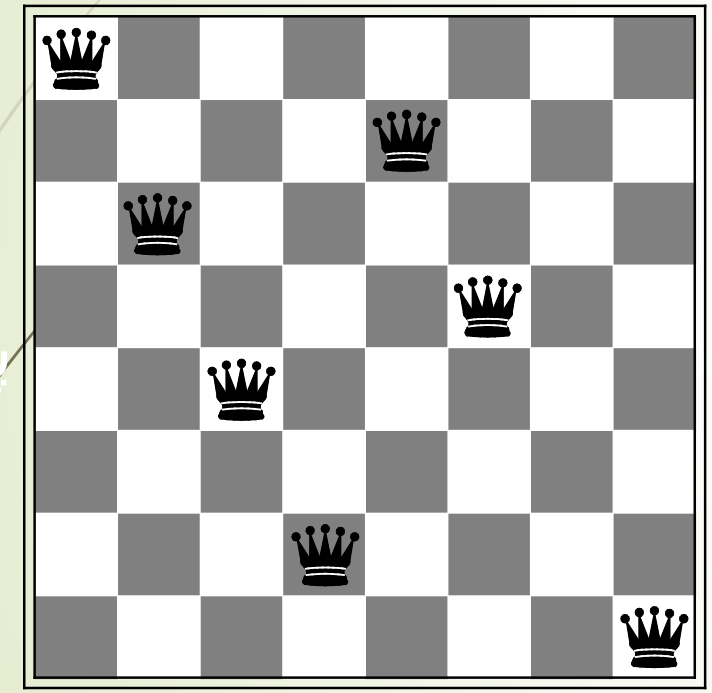
Goal State



Start State

$n!$ حالت دارد اما از هر حالت تنها می‌توان به $n!/2$ از حالات رسید.

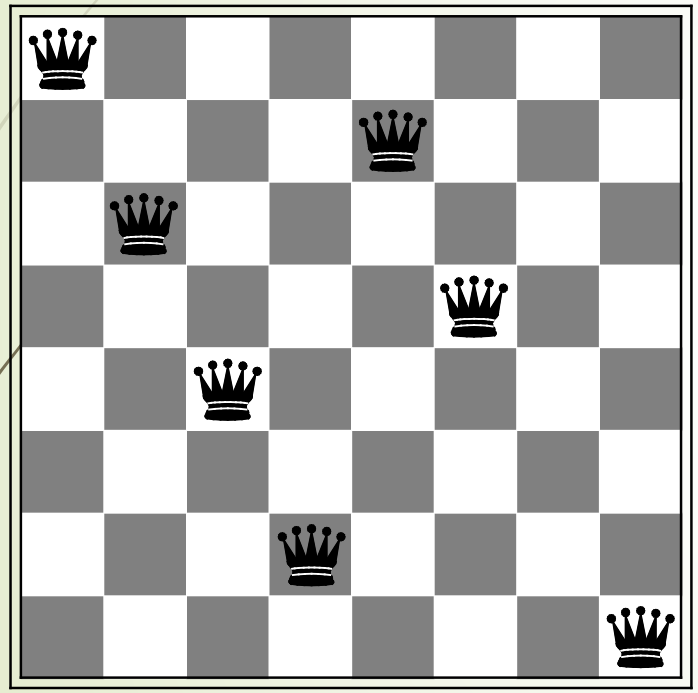
مسئله ۸ وزیر



- حالات؟؟
- حالت اولیه؟؟
- اعمال؟؟
- آزمون هدف؟؟
- هزینه مسیر؟؟

مسئله ۸ وزیر

- حالات؟؟ هر ترتیبی از ۰ تا ۸ وزیر روی صفحه
- حالت اولیه؟؟ صفحه بدون وزیر
- اعمال؟؟ اضافه کردن یک وزیر به یکی از مربع‌های خالی
- آزمون هدف؟؟ وجود ۸ وزیر بر روی صفحه بدون هیچ حمله‌ای میان آن‌ها
- هزینه مسیر؟؟ اهمیت ندارد
  - هزینه جستجو در مقابل هزینه مسیر راه‌حل



$64 \times 63 \times \ldots \times 57 \approx 1.8 \times 10^{14}$ states

جستجوی راه‌حل‌ها

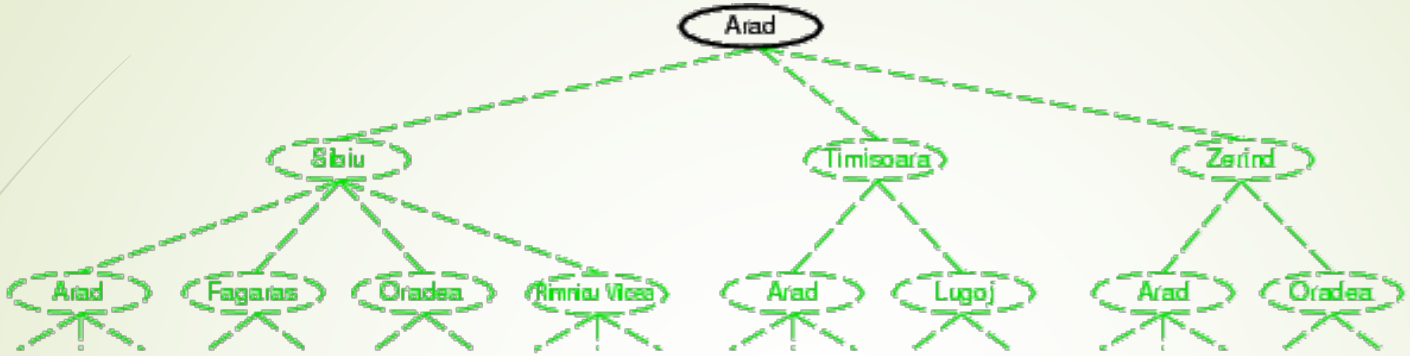
چگونه راه‌حل مسائل فرموله‌شده را بیابیم؟

الگوریتم جستجوی درختی

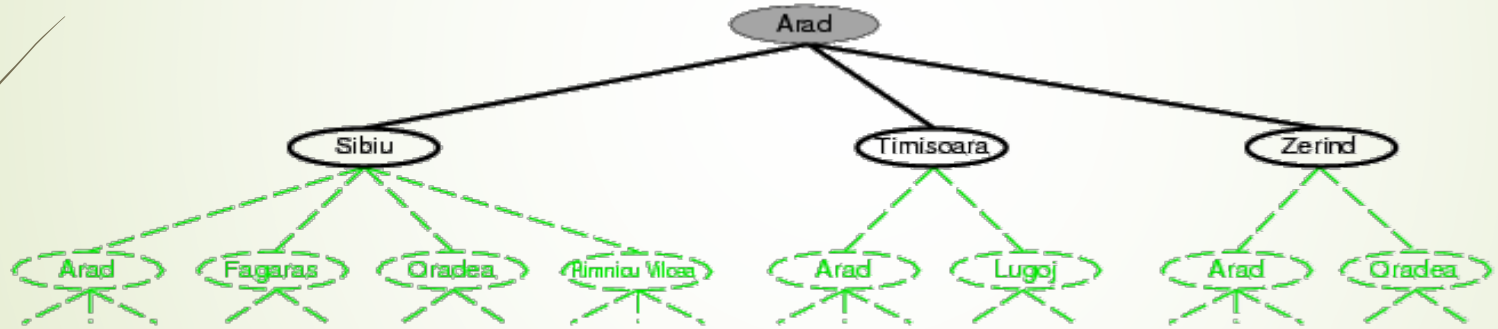
- ایده‌ی پایه
  - انجام جستجوی آفلاین و شبیه‌سازی‌شده‌ی فضای حالت با تولید پسین‌های حالاتی که قبلا کاوش شده‌اند (یعنی حالات بسط یافته).
    - جستجوی آفلاین، یک راه‌حل را قبل از گام برداشتن در دنیای واقعی محاسبه می‌کند و سپس آن را اجرا می‌کند.
- جستجوی فضای حالت از طریق تولید صریح درخت
  - حالت اولیه در ریشه
  - شاخه‌ها به‌عنوان اعمال
  - گره‌ها متناظر با حالات در فضای حالت مسئله

