



دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

حل مسئله از طریق جستجو

«هوش مصنوعی: یک رهیافت نوین»، فصل ۲

ارائه‌دهنده: سیده فاطمه موسوی

نیم‌سال اول ۱۳۹۹-۱۳۹۸

رئوس مطالب

- عامل‌های حل مسئله

- فرموله‌سازی مسئله

- مسائل نمونه

- جستجو برای راه‌حل‌ها

- جستجوی درختی و جستجوی گراف

- اندازه‌گیری کارایی حل مسئله

- الگوریتم‌های جستجوی پایه

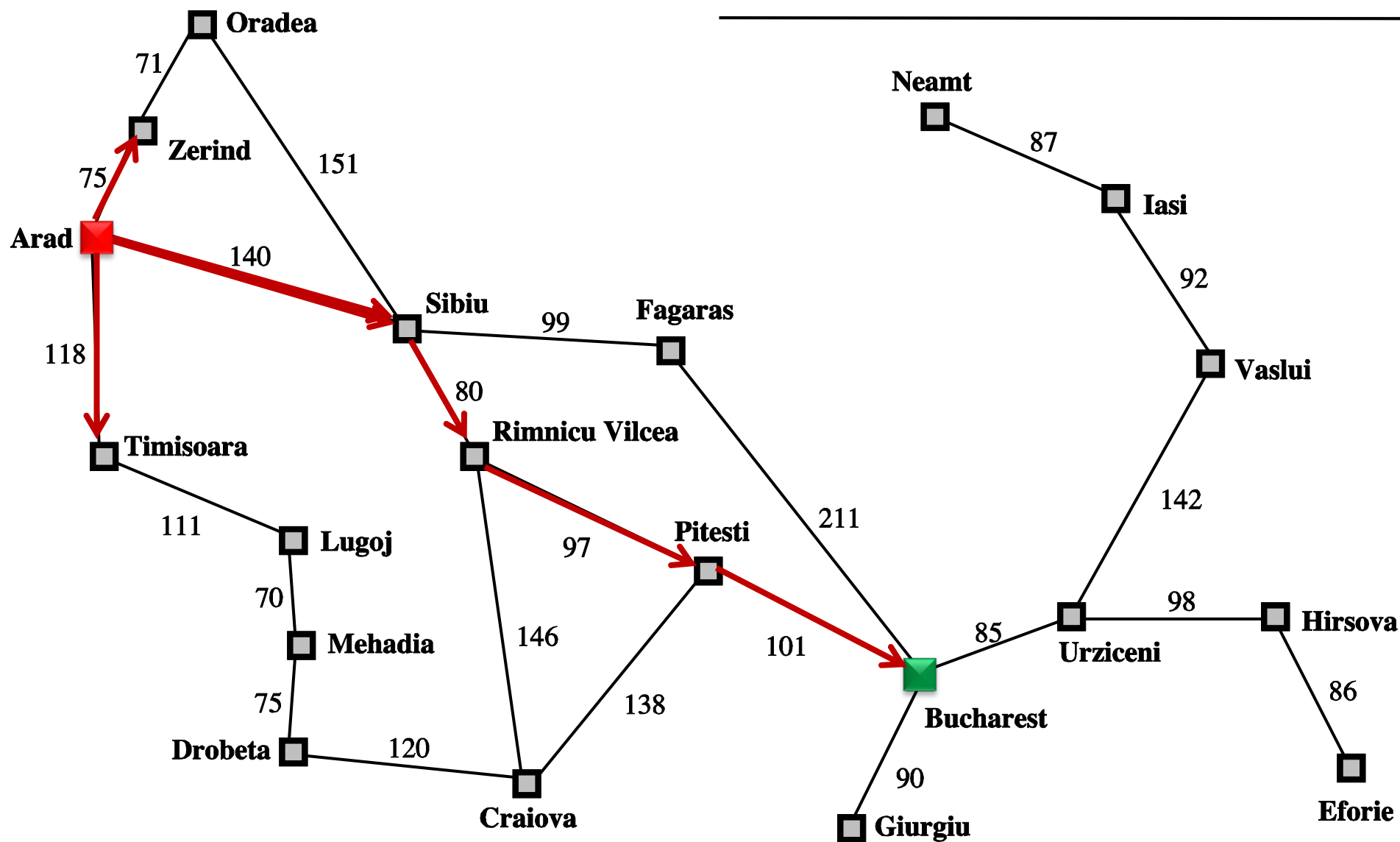
- راهبردهای جستجوی ناآگاهانه

- راهبردهای جستجوی آگاهانه

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

فرموله سازی مسائل

- جهان واقعی کاملاً پیچیده است.
- فرایند حذف جزئیات از یک بازنمایی برای حل مسئله را **تجريد** یا **انتزاع (abstraction)** می گویند.
- **حالت (انتزاعی) = مجموعه ای از حالات واقعی**
- حذف "همراهان در سفر"، "محل های 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 ...
- سطح مناسب انتزاع؟؟
- **معتبر بودن (valid)**: بتوانیم هر راه حل انتزاعی را به یک راه حل با جزئیات بیشتر بسط دهیم.
- **مفید بودن (useful)**: انجام هر یک از اعمال در راه حل آسان تر از مسئله ی اصلی باشد.

• یک عامل با چند گزینه‌ی بعدی از مقادیر نامعلوم، بدین شکل می‌تواند تصمیم بگیرد که چه کاری انجام دهد: در ابتدا توالی‌های مختلف ممکن از اقدامات را که به حالات با مقادیر معلوم منجر می‌شود بررسی کند و سپس بهترین آن‌ها را انتخاب کند.

• ویژگی‌هایی که فعلاً برای محیط در نظر خواهیم گرفت:

• شناخته‌شده (Known)

• داشتن یک نقشه از راه‌ها برای مسائل مسیریابی

• کاملاً قابل مشاهده (Observable)

• عامل همیشه وضعیت فعلی را بداند.

• قطعی (Deterministic)

• حاصل انجام یک عمل در یک وضعیت دقیقاً یک وضعیت مشخص باشد.

• گسسته (Discrete)

function SIMPLE-PROBLEM-SOLVING-AGENT (*percept*) **returns** an action
persistent: *seq*, an action sequence, initially empty
 state, some description of the current world state
 goal, a goal, initially null
 problem, a problem formulation

state \leftarrow UPDATE-STATE (*state*, *percept*) % Can search with closed eyes (open-loop)
if *seq* is empty **then** % Perceptions after each action provide no new information
 goal \leftarrow FORMULATE-GOAL (*state*)
 problem \leftarrow FORMULATE-PROBLEM (*state*, *goal*)
 seq \leftarrow SEARCH (*problem*)
action \leftarrow FIRST (*seq*)
seq \leftarrow REST (*seq*)
return *action*

مسائل و راه‌حل‌های خوش تعریف

- یک مسئله، به صورت 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 $s = \text{'at Bucharest'}$

- 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.

(فناوری اطلاعات ۱۸۹)

کدام عبارت برای حل یک مسئله با روش جستجو غلط است؟

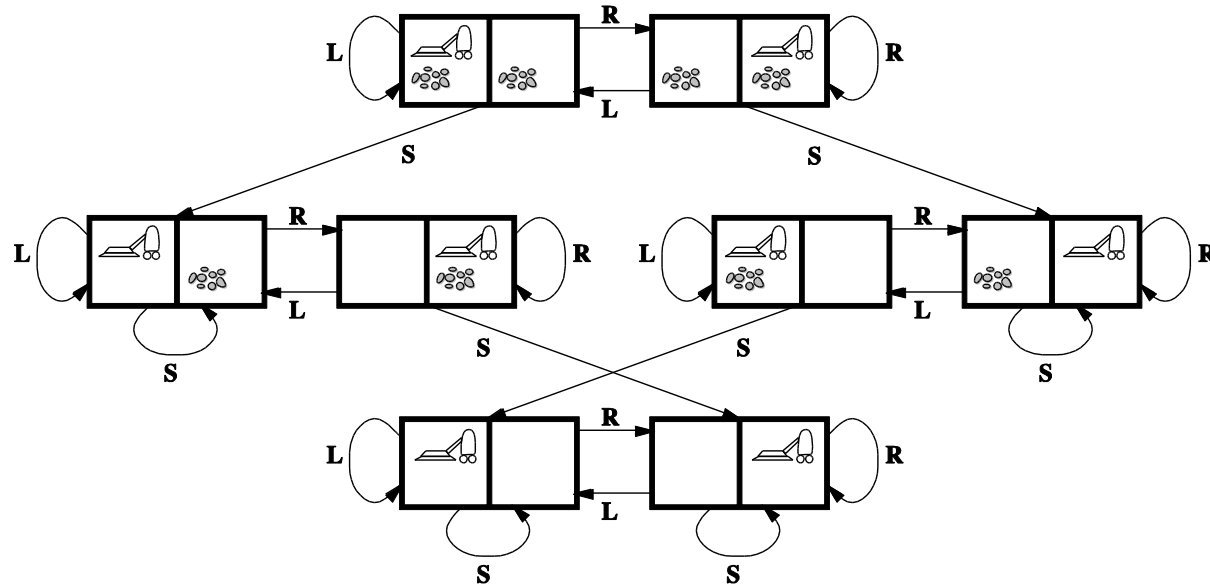
✓ (۱) حالت هدف باید مشخص باشد.

(۲) حالت‌های بعدی هر حالت باید مشخص باشند.

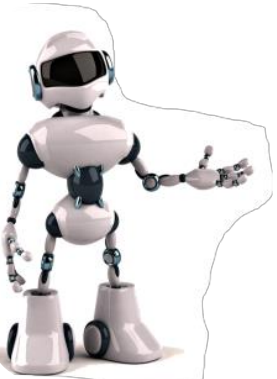
(۳) هزینه از یک حالت تا حالت بعدی باید مشخص باشد.

(۴) حالت شروع باید مشخص باشد.

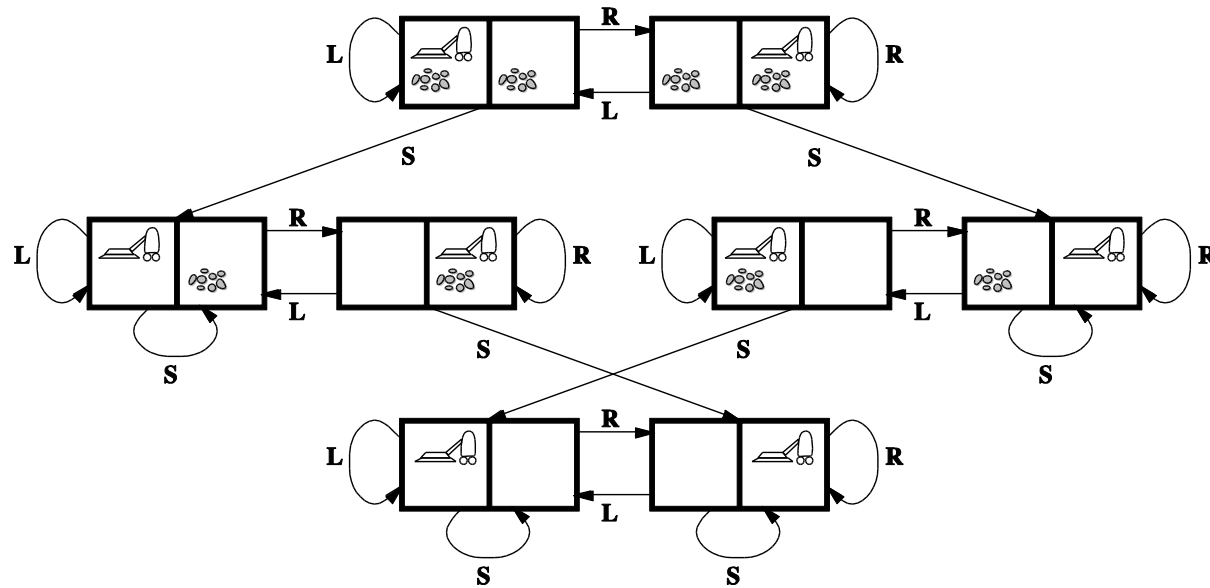
دنیای جاروبرقی



- 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 آن که دو محل تمیز هستند.

- **هزینه مسیر؟؟** تعداد اعمال برای رسیدن به هدف

پازل ۸ تایی

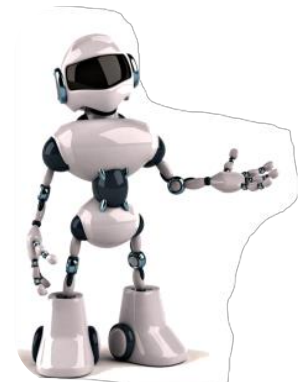
7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

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



پازل ۸ تایی

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

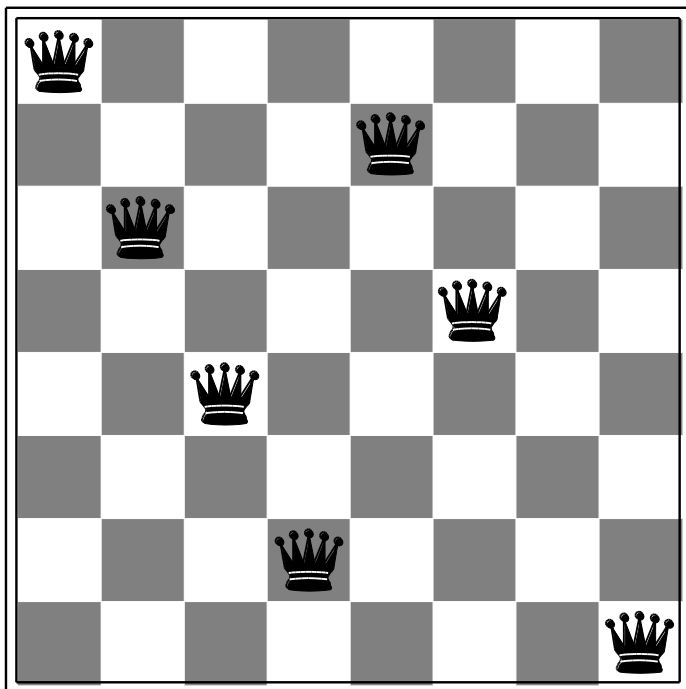
Goal State

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

- **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 آن که به یک حالت هدف مشخص رسیده است یا خیر.
- **هزینه مسیر؟؟** تعداد اعمال برای رسیدن به هدف

مسئله ۸ وزیر

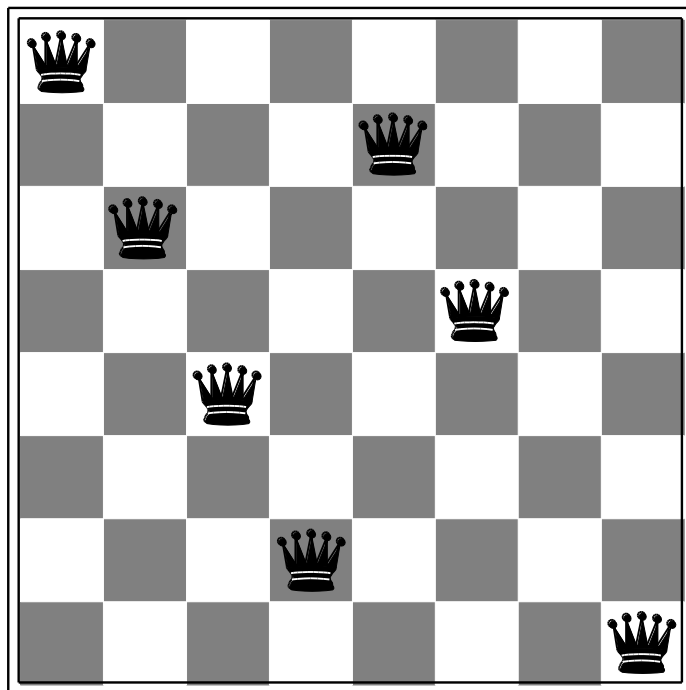
پیکربندی افزایشی در مقابل پیکربندی حالت کامل



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

مسئله ۸ وزیر

پیکربندی افزایشی

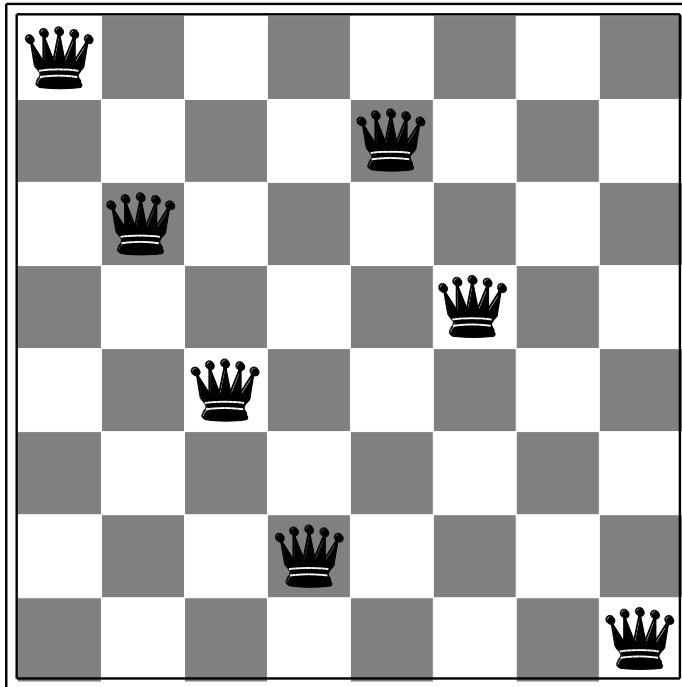


$$64 \times 63 \times \dots \times 57 \approx 1.8 \times 10^{14} \text{ states}$$

- **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

مسئله ۸ وزیر (پیکربندی دیگر)

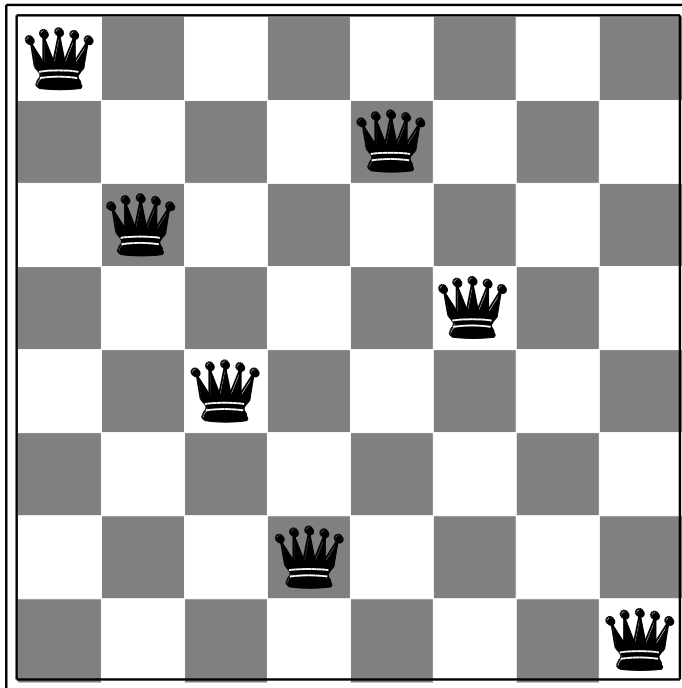
پیکربندی افزایشی



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

مسئله ۸ وزیر (پیکربندی دیگر)

