دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

جستجو در محیط‌های پیچیده

«هوش مصنوعی: یک رهیافت نوین»، بخش ۳.۴ – ۵.۴

ارائه‌دهنده: سیده فاطمه موسوی

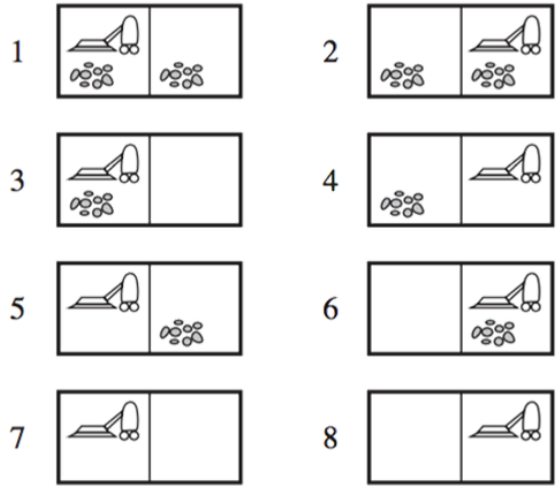
نیم‌سال اول ۱۴۰۰–۱۳۹۹

رئوس مطالب

- جستجو با اعمال غیرقطعی
- جستجو با مشاهدات جزئی
- عامل‌های جستجوی برخط و محیط‌های ناشناخته

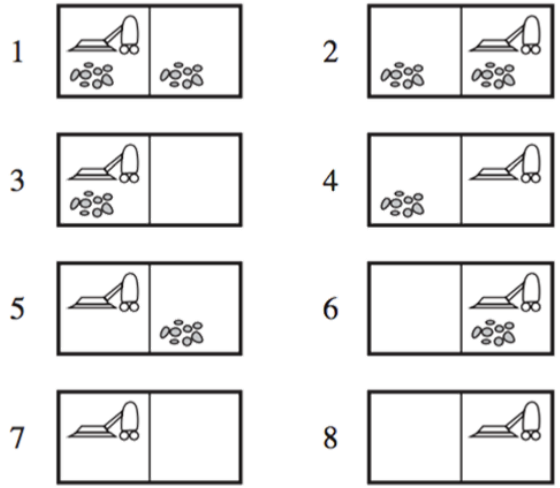
انواع مسئله

- محیط‌های قطعی و کاملا مشاهده‌پذیر (مسئله تک–حالت (Single-State))
  - عامل دقیقا وضعیتش را می‌داند حتی بعد از انجام یک دنباله عمل
  - راه‌حل یک دنباله است



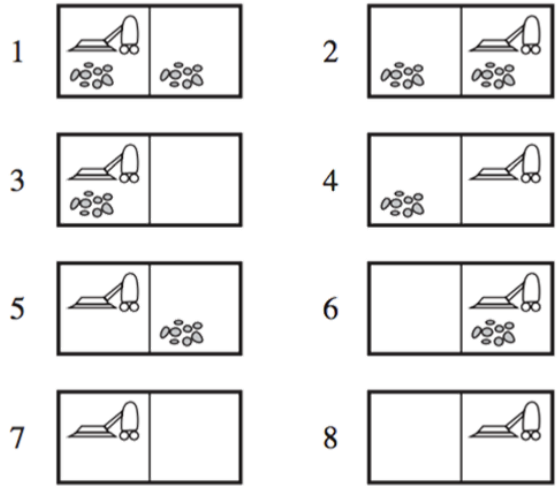
انواع مسئله

- محیط‌های قطعی و کاملا مشاهده‌پذیر (مسئله تک–حالت (Single-State))
  - عامل دقیقا وضعیتش را می‌داند حتی بعد از انجام یک دنباله عمل
  - راه‌حل یک دنباله است
- غیر قابل مشاهده یا فاقد سنسور (مسئله منطبق (Conformant))
  - ادراک عامل اصلا هیچ اطلاعاتی را فراهم نمی‌آورد.
  - راه‌حل یک دنباله است.



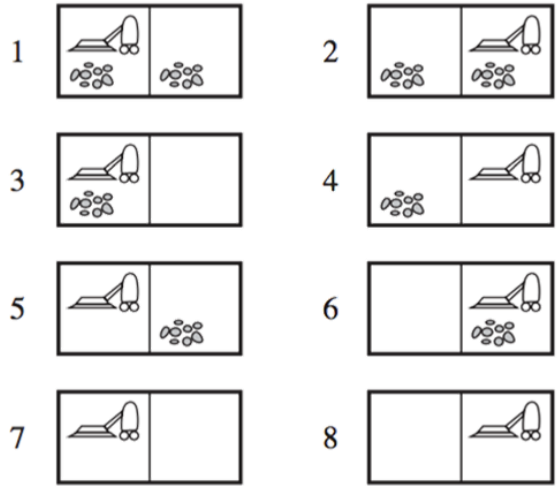
انواع مسئله

- محیط‌های قطعی و کاملا مشاهده‌پذیر (مسئله تک–حالت (Single-State))
  - عامل دقیقا وضعیتش را می‌داند حتی بعد از انجام یک دنباله عمل
  - راه‌حل یک دنباله است
- غیر قابل مشاهده یا فاقد سنسور (مسئله منطبق (Conformant))
  - ادراک عامل اصلا هیچ اطلاعاتی را فراهم نمی‌آورد.
  - راه‌حل یک دنباله است.
- غیرقطعی و/یا نیمه مشاهده‌پذیر (مسئله اقتضایی (Contingency))
  - ادراکات اطلاعات کاملی درباره وضعیت دقیق عامل فراهم نمی‌آورند.
  - راه‌حل می‌تواند یک برنامه اقتضایی (Contingency plan) باشد و نه یک دنباله



انواع مسئله

- محیط‌های قطعی و کاملا مشاهده‌پذیر (مسئله تک–حالت (Single-State))
  - عامل دقیقا وضعیتش را می‌داند حتی بعد از انجام یک دنباله عمل
  - راه‌حل یک دنباله است
- غیر قابل مشاهده یا فاقد سنسور (مسئله منطبق (Conformant))
  - ادراک عامل اصلا هیچ اطلاعاتی را فراهم نمی‌آورد.
  - راه‌حل یک دنباله است.
- غیرقطعی و/یا نیمه مشاهده‌پذیر (مسئله اقتضایی (Contingency))
  - ادراک اطلاعات کاملی درباره وضعیت دقیق عامل فراهم نمی‌آورند.
  - راه‌حل می‌تواند یک برنامه اقتضایی (Contingency plan) باشد و نه یک دنباله
- فضای حالت ناشناخته (مسئله اکتشاف(Exploration))

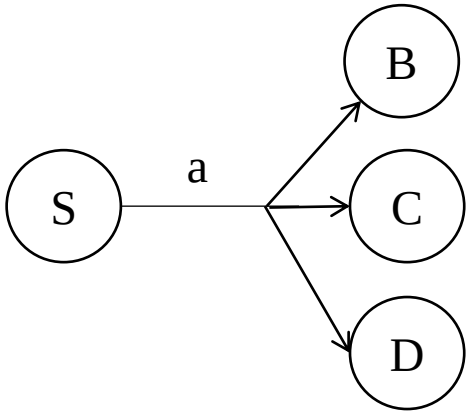


محیط‌های غیرقطعی یا نیمه مشاهده‌پذیر

- ادارک در محیط‌های نیمه مشاهده‌پذیر یا غیرقطعی مفید می‌شوند
  - نیمه مشاهده‌پذیر: محدود کردن مجموعه وضعیت‌های ممکنی که عامل ممکن است در آن باشد
  - غیرقطعی: کدام یک از نتایج ممکن عمل اتفاق افتاده است
- ادراک آینده نمی‌توانند از قبل مشخص شوند و اعمال آینده هم وابسته به همین ادراک است.
- راه‌حل یک برنامه‌ی اقتضایی است.
  - یک درخت تشکیل شده از عبارات if-then-else تودرتو
  - عامل وابسته به آن‌چه که دریافت کرده است چه کاری باید انجام دهد.