نمونه‌ای از درخت جستجو

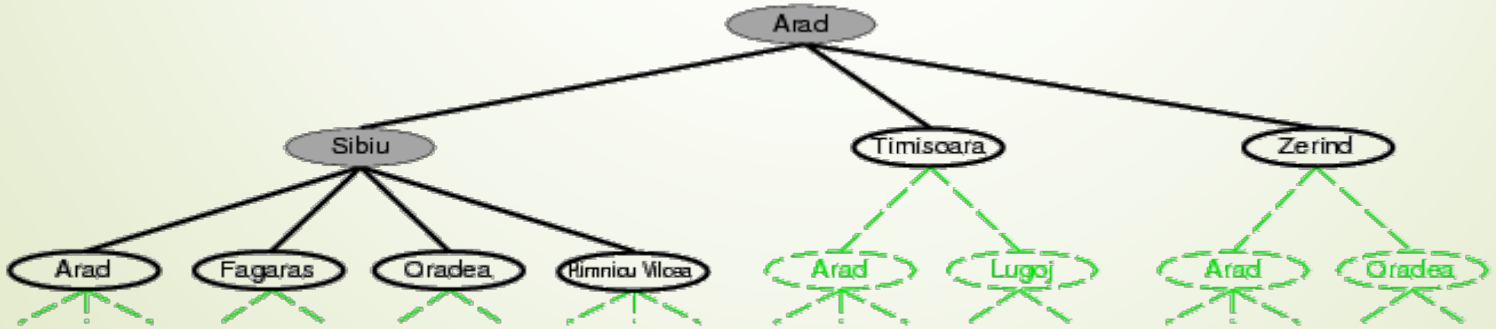
الف) حالت اولیه



ب) پس از بسط Arad

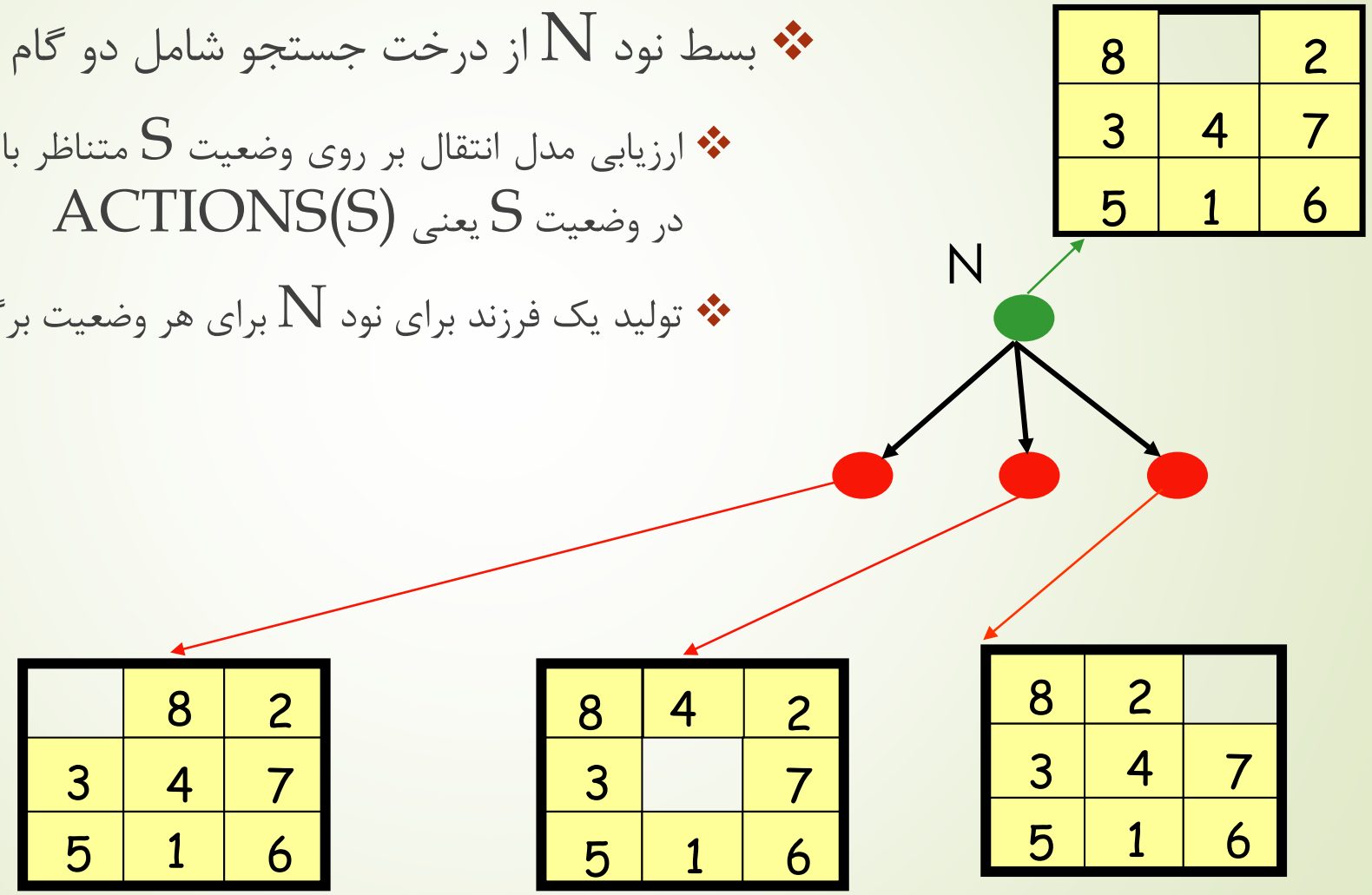


ج) پس از بسط Sibiu



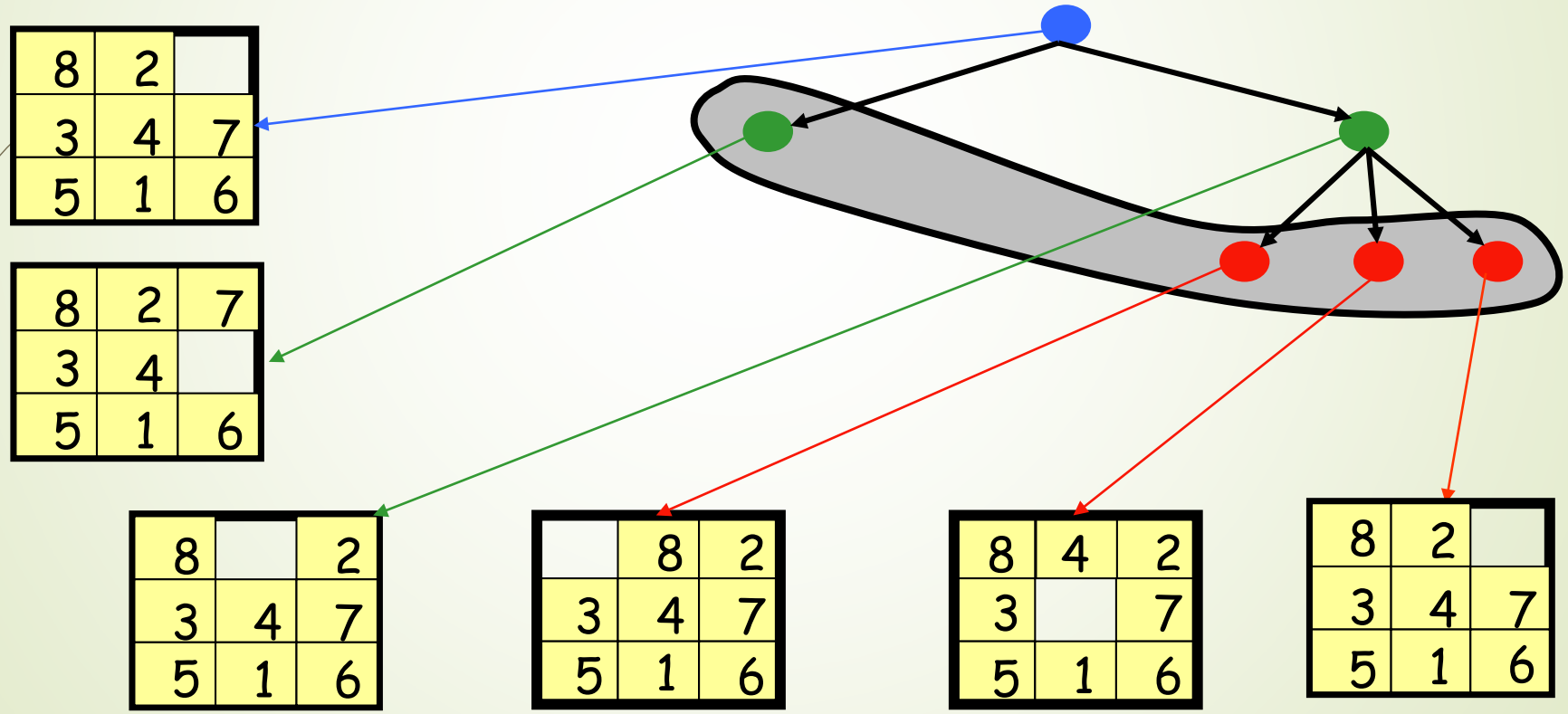
بسط نود

- بسط نود N از درخت جستجو شامل دو گام زیر است:
- ارزیابی مدل انتقال بر روی وضعیت S متناظر با نود N برای اعمال قابل انجام در وضعیت S یعنی ACTIONS(S)
- تولید یک فرزند برای نود N برای هر وضعیت برگردانده شده توسط مدل انتقال



مجموعه مرزی درخت جستجو

- Frontier مجموعه‌ای از نودهای جستجو است که تاکنون بسط داده نشده‌اند.
  - مجموعه‌ی همه‌ی نودهای برگ موجود برای بسط در هر مرحله



تابع درخت جستجو

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

initialize the frontier using the initial state of problem

loop do

if the frontier is empty **then return** failure

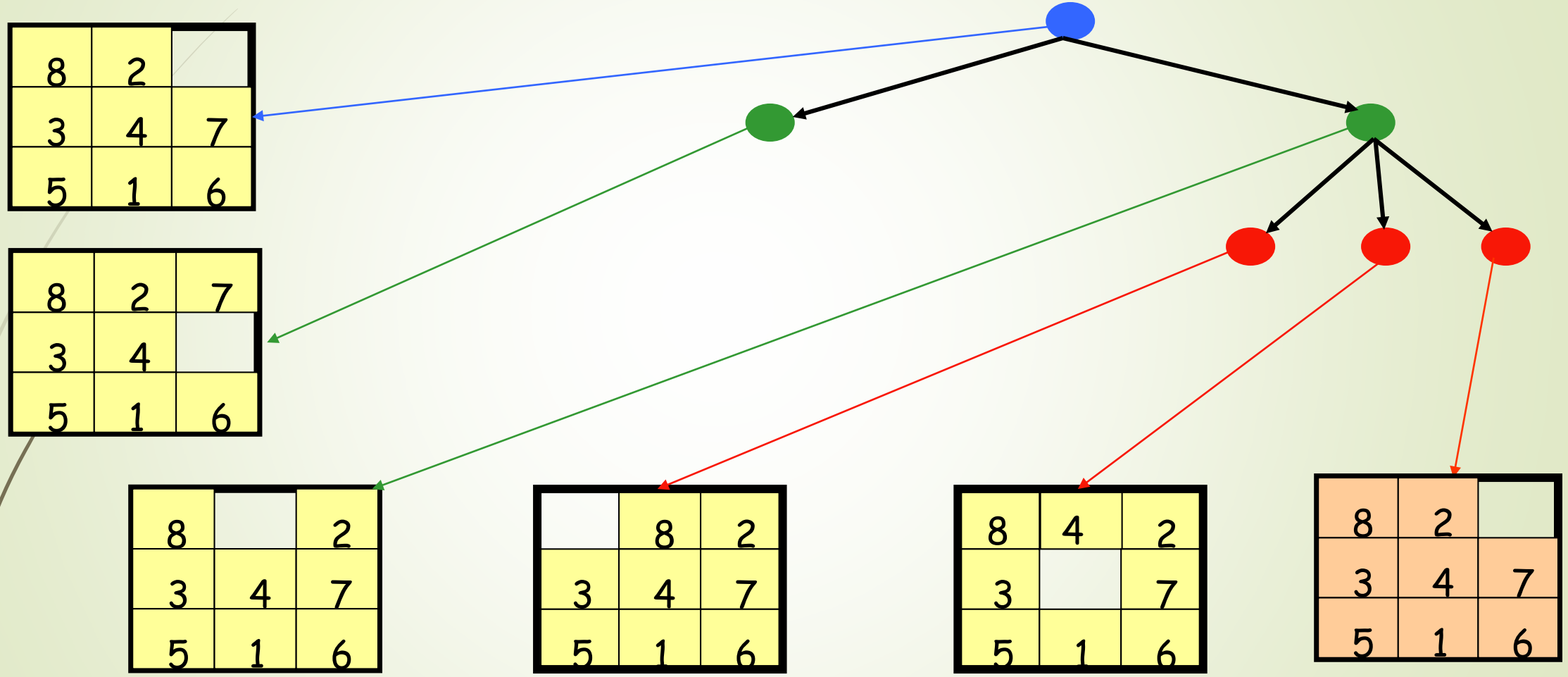
choose a leaf node and remove it from the frontier → Search Strategy

if the node contains a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

❖ **راهبرد جستجو:** الگوریتم چگونه تعیین می‌کند کدام یک از نودها را در مرحله‌ی بعد برای بسط انتخاب کند.

مشکل درخت جستجو



مشکل درخت جستجو (ادامه)

- مسیرهای زائد (Redundant paths) در درخت جستجو: بیشتر از یک راه برای رسیدن از یک حالت به حالت دیگر وجود دارد.
  - اگر به حالات اجازه داده شود دوباره ملاقات شوند، درخت جستجو ممکن است نامتناهی باشد حتی اگر فضای حالت متناهی باشد.
  - می‌تواند باعث تبدیل یک مسئله حل‌شدنی به یک مسئله‌ی غیرقابل حل شود.
- راه‌حل: استفاده از یک مجموعه کاوش‌شده (Explored set) برای یادآوری هر حالت بسط یافته

جستجوی گرافی

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of problem

initialize the explored set to be empty

loop do

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

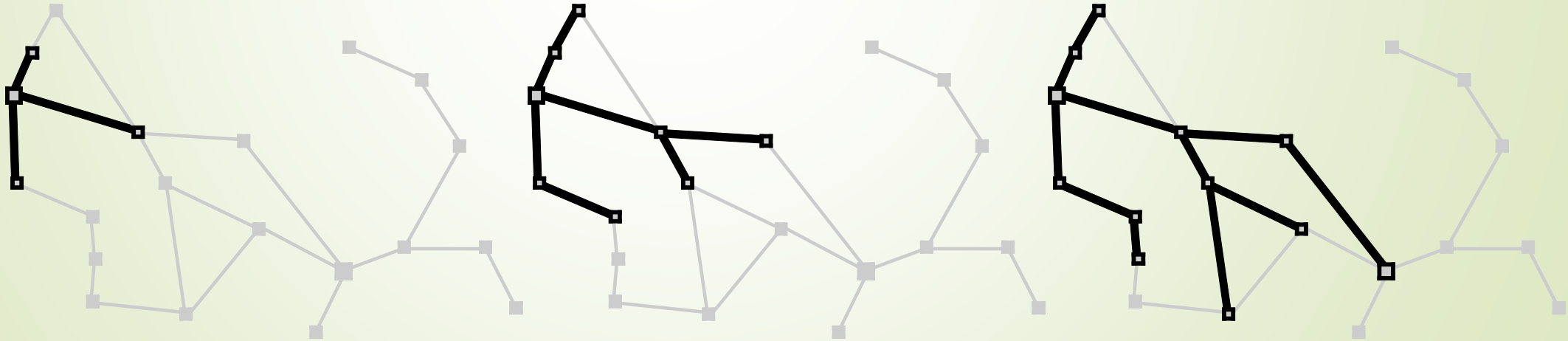
if the node contains a goal state **then return** the corresponding solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to the frontier ***only if not in the frontier or explored set***

جستجوی گرافی – مثال

- درخت جستجوی ساخته‌شده توسط الگوریتم GRAPH-SEARCH شامل حداکثر یک کپی از هر حالت است.
  - مشابه با رشد یک درخت به‌طور مستقیم بر روی گراف فضای حالت است.



اندازه‌گیری کارایی حل مسئله

- ارزیابی کارایی الگوریتم
  - کامل بودن (Completeness): اگر راه‌حل وجود داشته باشد آیا الگوریتم تضمین می‌کند که راه‌حل وجود دارد؟
  - بهینگی (Optimality): آیا استراتژی راه‌حل با کم‌ترین هزینه مسیر را می‌یابد؟
  - پیچیدگی زمانی (Time complexity): چقدر طول می‌کشد تا راه‌حل پیدا شود؟
  - پیچیدگی فضایی (Space complexity): چقدر حافظه برای انجام جستجو مورد نیاز است؟
- پیچیدگی زمان و فضا بر حسب موارد زیر اندازه‌گیری می‌شود:
  - b: ماکزیمم ضریب انشعاب (branching factor) درخت جستجو
  - d: عمق کم‌عمق‌ترین نود هدف
  - m: ماکزیمم عمق فضای حالت (ممکن است بی‌نهایت باشد)

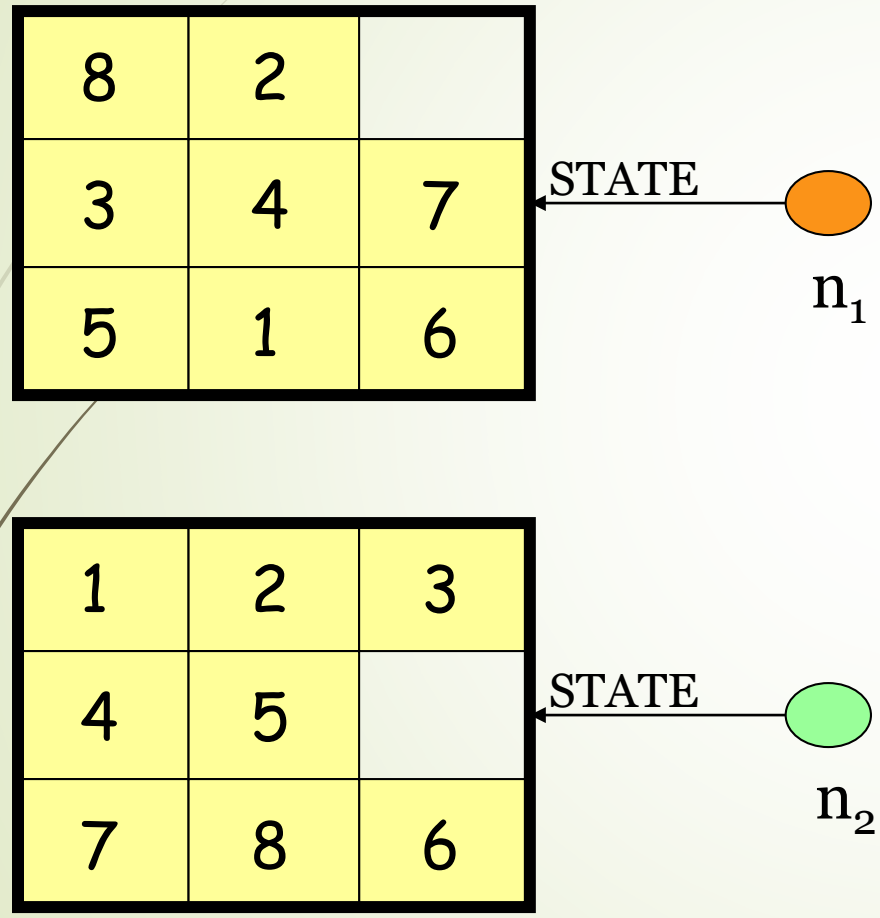
الگوریتم‌های جستجوی پایه

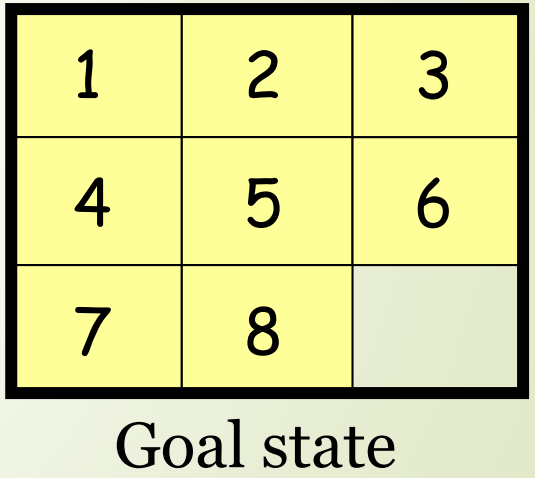
ناآگاهانه در مقابل آگاهانه

- راهبردهای ناآگاهانه یا کور (blind)
  - هیچ اطلاعات اضافه‌ای در مورد حالت به جز آن‌چه که در تعریف مسئله آمده است ندارند.
  - این راهبردها تنها قادر به تولید پسین‌ها براساس یک ترتیب مشخص و تشخیص حالت هدف از غیر هدف هستند.
  - تنها براساس مکان نودها در درخت جستجو عمل می‌کنند.
- راهبردهای آگاهانه یا هیوریستیک (heuristic)
  - راهبردهایی که می‌دانند یک حالت غیرهدف "امیدوارکننده‌تر" از دیگری است.