پیکربندی افزایشی



2057 states

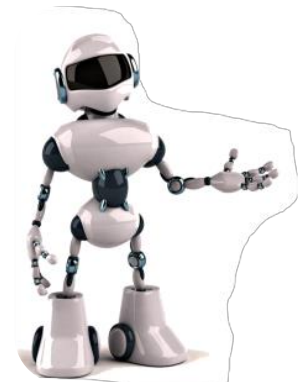
- **حالات؟؟** چیدمان ۰ تا ۸ وزیر در سمت چپ‌ترین ستون‌ها، به طوری که در هر ستون یک وزیر بوده و هیچ یک از وزیرها به دیگری حمله نکند.
- **حالت اولیه؟؟** صفحه بدون وزیر
- **اعمال؟؟** اضافه کردن وزیر در سمت چپ‌ترین ستون خالی به گونه‌ای که به دیگر وزیرها حمله نکند.
- **آزمون هدف؟؟** وجود ۸ وزیر بر روی صفحه
- **هزینه مسیر؟؟** اهمیت ندارد

مسئله Knuth

با شروع از عدد چهار و انجام یک دنباله از عملیات فاکتوریل، جذر و کف می‌توان به هر عدد صحیح مثبت دلخواه رسید.

$$\left[\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right] = 5$$

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



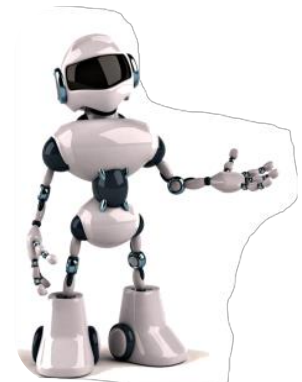
مسئله Knuth

با شروع از عدد چهار و انجام یک دنباله از عملیات فاکتوریل، جذر و کف می‌توان به هر عدد صحیح مثبت دلخواه رسید.

$$\left[\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right] = 5$$

فضای حالت نامتناهی

- **حالات؟؟** اعداد مثبت
- **حالت اولیه؟؟** عدد ۴
- **اعمال؟؟** انجام فاکتوریل، ریشه دوم یا کف (فاکتوریل تنها برای اعداد صحیح استفاده می‌شود).
- **آزمون هدف؟؟** حالت یک عدد صحیح دلخواه باشد.
- **هزینه مسیر؟؟** اهمیت ندارد



جستجوی راه حل ها

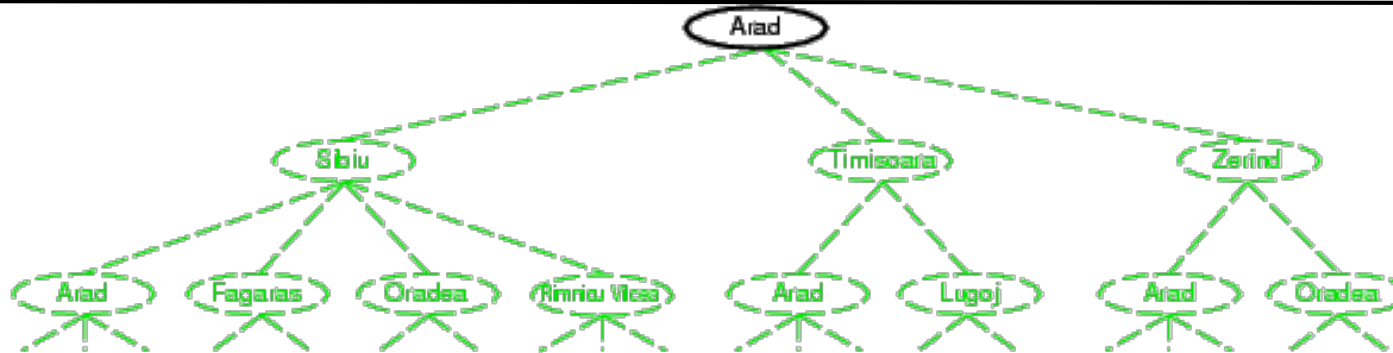
چگونه راه حل مسائل فرموله شده را بیابیم؟

الگوریتم جستجوی درختی

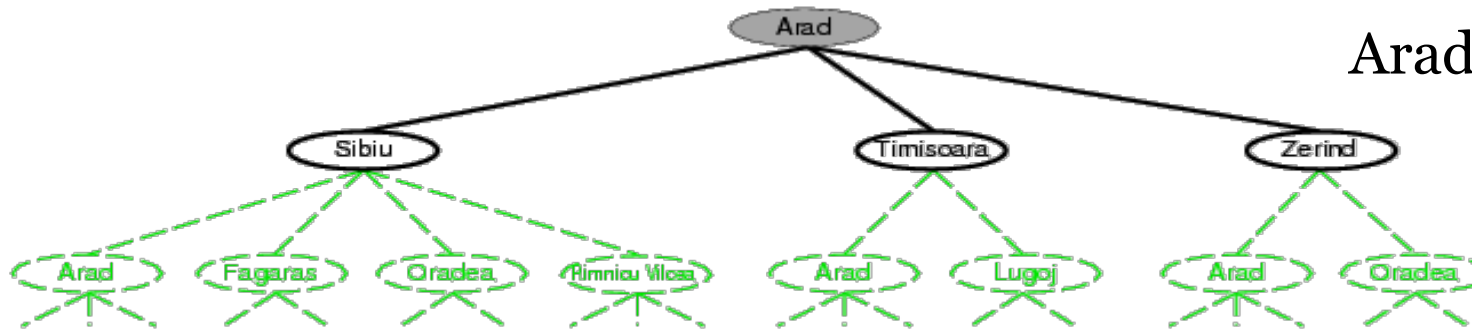
- ایده‌ی پایه
 - انجام جستجوی آفلاین و شبیه‌سازی‌شده‌ی فضای حالت با تولید پسین‌های حالاتی که قبلاً کاوش شده‌اند (یعنی حالات بسط یافته).
 - جستجوی آفلاین، یک راه‌حل را قبل از گام برداشتن در دنیای واقعی محاسبه می‌کند و سپس آن را اجرا می‌کند.
- جستجوی فضای حالت از طریق تولید صریح درخت
 - حالت اولیه در ریشه
 - شاخه‌ها به عنوان اعمال
 - گره‌ها متناظر با حالات در فضای حالت مسئله
- یادآوری: پیچیدگی فضا وابسته به نحوه‌ی نمایش حالت است.

نمونه‌ای از درخت جستجو

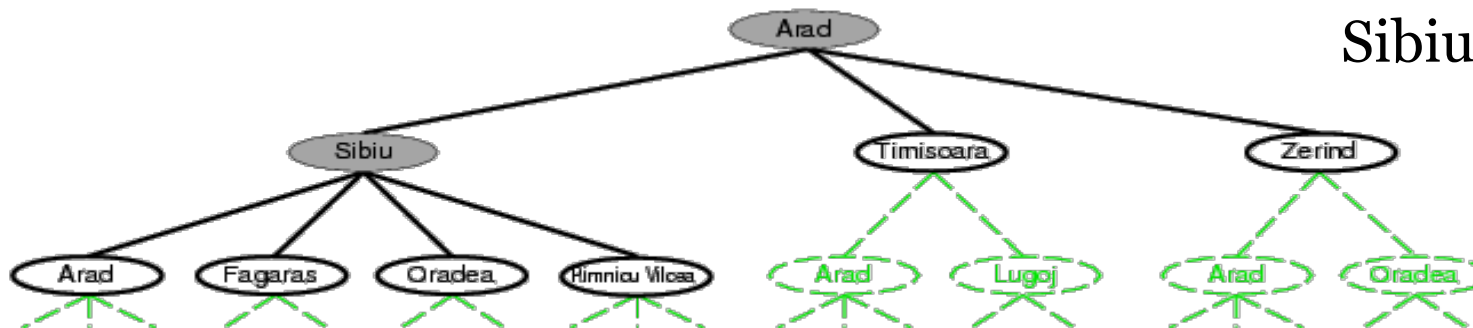
الف) حالت اولیه



ب) پس از بسط Arad

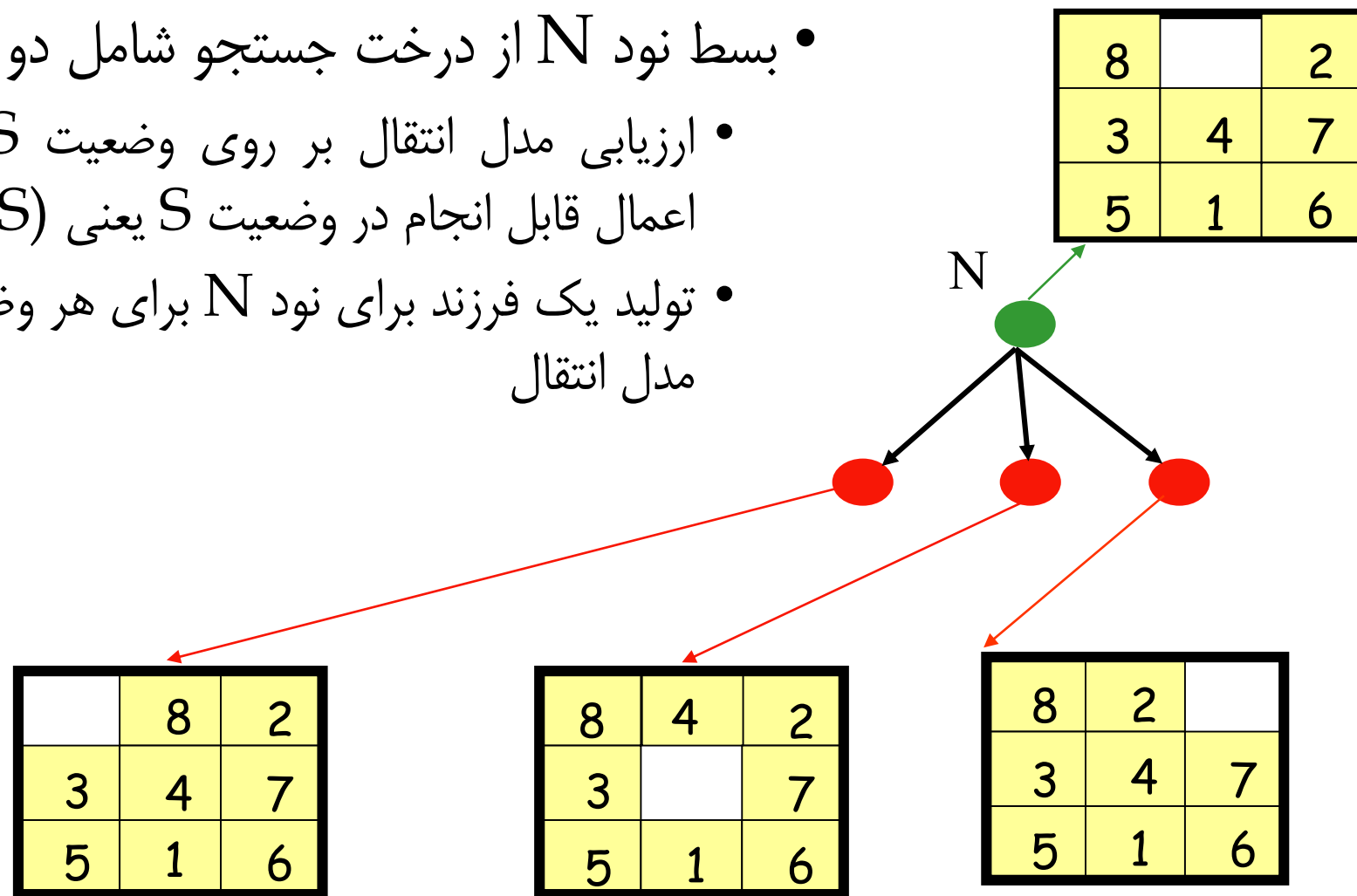


ج) پس از بسط Sibiu



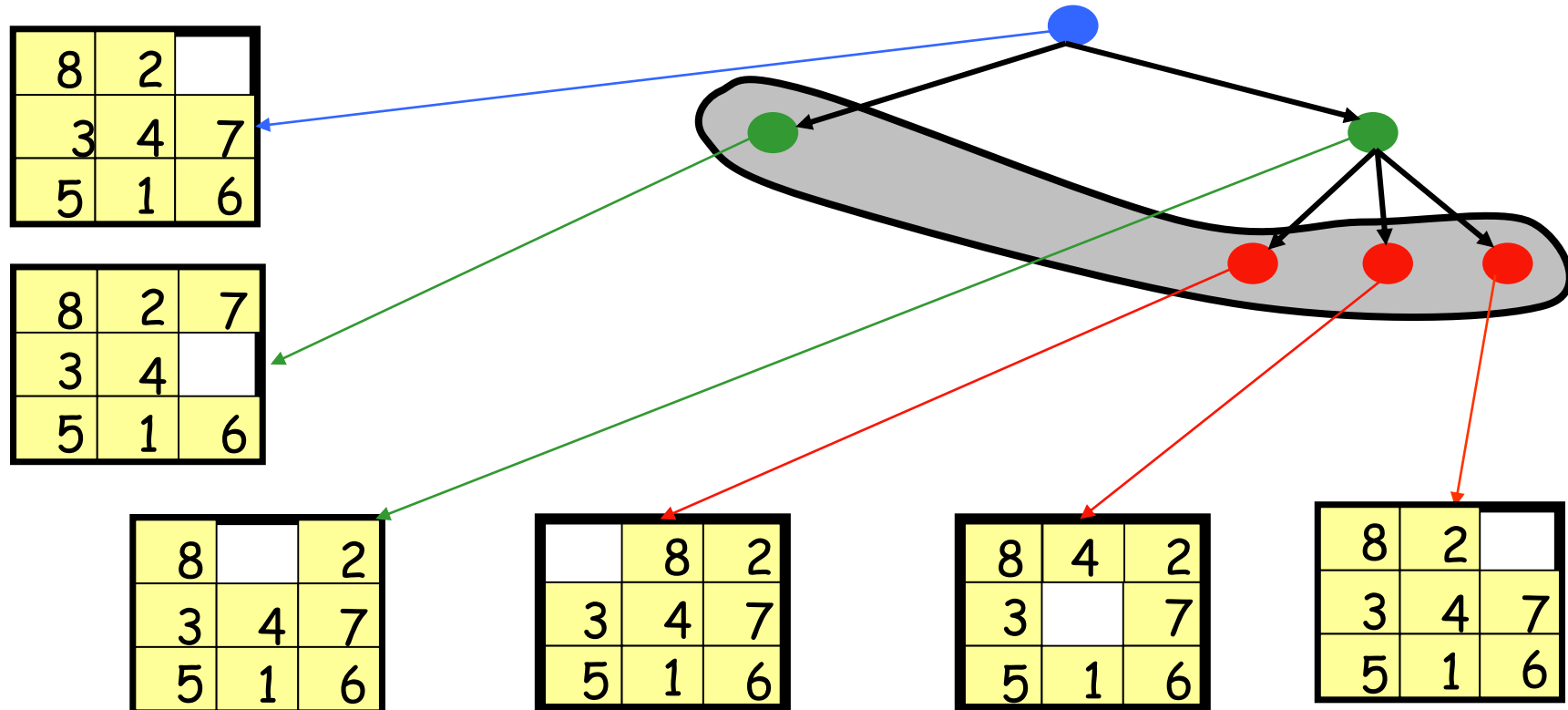
بسط نود

- 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 تا کنون 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

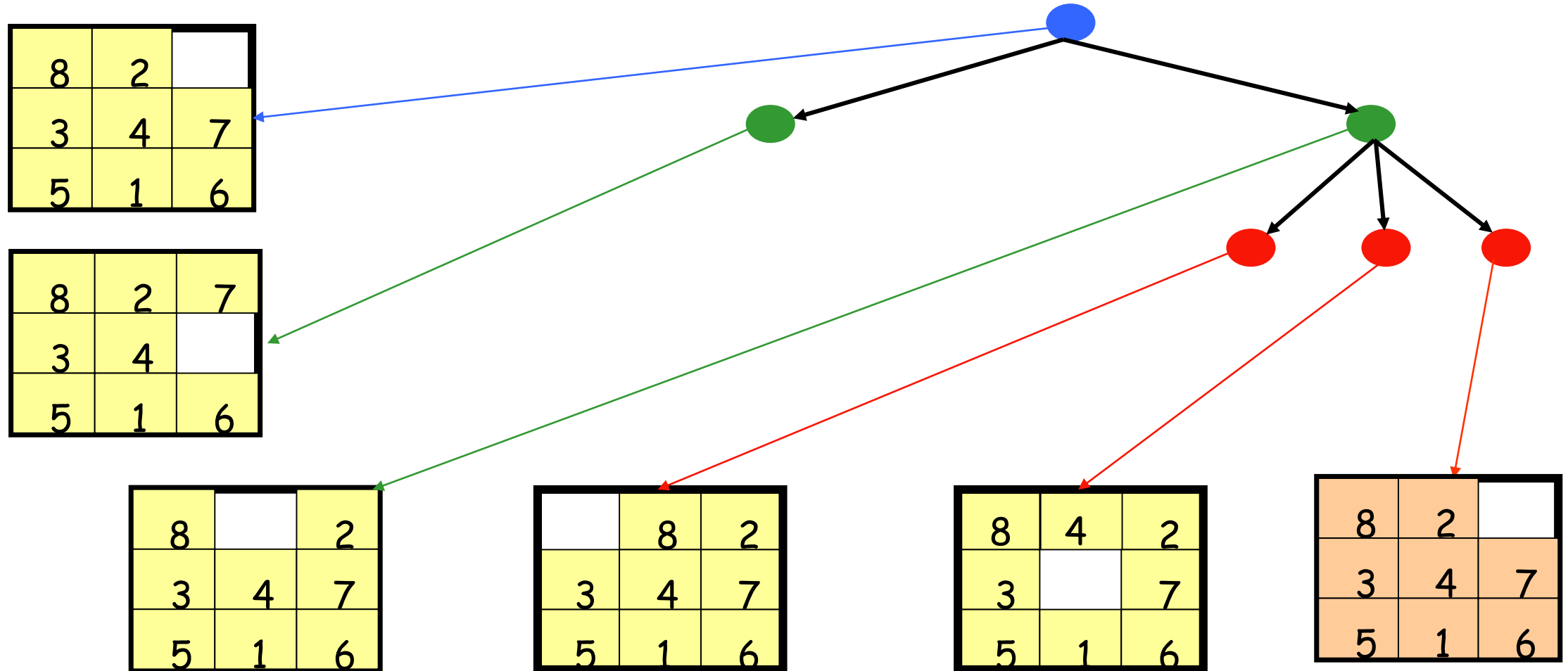
choose a leaf node and remove it from the **frontier** → Search Strategy

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 می کند کدام یک از نودها را در مرحله ی بعد برای بسط انتخاب کند.