جستجو با اعمال غیر قطعی

جستجو با اعمال غیرقطعی

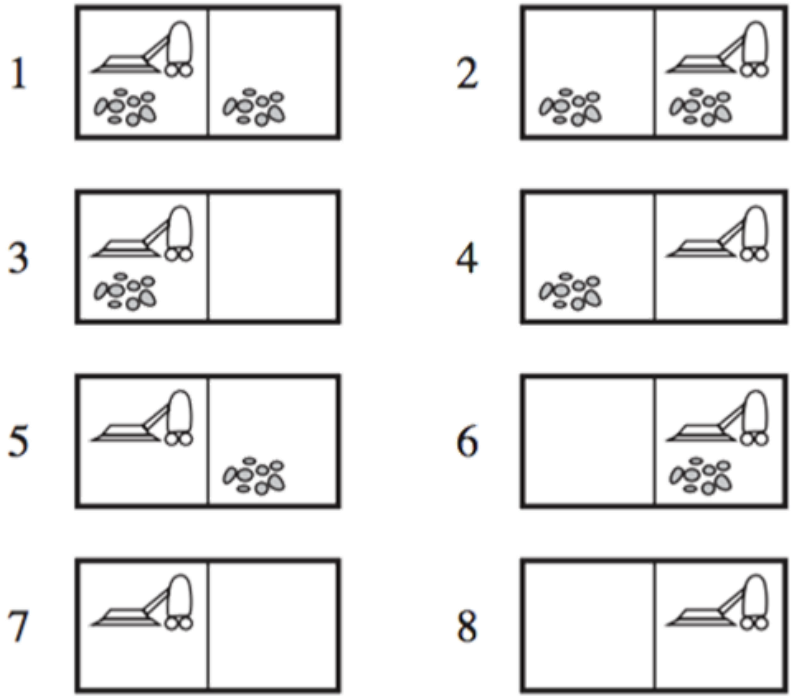


- در محیط‌های غیرقطعی، نتیجه‌ی هر عمل می‌تواند تغییر کند.
  - ادراک آینده می‌توانند مشخص کنند کدام نتیجه اتفاق افتاده است.

- تعمیم مدل‌سازی انتقال (Transition function)
  - استفاده از تابع $RESULTS : S \times A \rightarrow 2^S$ به‌جای $RESULT : S \times A \rightarrow S$

- درخت جستجو یک درخت AND-OR خواهد شد.
  - راه‌حل یک زیردرخت شامل برنامه اقتضایی خواهد بود (عبارات if-then-else تودتو)

دنیای جاروبرقی غیرقطعی (The erratic vacuum world)



$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } [\,]]$

اگر عامل در وضعیت اولیه یک باشد

- حالات {1,2,…,8}
- اعمال {Left, Right, Suck}
- هدف {7} یا {8}
- غیرقطعی بودن عمل Suck
  - هنگامی که بر روی یک خانه کثیف اعمال شود آن را تمیز می‌کند و گاهی اوقات خانه‌ی مجاور را نیز تمیز می‌کند.
  - هنگامی که بر روی یک خانه تمیز اعمال می‌شود این عمل گاهی اوقات آن خانه را کثیف می‌کند.
  - برای مثال: Results{1,suck}={5,7}

درخت جستجوی AND-OR

نود OR: انتخاب‌های عامل از اعمال در هر وضعیت

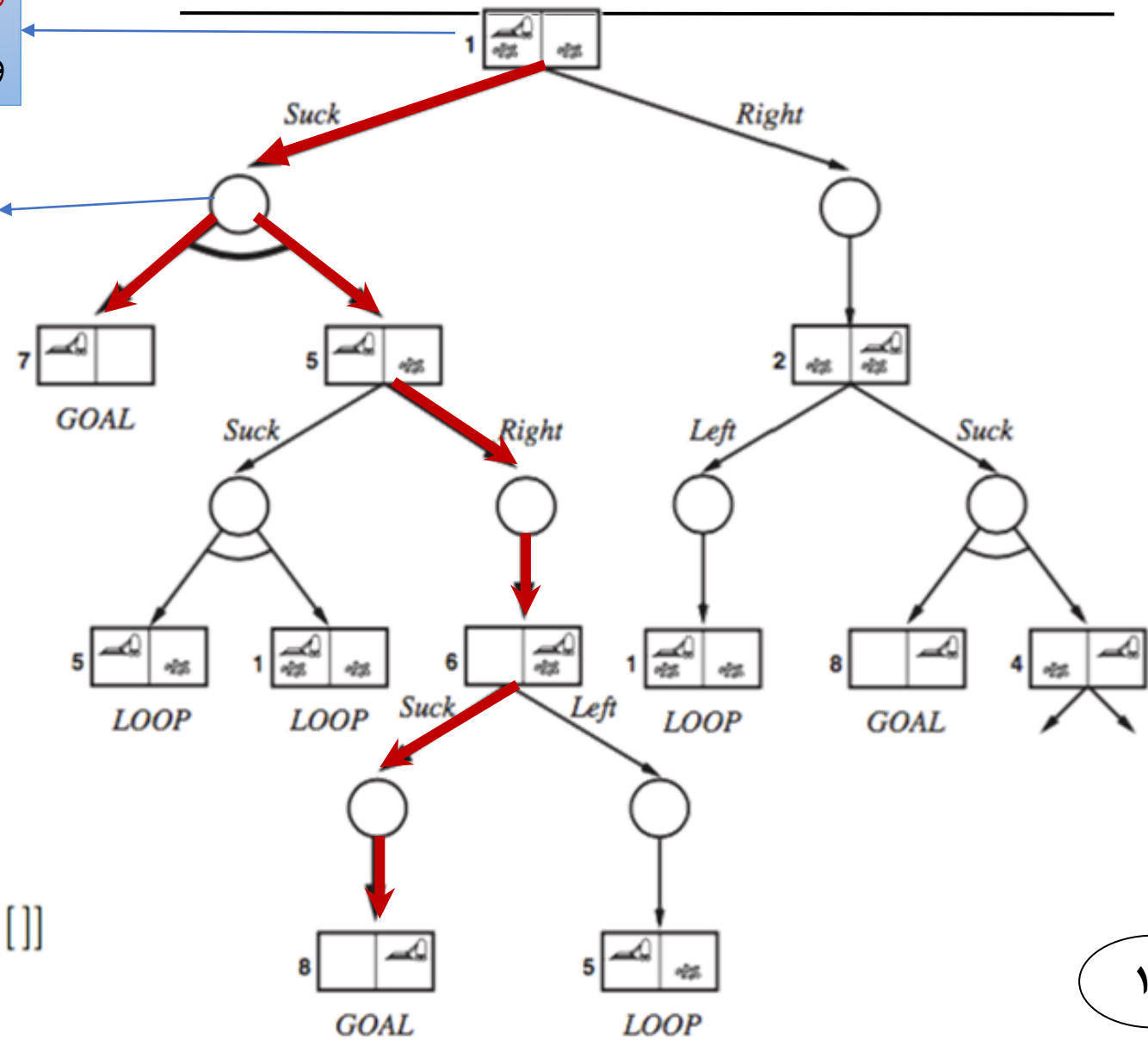
نود AND: انتخاب محیط از نتایج ممکن اعمال عامل

راه‌حل جستجوی AND-OR زیردرختی است که

-هر برگ آن یک نود هدف است

- در هریک از نودهای OR یک عمل را مشخص می‌کند.