ناآگاهانه در مقابل آگاهانه (مثال)

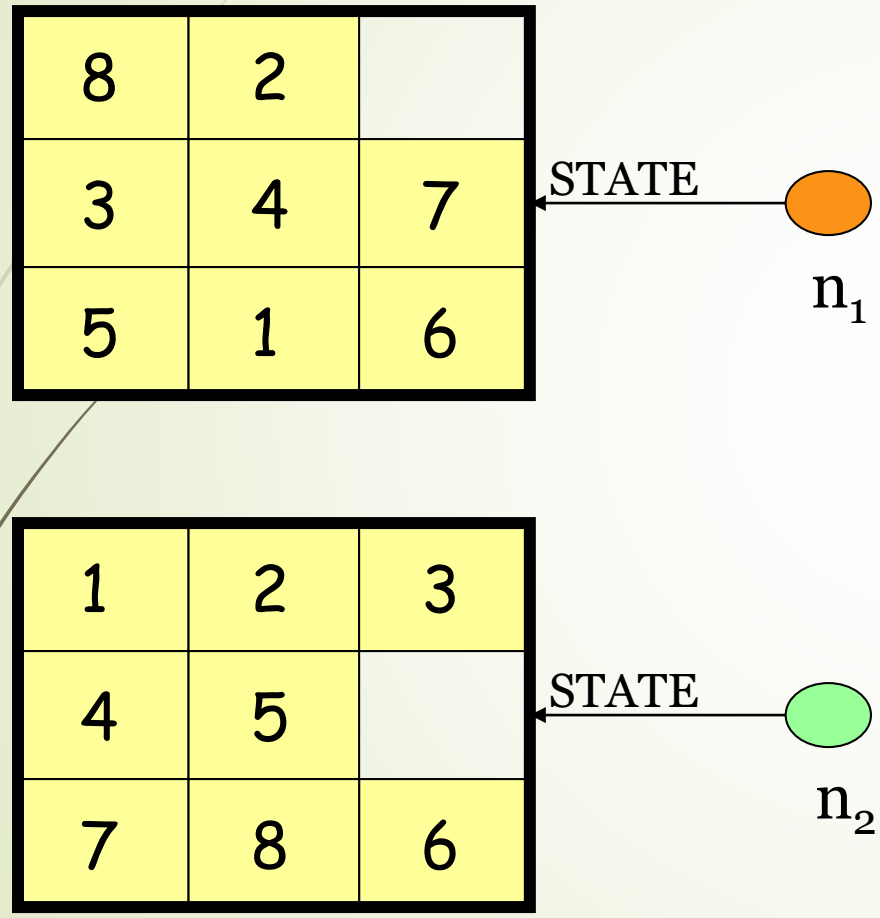
- برای یک جستجوی ناآگاهانه، n_1 و n_2 تنها دو نود در مکانی از درخت جستجو می‌باشند.





ناآگاهانه در مقابل آگاهانه (مثال)

- برای یک جستجوی آگاهانه با شمارش تعداد کاشی‌هایی که در مکان اشتباه قرار گرفته‌اند n_2 امیدوارکننده‌تر از n_1 است.



1	2	3
4	5	6
7	8	

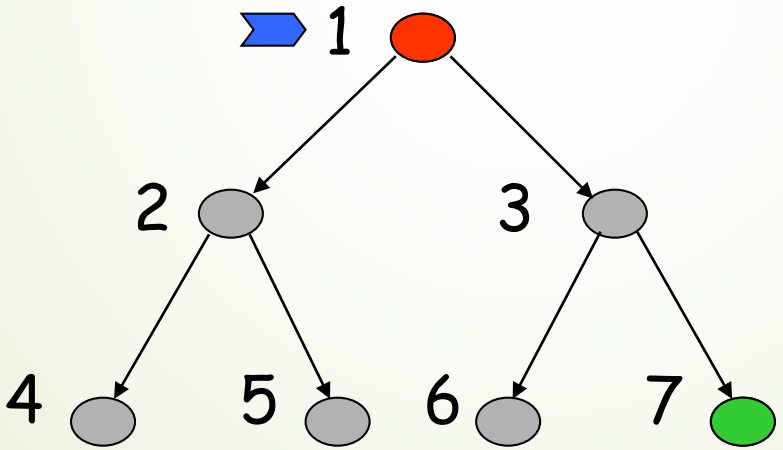
Goal state

جستجوی ناآگاهانه

- دسته‌بندی براساس الگوریتم بسط نود
  - جستجوی اول سطح (Breadth-first search)
  - جستجوی هزینه یکنواخت (Uniform-cost search)
  - جستجوی اول عمق (Depth-first search)
  - جستجوی عمقی محدود (Depth-limited search)
  - جستجوی عمیق‌شونده‌ی تکراری (Iterative deepening search)
  - جستجوی دوطرفه (Bidirectional search)

جستجوی اول سطح – BFS

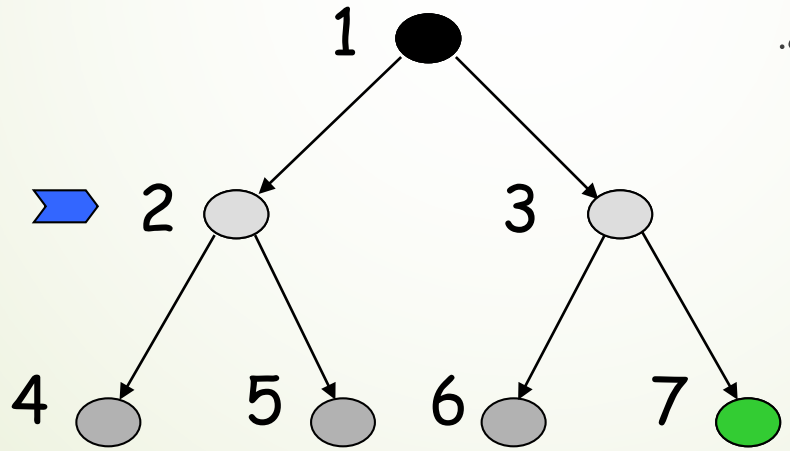
- بسط کم‌عمق‌ترین گره بسط‌نیافته
  - آزمون هدف به هنگام تولید نود اعمال می‌شود نه هنگام انتخاب آن برای بسط
- پیاده‌سازی: مجموعه‌ی مرزی یک صف FIFO خواهد بود.
  - گره‌های جدید در انتهای صف قرار می‌گیرند.



Frontier = {1}
Explored= {}

جستجوی اول سطح – BFS

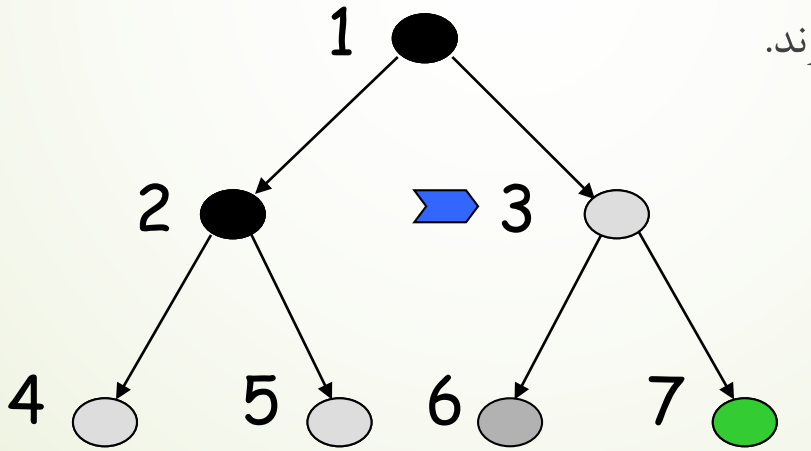
- بسط کم‌عمق‌ترین گره بسط‌نیافته
  - آزمون هدف به هنگام تولید نود اعمال می‌شود نه هنگام انتخاب آن برای بسط
- پیاده‌سازی: مجموعه‌ی مرزی یک صف FIFO خواهد بود.
  - گره‌های جدید در انتهای صف قرار می‌گیرند.



Frontier= {2, 3}
Explorer= {1}

جستجوی اول سطح – BFS

- بسط کم‌عمق‌ترین گره بسط‌نیافته
  - آزمون هدف به هنگام تولید نود اعمال می‌شود نه هنگام انتخاب آن برای بسط
- پیاده‌سازی: مجموعه‌ی مرزی یک صف FIFO خواهد بود.
  - گره‌های جدید در انتهای صف قرار می‌گیرند.

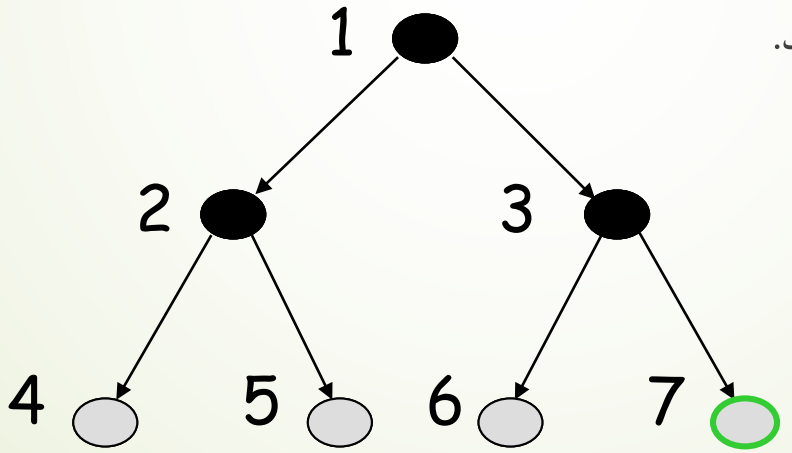


Frontier = {3, 4, 5}

Explorer= {1,2}

جستجوی اول سطح – BFS

- بسط کم‌عمق‌ترین گره بسط‌نیافته
  - آزمون هدف به هنگام تولید نود اعمال می‌شود نه هنگام انتخاب آن برای بسط
- پیاده‌سازی: مجموعه‌ی مرزی یک صف FIFO خواهد بود.
  - گره‌های جدید در انتهای صف قرار می‌گیرند.