مشکل درخت جستجو



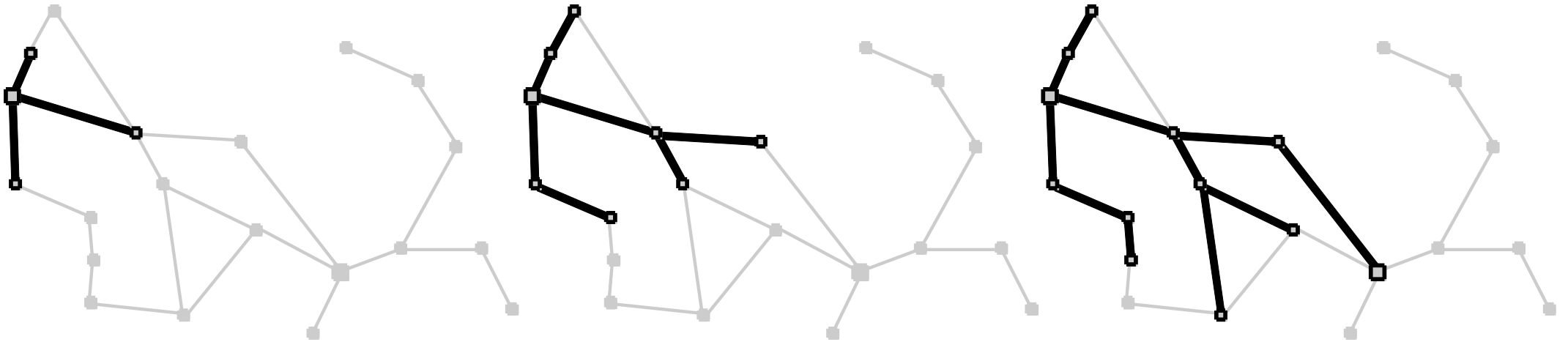
مشکل درخت جستجو (ادامه)

- **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.
- ممکن است به خاطر تعریف بد مسئله یا اساس مسئله باشد.
- برای مثال: فرموله کردن مسئله ۸ وزیر به گونه‌ای که یک وزیر بتواند در هر ستونی قرار گیرد.
- برای مثال: مسائل یافتن مسیر یا پازل‌های بلوک لغزنده
- می‌تواند باعث تبدیل یک مسئله حل‌شدنی به یک مسئله غیرقابل حل شود.
- راه‌حل: استفاده از یک **مجموعه کاوش‌شده (Explored set)** برای یادآوری هر حالت بسط یافته

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

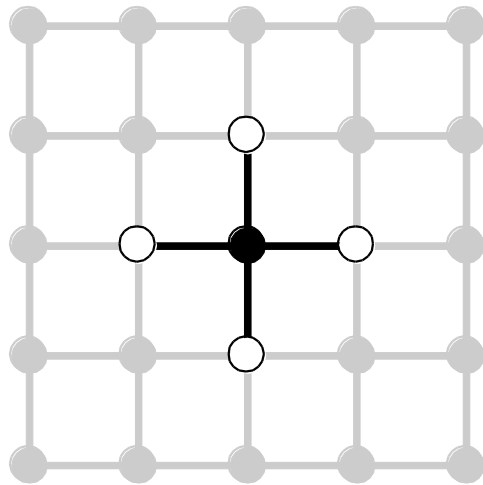
جستجوی گرافی – مثال اول

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

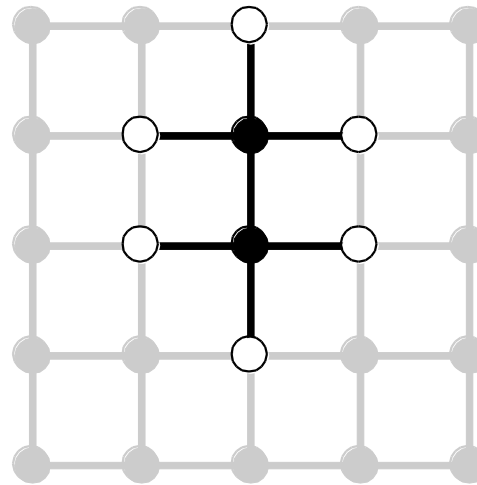


جستجوی گرافی – مثال دوم

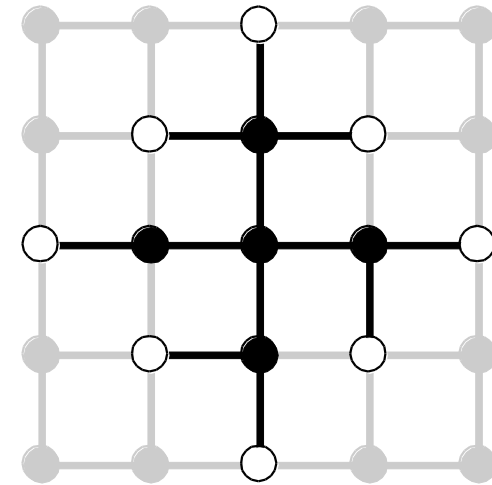
- یک درخت جستجو از عمق d که شامل حالات تکراری است 4^d برگ دارد، اما تنها حدود $2d^2$ حالت مجزا در d گام با شروع از هر حالت وجود دارد. (چرا؟)



(a)



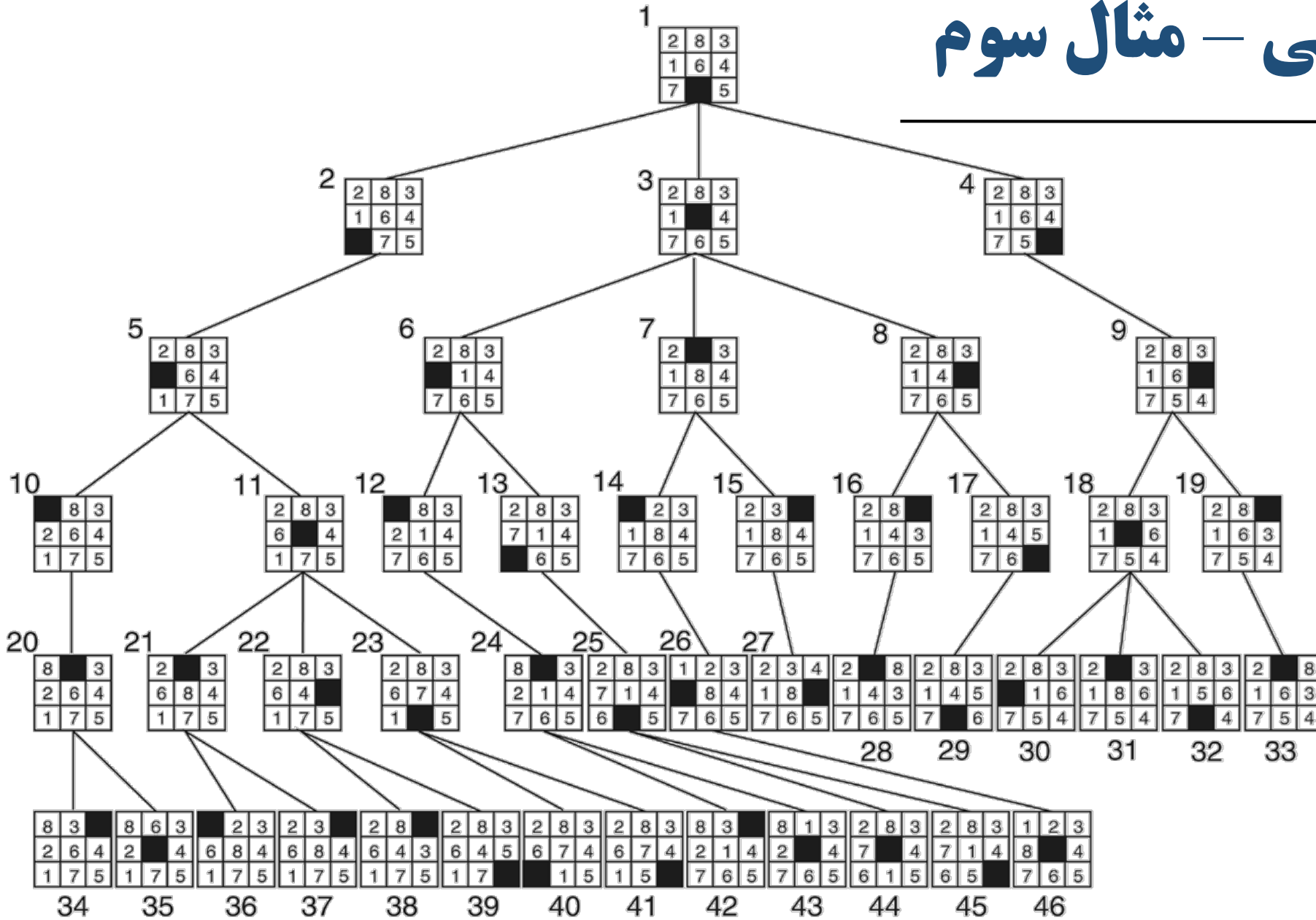
(b)



(c)

- ویژگی تفکیک GRAPH-SEARCH

جستجوی گرافی – مثال سوم



2	8	3
1	6	4
7		5

Start

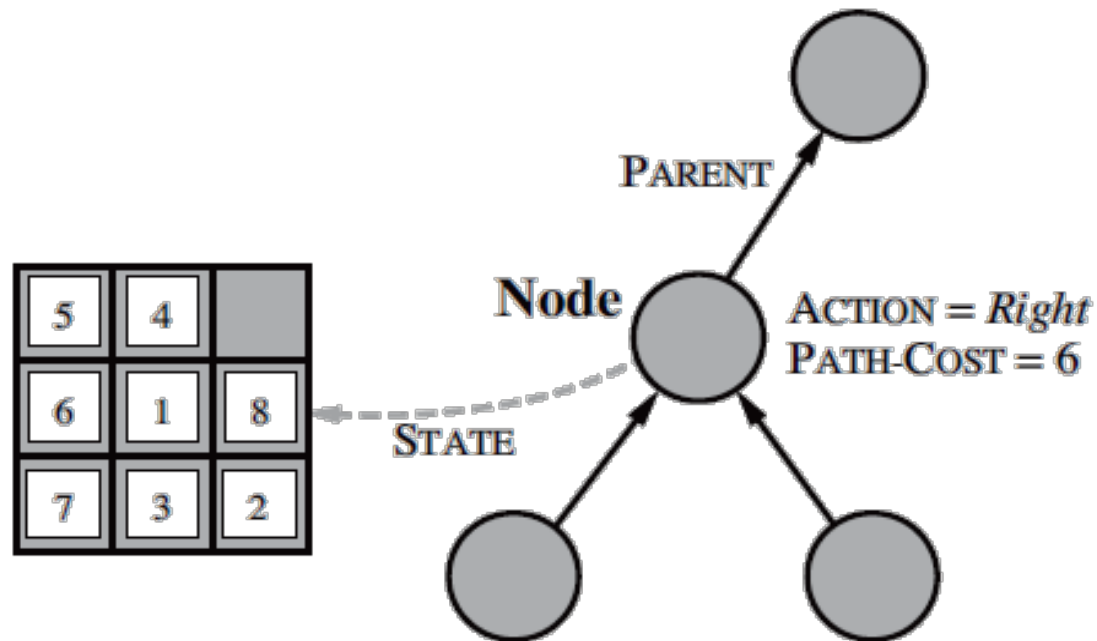
1	2	3
8		4
7	6	5

Goal

Goal

فضای حالت در مقابل درخت جستجو

- یک حالت (state) متناظر با یک پیکربندی از دنیا است.
- یک گره (node) ساختار داده‌ای متعلق به درخت جستجو است.



- آیا می‌توان دو گره با یک حالت یکسان داشت؟

ساختارهای لازم برای الگوریتم‌های جستجو

- محاسبه امان‌های اساسی برای نودهای فرزند

function CHILD-NODE(*problem, parent, action*) **returns** a node

return a node with

STATE = *problem.RESULT(parent.STATE, action)*,

PARENT = *parent*, ACTION = *action*,

PATH-COST = *parent.PATH-COST* + *problem.STEP-COST(parent.STATE, action)*

- ساختار داده مناسب برای هر کدام از مجموعه‌های مرزی و کاوش شده چیست؟

- مجموعه مرزی: صف مانند FIFO، LIFO و ..

- مجموعه کاوش شده: Hash table

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

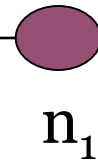
- راهبردهای ناآگاهانه یا کور (blind)
 - هیچ اطلاعات اضافه‌ای در مورد حالت به جز آنچه که در تعریف مسئله آمده است ندارند.
 - این راهبردها تنها قادر به تولید پسین‌ها براساس یک ترتیب مشخص و تشخیص حالت هدف از غیر هدف هستند.
 - تنها براساس مکان نودها در درخت جستجو عمل می‌کنند.
- راهبردهای آگاهانه یا هیوریستیک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

8	2	
3	4	7
5	1	6

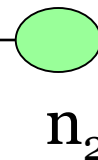
STATE



n_1

1	2	3
4	5	
7	8	6

STATE



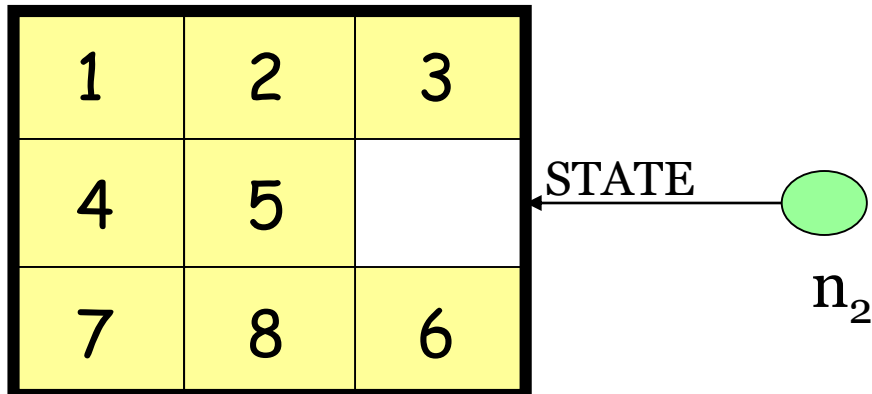
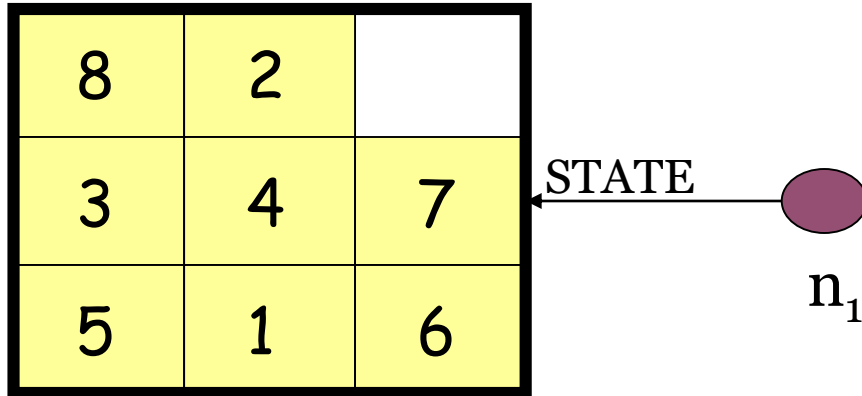
n_2

1	2	3
4	5	6
7	8	

Goal state

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	

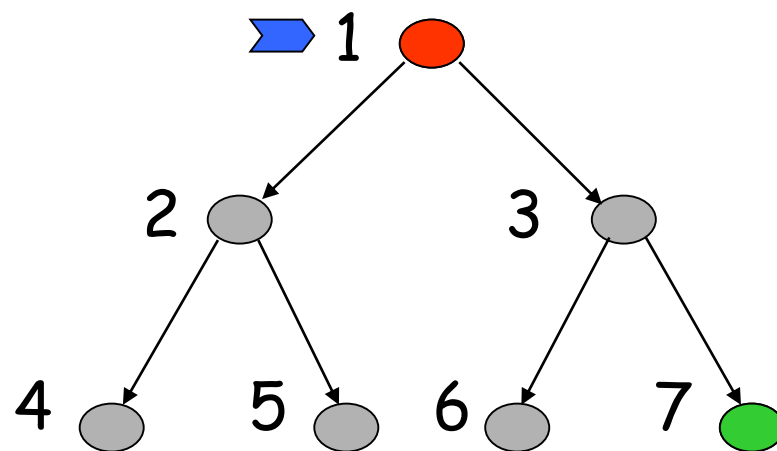
Goal state

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 عمیق شونده‌ی تکراری (Iterative deepening search)
 - جستجوی دوطرفه (Bidirectional search)

جستجوی اول سطح – BFS

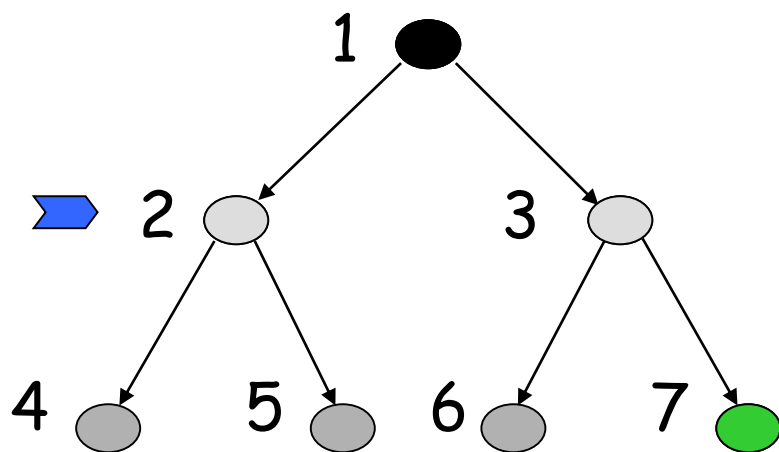
- بسط کم عمق ترین گره بسط نیافته
- آزمون هدف به هنگام تولید نود اعمال می شود نه هنگام انتخاب آن برای بسط
- پیاده سازی: مجموعه ی مرزی یک صف FIFO خواهد بود.
- گره های جدید در انتهای صف قرار می گیرند.



Frontier = {1}
Explorer = {}

جستجوی اول سطح – BFS

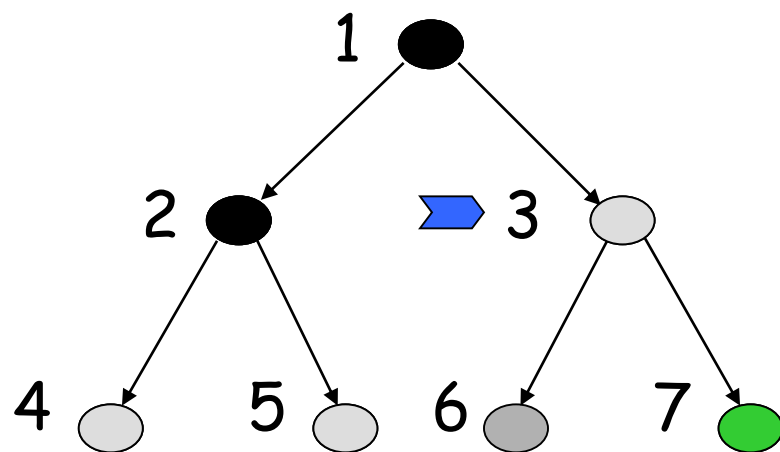
- بسط کم عمق ترین گره بسط نیافته
- آزمون هدف به هنگام تولید نود اعمال می شود نه هنگام انتخاب آن برای بسط
- پیاده سازی: مجموعه ی مرزی یک صف FIFO خواهد بود.
- گره های جدید در انتهای صف قرار می گیرند.



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

جستجوی اول سطح – BFS

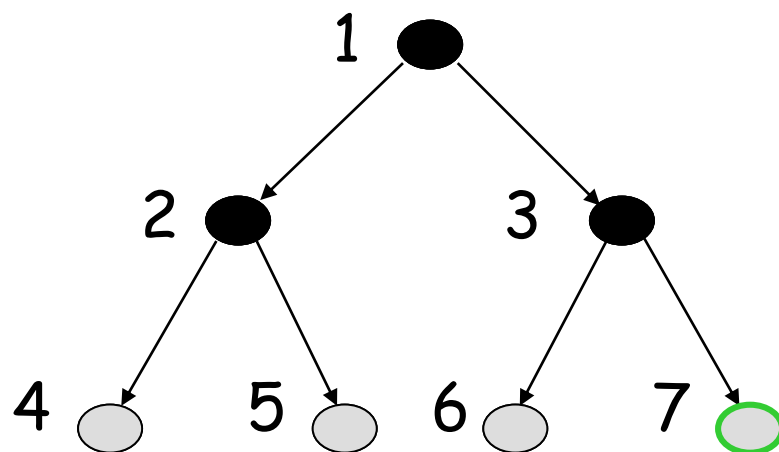
- بسط کم عمق ترین گره بسط نیافته
- آزمون هدف به هنگام تولید نود اعمال می شود نه هنگام انتخاب آن برای بسط
- پیاده سازی: مجموعه ی مرزی یک صف FIFO خواهد بود.
- گره های جدید در انتهای صف قرار می گیرند.