- نتیجه‌ی هر عمل در هر شاخه از نودهای AND مشخص باشد.



$$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } [\,]]$$

جستجوی اول–عمق گرافی AND-OR

function AND-OR-GRAPH-SEARCH($problem$) **returns** a conditional plan or failure

OR-SEARCH($problem$.INITIAL-STATE, $problem$, [ ])

function OR-SEARCH($state$, $problem$, $path$) **returns** a conditional plan or failure

if $problem$.GOAL-TEST($state$) **then return** the empty plan

if $state$ is on $path$ **then return** $failure$

for each $action$ **in** $problem$.ACTIONS($state$) **do**

$plan \leftarrow$ AND-SEARCH(RESULTS($state$, $action$), $problem$, [$state$ | $path$])

if $plan \neq failure$ **then return** [$action$ | $plan$]

return $failure$

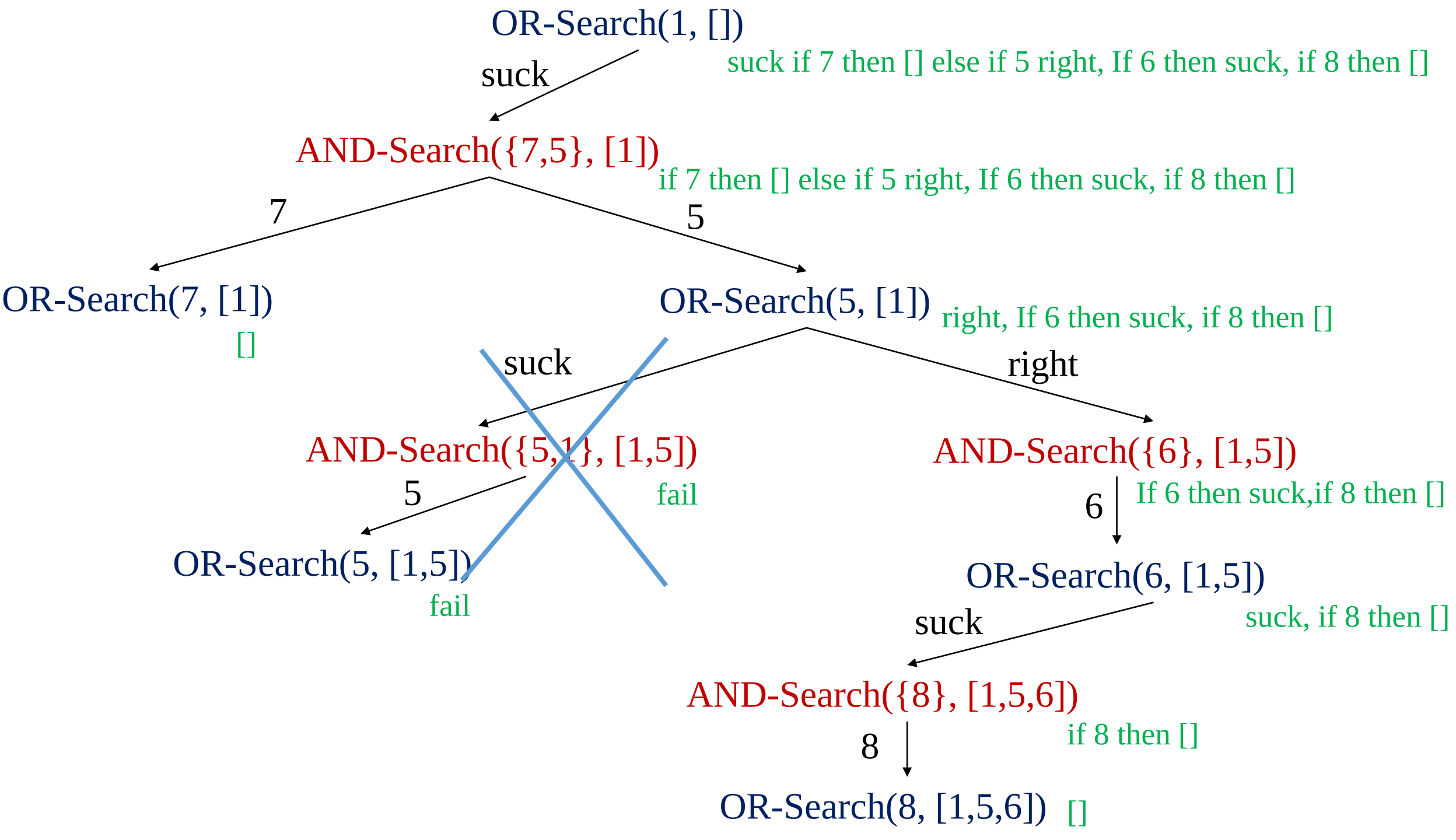
function AND-SEARCH($states$, $problem$, $path$) **returns** a conditional plan, or failure

for each s_i in $states$ **do**

$plan_i \leftarrow$ OR-SEARCH(s_i , $problem$, $path$)

if $plan_i = failure$ **then return** $failure$

return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** ... **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]


OR-Search(1, [])
suck
suck if 7 then [] else if 5 right, If 6 then suck, if 8 then []
AND-Search({7,5}, [1])
if 7 then [] else if 5 right, If 6 then suck, if 8 then []
7
5
OR-Search(7, [1])
[]
OR-Search(5, [1])
right, If 6 then suck, if 8 then []
suck
right
AND-Search({5,1}, [1,5])
fail
5
OR-Search(5, [1,5])
fail
AND-Search({6}, [1,5])
6
If 6 then suck,if 8 then []
OR-Search(6, [1,5])
suck
suck, if 8 then []
AND-Search({8}, [1,5,6])
8
if 8 then []
OR-Search(8, [1,5,6])
[]