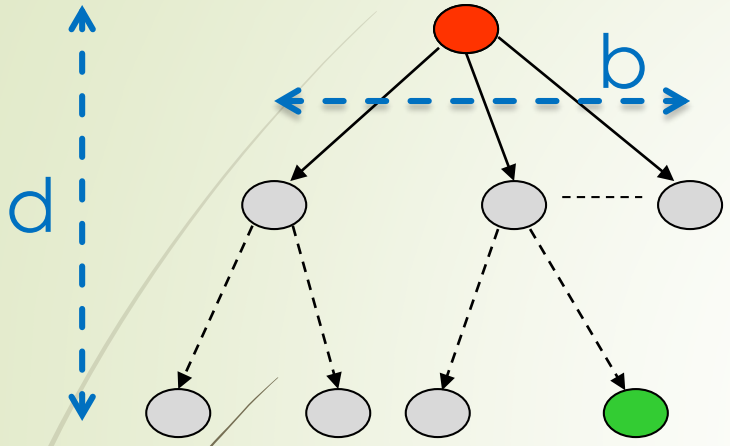
Frontier= {4, 5, 6, 7}

Explorer= {1,2,3}

BFS الگوریتم

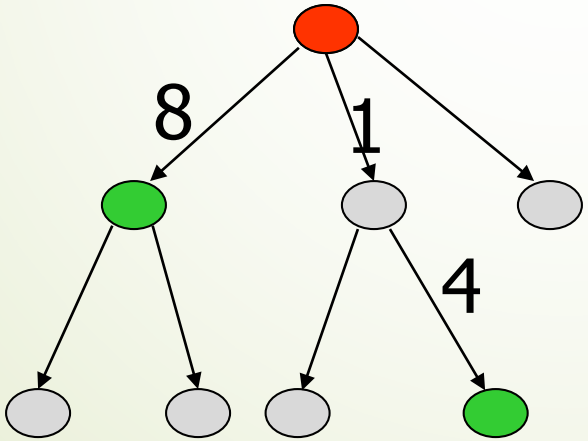
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or
    failure
node ←a node with STATE = problem.INITIAL-STATE, PATH-COST =
    0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ←a FIFO queue with node as the only element
explored ←an empty set
loop do
    if EMPTY?( frontier) then return failure
    node←POP( frontier ) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ←CHILD-NODE(problem, node, action)
        if child .STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return
                SOLUTION(child)
            frontier ←INSERT(child , frontier)
```

ارزیابی BFS



- کامل؟
  - بله اگر b و d متناهی باشند.

- بهینگی؟
  - بله اگر هزینه مسیر یک تابع غیرکاهشی از عمق گره باشد.
  - برای مثال هزینه تمامی اعمال یکسان باشد.



ارزیابی BFS

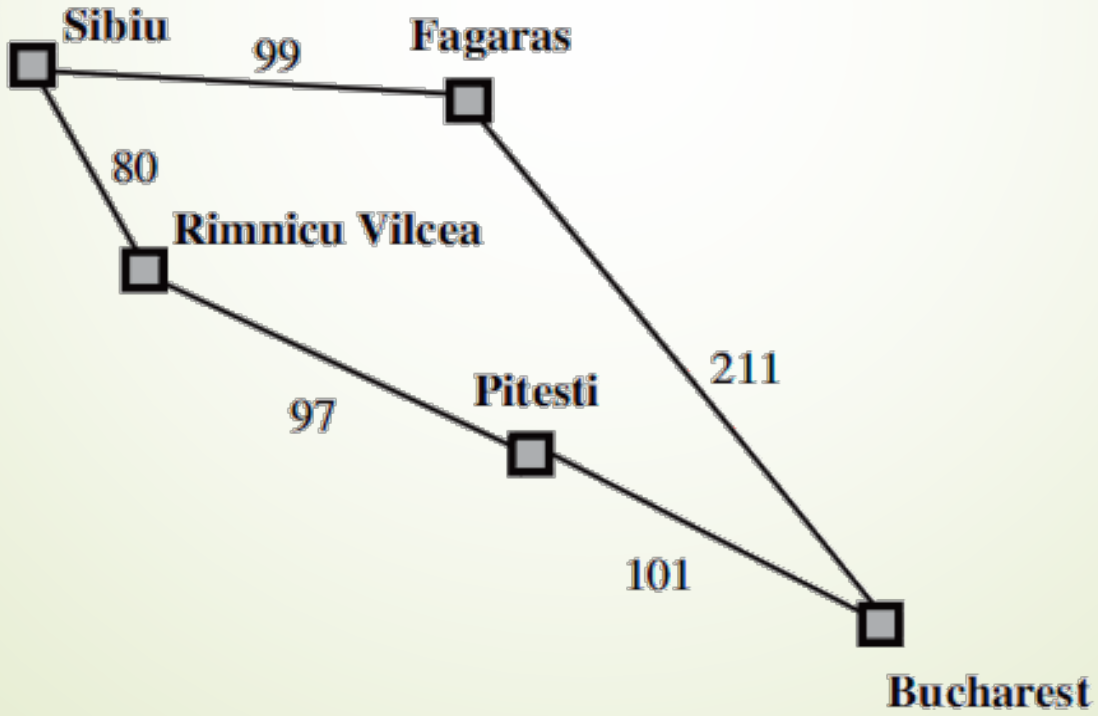
- پیچیدگی زمانی؟
  - وابسته به این است که چه موقع آزمون هدف بر روی نود اعمال شود.
  - هنگام تولید نود: $b+b^2+b^3+\ldots+b^d = O(b^d)$
  - هنگام انتخاب نود برای بسط: $b+b^2+b^3+\ldots+b^d + (b^{d+1}-b) = O(b^{d+1})$
- پیچیدگی فضایی؟
  - هر گره ای که تولید شده باید در حافظه باقی بماند تا بتوان فرزندان بعدی آن گره را تولید کرد.
  - فضای کلی = فضای مجموعه کاوش‌شده + فضای مجموعه مرزی
  - جستجوی گرافی: $O(b^{d-1})+ O(b^d)= O(b^d)$
  - جستجوی درختی فضای بیشتری استفاده نمی‌کند در حالی‌که ممکن است باعث شود زمان اضافه‌تر زیادی داشته باشد.

جستجوی هزینه یکنواخت – UCS

- نود n با کم‌ترین هزینه $g(n)$ را بسط می‌دهد.
  - هر یال دارای هزینه $c>0$ است.
  - $g(n)=\sum costs\ of\ arcs$
  - هم‌ارز با BFS است اگر هزینه تمامی اعمال آن یکسان باشد.
- پیاده‌سازی: صف اولویت مرتب‌شده با هزینه مسیر $g(n)$ برای نودهای مجموعه مرزی

جستجوی هزینه یکنواخت – UCS

- دو تفاوت عمده UCS با BFS
  - آزمون هدف بر روی یک نود هنگام انتخاب آن برای بسط اعمال می‌شود، نه هنگام تولید نود (چرا؟)
  - فرض کنید نودی تولید شده باشد که وضعیت آن نود قبلا در مجموعه frontier وارد شده باشد، در این صورت اگر نود جدید حاوی مسیر بهتری نسبت به نود قبلی باشد، آن نود را با نود موجود در frontier جایگزین می‌کنیم.



الگوریتم جستجوی هزینه یکنواخت

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ←a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