Frontier = {3, 4, 5}
Explorer = {1, 2}

جستجوی اول سطح – BFS

- بسط کم عمق ترین گره بسط نیافته
- آزمون هدف به هنگام تولید نود اعمال می شود نه هنگام انتخاب آن برای بسط
- پیاده سازی: مجموعه ی مرزی یک صف FIFO خواهد بود.
- گره های جدید در انتهای صف قرار می گیرند.



Frontier= {4, 5, 6, 7}
Explorer= {1,2,3}

```
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

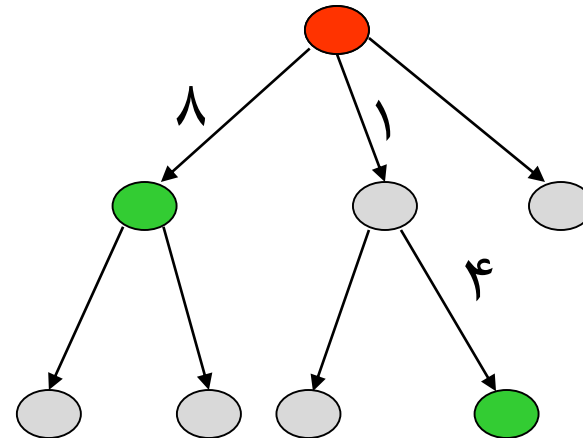
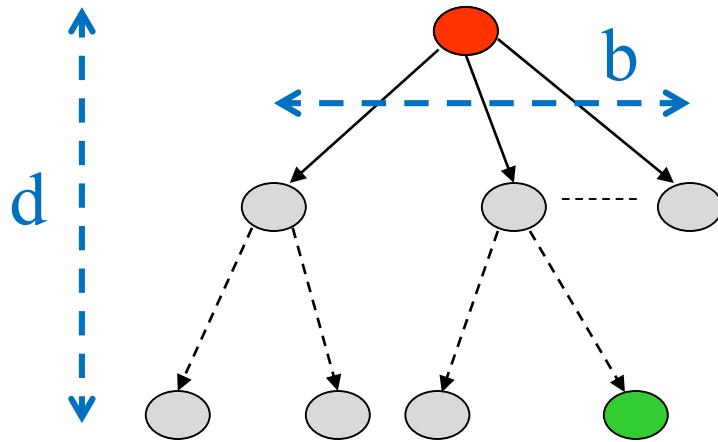
- کامل؟

- 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: $b + b^2 + b^3 + \dots + b^d = O(b^d)$

- هنگام انتخاب نود برای بسط: $b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b) = O(b^{d+1})$

- 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 کاوش شده + فضای مجموعه مرزی

- جستجوی گرافی: $O(b^{d-1}) + O(b^d) = O(b^d)$

- جستجوی درختی فضای بیشتری استفاده نمی کند در حالی که ممکن است باعث شود زمان اضافه تر زیادی داشته باشد.

زمان و حافظه مورد نیاز BFS

- در جستجوی اول سطح نیازمندی‌های حافظه مشکل بزرگ‌تری نسبت به زمان اجرا است.
- در حالت کلی، مسائل جستجوی با پیچیدگی نمایی به جز برای نمونه‌های کوچک توسط روش‌های ناآگاهانه قابل حل نیستند.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

فرض کنید در هر ثانیه یک میلیون نود بتواند تولید شود و هر نود نیاز به ۱۰۰۰ بایت از حافظه دارد.

تست ۲

فرض کنید برای مسئله‌ای با جستجوی اول پهنا (breadth-first) و تست هدف در لحظه تولید نیاز به بسط دادن ۳۲ گره باشد. اگر فاکتور انشعاب درخت جستجو ثابت باشد و عمق درخت برابر ۵ و عمق هدف برابر ۴ باشد، کدام یک از گزینه‌های زیر مقدار فاکتور انشعاب (b) را نشان می‌دهد؟ (فرض کنید ریشه‌ی درخت در عمق صفر واقع شده است.)

(مهندسی کامپیوتر دولتی ۹۱)

(۱) $b=2$

(۲) $b>5$

(۳) $2<b<3$

(۴) $3\leq b\leq 5$ ✓

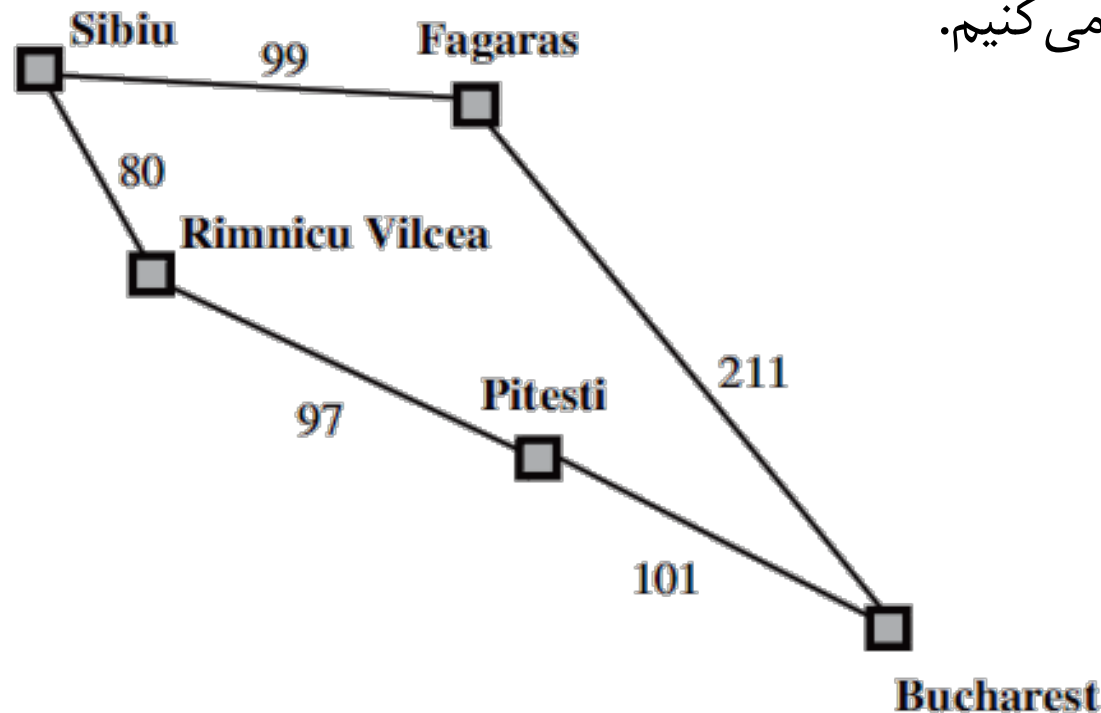
جستجوی هزینه یکنواخت – UCS

- نود n با کمترین هزینه $g(n)$ را بسط می‌دهد.
- هر یال دارای هزینه $c > 0$ است.
- $g(n) = \sum \text{costs of arcs}$
- 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.
- فرض کنید نودی تولید شده باشد که وضعیت آن نود قبلاً 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.

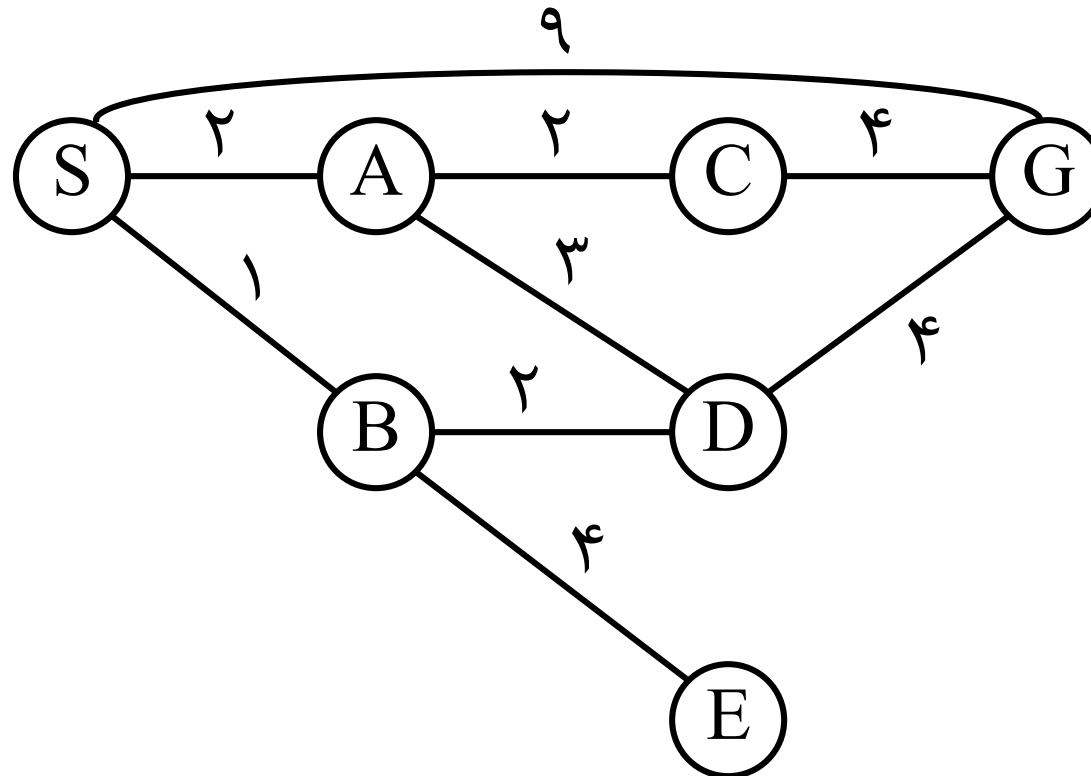


الگوریتم جستجوی هزینه یکنواخت

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure
node \leftarrow a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier \leftarrow a priority queue ordered by PATH-COST, with *node* as the only element
explored \leftarrow an empty set
loop do
 if EMPTY?(*frontier*) **then return** failure
 node \leftarrow 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 \leftarrow CHILD-NODE(*problem*, *node*, action)
 if *child*.STATE is not **in** *explored* or *frontier* **then**