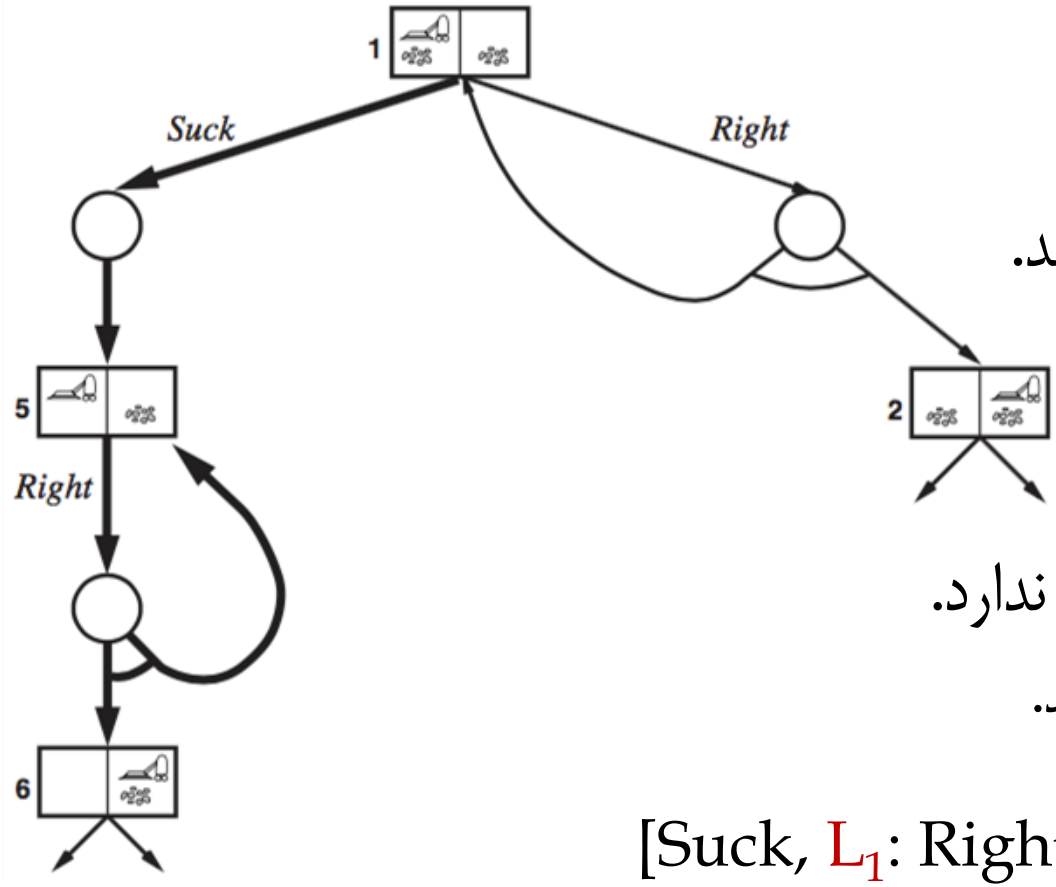
جستجوی اول–عمق گرافی AND-OR

- اغلب، چرخه‌ها (cycle) را در مسائل غیرقطعی مشاهده می‌کنیم.
  - برای مثال هنگامی‌که یک عمل گاهی اوقات هیچ تاثیری نداشته باشد.

- اگر وضعیت فعلی برابر با یکی از وضعیت‌های دیده شده روی مسیر از ریشه باشد آن‌گاه الگوریتم failure برمی‌گرداند.
  - اگر یک مسیر غیرچرخه به هدف وجود داشته باشد، از همان وضعیت مشابهی که قبلا مشاهده‌شده می‌توان به آن رسید.

- الگوریتم در هر فضای حالت متناهی پایان می‌یابد.
  - هر مسیر به یک هدف (goal) یا dead-end یا یک حالت تکراری (repeated state) می‌رسد.

دنیای جاروبرقی لغزان (Slippery)



- اعمال Left و Right گاهی اوقات با شکست مواجه می‌شوند و عامل در همان مکان باقی می‌ماند.
  - برای مثال Results{1,right}={1,2}

- در چنین مواردی هیچ راه‌حل بدون چرخه‌ای وجود ندارد.
  - راه‌حل چرخه‌ای: آن‌قدر ادامه بده تا یک عمل کار کند.

[Suck, L_1: Right, if state = 5 then L_1 else Suck

[Suck, while state = 5 do Right, Suck]

جستجو با مشاهدات جزئی

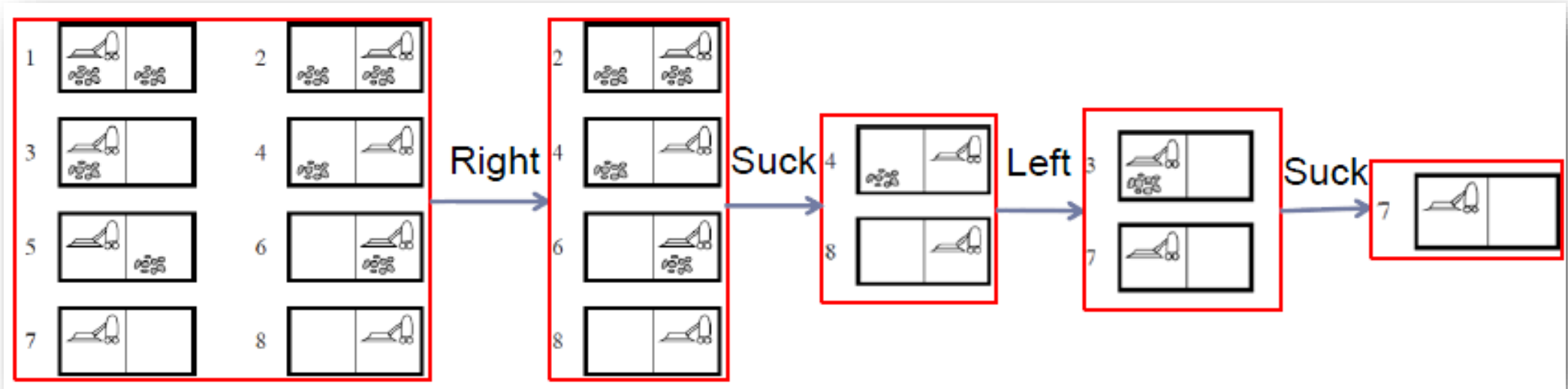
جستجو با مشاهدات جزئی

- عامل همیشه وضعیت دقیق خود را نمی‌داند.
  - اگر عامل در یکی از چند وضعیت ممکن باشد آن‌گاه یک عمل ممکن است منجر به یکی از چند نتیجه‌ی ممکن شود. (حتی اگر محیط قطعی باشد)

- وضعیت باور (Belief state)
  - باور فعلی عامل در مورد وضعیت‌هایی (وضعیت‌های فیزیکی) که ممکن است در آن باشد.
    - با فرض دنباله‌ی اعمال و ادراکاتی که تا کنون انجام داده است.

جستجو فاقد مشاهده یا فاقد سنسور

- ادراکات عامل هیچ‌گونه اطلاعاتی فراهم نمی‌آورد.
- دنیای جاروبرقی را به‌نحوی درنظر بگیرید که عامل جغرافیای محیط اطراف خود را می‌داند اما مکان خود و توزیع آشغال‌ها را نمی‌داند.
  - حالت اولیه: initial belief = {1,2,3,4,5,6,7,8}



فرموله‌سازی مسئله‌ی فاقد سنسور

- جستجو در فضای حالت باور به‌جای فضای حالت فیزیکی
  - در این فضا مسئله کاملا مشاهده‌پذیر است. چرا؟
  - راه‌حل همیشه یک دنباله عمل است. اگر محیط غیرقطعی باشد چطور؟
- فرض کنید مسئله‌ی فیزیکی P به‌صورت زیر تعریف شده باشد
  - حالات: N حالت فیزیکی
  - اعمال: ACTIONS_P
  - مدل انتقال: RESULTS_P
  - آزمون هدف: GOAL-TEST_P
  - هزینه‌ی گام: STEP-COST_P
- در این‌صورت مسئله‌ی بدون سنسور را می‌توان براساس این مسئله تعریف نمود.

فرموله‌سازی مسئله‌ی فاقد سنسور ...

- حالات: هر مجموعه ممکن از وضعیت‌های فیزیکی، 2^N
- حالت اولیه: معمولا مجموعه همه‌ی وضعیت‌های فیزیکی
- اعمال: $ACTIONS(b) = \bigcup_{s \in b} ACTIONS_P(s)$
  - اعمال غیر قانونی؟ اگر $b=\{s_1, s_2\}$ آن‌گاه $ACTIONS_P(s_1) \neq ACTIONS_P(s_2)$
  - اگر اعمال غیرقانونی هیچ تاثیری روی محیط نداشته باشند می‌توان از آن‌ها اجتماع گرفت
  - اگر یک عمل غیرقانونی تاثیر بدی در یک حالت داشته باشد بهتر است از اشتراک اعمال استفاده نمود.