frontier ←a priority queue ordered by PATH-COST, with *node* as the only element

explored ←an empty set

loop do

if EMPTY?(*frontier*) **then return** failure

node←POP(*frontier*) /* chooses the lowest-cost node in frontier */

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

add *node*.STATE to *explored*

for each action **in** *problem*.ACTIONS(*node*.STATE) **do**

child ←CHILD-NODE(*problem*, *node*, *action*)

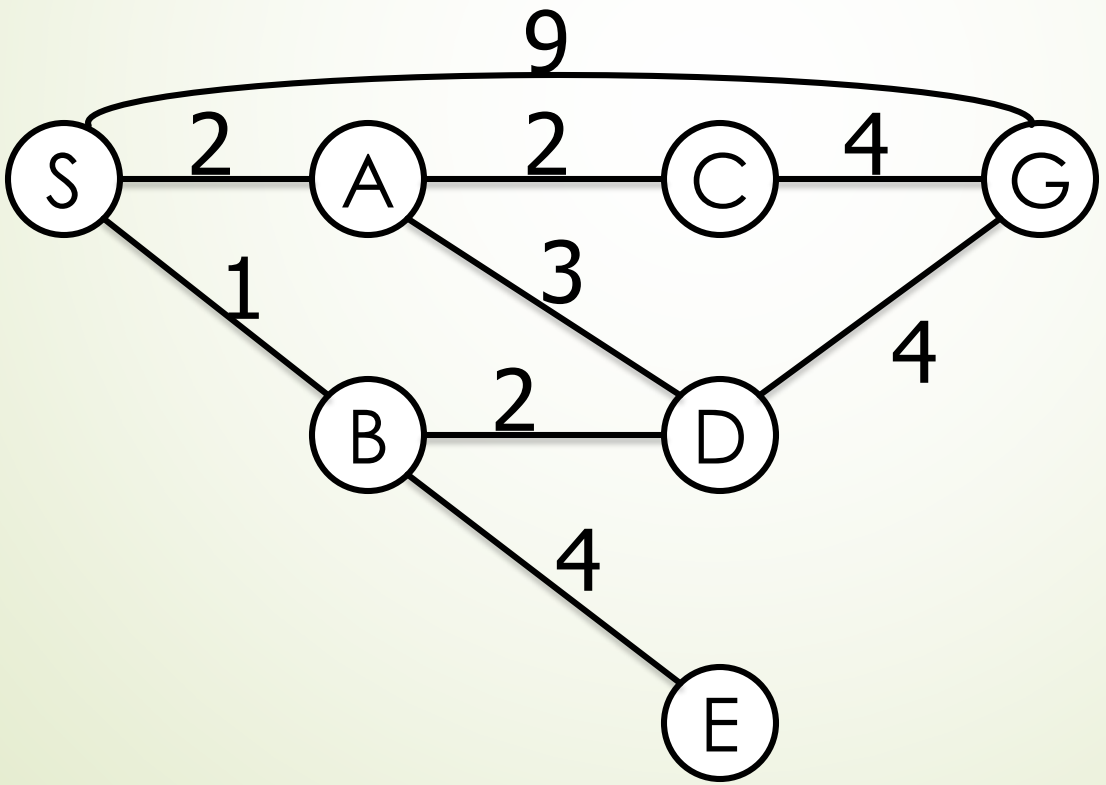
if *child* .STATE is not **in** *explored* or *frontier* **then**

frontier ←INSERT(child , frontier)

else if *child* .STATE is **in** *frontier* with higher PATH-COST **then** replace that *frontier node* with *child*

جستجوی هزینه یکنواخت – تمرین

شکل زیر مسئله جستجویی را نشان می‌دهد که به صورت گراف مدل شده است. وضعیت شروع S بوده و تنها وضعیت هدف G است. اعداد نشان داده‌شده بر روی یال‌ها هزینه‌ی هر عمل را نشان می‌دهد. مسیر برگردانده‌شده توسط الگوریتم جستجوی گرافی هزینه یکنواخت را به‌دست آورید.



تست

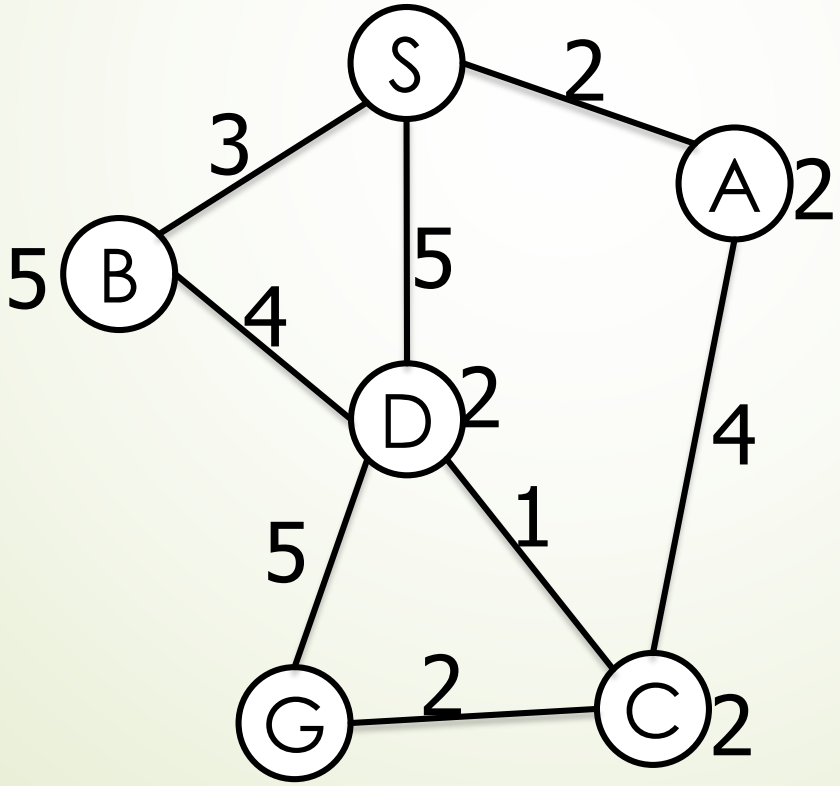
گراف زیر را درنظر بگیرید. گره S وضعیت شروع، گره G وضعیت هدف، اعداد کنار یال‌ها هزینه عبور از آن یال و اعداد کنار گره‌ها تابع h را نشان می‌دهند. در صورت استفاده از روش جستجوی uniform cost search ترتیب ملاقات گره‌ها به‌صورت کدام یک از موارد زیر خواهد بود؟

۱) S, A, B, D, G

۲) S, A, B, C, D, G

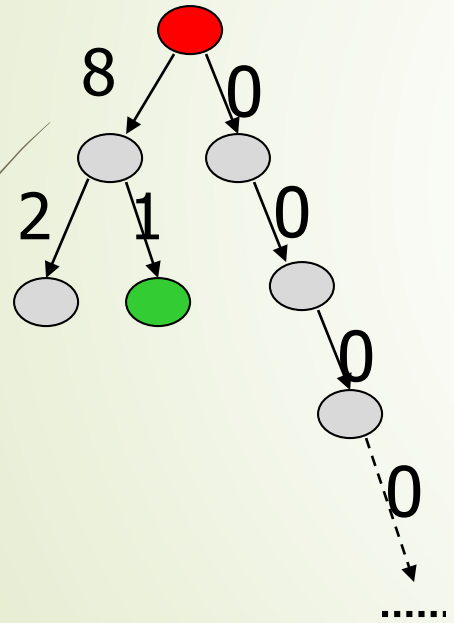
۳) S, A, B, D, C, G ✓

۴) S, A, D, C, G



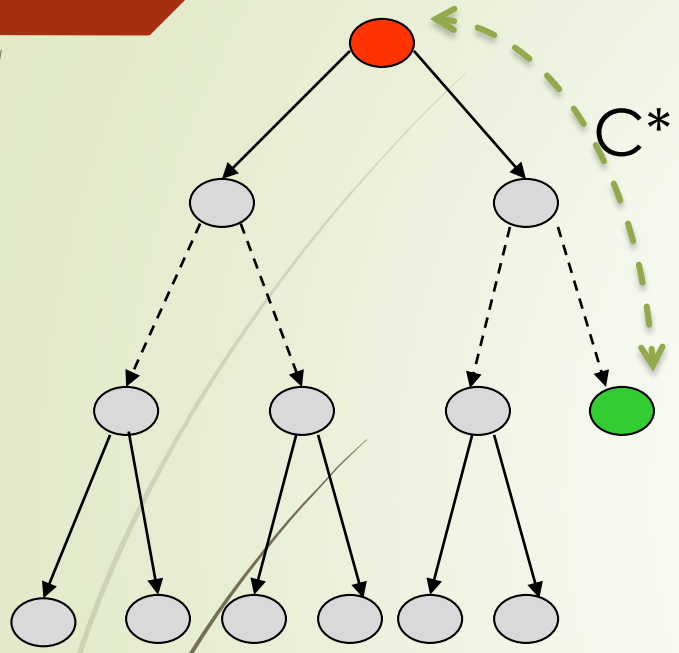
ارزیابی جستجوی هزینه یکنواخت

- کامل؟
  - بله اگر $0 < \varepsilon \leq$ step-cost باشد تا از یک دنباله نامتناهی از اعمال با هزینه صفر اجتناب شود.



- بهینگی؟
  - بله گره‌ها به ترتیب افزایش $g(n)$ بسط می‌یابند.

ارزیابی جستجوی هزینه یکنواخت



- پیچیدگی زمانی؟
  - فرض کنید C^* هزینه‌ی راه‌حل بهینه بوده و هر عمل حداقل ε هزینه داشته باشد.
  - تعداد گامها: $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
  - تعداد گامها وقتی تمامی هزینه‌ها یکسان باشد: $O(b^{d+1})$
  - و هزینه مرتب نگه داشتن مجموعه frontier در هر گام $O(\log(n))$
    - n: اندازه مجموعه frontier : $O(b^{\lfloor C^*/\varepsilon \rfloor})$
  - هزینه کل: $O(b^{\lfloor C^*/\varepsilon \rfloor} \log(b^{\lfloor C^*/\varepsilon \rfloor}))$

- پیچیدگی فضایی؟