 frontier \leftarrow INSERT(*child* , *frontier*)
 else if *child*.STATE is **in** *frontier* with higher PATH-COST **then**
 replace that *frontier* node with *child*

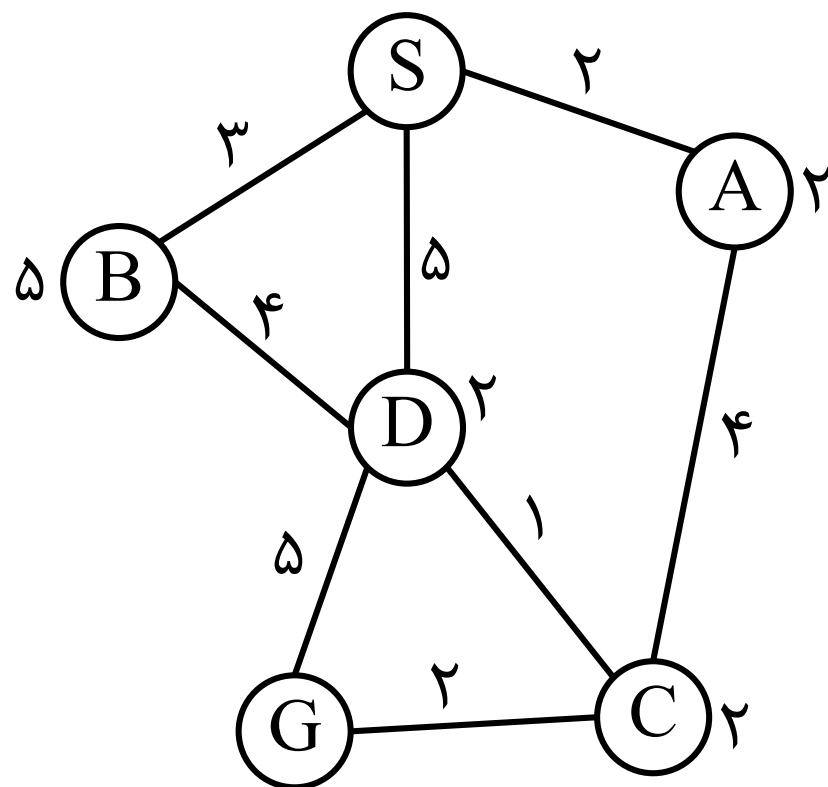
جستجوی هزینه یکنواخت – تمرین

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 داده شده بر روی یال‌ها هزینه‌ی هر عمل را نشان می‌دهد. مسیر برگردانده شده توسط الگوریتم جستجوی گرافی هزینه یکنواخت را به دست آورید.



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

(کامپیوتر ۹۵)



۱) S, A, B, D, G

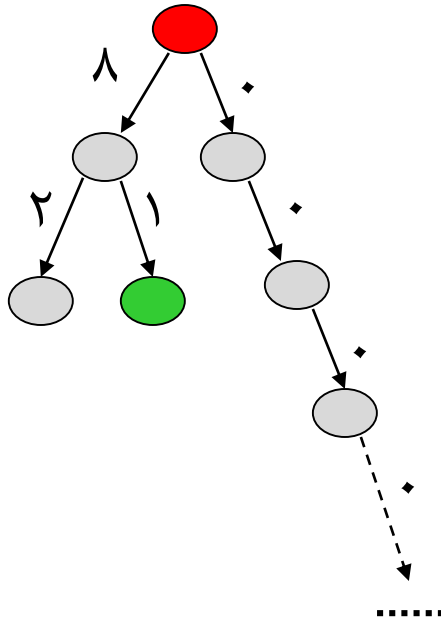
۲) S, A, B, C, D, G

۳) S, A, B, D, C, G ✓

۴) S, A, D, C, G

ارزیابی جستجوی هزینه یکنواخت

- کامل؟
- بله اگر $\text{step-cost} \geq \varepsilon > 0$ باشد تا از یک دنباله نامتناهی از اعمال با هزینه صفر اجتناب شود.



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

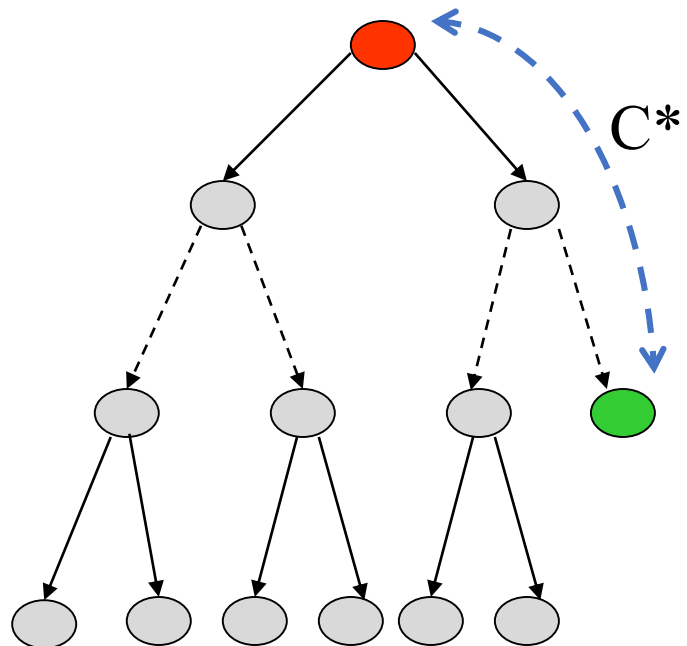
ارزیابی جستجوی هزینه یکنواخت

- پیچیدگی زمانی؟

- فرض کنید C^* هزینه‌ی راه‌حل بهینه بوده و هر عمل حداقل ϵ هزینه داشته باشد.

- بدترین حالت: $O(b^{1+\lceil C^*/\epsilon \rceil})$

- هنگامی که تمامی هزینه‌ها یکسان باشد: $O(b^{d+1})$



- پیچیدگی فضایی؟

- مشابه با پیچیدگی زمانی: $O(b^{1+\lceil C^*/\epsilon \rceil})$

اثبات بهینگی جستجوی هزینه یکنواخت

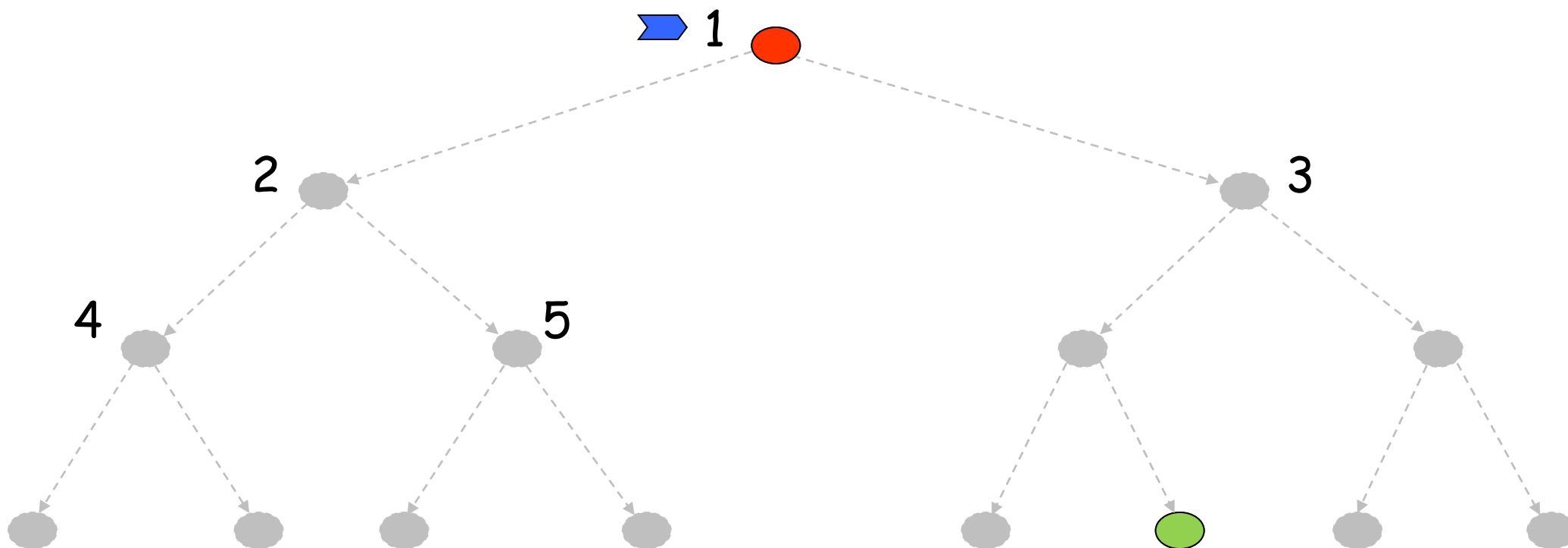
• لم: هرگاه UCS نود n را برای بسط انتخاب کند هزینه‌ی بهینه تا آن نود را یافته است.

اثبات با برهان خلف: فرض کنید هنگامی که نود n برای بسط انتخاب می‌شود، مسیر بهینه از ریشه تا n به دست 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 مبنی بر غیربهینه بودن مسیر حاصل شده تا n غلط است.

← الگوریتم UCS نودها را به ترتیب هزینه‌ی مسیر بهینه‌شان بسط می‌دهد. از این رو اولین نود هدف انتخابی برای بسط باید دارای راه حل بهینه باشد.

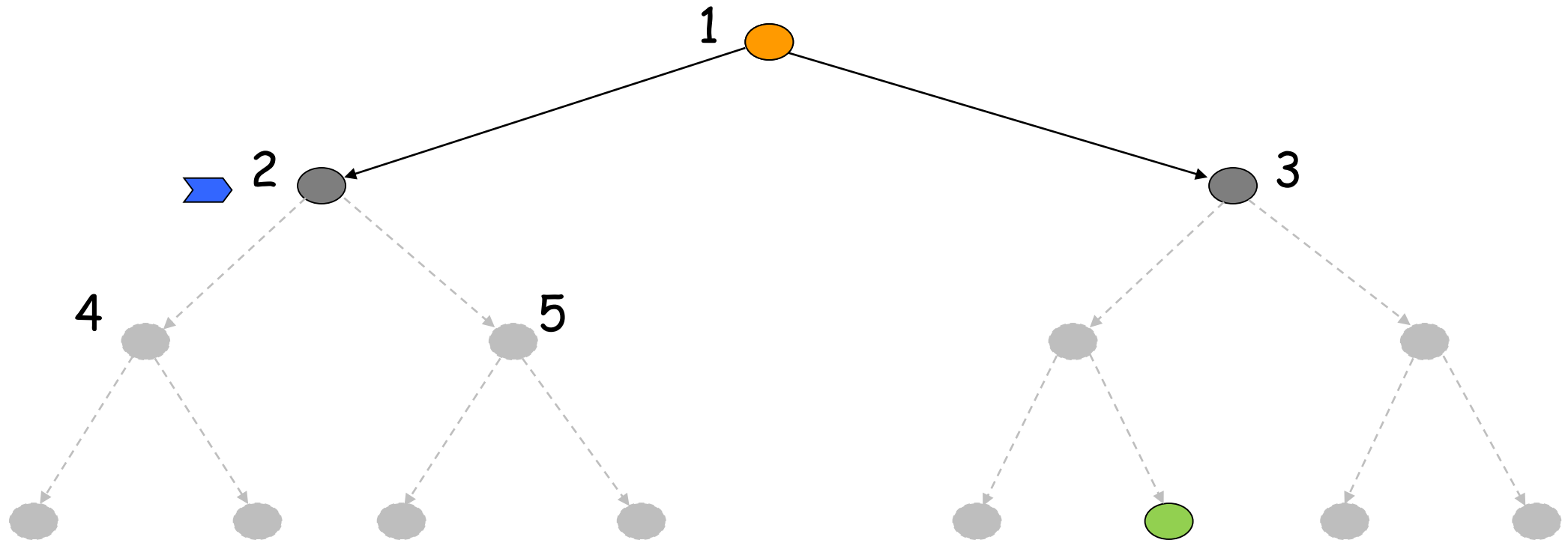
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



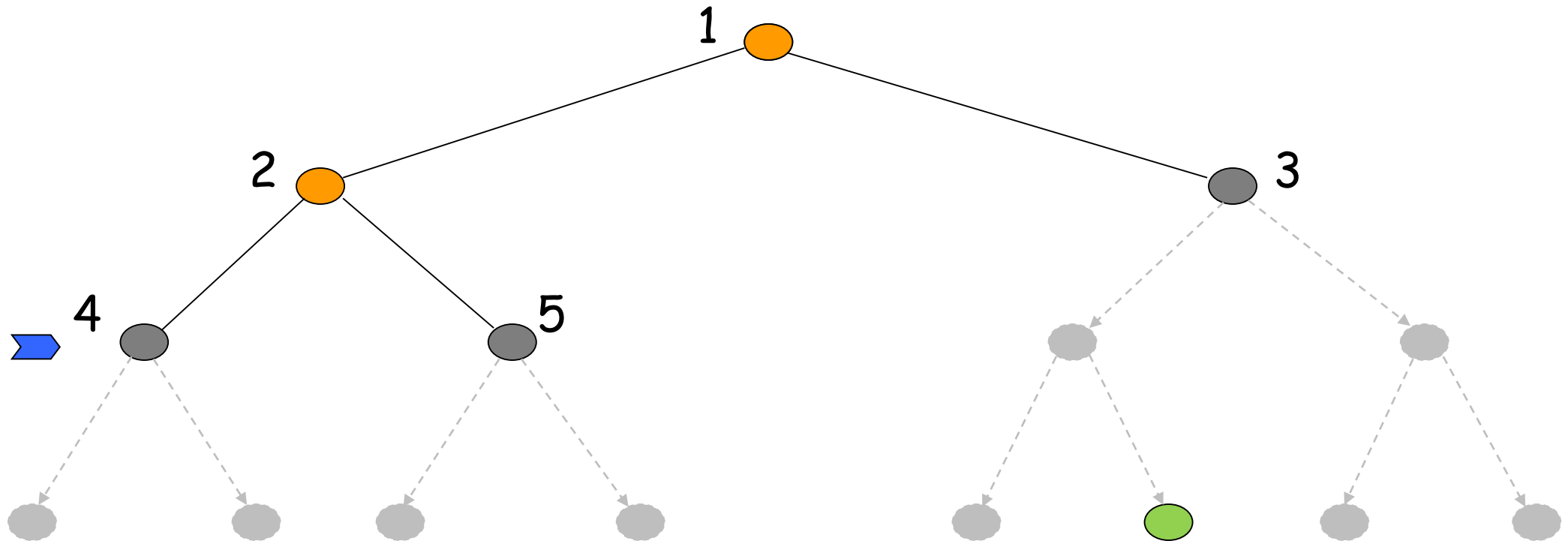
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



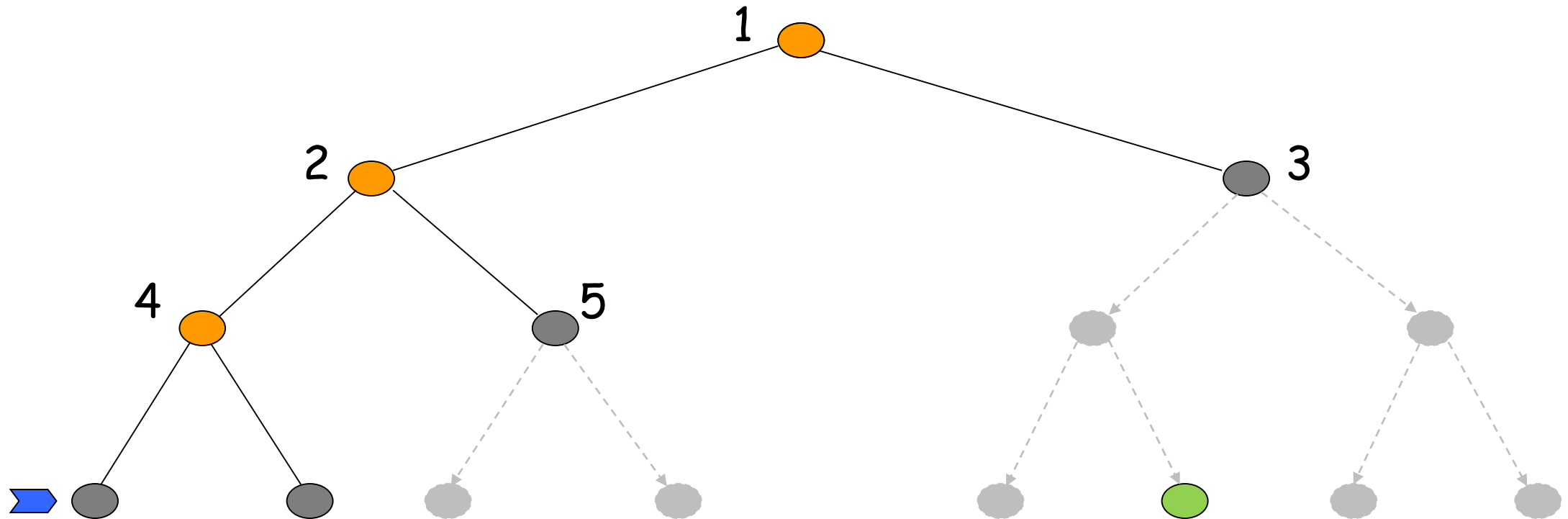
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



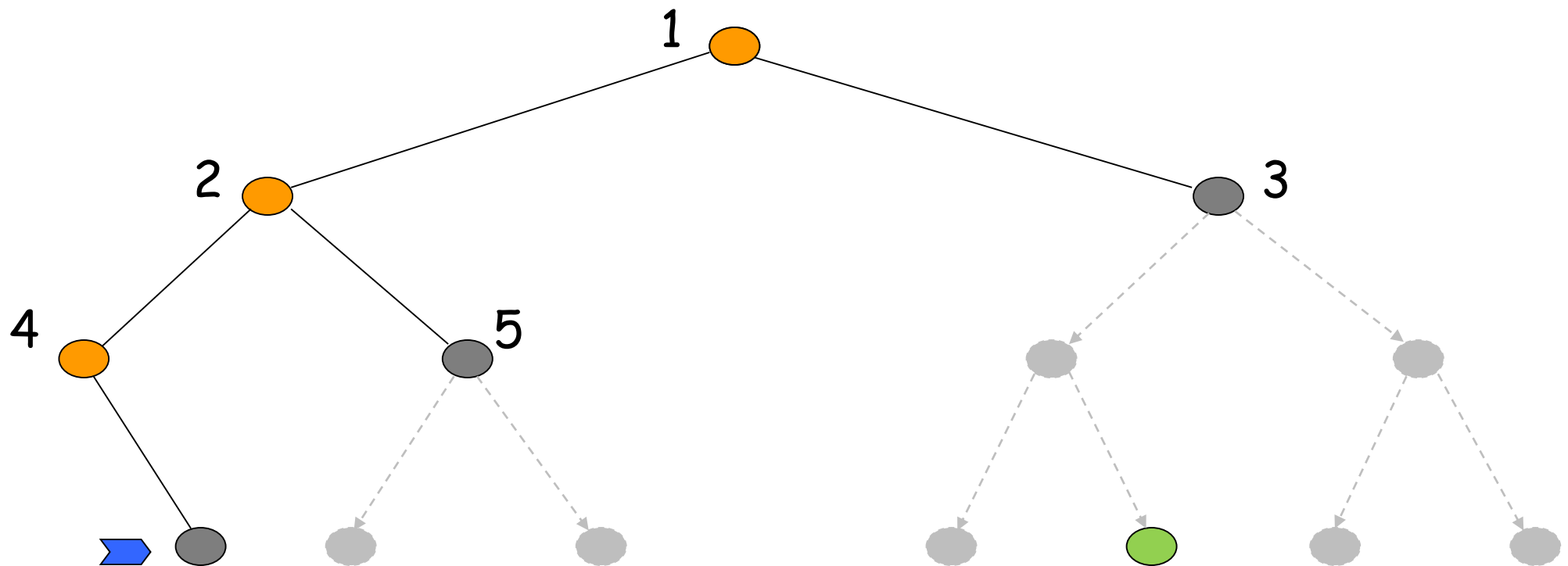
جستجوی اول عمق – DFS

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



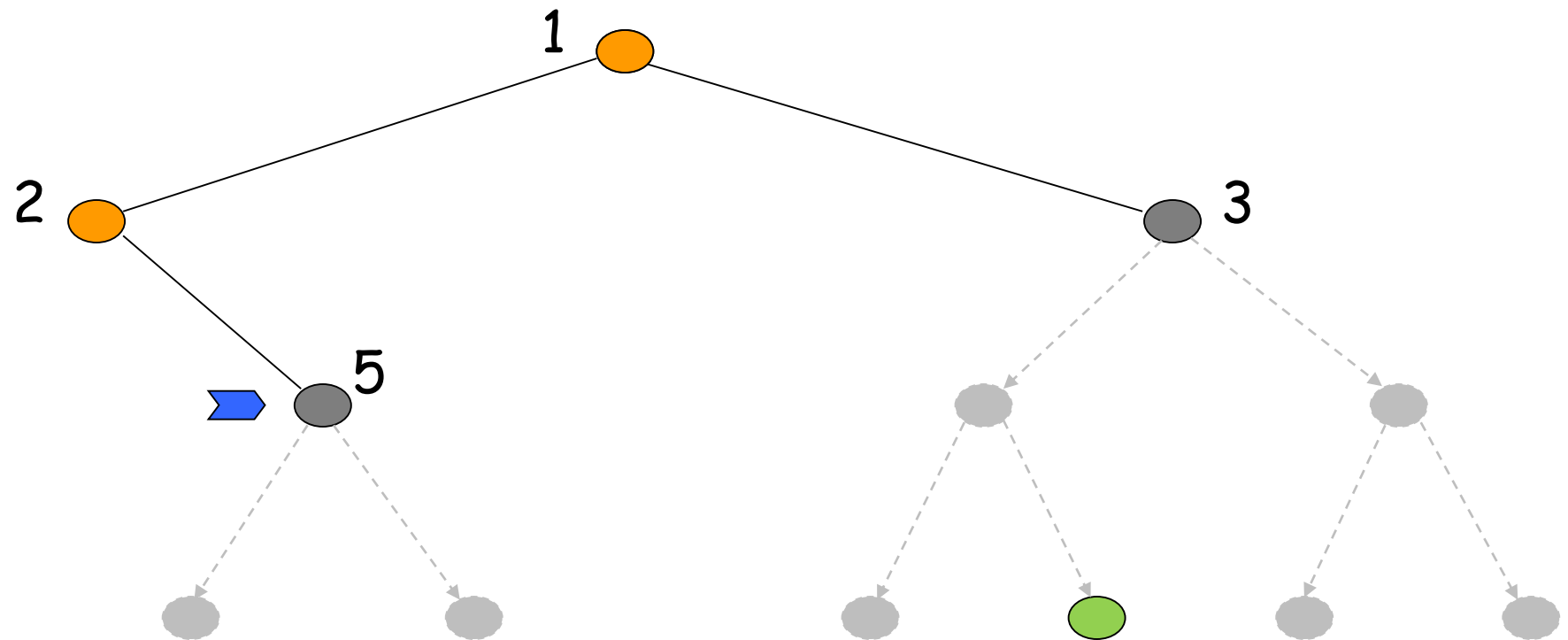
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



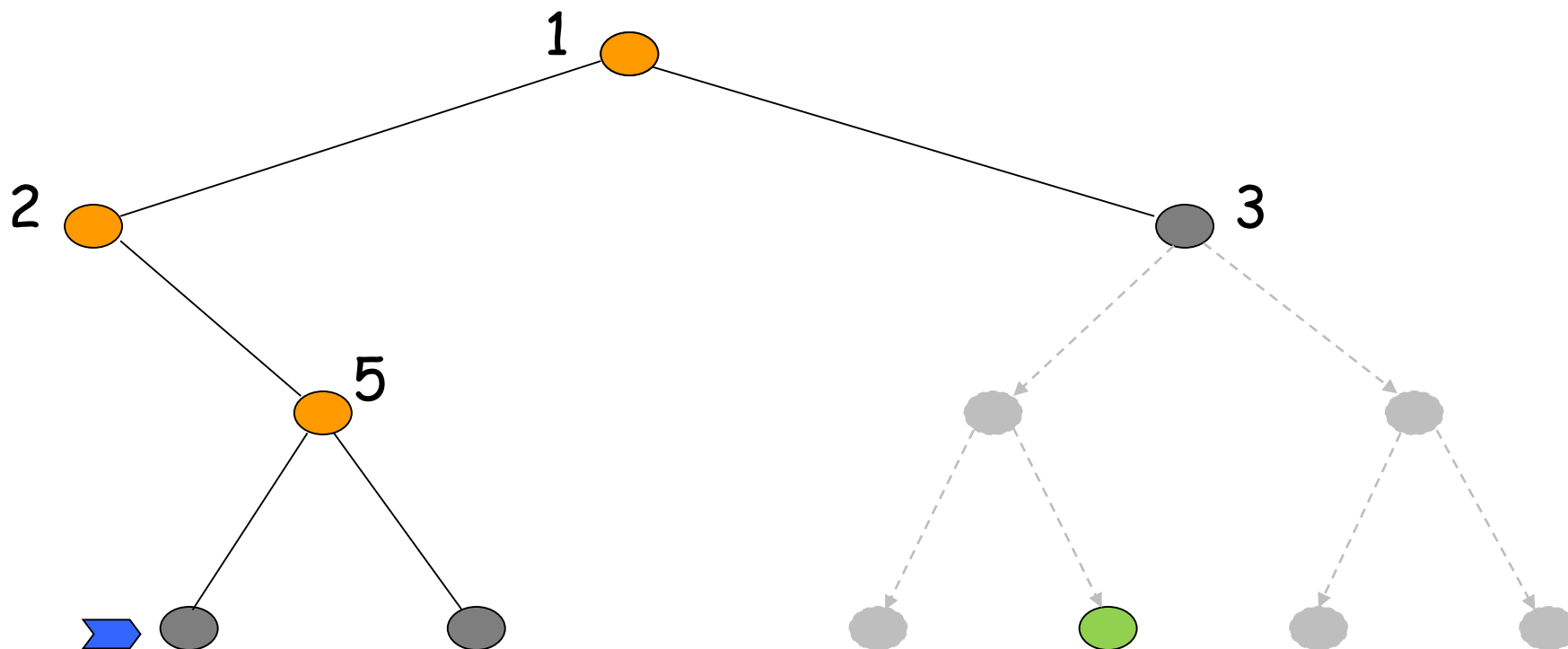
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



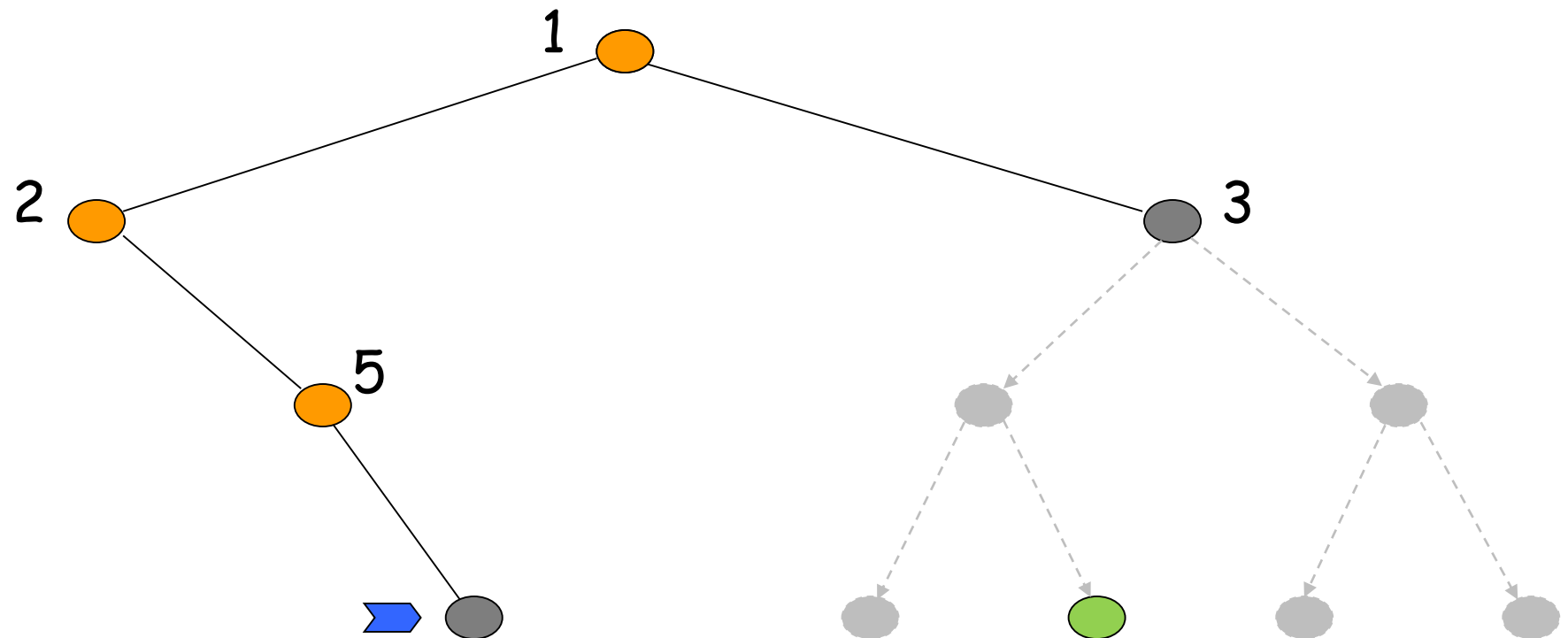
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



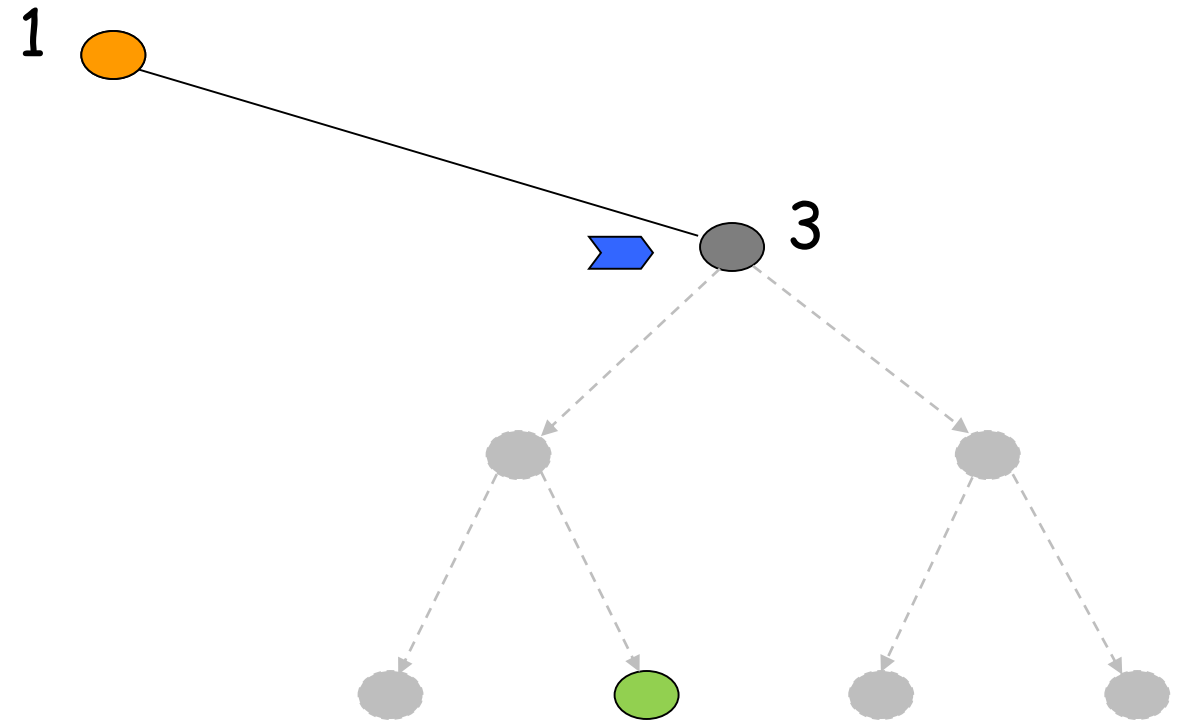
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



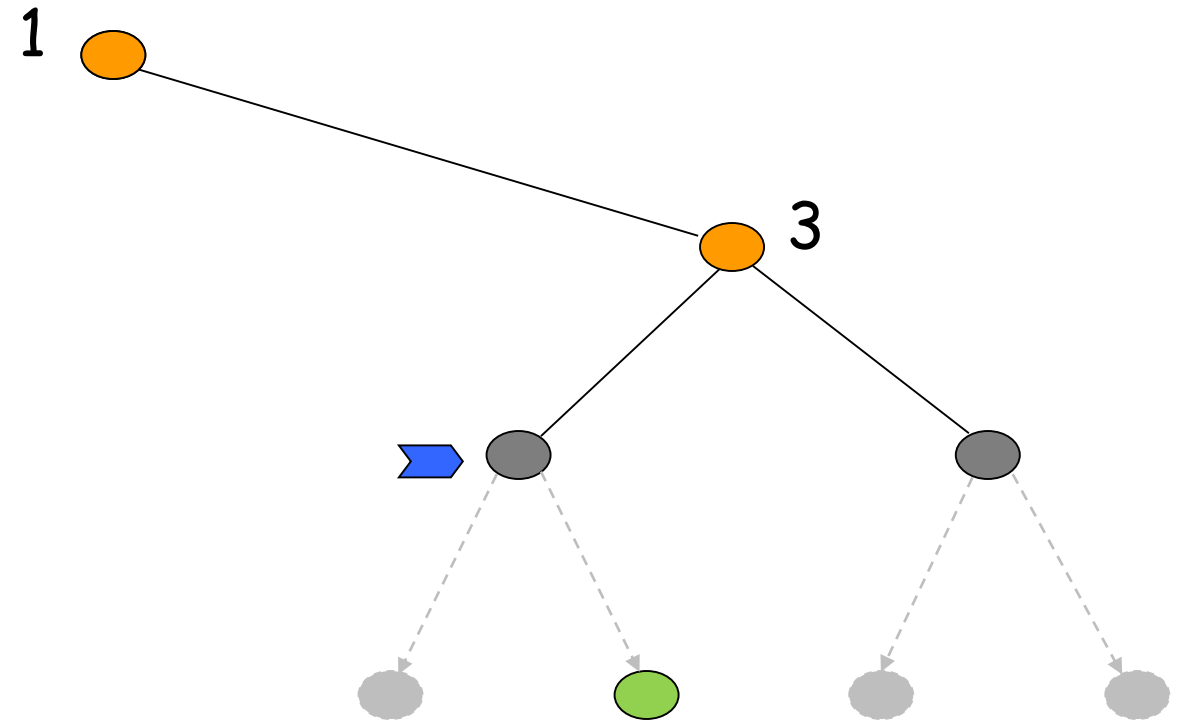