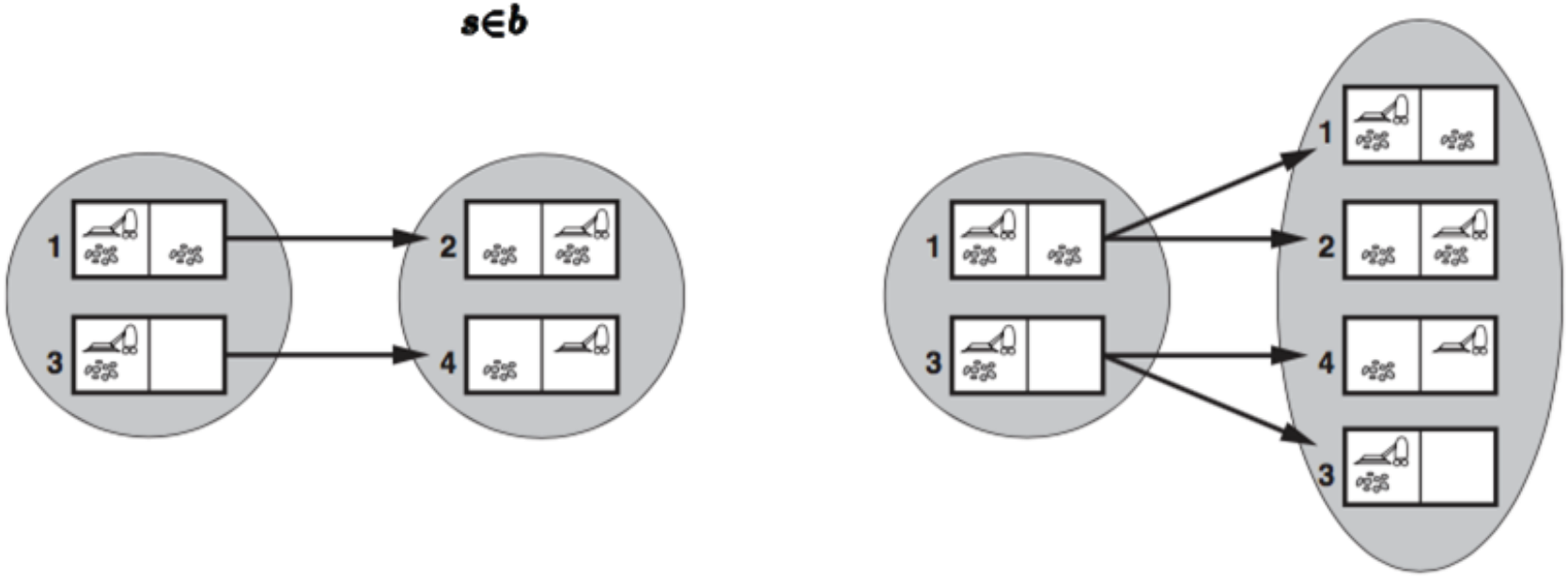
فرموله‌سازی مسئله‌ی فاقد سنسور ...

- مدل انتقال $b' = PREDICT_P(b,a)$
- اعمال قطعی

$$b' = \text{RESULT}(b,a) = \{s' : s' = \text{RESULT}_P(s,a) \text{ and } s \in b\}$$

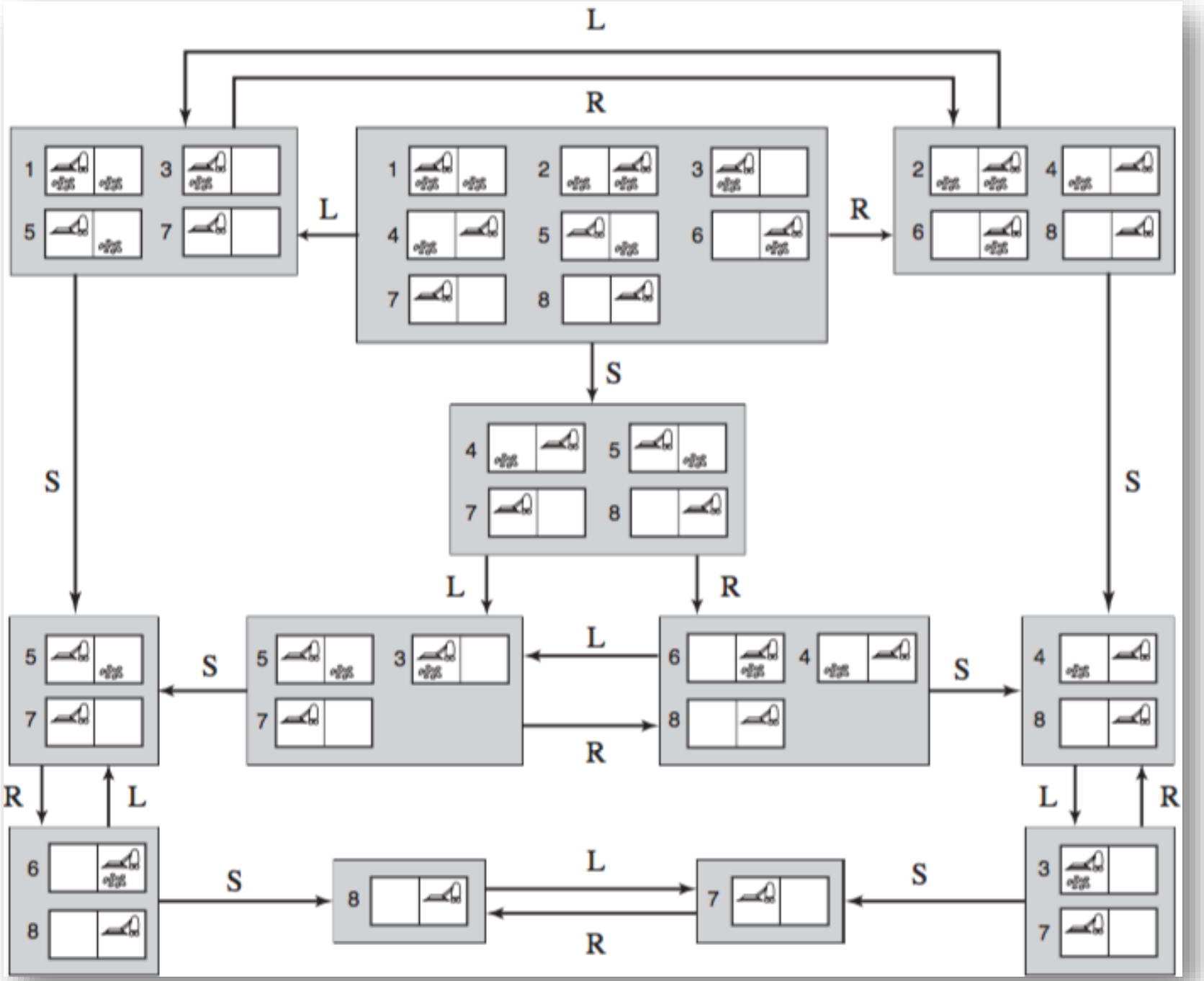
- اعمال غیرقطعی

$$b' = \text{RESULT}(b,a) = \{s' : s' \in \text{RESULTS}_P(s,a) \text{ and } s \in b\} = \bigcup_{s \in b} \text{RESULTS}_P(s,a) ,$$



فرموله‌سازی مسئله‌ی فاقد سنسور ...

- آزمون هدف: اگر تمامی وضعیت‌های موجود در حالت باور شرط GOAL-TEST_P را برقرار سازند، آن حالت باور هدف است.
- هزینه‌ی مسیر: هزینه‌های گام برابر با STEP-COST_P خواهد بود اگر هزینه‌ی یک عمل در تمامی حالات یکسان باشد.
- با این فرموله‌سازی هر یک از الگوریتم‌های جستجوی فصل ۳ قابل به‌کارگیری هستند.
- شکل اسلاید بعد فضای حالت باور دنیای جاروبرقی بدون سنسور و قطعی را نشان می‌دهد.
  - تعداد حالات باور ممکن: 2^8
  - تعداد حالات باور واقعی مسئله: ۱۲


L
R
1
3
5
7
L
1
2
3
4
5
6
7
8
R
2
4
6
8
S
S
4
5
7
8
S
L
R
5
7
S
5
3
7
L
R
6
4
8
S
4
8
R
L
L
R
6
8
S
8
L
R
7
S
3
7

مشکلات جستجو در فضای حالت باور

تعداد حالات باور قابل رسیدن (reachable)

- راه‌حل
  - استفاده از الگوریتم‌های جستجوی گرافی تعمیم‌یافته با استفاده از ویژگی زیر
    - اگر یک دنباله عمل، یک راه‌حل برای حالت باور b باشد آن‌گاه یک راه‌حل برای هر زیرمجموعه از b نیز هست. (تمرین: شبه‌کد این الگوریتم را بنویسید)
- در اکثر موارد ضریب انشعاب و طول راه‌حل در فضای حالت باور و فضای حالت فیزیکی خیلی متفاوت نیستند.