$$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$$

اثبات بهینگی جستجوی هزینه یکنواخت

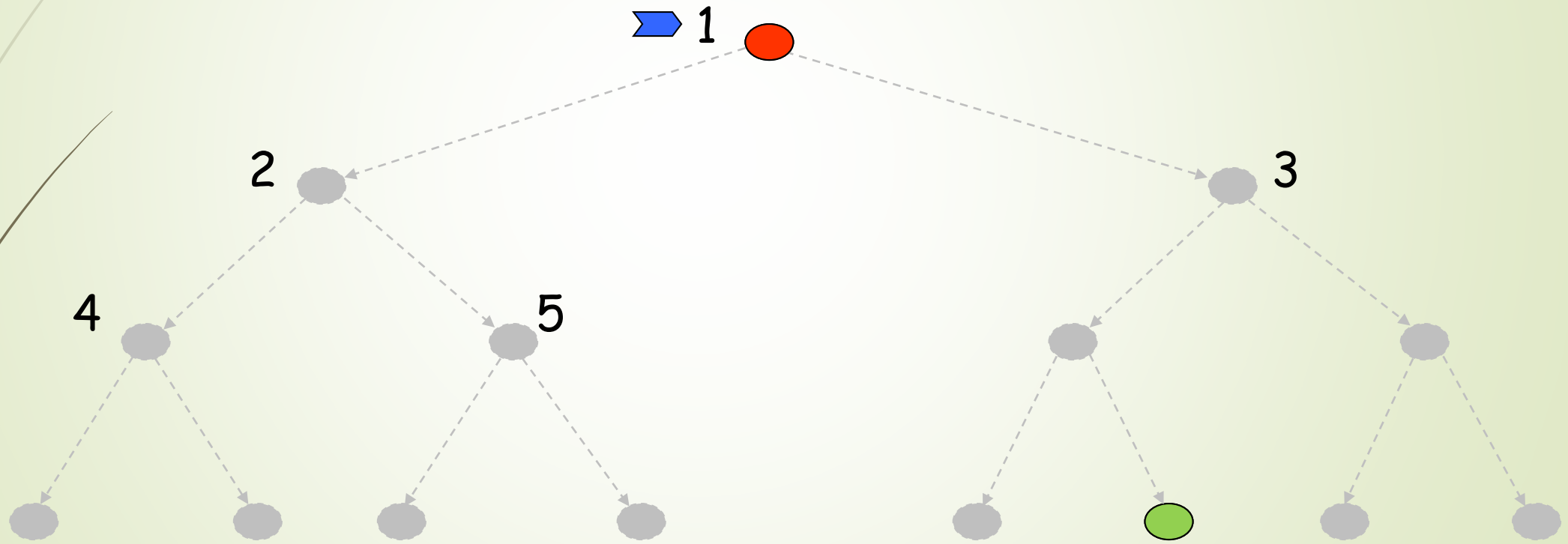
❖ لم: هرگاه UCS نود n را برای بسط انتخاب کند هزینه‌ی بهینه تا آن نود را یافته است.

اثبات با برهان خلف: فرض کنید هنگامی که نود n برای بسط انتخاب می‌شود، مسیر بهینه از ریشه تا n به‌دست نیامده باشد آنگاه باید بتوان از طریق نود دیگری مانند n' که در مجموعه‌ی frontier فعلی قرار دارد با یک مسیر بهینه به حالت موجود در n رسید. طبق تعریف $g(n')<g(n)$ بوده و چون هزینه‌های گام غیرمنفی است، افزودن نودهای بیشتر در ادامه مسیر از n' هزینه‌ی کم‌تری را ایجاد نمی‌کند. در چنین حالتی n' طبق روال انتخاب UCS باید زودتر از n انتخاب می‌شد و فرض ما مبنی‌بر غیربهینه بودن مسیر حاصل‌شده تا n غلط است.

← الگوریتم UCS نودها را به ترتیب هزینه‌ی مسیر بهینه‌شان بسط می‌دهد. از این‌رو اولین نود هدف انتخابی برای بسط باید دارای راه‌حل بهینه باشد.

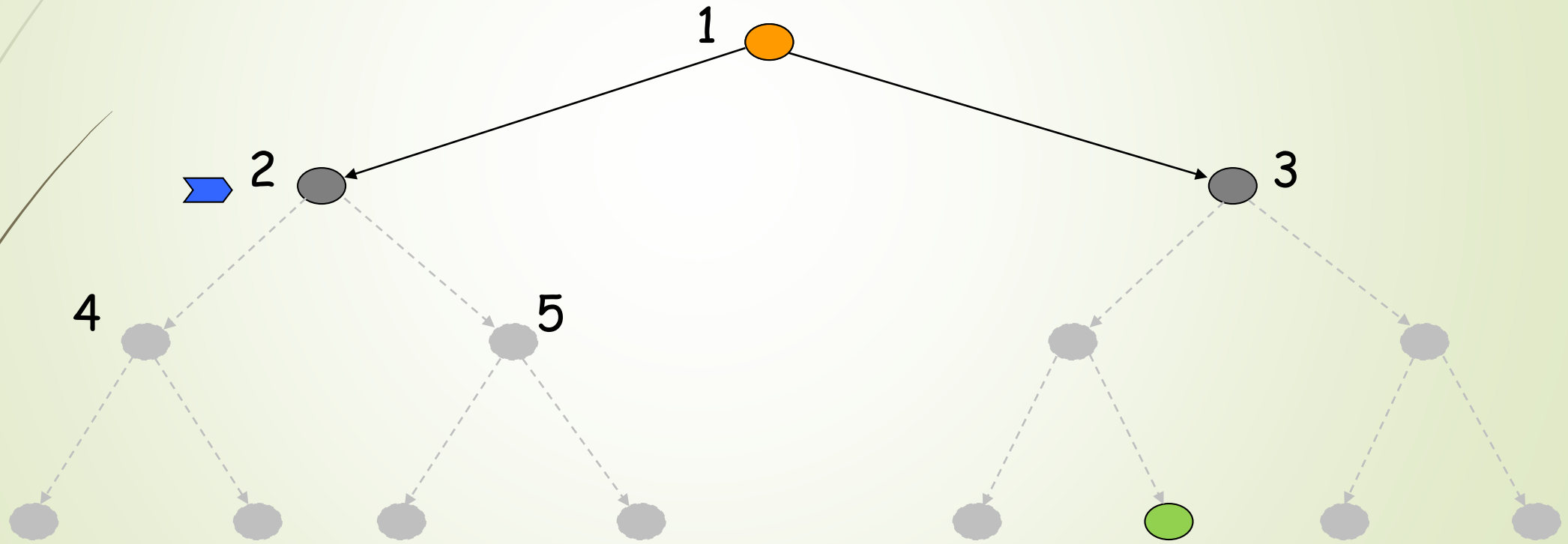
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



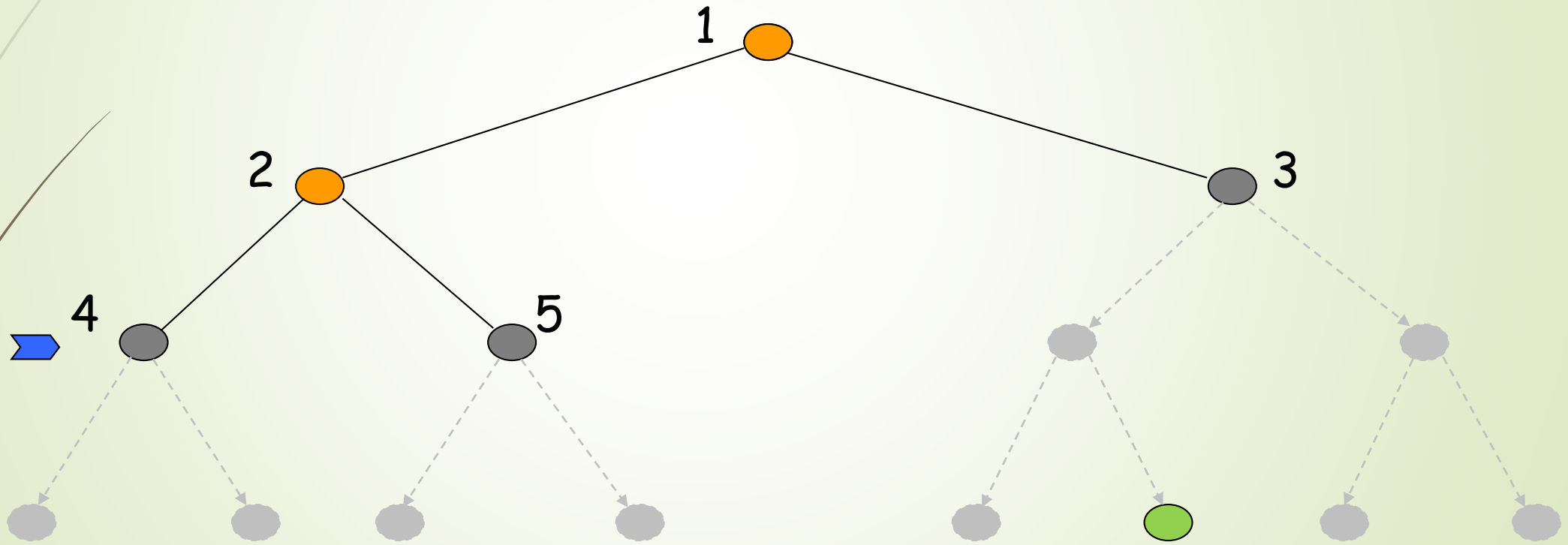
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



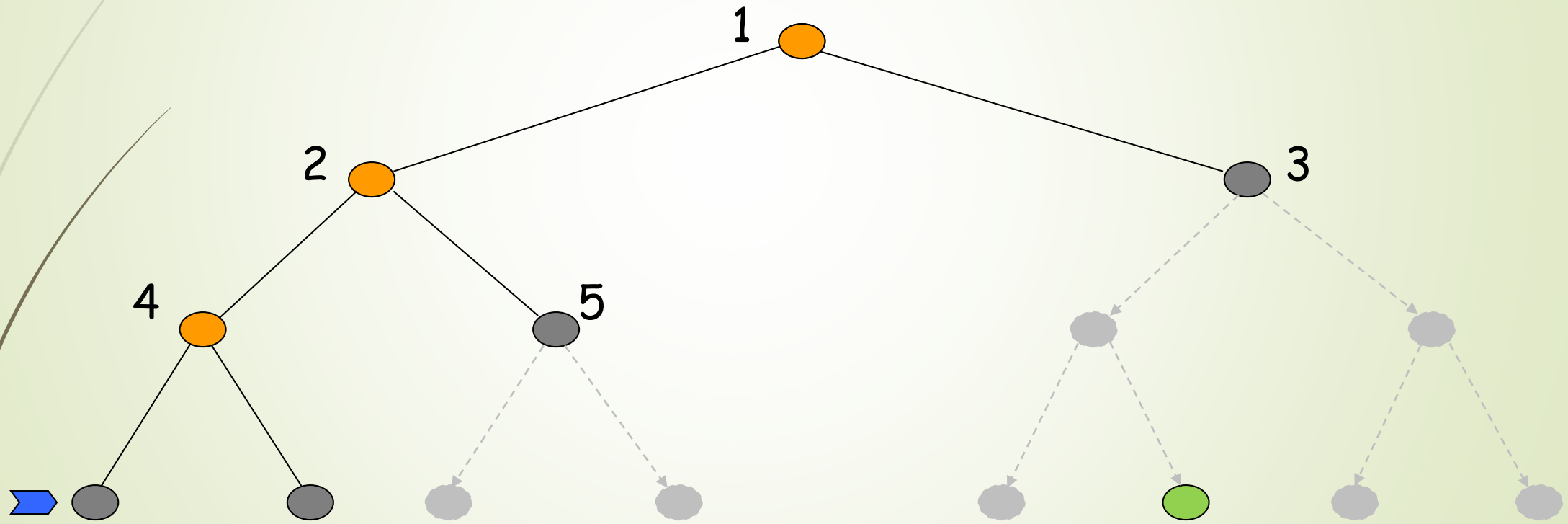
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



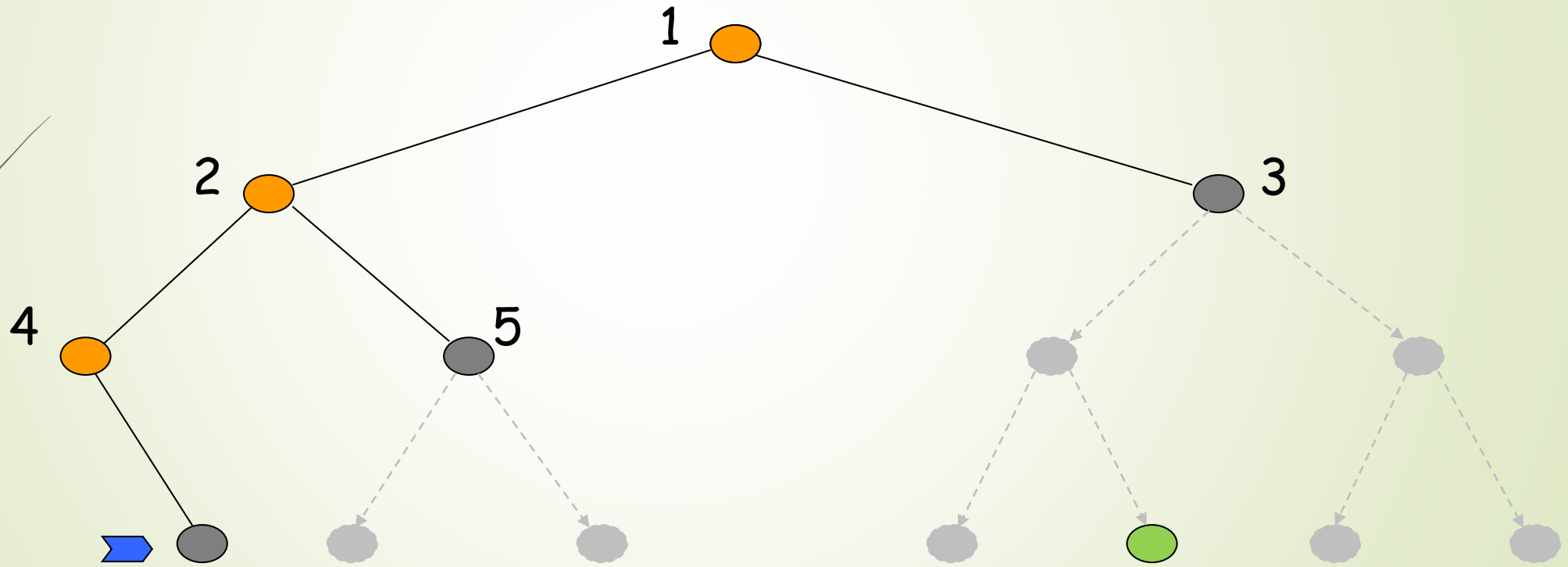
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



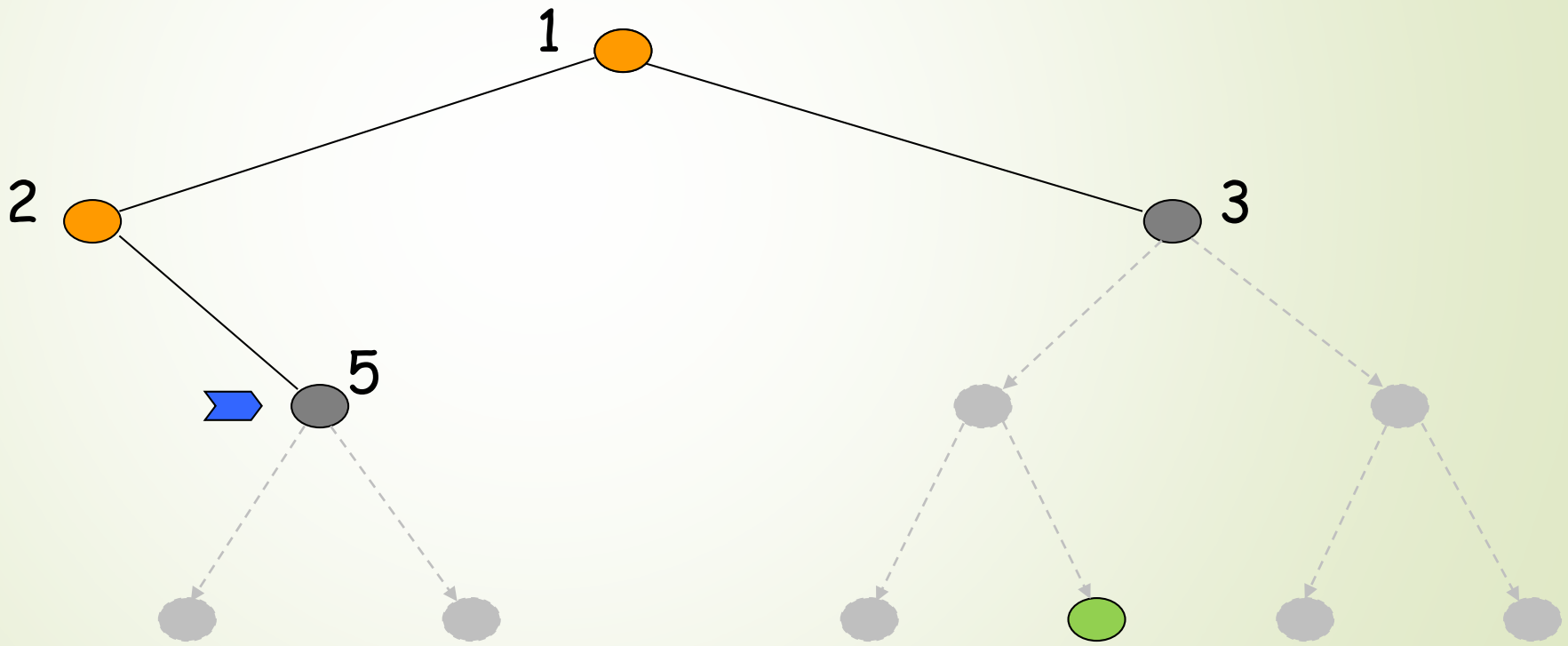
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



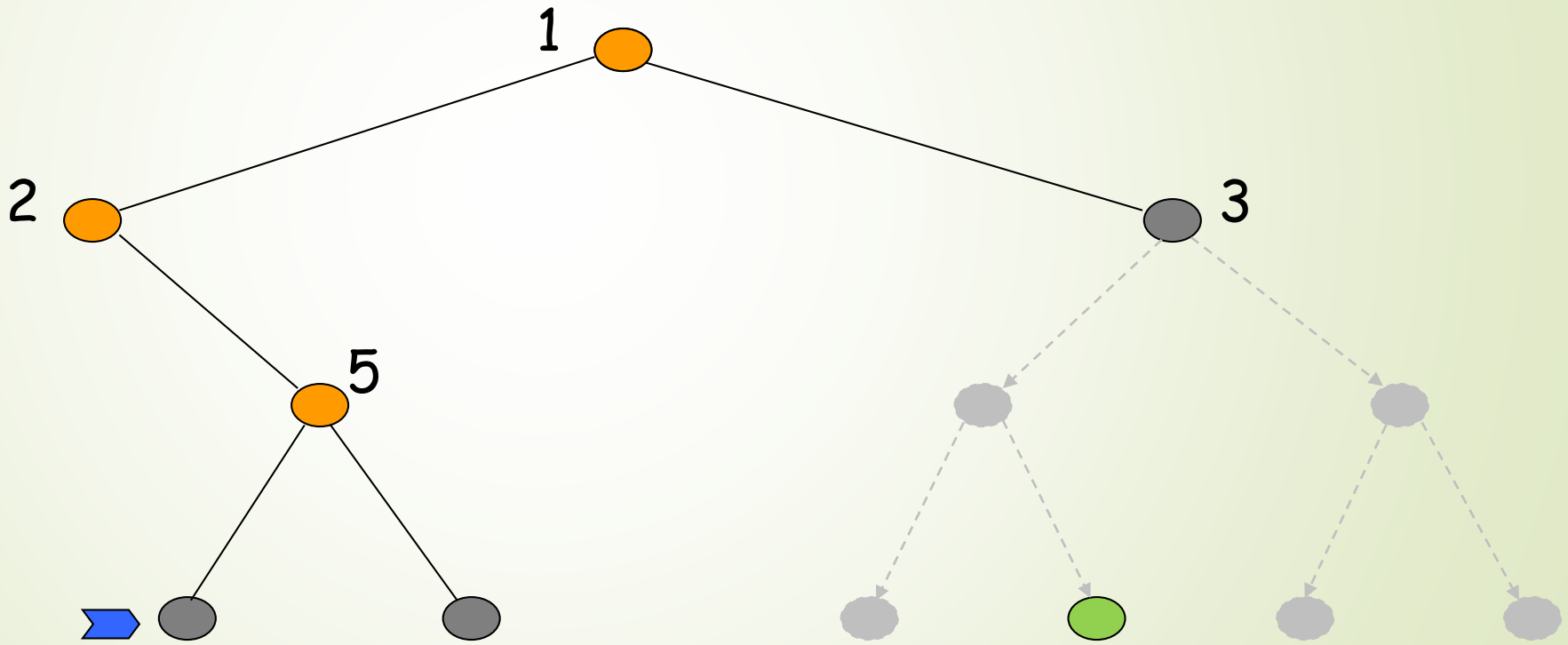
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



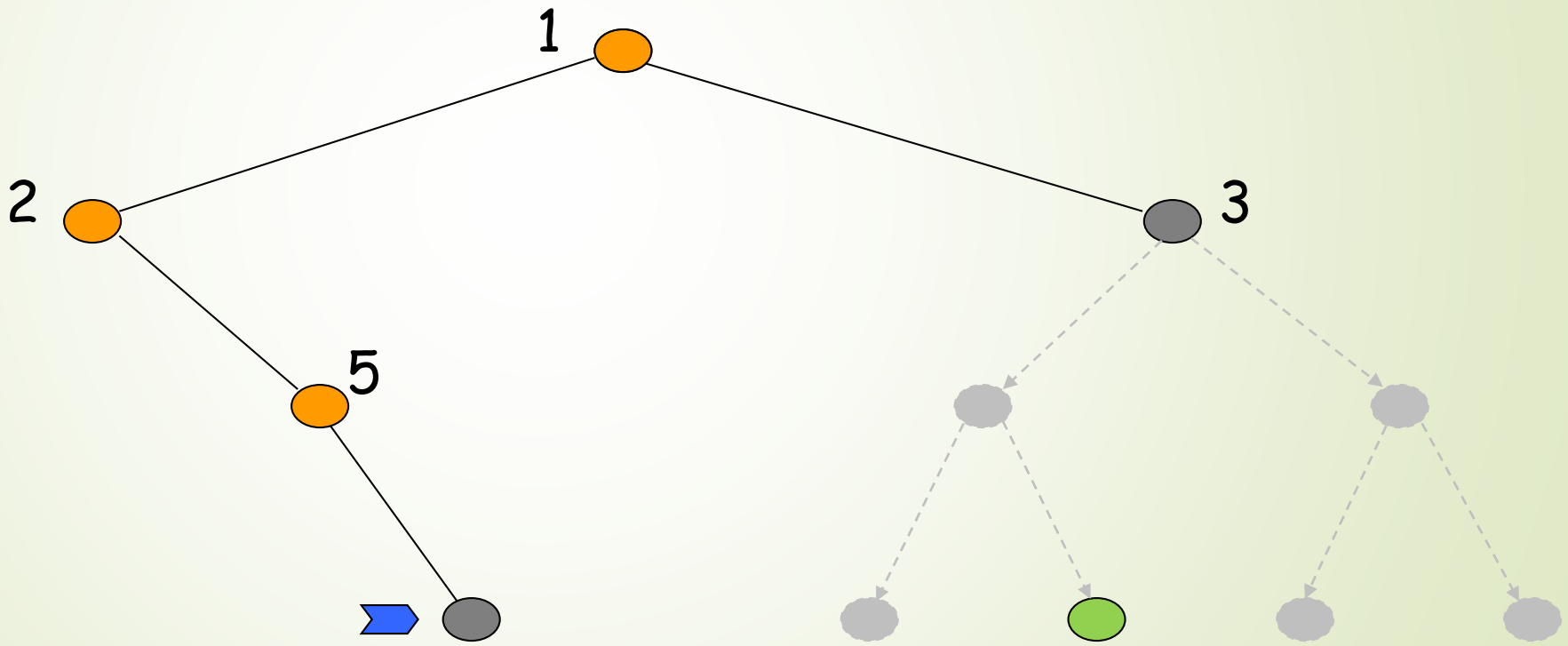
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



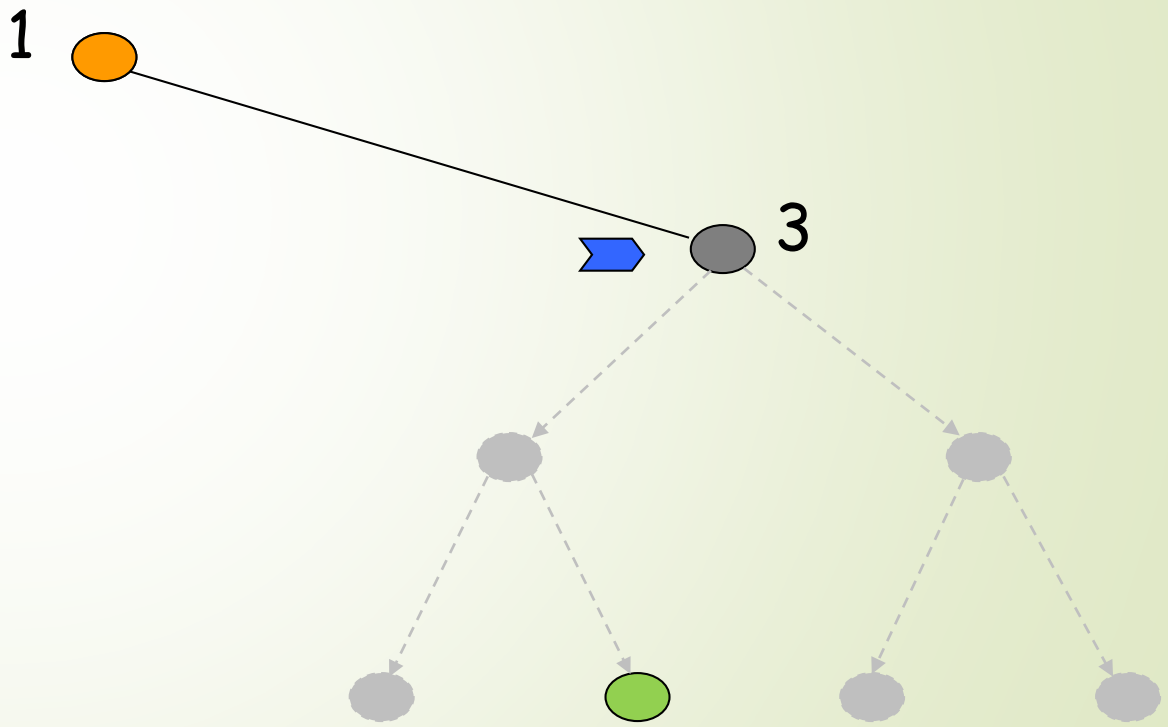
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



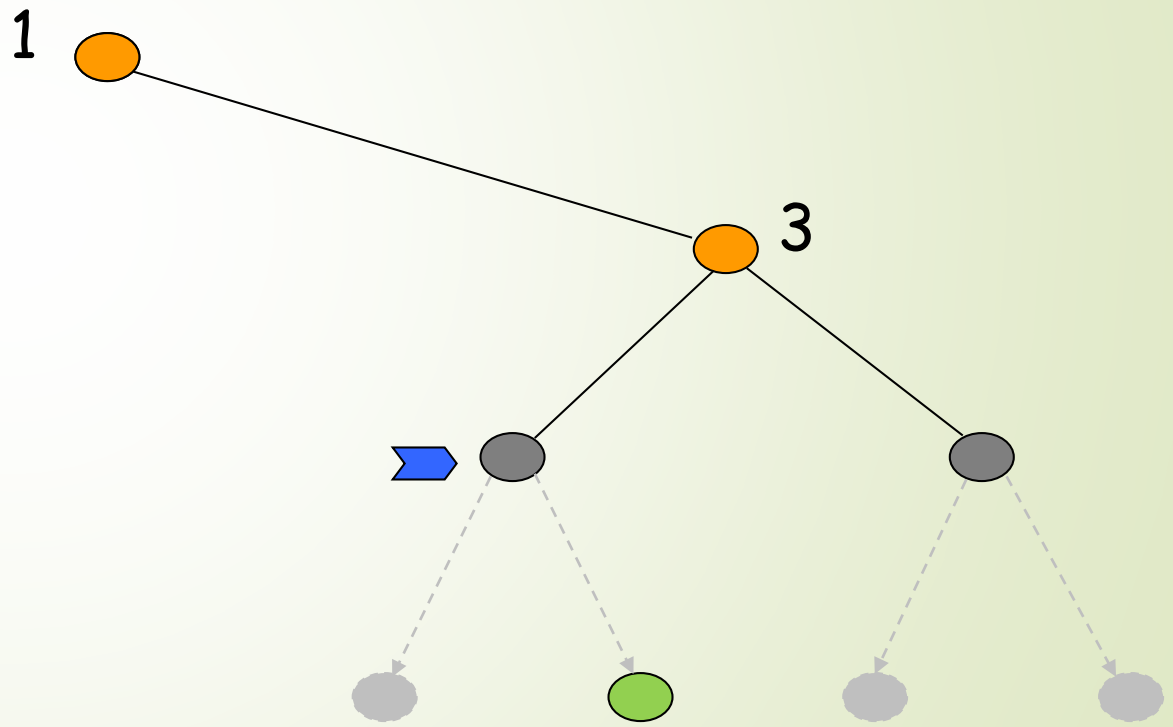
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



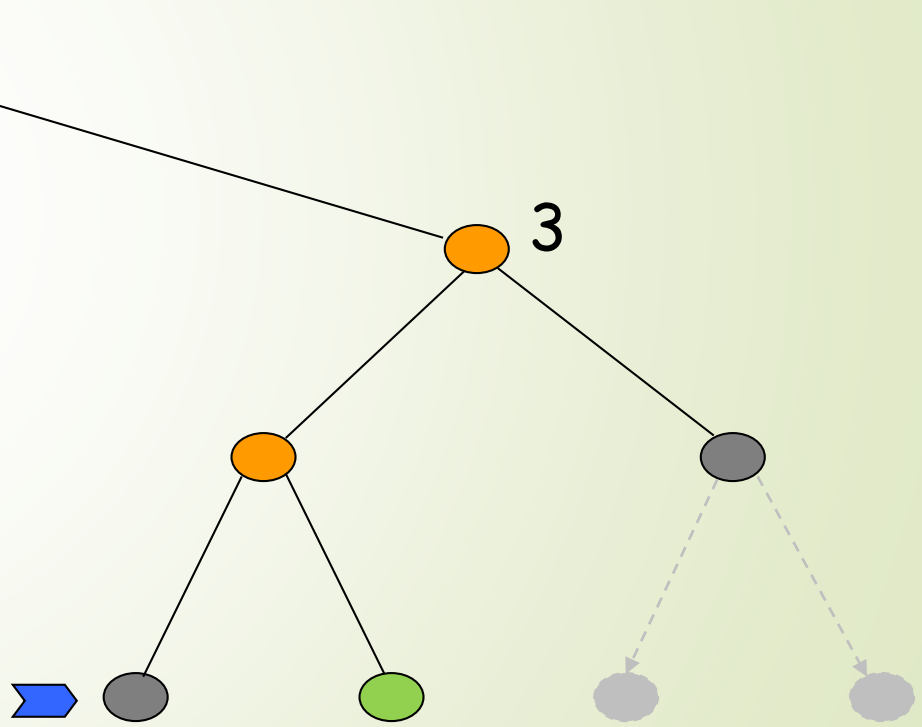
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



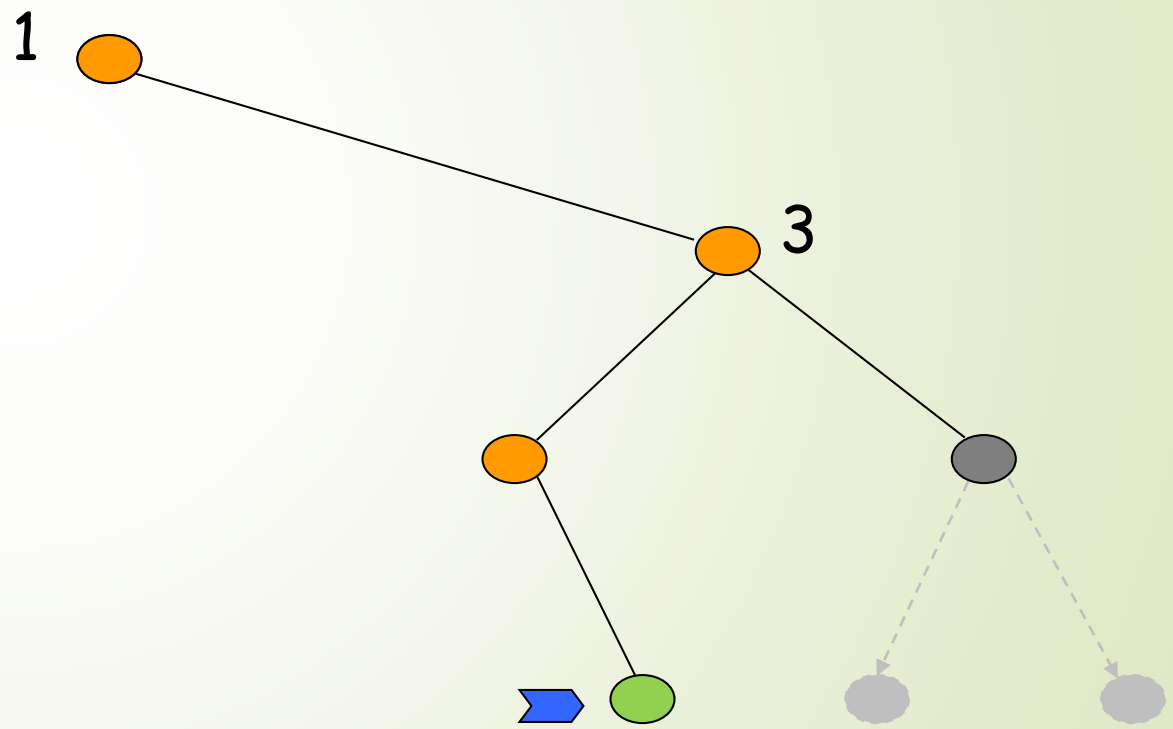
جستجوی اول عمق – DFS

- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.

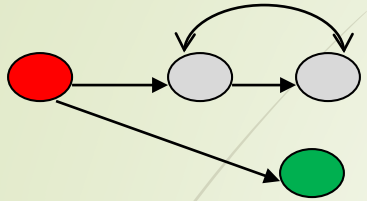


جستجوی اول عمق – DFS

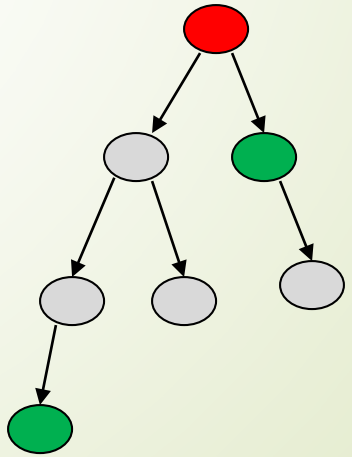
- عمیق‌ترین نود بسط نیافته در frontier را انتخاب می‌کند.
- پیاده‌سازی: frontier یک صف LIFO یا همان پشته است.



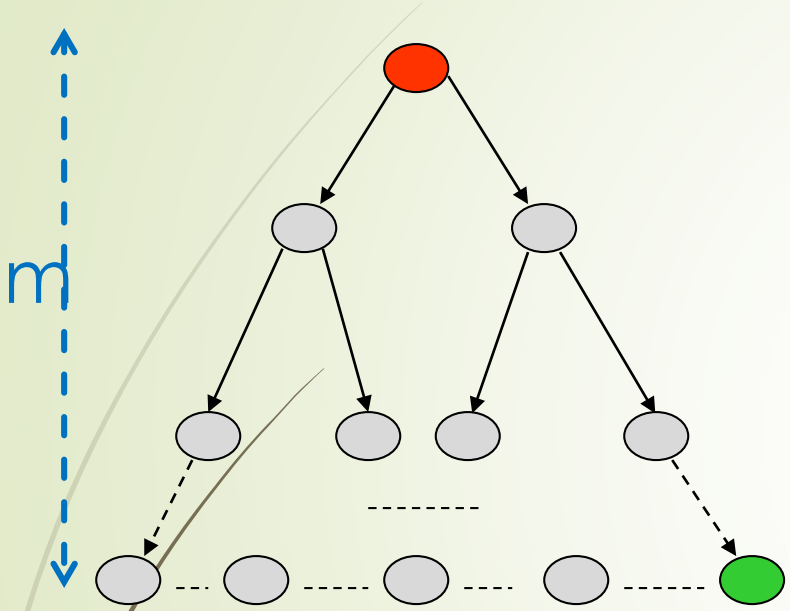