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



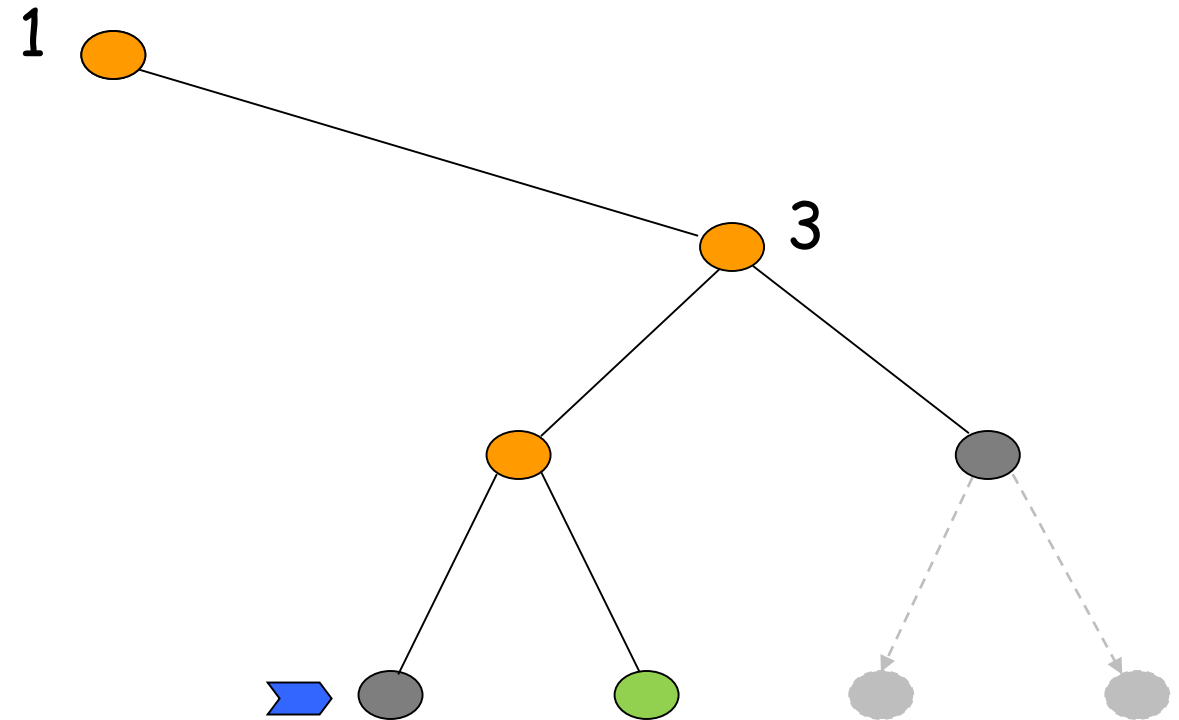
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



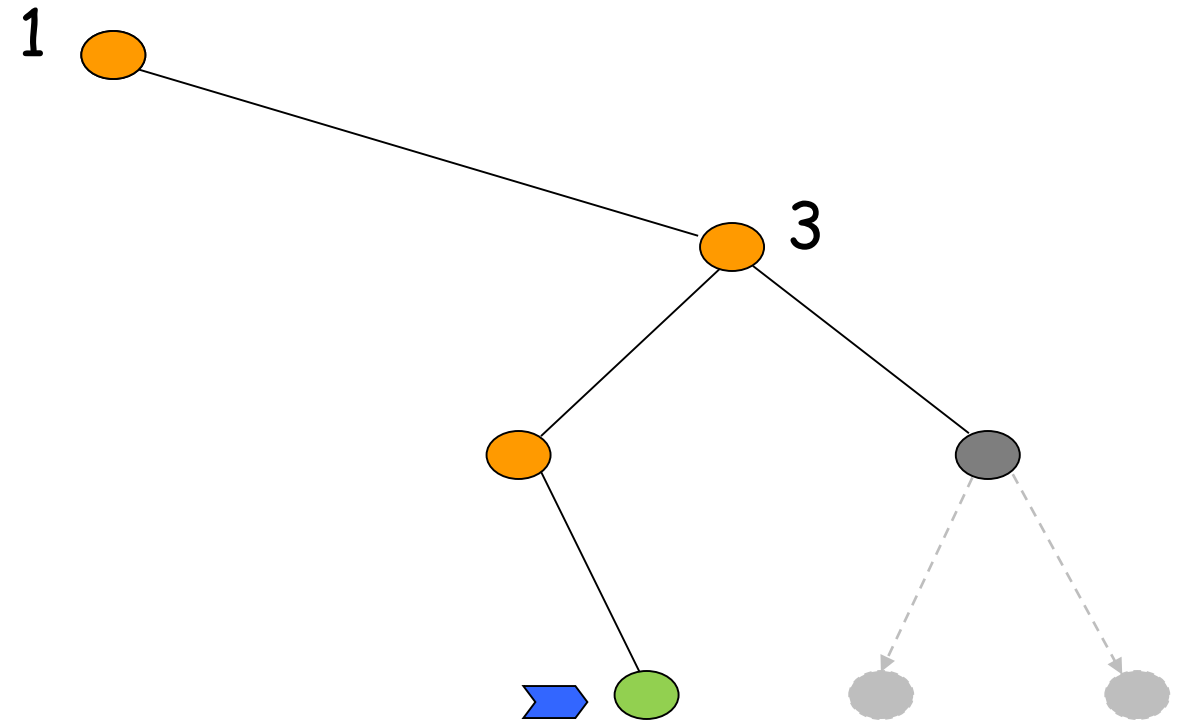
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



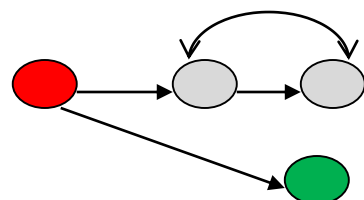
جستجوی اول عمق – DFS

- عمیق ترین نود بسط نیافته در frontier را انتخاب می کند.
- پیاده سازی: frontier یک صف LIFO یا همان پشته است.



ارزیابی جستجوی اول عمق

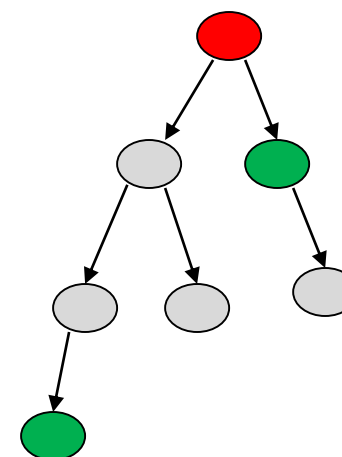
• کامل؟



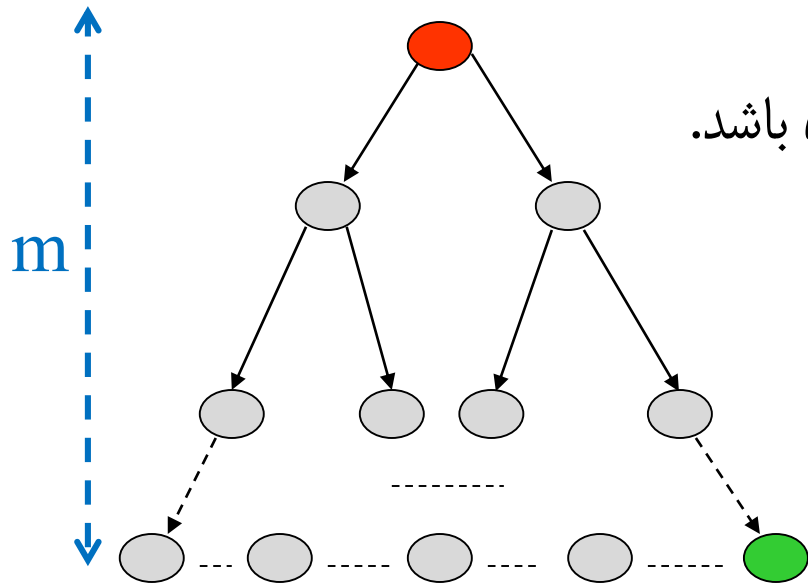
- جستجوی درختی: به دلیل وجود حالات تکراری کامل نیست.
- جستجوی گرافی: تنها در فضاهاى حالت متناهی کامل است چون در نهایت تمام گره‌ها بسط می‌یابند.
- شکست جستجوی اول عمق با هر دو نوع جستجو در فضاهاى حالت نامتناهی
- احتمال گیر کردن در یک مسیر نامتناهی بدون هدف

• بهینگی؟

• خیر



ارزیابی جستجوی اول عمق



- پیچیدگی زمانی؟

- جستجوی درختی: $O(b^m)$ مقدار m می تواند خیلی بزرگتر از d باشد.
- جستجوی گرافی: با سایز فضای حالت محدود شده است.

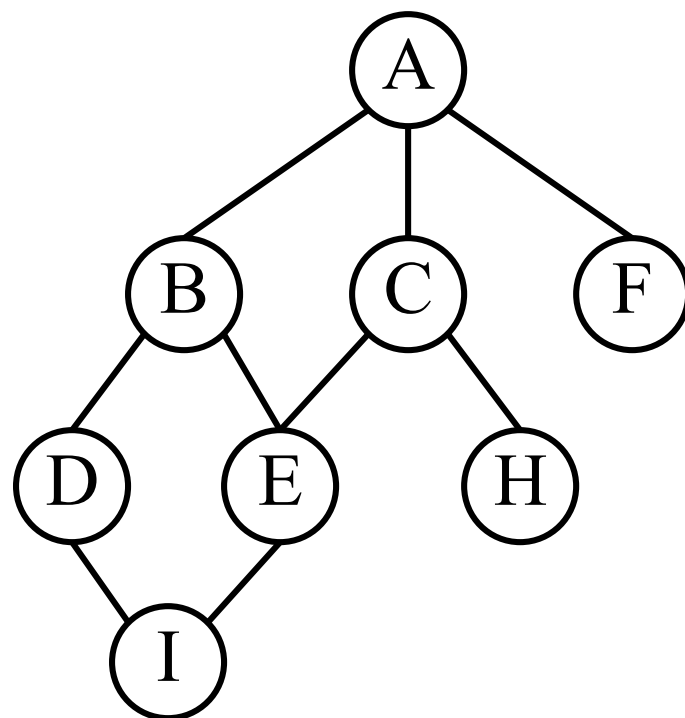
- پیچیدگی فضایی؟

- جستجوی درختی: $O(bm)$

- تنها نیاز به نگه داشتن یک مسیر از ریشه تا گره ای دارد که در حال بررسی آن است به علاوه همزادهایی از نودهای موجود در این مسیر که هنوز گسترش نیافته اند.

- جستجوی گرافی: همه ی نودها در مجموعه **explored** ذخیره می شوند.

اگر در گراف زیر جستجو در عمق (Depth First Search) را از رأس C شروع کنیم، کدام گره‌ها به ترتیب 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

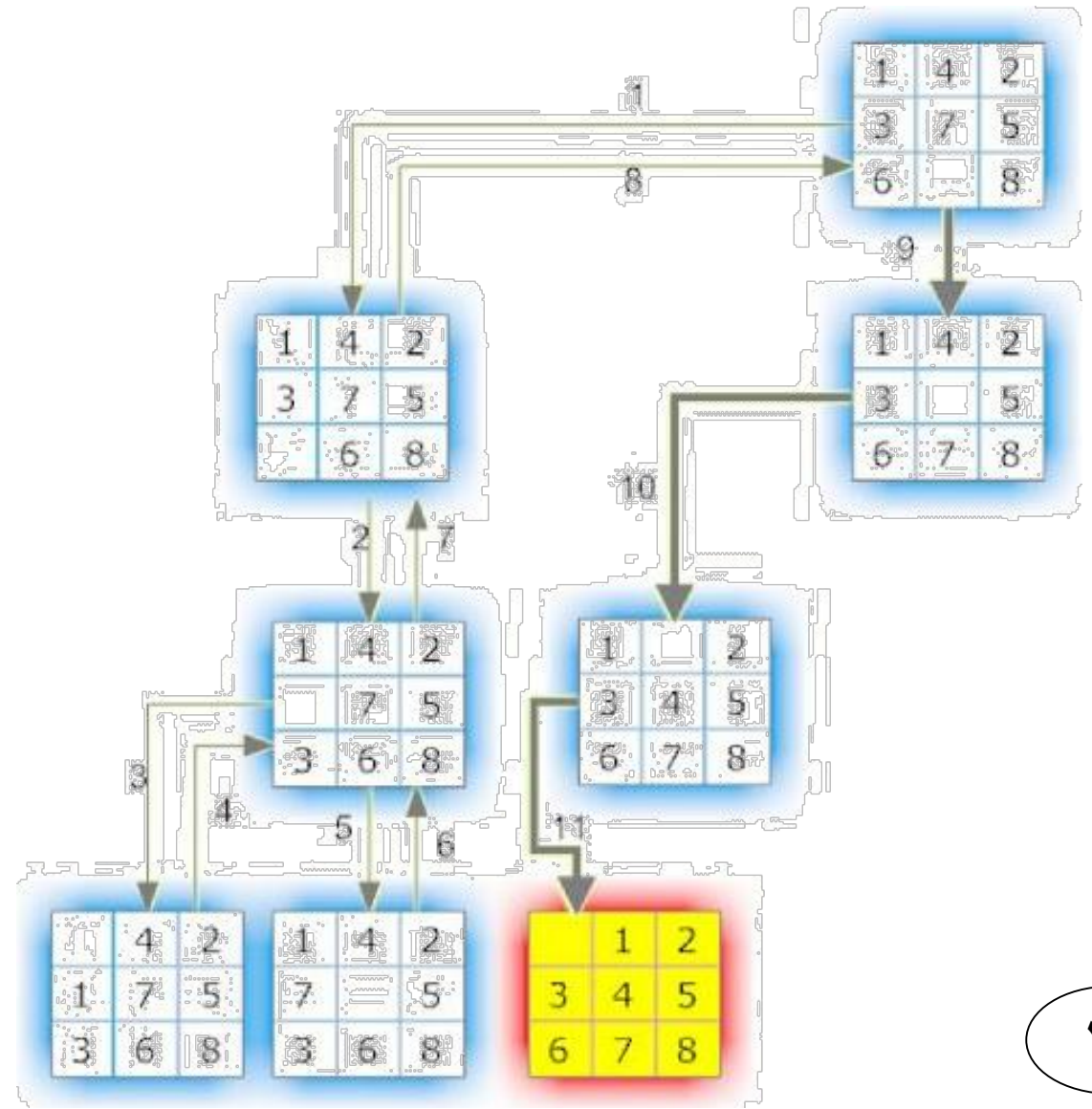
- نسخه‌ای از جستجوی اول عمق جستجوی عقب‌گرد (Backtracking) است.

- 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: $O(m)$

Picture from "Intelligent Systems: a Modern Approach" by Crina Grosan, Ajith Abraham



جستجوی عمقی محدود شده – DLS

- انجام جستجوی عمقی با در نظر گرفتن یک محدودیت عمق از پیش تعریف شده مانند l
 - با گره‌هایی که در عمق l هستند به گونه‌ای رفتار می‌شود که گویی هیچ پسینی ندارند.
 - مسئله مسیر نامتناهی را حل می‌کند.
- در برخی مسائل، دانش مسئله می‌تواند برای تعیین l به کار 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 کم عمق ترین هدف پایین تر 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 محدود شده – DLS

function DEPTH-LIMITED-SEARCH(*problem*, *limit*) **returns** a solution, or failure/cutoff
 return RECURSIVE-DLS(MAKE-NODE(*problem*.INITIAL-STATE), *problem*, *limit*)

function RECURSIVE-DLS(*node*, *problem*, *limit*) **returns** a solution, or failure/cutoff
 if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

else if *limit* = 0 **then return** cutoff

else

cutoff_occurred? ← false

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