مشکلات جستجو در فضای حالت باور ...

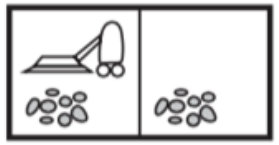
تعداد حالات فیزیکی در هر یک از حالات باور (مشکل اصلی)

- برای مثال حالت باور اولیه در یک دنیای جاروبرقی ۱۰×۱۰ شامل ۱۰۰×۲^۱۰۰ حالت فیزیکی است.
- راه‌حل:
  - نمایش حالت باور با توصیف فشرده‌تر (با استفاده از نمایش‌های رسمی)
  - جستجوی حالت باور افزایشی (incremental) برای کشف سریع عدم موفقیت (failure)
    - ابتدا یک راه‌حل برای حالت اول در مجموعه حالات حالت باور اولیه پیدا کن
    - بررسی کن که آیا این راه‌حل برای سایر حالات موجود در حالت باور کار می‌کند یا خیر.
    - اگر نه برگرد و یک راه‌حل دیگر برای حالت اول پیدا کن و این کار را ادامه بده.

جستجو همراه با مشاهدات

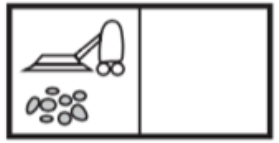
- برای یک مسئله نیمه مشاهده‌پذیر باید مشخص کنیم چگونه محیط ادارکات را برای عامل تولید می‌کند.
  - استفاده از تابع PERCEPT(s) که ادراک دریافت‌شده از وضعیت s را برمی‌گرداند.
    - کاملا مشاهده‌پذیر: PERCEPT(s)=s
    - فاقد حس‌گر: PERCEPT(s)=null
- مثال: دنیای جاروبرقی با حس‌گر محلی
  - وجود حس‌گری برای تشخیص مکان
  - وجود حس‌گری برای تشخیص گردوخاک در مکانی که در آن حضور دارد
  - عدم وجود حس‌گر برای کشف گردوخاک در محل‌های دیگر

1



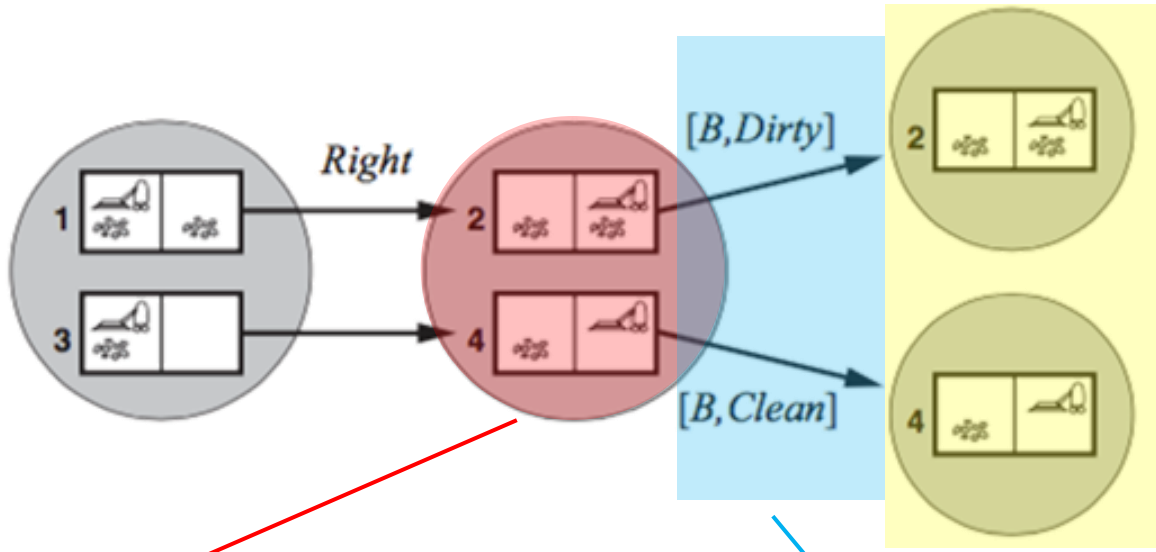
[A, Dirty]

3



مثال– دنیای جاروبرقی با حس‌گر محلی گردوخاک

دنیای قطعی



$UPDATE(\hat{b})$

حالت باور حاصل از هر ادراک عامل چیست؟

$PREDICT(b, \text{Right})$

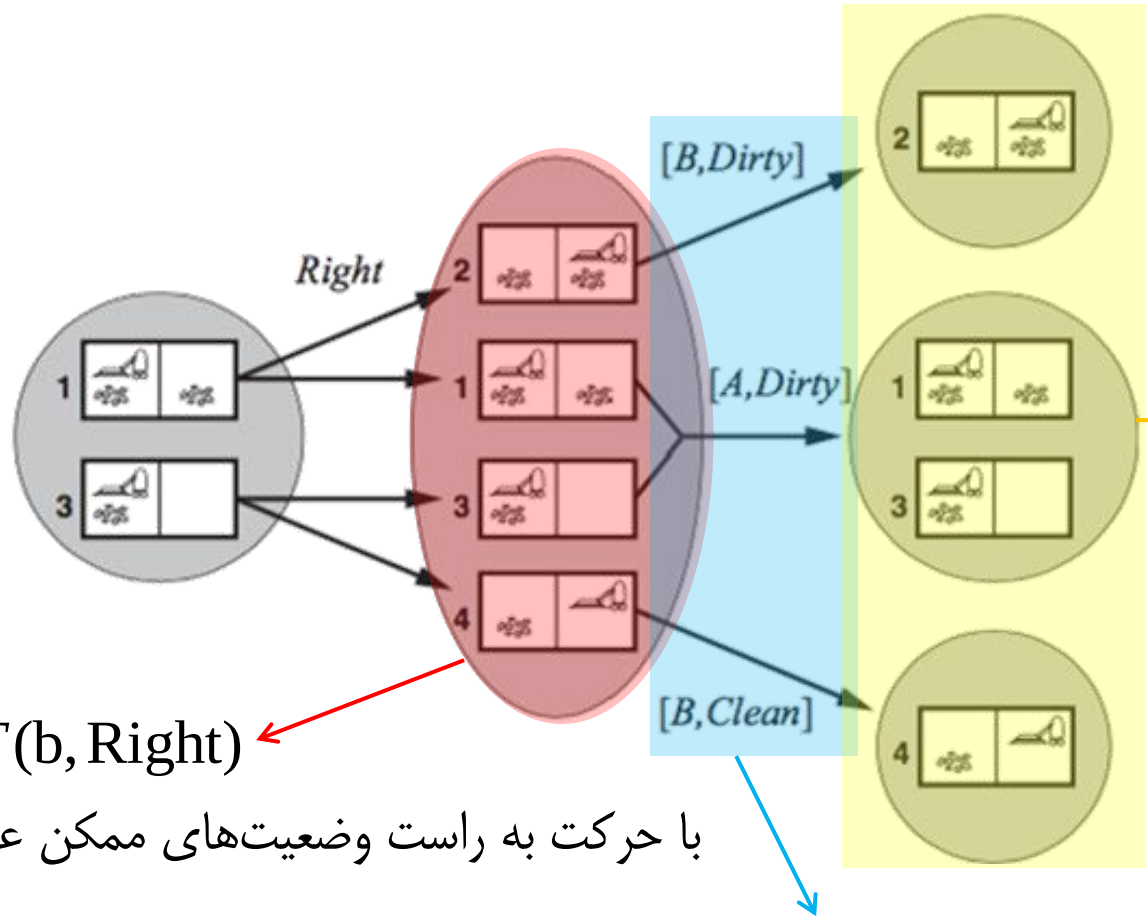
با حرکت به راست وضعیت‌های ممکن عامل چه هستند؟