ارزیابی جستجوی اول عمق



- کامل؟
  - جستجوی درختی: به دلیل وجود حالات تکراری کامل نیست.
  - جستجوی گرافی: تنها در فضاهای حالت متناهی کامل است چون در نهایت تمام گره‌ها بسط می‌یابند.
  - شکست جستجوی اول عمق با هر دو نوع جستجو در فضاهای حالت نامتناهی
    - احتمال گیر کردن در یک مسیر نامتناهی بدون هدف
    - در حالی که جواب در عمق های کمتر وجود داشته است.
- بهینگی؟
  - خیر



ارزیابی جستجوی اول عمق



- پیچیدگی زمانی؟
  - جستجوی درختی: $O(b^m)$ مقدار m می‌تواند خیلی بزرگ‌تر از d باشد.
    - m عمق فضای حالت(شامل وضعیت های احتمالا تکراری)
    - d عمق کم عمق ترین نود هدف
  - جستجوی گرافی: با سایز فضای حالت محدود شده است.

- پیچیدگی فضایی؟
  - جستجوی درختی: $O(bm)$
    - تنها نیاز به نگه داشتن یک مسیر از ریشه تا گره‌ای دارد که در حال بررسی آن است به‌علاوه همزادهایی از نودهای موجود در این مسیر که هنوز گسترش نیافته‌اند.
  - جستجوی گرافی: همه‌ی نودها در مجموعه explored ذخیره می‌شوند.

تست

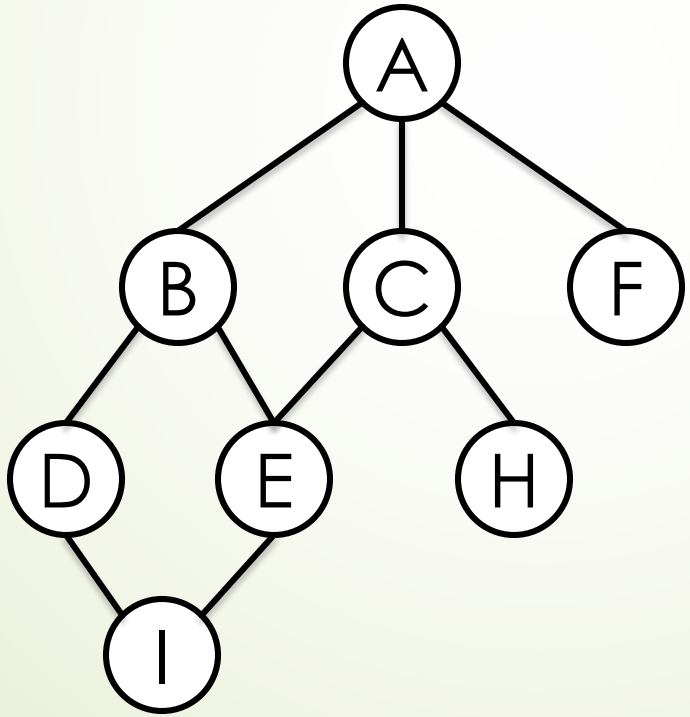
اگر در گراف زیر جستجو در عمق (Depth First Search) را از رأس C شروع کنیم، کدام گره‌ها به‌ترتیب از چپ به راست رویت می‌شوند؟ (فرض کنید فرزندان یک گره براساس ترتیب حروف الفبا انتخاب می‌شوند).

۱) A B C D E F H I

۲) C A B D I E F H ✓

۳) C A E H B F I D

۴) C A B D E H I F



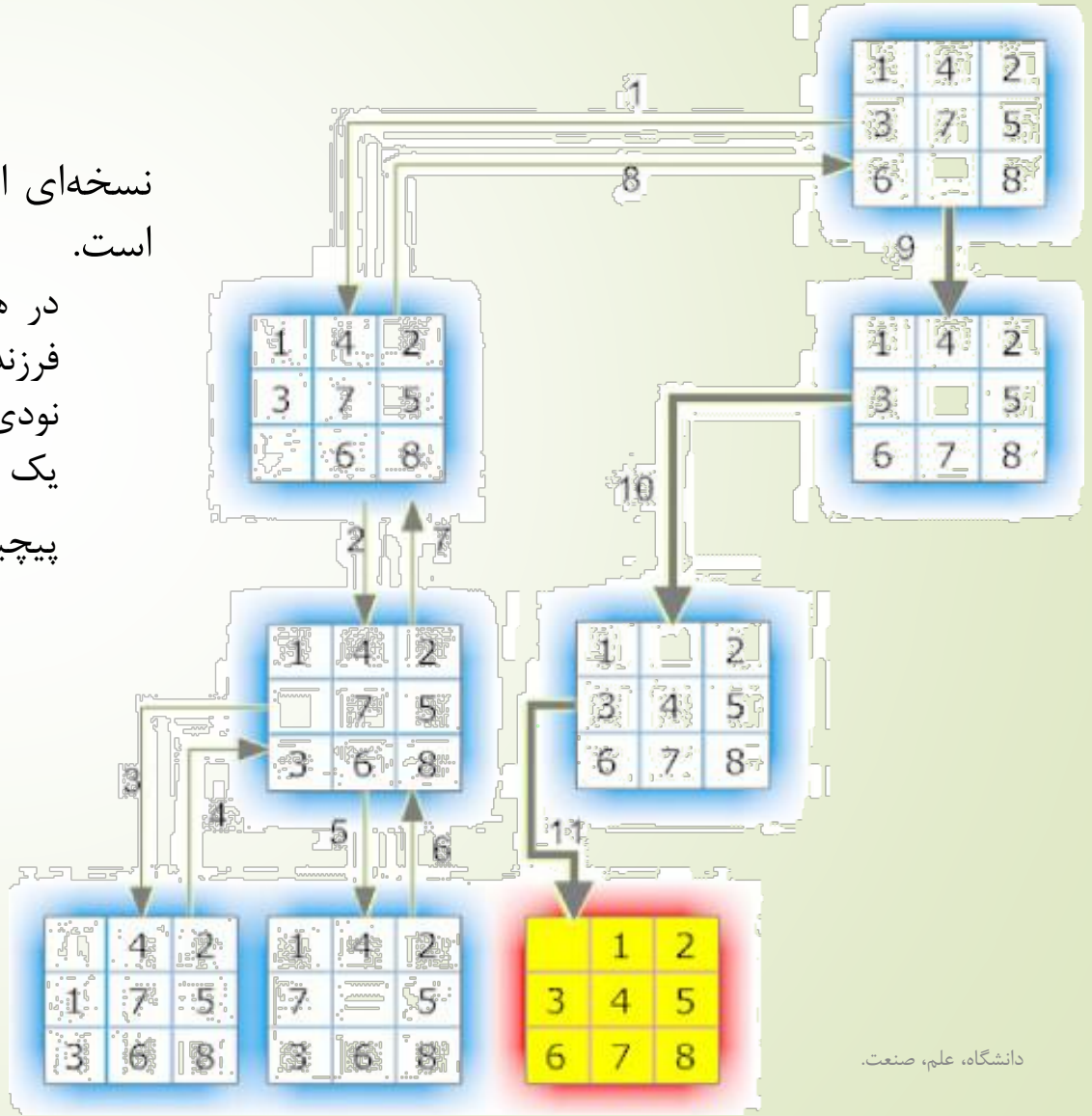
جستجوی عقب‌گرد

نسخه‌ای از جستجوی اول عمق جستجوی عقب‌گرد (Backtracking) است.

در هنگام بسط یک گره به‌جای تولید تمام فرزندان آن گره، هر بار فقط یکی از فرزندان آن تولید می‌شود.

نودی که به‌صورت جزئی بسط یافته است، به خاطر می‌سپارد که در مرحله‌ی بعد کدام یک از فرزندانش باید بسط یابد.

پیچیدگی فضایی: $O(m)$



جستجوی عمقی محدود شده – DLS
Depth-Limited Search

- انجام جستجوی عمقی با در نظر گرفتن یک محدودیت عمق از پیش تعریف‌شده مانند l
  - با گره‌هایی که در عمق l هستند به‌گونه‌ای رفتار می‌شود که گویی هیچ پسینی ندارند.
  - مسئله مسیر نامتناهی را حل می‌کند.

- در برخی مسائل، دانش مسئله می‌تواند برای تعیین l به‌کار رود.
  - برای مثال در مسئله جاده‌های رومانی می‌توان حداکثر عمق درخت جستجو را برابر با "قطر گراف فضای حالت" در نظر گرفت.
    - قطر گراف برابر با حداکثر تعداد یال‌های بین دو گره دلخواه در آن گراف است.
  - یافتن حداکثر عمق در بسیاری از مسائل ممکن نیست.

جستجوی عمقی محدود شده – DLS

- کامل؟
  - خیر، اگر $l<d$ باشد بدین معناست که کم‌عمق‌ترین هدف پایین‌تر از محدوده عمق قرار دارد.