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

result ← RECURSIVE-DLS(*child* , *problem*, *limit* – 1)

if *result* = cutoff **then** *cutoff_occurred?* ← true

else if *result* ≠ failure **then return** *result*

if *cutoff_occurred?* **then return** cutoff **else return** failure

جستجوی عمقی تکراری – IDS

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

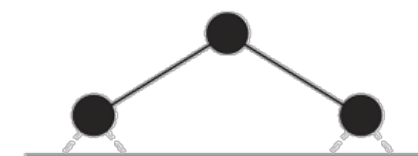
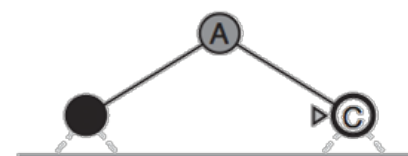
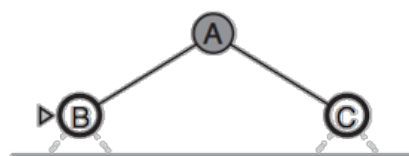
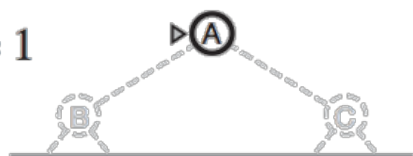
- یک استراتژی کلی برای یافتن بهترین محدودیت برای عمق l ارائه می دهد.
- هدف یافتن عمق d یا همان کم عمق ترین نود هدف است.
- از مزایای DFS و BFS بهره می برد.
- DFS: نیاز به حافظه ی کم
- BFS: کامل بودن و همچنین بهینگی در صورت استفاده از توابع هزینه مناسب
- سر بار زیادی ایجاد نمی کند چون اکثر نودها در پایین ترین سطح ایجاد می شوند.

جستجوی عمقی تکراری – IDS

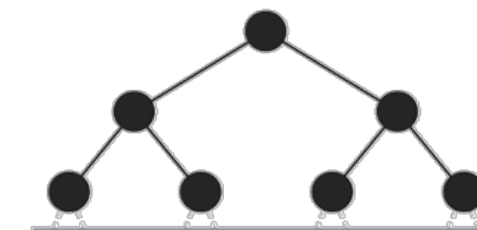
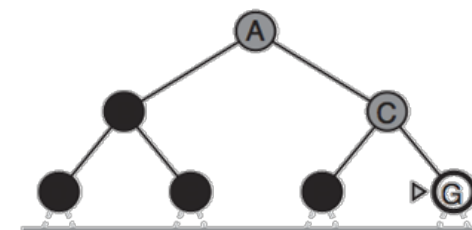
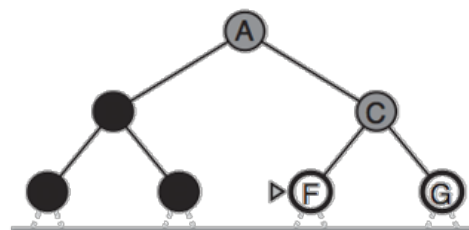
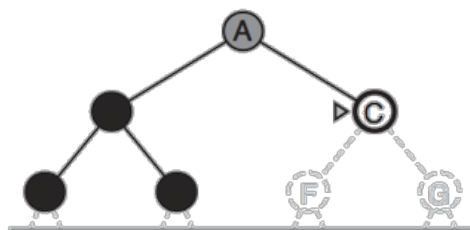
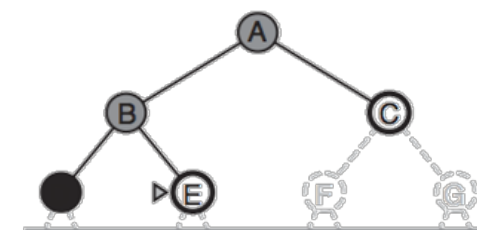
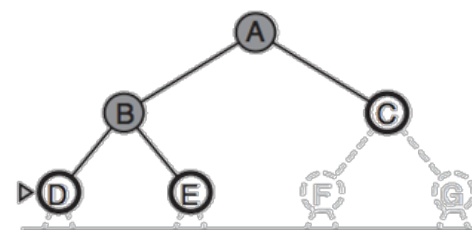
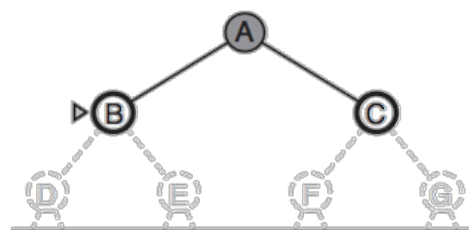
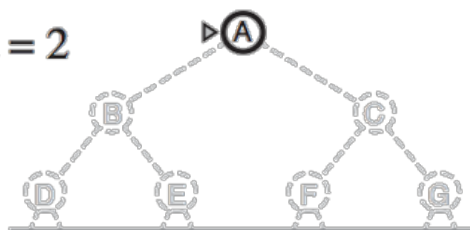
Limit = 0



Limit = 1

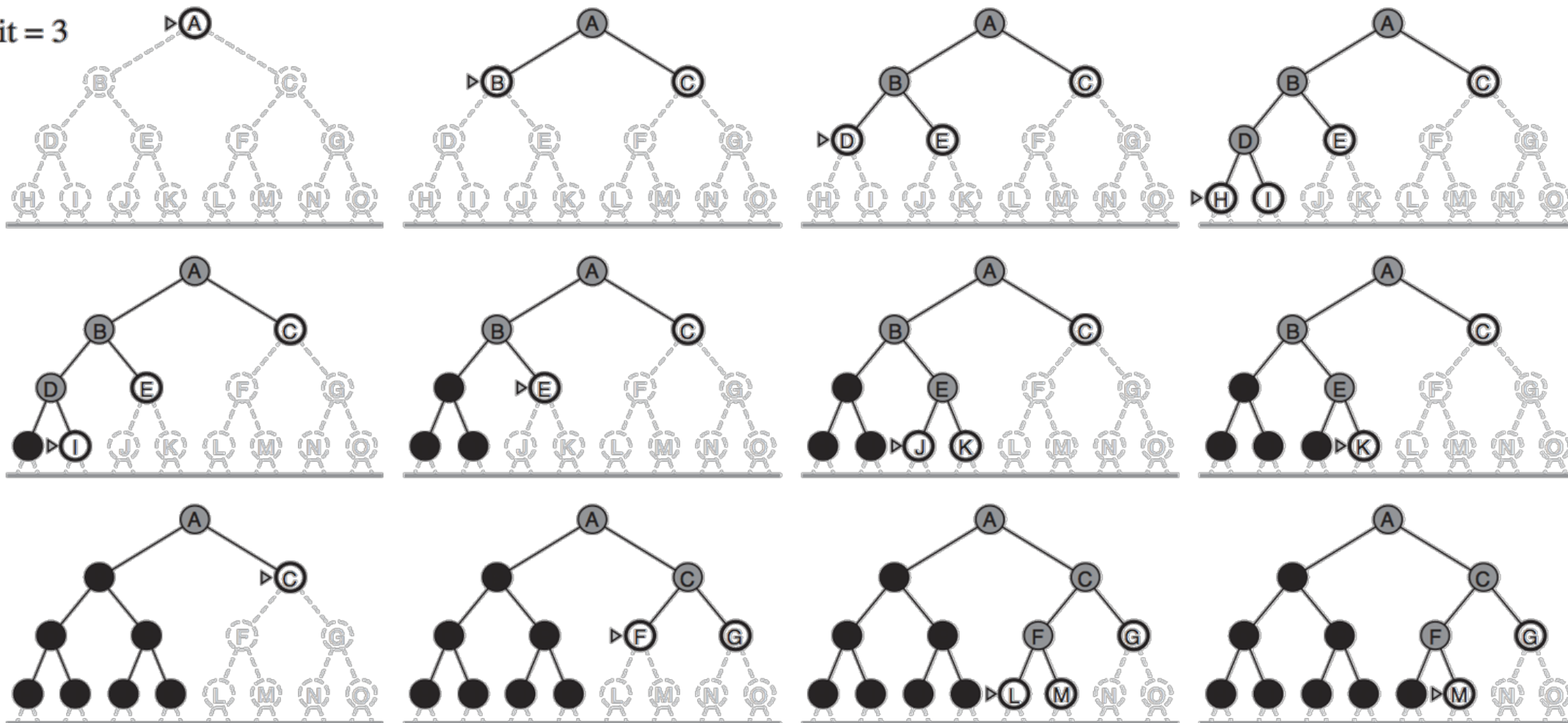


Limit = 2



جستجوی عمقی تکراری – IDS

Limit = 3



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: $O(b^d)$

- سطح d : یک بار

- سطح $d-1$: دو بار

- سطح $d-2$: سه بار

- ...

- سطح دو: $d-1$ بار

- سطح یک: d بار

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

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

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.
- به طور کلی وقتی فضای جستجو بزرگ باشد و عمق راه حل شناخته شده نباشد، جستجوی IDS یک روش جستجوی ناآگاهانه مناسب یا preferred محسوب می شود.

پیچیدگی زمانی در جستجوی تعمیق تکراری (iterative deepening) به کدام یک از عوامل زیر بستگی دارد؟
(کامپیوتر ۸۶)

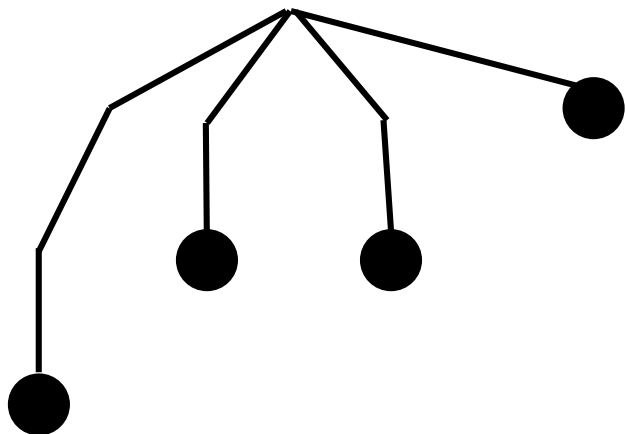
(۱) بیشترین عمق درخت

(۲) سایز فضای حالت

(۳) تابع مکاشفهای انتخاب شده

(۴) عمق کم عمق ترین گره هدف ✓

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؟ (کامپیوتر ۸۵)



۱) عمق نخست (Depth first)

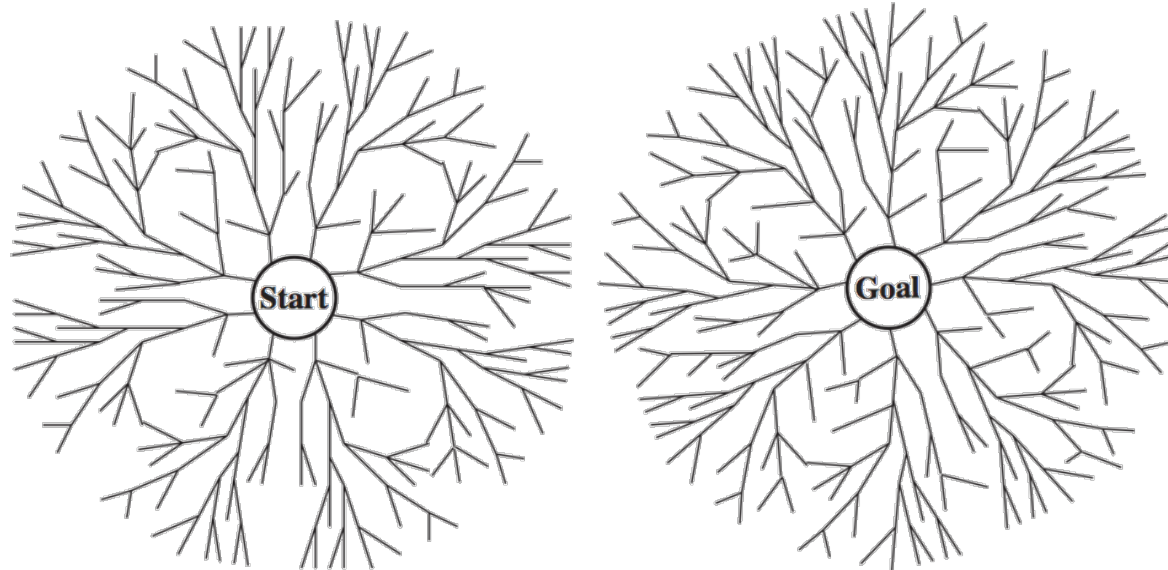
۲) عرض نخست (Breadth first)

۳) جستجوی هزینه یکنواخت (Uniform cost) ✓

۴) تعمیق تکراری (Iterative deepening)

جستجوی دو طرفه – bidirectional

- انجام همزمان دو جستجو: یکی از وضعیت اولیه به سمت هدف (forward یا روبه جلو) و دیگری از وضعیت هدف به سمت وضعیت اولیه (backward یا روبه عقب)
- 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.



BIDIRECTIONAL_SEARCH