$POSSIBLE\_PERCEPTS(\hat{b})$

مجموعه‌ی ادراکاتی که می‌توان در حالت جدید فرض نمود چه هستند؟

مثال– دنیای جاروبرقی با حس‌گر محلی گردوخاک

دنیای غیرقطعی: جاروبرقی لغزان



$UPDATE(\hat{b})$

حالت باور حاصل از هر ادراک عامل چیست؟

$PREDICT(b, \text{Right})$

با حرکت به راست وضعیت‌های ممکن عامل چه هستند؟

$POSSIBLE\_PERCEPTS(\hat{b})$

مجموعه‌ی ادراکاتی که می‌توان در حالت جدید فرض نمود چه هستند؟

مدل انتقال محیط با مشاهده‌ی جزئی

- مرحله‌ی پیش‌بینی (Prediction stage): بعد از انجام یک عمل، حالت باور چه تغییری می‌کند؟

$$\hat{b} = PREDICT(b, a)$$

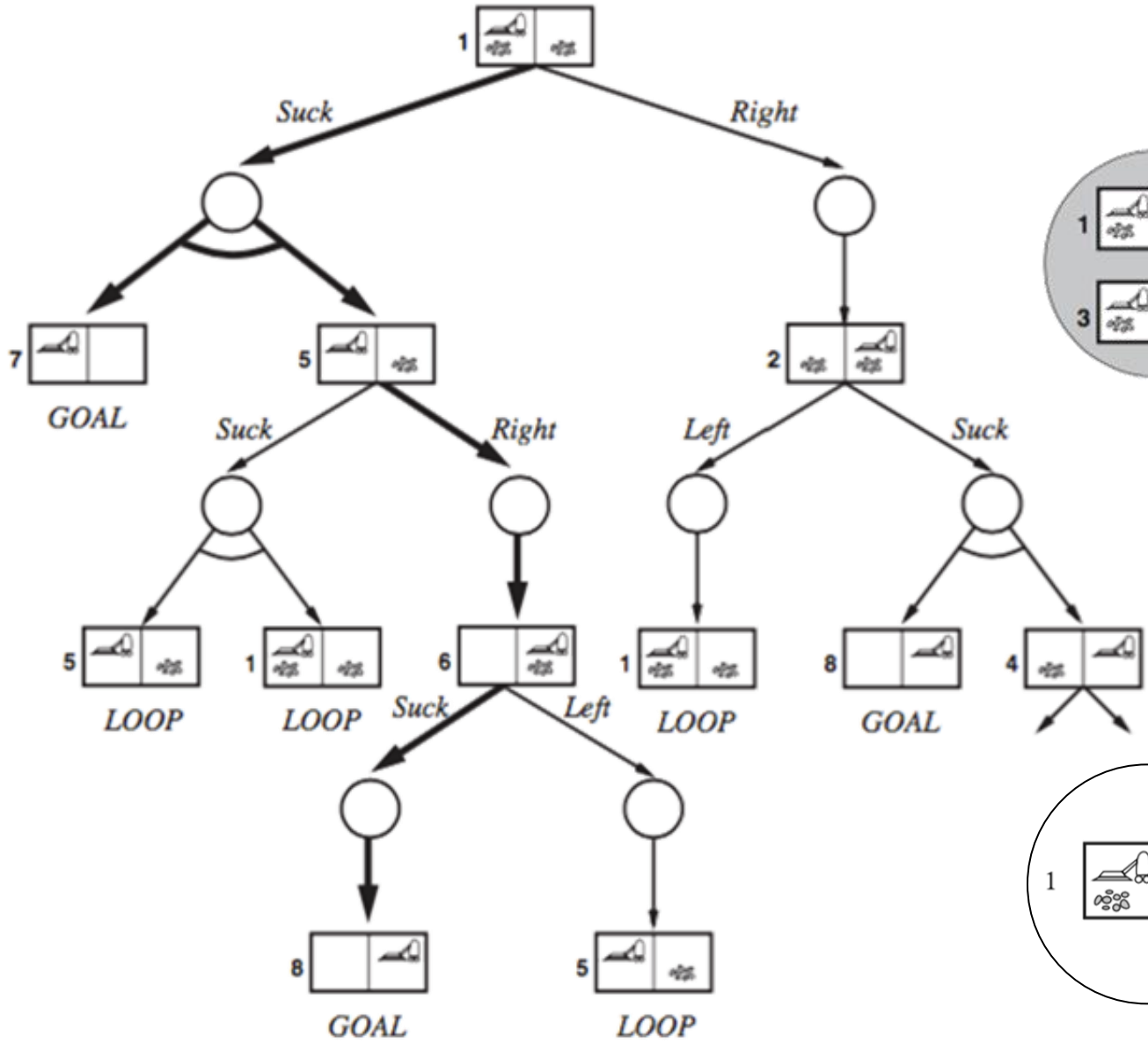
- مرحله‌ی پیش‌بینی مشاهدات (Observation prediction stage): ادارک ممکن در یک حالت باور چه هستند؟

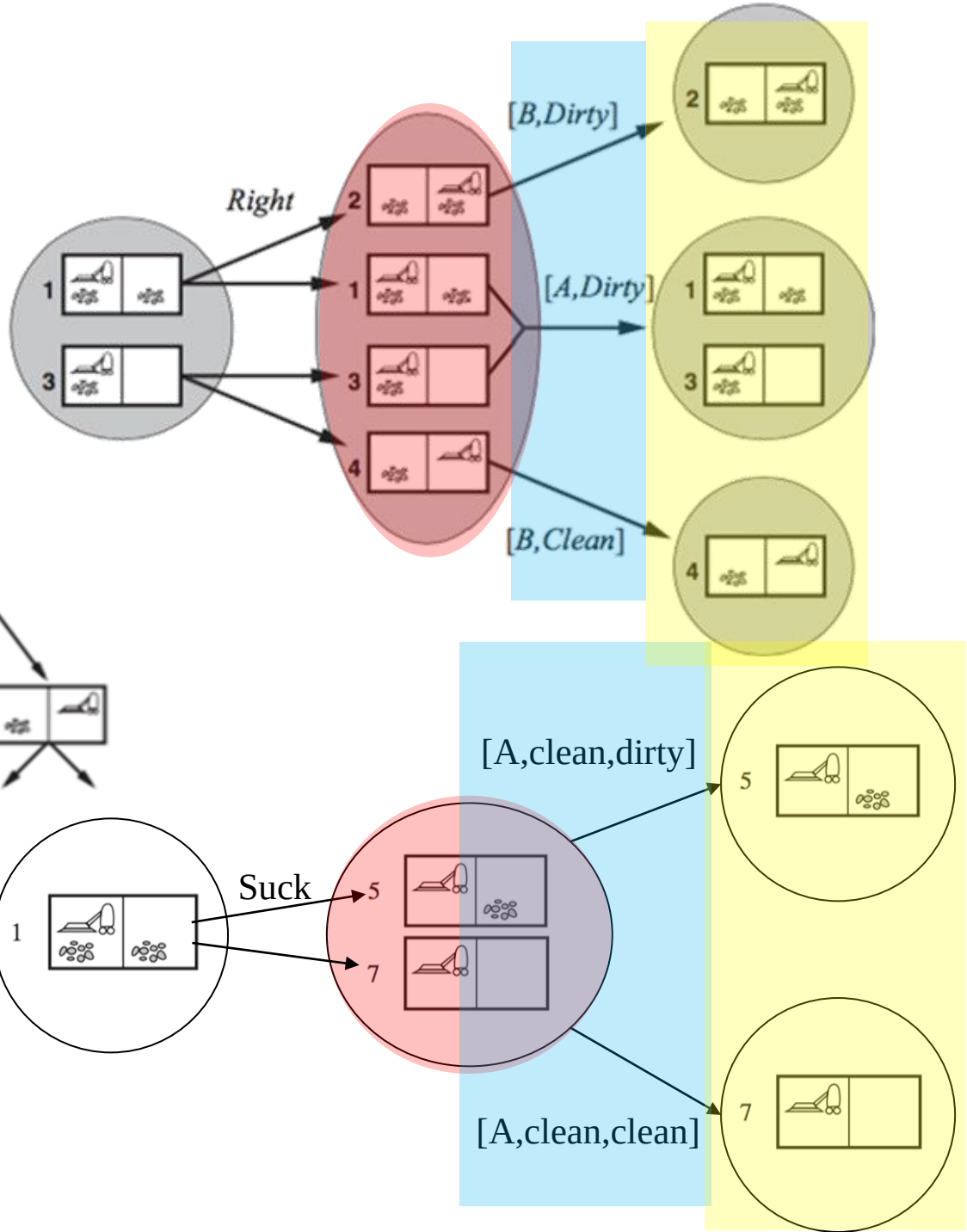
$$\text{POSSIBLE-PERCEPTS}(\hat{b}) = \{o : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b}\}$$