- بهینگی؟
  - خیر، اگر $l>d$ باشد به دلیلی مشابه با DFS بهینه نیست.
- پیچیدگی زمانی؟

$$O(b^l)$$

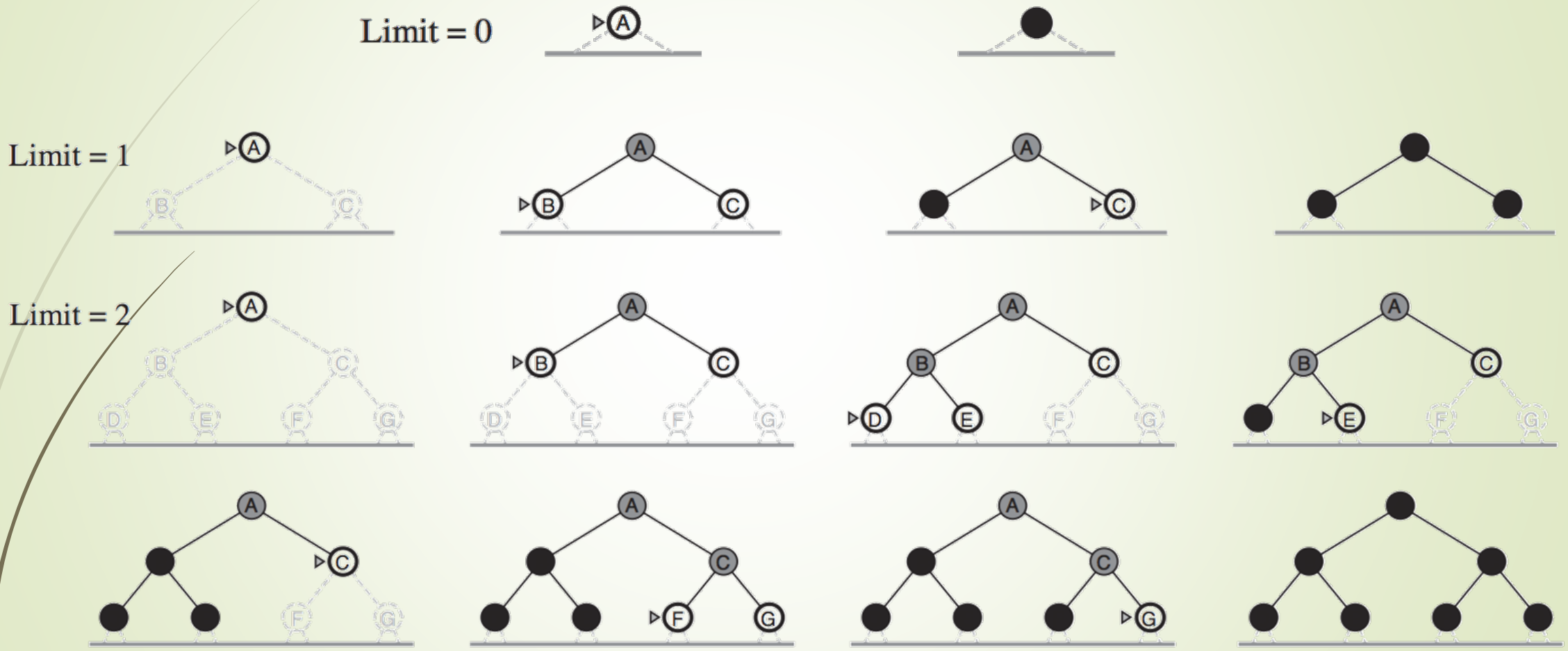
- پیچیدگی فضایی؟

$$O(bl)$$

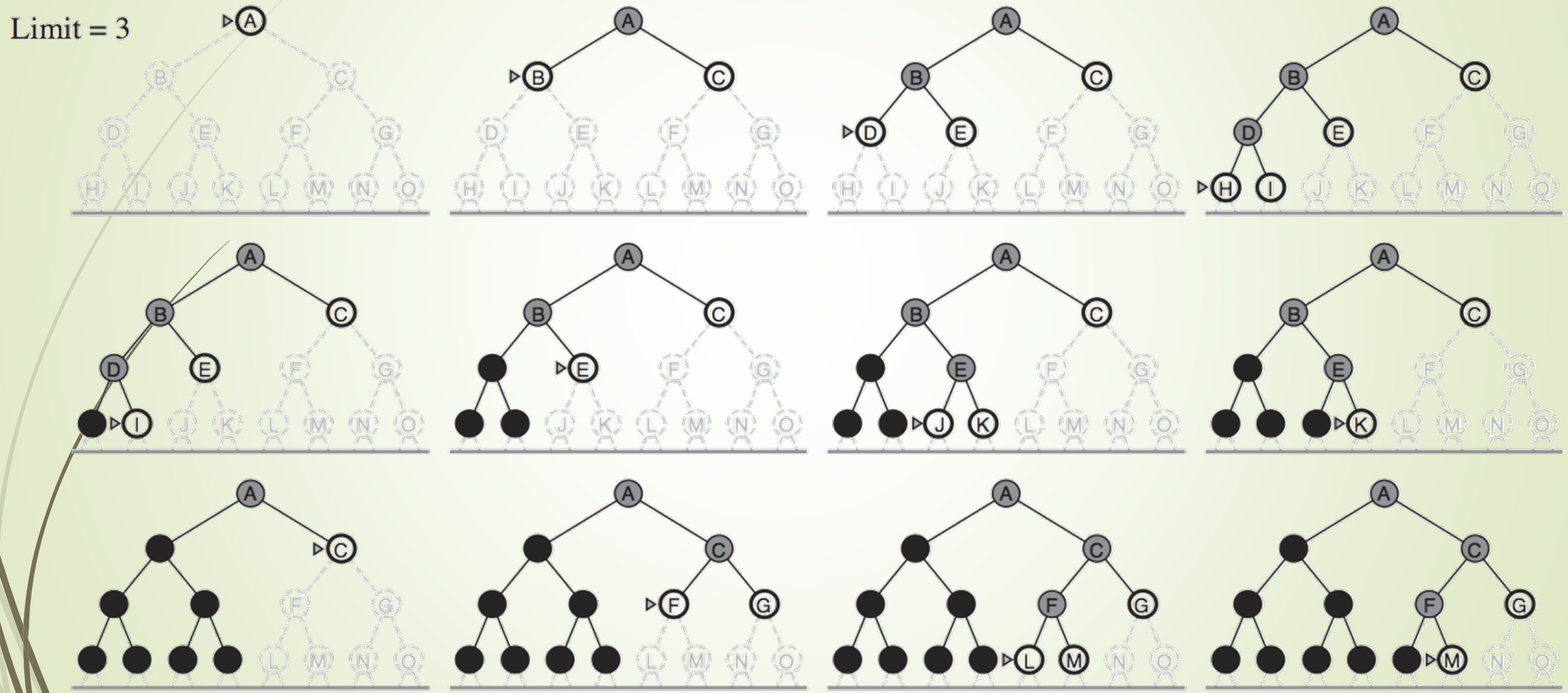
جستجوی عمقی تکراری– IDS
Iterative Deepening Search

- یک استراتژی کلی برای یافتن بهترین محدودیت برای عمق l ارائه می‌دهد.
  - هدف یافتن عمق d یا همان کم‌عمق‌ترین نود هدف است.
- از مزایای DFS و BFS بهره می‌برد.
  - DFS: نیاز به حافظه‌ی کم
  - BFS: کامل بودن - بهینگی
- سربار ؟

جستجوی عمقی تکراری– IDS



جستجوی عمقی تکراری– IDS



ارزیابی جستجوی عمقی تکراری

- کامل؟
  - بله اگر b و d متناهی باشند.
- پیچیدگی زمانی؟
  - الگوریتم به دلیل تولید مجدد حالات هزینه‌بر است.
  - نودهای تولید شده:
    - سطح d: یک بار
    - سطح d-1: دو بار
    - سطح d-2: سه بار
    - ...
    - سطح دو: d-1 بار
    - سطح یک: d بار

$$O(b^d)$$

$$N(IDS) = (d)b + (d-1)b^2 + \ldots + (1)b^d$$

$$N(BFS) = b + b^2 + \ldots + b^d$$

Num. Comparison for b=10 and d=5 solution at far right

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

$$N(BFS) = 10 + 100 + 1000 + 10000 + 100000 = 111110$$

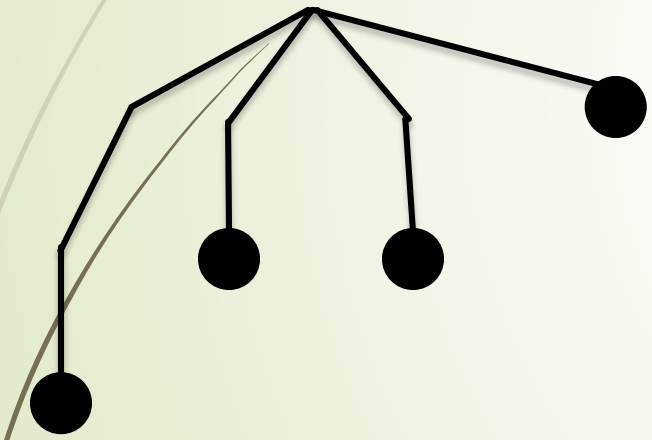
ارزیابی جستجوی عمقی تکراری

- پیچیدگی فضایی؟
  - مانند جستجوی اول عمق: $O(bd)$
- بهینگی؟
  - بله اگر هزینه مسیر یک تابع غیرکاهشی از عمق گره باشد.
  - برای مثال هزینه تمامی اعمال یکسان باشد.

- به‌طور کلی وقتی فضای جستجو بزرگ باشد و عمق راه‌حل شناخته‌شده نباشد، جستجوی IDS یک روش جستجوی ناآگاهانه مناسب یا preferred محسوب می‌شود.

تست

در حین انجام یک روش جستجو، درخت جستجوی حاصل به شکل مقابل رشد یافته است. رأس‌هایی که نامزد بسط داده شدن هستند با رنگ سیاه مشخص شده‌اند. این جستجو چه روشی می‌تواند باشد؟



۱) عمق نخست (Depth first)

۲) عرض نخست (Breath first)

✓ ۳) جستجوی هزینه یکنواخت (Uniform cost)

۴) تعمیق تکراری (Iterative deepening)

bidirectional – جستجوی دو طرفه

- انجام هم‌زمان دو جستجو: یکی از وضعیت اولیه به سمت هدف (forward یا روبه‌جلو) و دیگری از وضعیت هدف به سمت وضعیت اولیه (backward یا روبه‌عقب)
- امیدواریم که دو جستجو در میانه‌ی راه به همدیگر برسند.

$$b^{d/2} + b^{d/2} \neq b^{d}$$

- یکی یا هر دوی جستجوها، قبل از بسط هر گره بررسی می‌کند که آیا آن گره در مجموعه‌ی frontier جستجوی دیگر وجود دارد یا خیر؛ اگر چنین بود یک راه‌حل پیدا شده است.