```
1   $Q_I.Insert(x_I)$  and mark  $x_I$  as visited
2   $Q_G.Insert(x_G)$  and mark  $x_G$  as visited
3  while  $Q_I$  not empty and  $Q_G$  not empty do
4      if  $Q_I$  not empty
5           $x \leftarrow Q_I.GetFirst()$ 
6          if  $x = x_G$  or  $x \in Q_G$ 
7              return SUCCESS
8          forall  $u \in U(x)$ 
9               $x' \leftarrow f(x, u)$ 
10             if  $x'$  not visited
11                 Mark  $x'$  as visited
12                  $Q_I.Insert(x')$ 
13             else
14                 Resolve duplicate  $x'$ 
15         if  $Q_G$  not empty
16              $x' \leftarrow Q_G.GetFirst()$ 
17             if  $x' = x_I$  or  $x' \in Q_I$ 
18                 return SUCCESS
19             forall  $u^{-1} \in U^{-1}(x')$ 
20                  $x \leftarrow f^{-1}(x', u^{-1})$ 
21                 if  $x$  not visited
22                     Mark  $x$  as visited
23                      $Q_G.Insert(x)$ 
24             else
25                 Resolve duplicate  $x$ 
26 return FAILURE
```

الگوریتم جستجوی دو طرفه

• الگوریتم روبه‌رو نسخه‌ای از الگوریتم جستجوی دو طرفه است که از هر طرف جستجوی اول سطح در آن انجام می‌گیرد.

• ابتدا یک گره از جستجوی روبه‌جلو بسط داده می‌شود سپس یک گره از جستجوی روبه عقب.

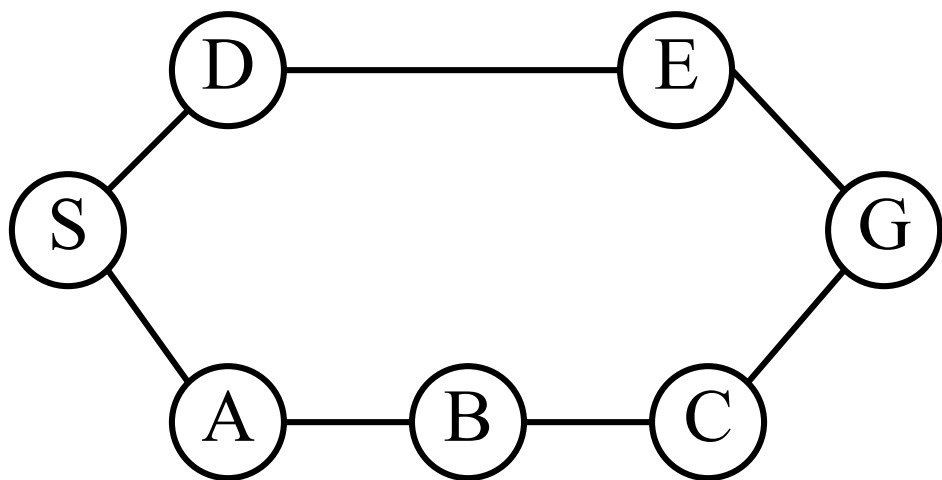
• در هر یک از جستجوها، قبل از بسط هر گره بررسی می‌کند که آیا آن گره در مجموعه‌ی frontier جستجوی دیگر وجود دارد یا خیر؛ اگر چنین بود یک راه‌حل پیدا شده است.

• $U(x)$: مجموعه اعمال قابل انجام در وضعیت x

• $f(x, u)$: به‌کارگیری عمل u بر روی وضعیت x و رسیدن به یک وضعیت جدید

جستجوی دو طرفه – مثال اول

- جستجوی دوطرفه اول سطح را بر روی گراف حالت زیر انجام دهید.
- فرض کنید ابتدا یک گره از جستجوی روبه جلو بسط داده می شود سپس یک گره از جستجوی روبه عقب.



جستجوی دو طرفه – مثال اول

Iter 0: $Q_{\text{forward}} = \{S\}$   $Q_{\text{backward}} = \{G\}$

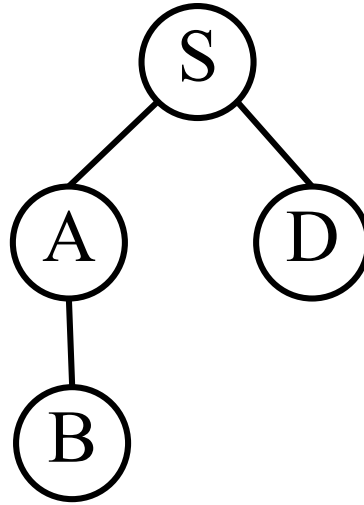
Iter 1: $Q_{\text{forward}} = \{A, D\}$   $Q_{\text{backward}} = \{G\}$

$Q_{\text{forward}} = \{A, D\}$   $Q_{\text{backward}} = \{C, E\}$

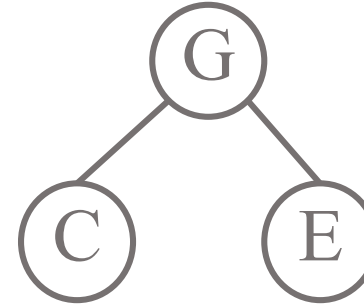
جستجوی دو طرفه – مثال اول

Iter 2:

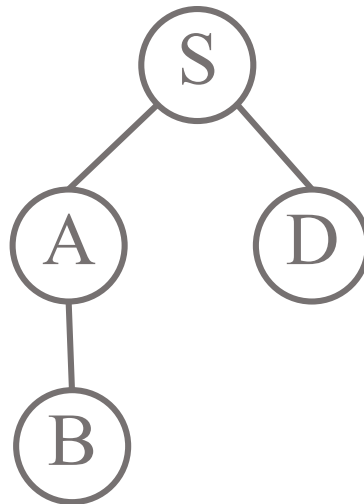
$$Q_{\text{forward}} = \{D, B\}$$



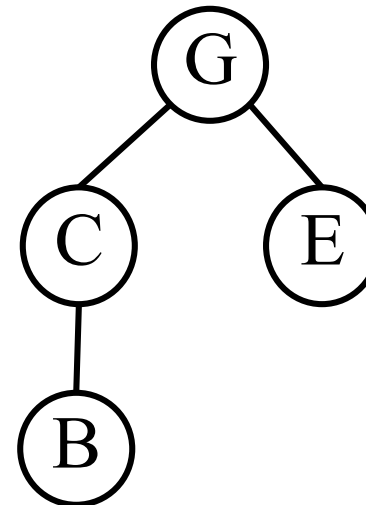
$$Q_{\text{backward}} = \{C, E\}$$



$$Q_{\text{forward}} = \{D, B\}$$



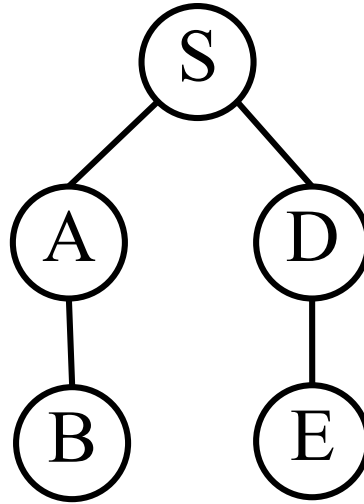
$$Q_{\text{backward}} = \{E, B\}$$



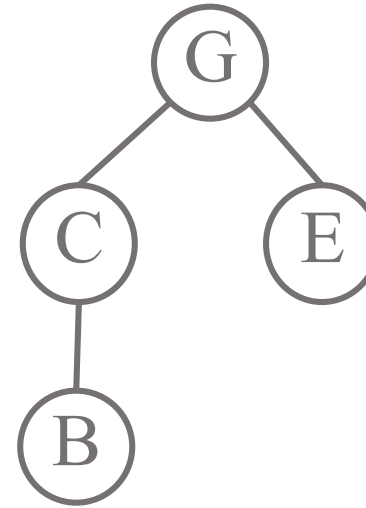
جستجوی دو طرفه – مثال اول

Iter 3:

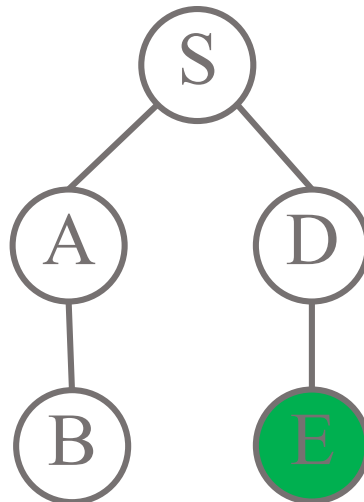
$$Q_{\text{forward}} = \{B, E\}$$



$$Q_{\text{backward}} = \{E, B\}$$



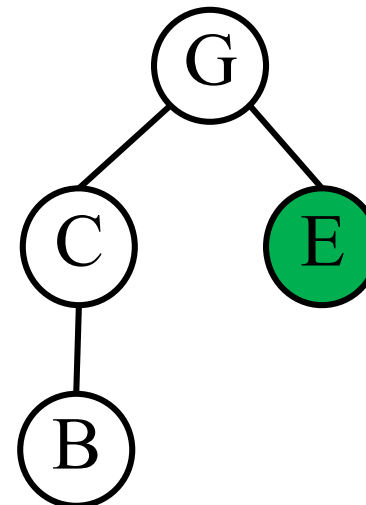
$$Q_{\text{forward}} = \{B, \textcolor{green}{E}\}$$



$$Q_{\text{backward}} = \{\textcolor{green}{E}, B\}$$

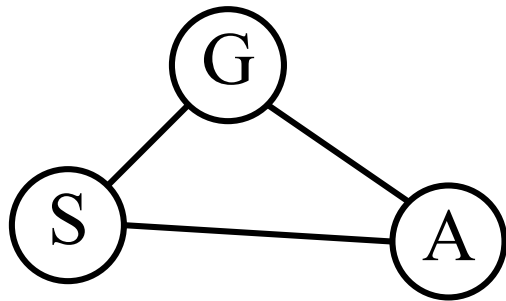
مسیر بهینه یافت شد!

S-D-E-G



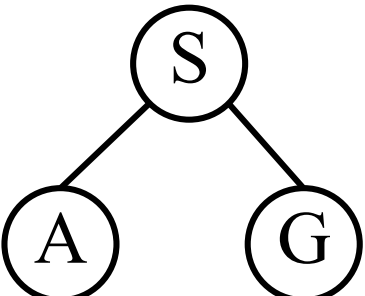

جستجوی دو طرفه – مثال دوم

- اگر ابتدا یک گره از جستجوی روبه جلو بسط داده شود سپس یک گره از جستجوی روبه عقب، آیا جستجوی دوطرفه اول سطح بر روی گراف حالت زیر، جواب بهینه را برخواهد گرداند؟
- اگر جستجوها به طور همزمان به بسط گرهها پردازند چطور؟



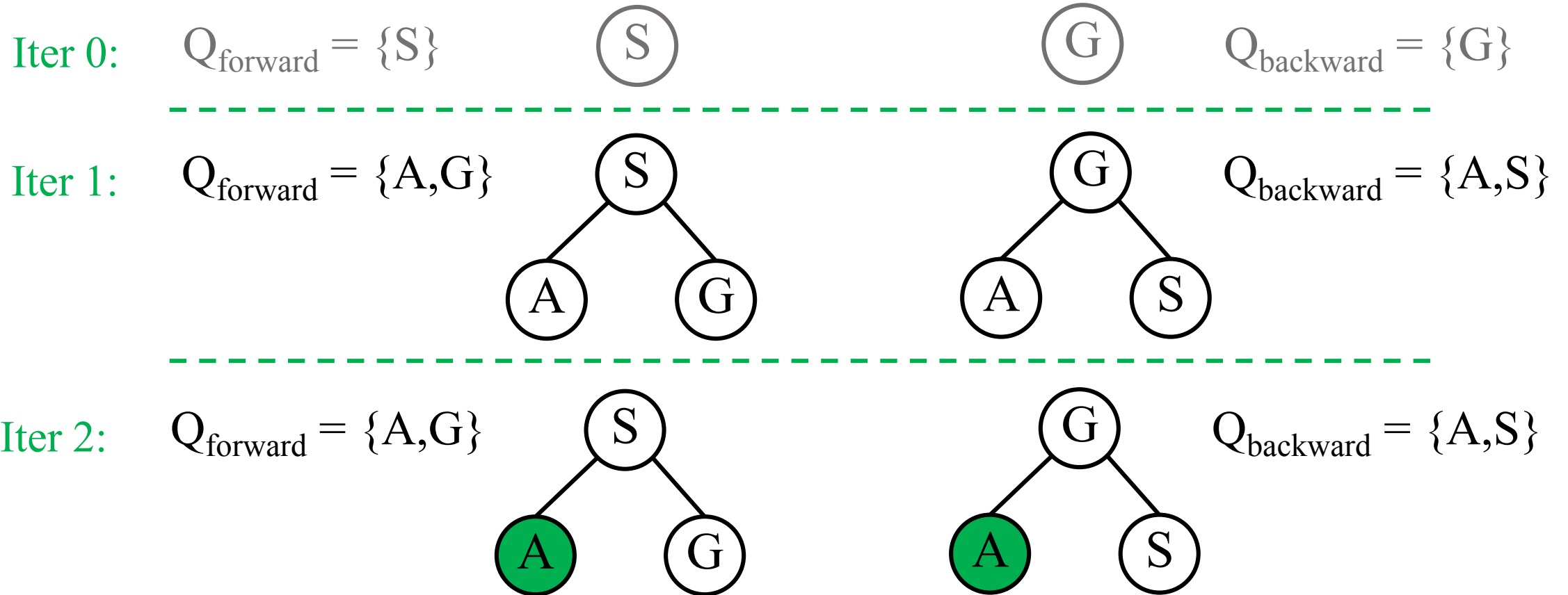
جستجوی دو طرفه – مثال دوم (بسط غیرهمزمان)

Iter 0: $Q_{\text{forward}} = \{S\}$   $Q_{\text{backward}} = \{G\}$

Iter 1: $Q_{\text{forward}} = \{A, G\}$   $Q_{\text{backward}} = \{G\}$

$Q_{\text{forward}} = \{A, G\}$   $Q_{\text{backward}} = \{G\}$
مسیر بهینه یافت شد!
S-G

جستجوی دو طرفه – مثال دوم (بسط همزمان)

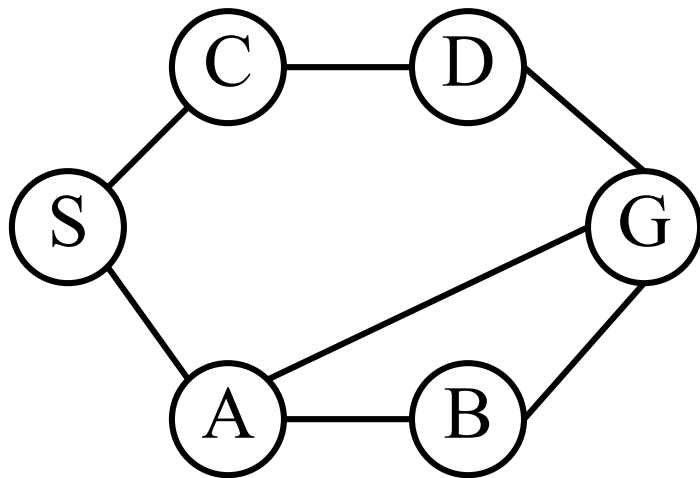


جستجوی دو طرفه – مثال دوم (بسط همزمان)

- همان طور که مشاهده می شود در این حالت در صورت انتخاب گره A جواب نیمه بهینه $(S-A-G)$ برگردانده می شود. در حالی که اگر گره بعدی در صف بررسی می شد جواب بهینه به دست می آمد.
- برای جلوگیری از بروز چنین مشکلاتی و تضمین بهینگی جستجوی دو طرفه ی اول سطح در بسط همزمان، باید در صورت رسیدن به جواب به بررسی دیگر گره های صف پرداخت تا در صورت وجود مسیر بهینه تر آن مسیر را برگرداند.

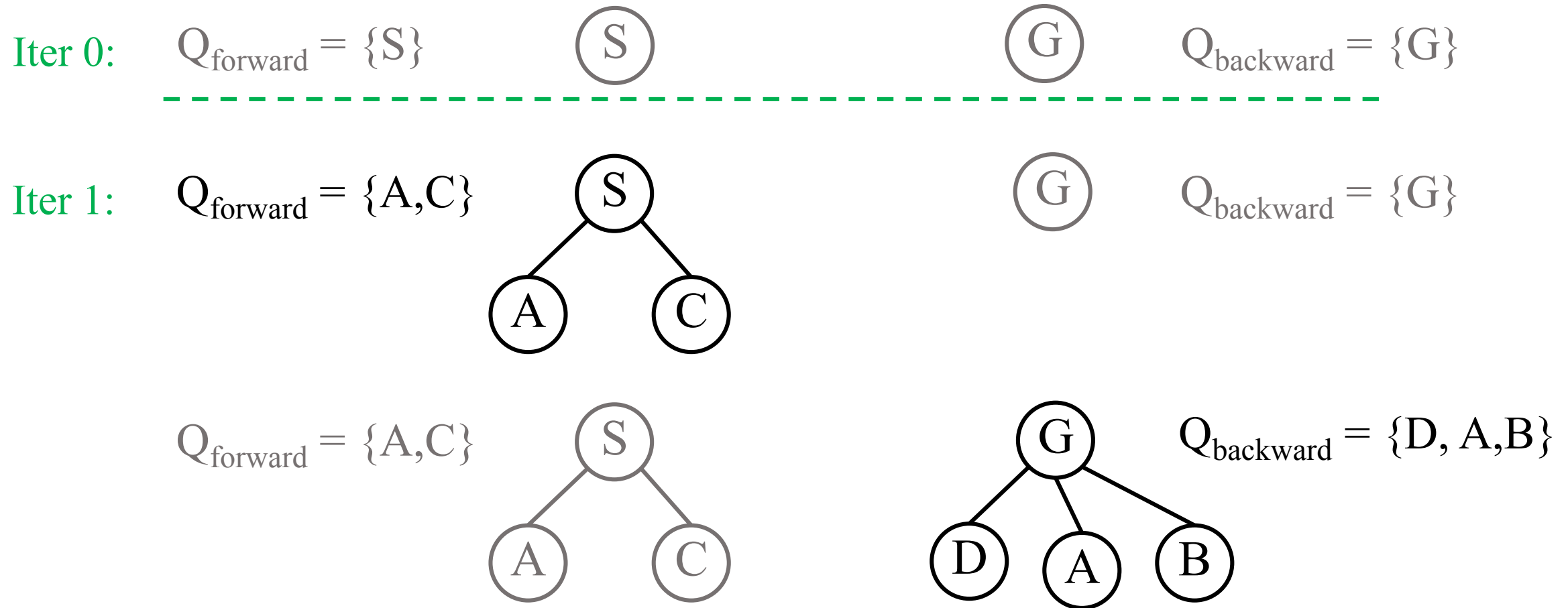
جستجوی دو طرفه – مثال سوم

با به کارگیری DFS، جستجوی دوطرفه را بر روی گراف حالت زیر انجام دهید. (بسط غیرهمزمان)



- همان طور که در اسلایدهای بعد مشاهده می شود امکان ایجاد مسیر نیمه بهینه در استفاده از جستجوی اول عمق دوطرفه وجود دارد.

جستجوی دو طرفه – مثال سوم

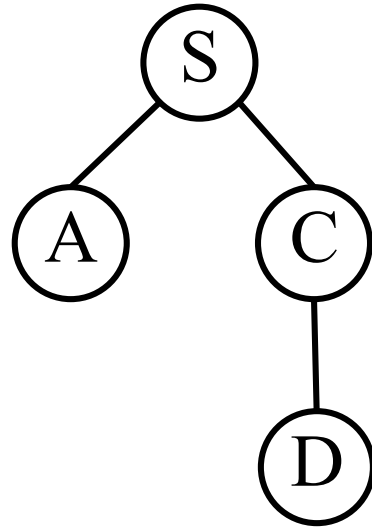


توجه کنید که هنگام انتخاب گره برای بسط، بررسی می‌شود که آیا آن نود قبلاً در مجموعه frontier جستجوی مقابل وجود دارد یا خیر.

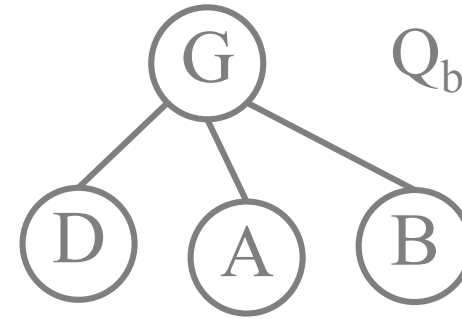
جستجوی دو طرفه – مثال سوم

Iter 2:

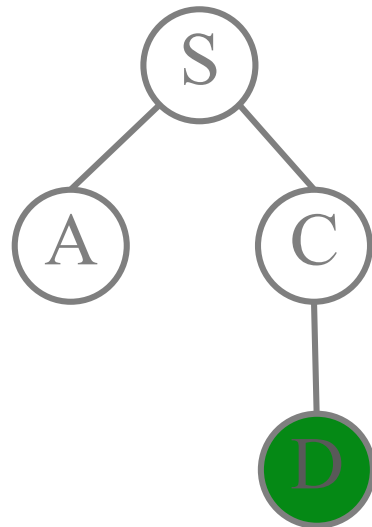
$$Q_{\text{forward}} = \{A, D\}$$



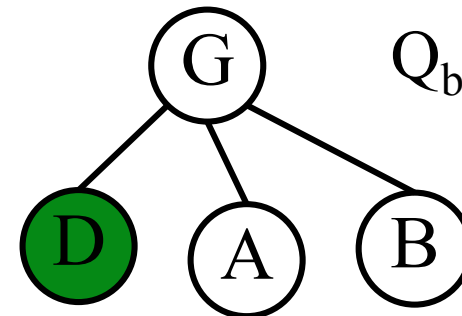
$$Q_{\text{backward}} = \{D, A, B\}$$



$$Q_{\text{forward}} = \{A, \textcolor{green}{D}\}$$



$$Q_{\text{backward}} = \{\textcolor{green}{D}, A, B\}$$



جستجوی دو طرفه – bidirectional

- پیاده‌سازی؟
- بررسی عضویت یک گره در درخت جستجوی دیگر با یک جدول هش، می‌تواند در زمان ثابتی انجام شود.
- پیشین‌های یک نود باید به‌طور موثری قابل محاسبه باشند.
- ساده‌ترین حالت زمانی است که تمامی اعمال در فضای حالت معکوس‌پذیر باشند.
- اگر چندین حالت هدف وجود داشته باشد که صریحاً فهرست شده باشند می‌توان یک حالت هدف ساختگی جدید بسازیم که پیشین‌های بلافاصله‌ی آن‌ها حالات هدف واقعی هستند.
- هدف انتزاعی (کیش و مات)؟!

ارزیابی جستجوی دو طرفه

- کامل؟

- بستگی به دو الگوریتم جستجو دارد: اگر هر دو جستجو BFS باشند کامل است.
- ترکیبات دیگر ممکن است باعث عدم کامل بودن شود.

- بهینگی؟

- بستگی به دو الگوریتم جستجو دارد: اگر هر دو جستجو BFS باشند، تابع هزینه به طور مناسب انتخاب شده باشد و بررسی صف‌ها در بسط هم‌زمان به‌درستی انجام گرفته باشد، بهینه است.
- ترکیبات دیگر ممکن است باعث عدم بهینگی شود.

ارزیابی جستجوی دو طرفه

- پیچیدگی زمانی؟
 - به طور خوش بینانه $O(b^{d/2})$
- پیچیدگی فضایی؟
 - تمامی نودهای یکی از درخت‌ها باید همیشه در حافظه نگه داشته شود تا نودهای جدید درخت دیگر با آن‌ها مقایسه و نود مشترک پیدا شود. پس حداکثر $O(b^{d/2})$ نود در حافظه خواهد بود.
 - مهم‌ترین ضعف این روش نیاز به فضا است.

مقایسه راهبردهای جستجوی ناآگاهانه

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative-Deepening	Bidirectional (If applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+C^*/\varepsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+C^*/\varepsilon})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Superscript caveats are as follows:

^a complete if b is finite

^b complete if step costs $\geq \varepsilon$ for positive ε

^c optimal if step costs are all identical

^d if both directions use breadth-first search