- مرحله‌ی به‌روزرسانی (Update stage): بعد از هر ادراک، حالت باور چه تغییری می‌کند؟

$$b_o = \text{UPDATE}(\hat{b}, o) = \{s : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b}\}$$

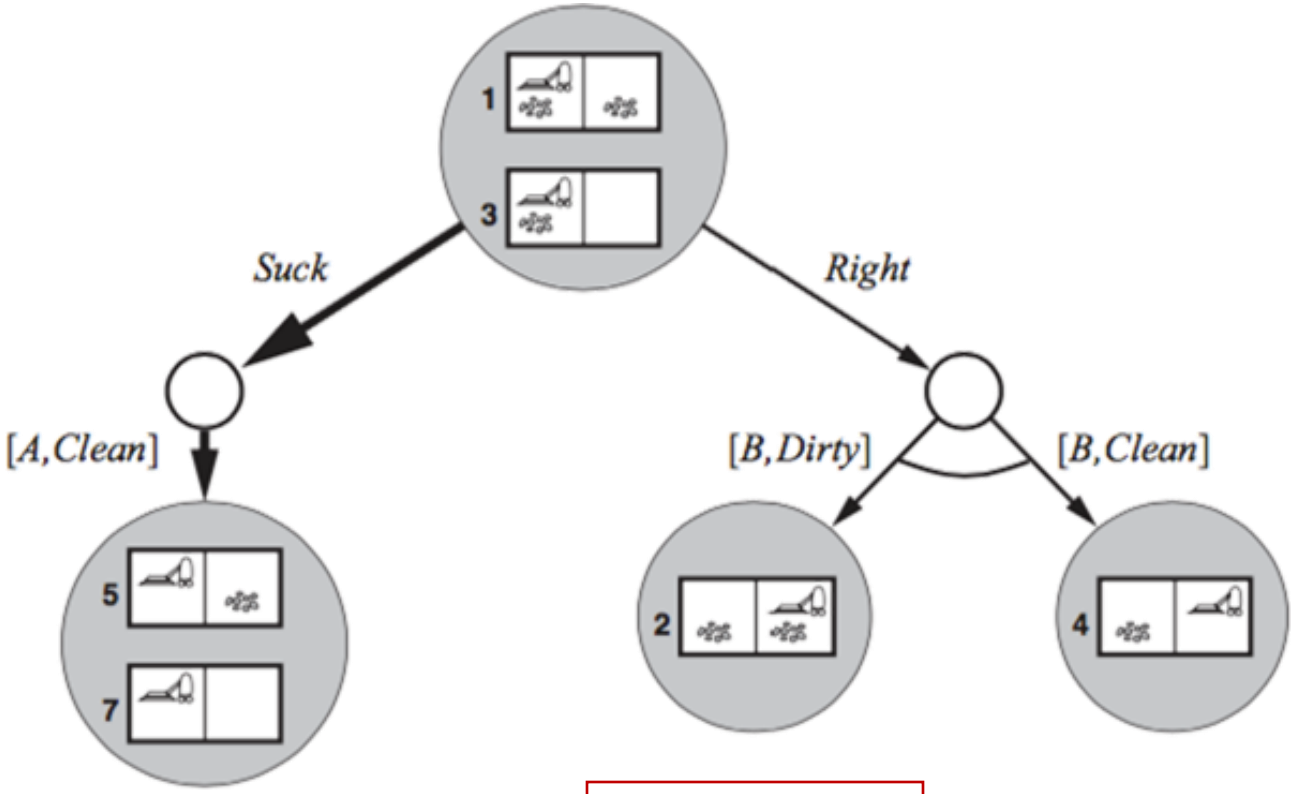
$$\text{RESULTS}(b, a) = \{b_o : b_o = \text{UPDATE}(\text{PREDICT}(b, a), o) \text{ and } o \in \text{POSSIBLE-PERCEPTS}(\text{PREDICT}(b, a))\}$$

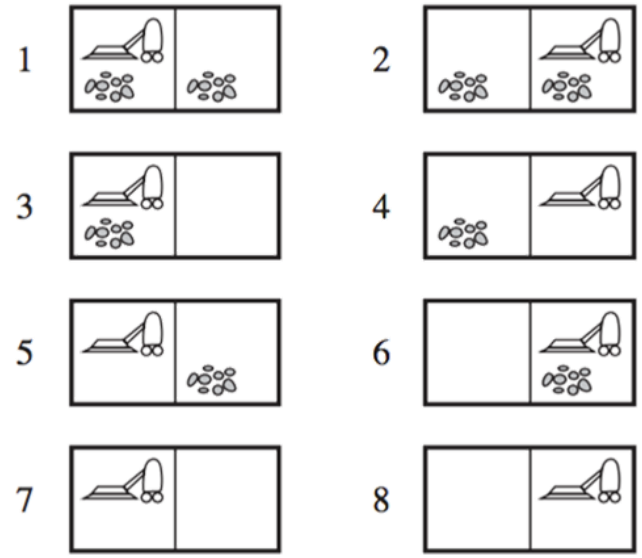

1
Suck
Right
7
GOAL
5
2
Suck
Right
Left
Suck
5
LOOP
1
LOOP
6
1
LOOP
8
GOAL
4
Suck
Left
8
GOAL
5
LOOP


Right
1
3
2
1
3
4
[B,Dirty]
[A,Dirty]
[B,Clean]
2
1
3
4
Suck
1
5
7
[A,clean,dirty]
[A,clean,clean]
5
7

حل مسائل نیمه مشاهده‌پذیر

- اعمال الگوریتم AND-OR Tree Search بر روی فضای حالت باور
- بخشی از درخت جستجوی AND-OR از محیط جاروبرقی با ادراک اولیه [A,Dirty]





[*Suck*, *Right*, **if** $Bstate = \{6\}$ **then** *Suck* **else** [ ]]

عامل محیط‌های نیمه مشاهده‌پذیر

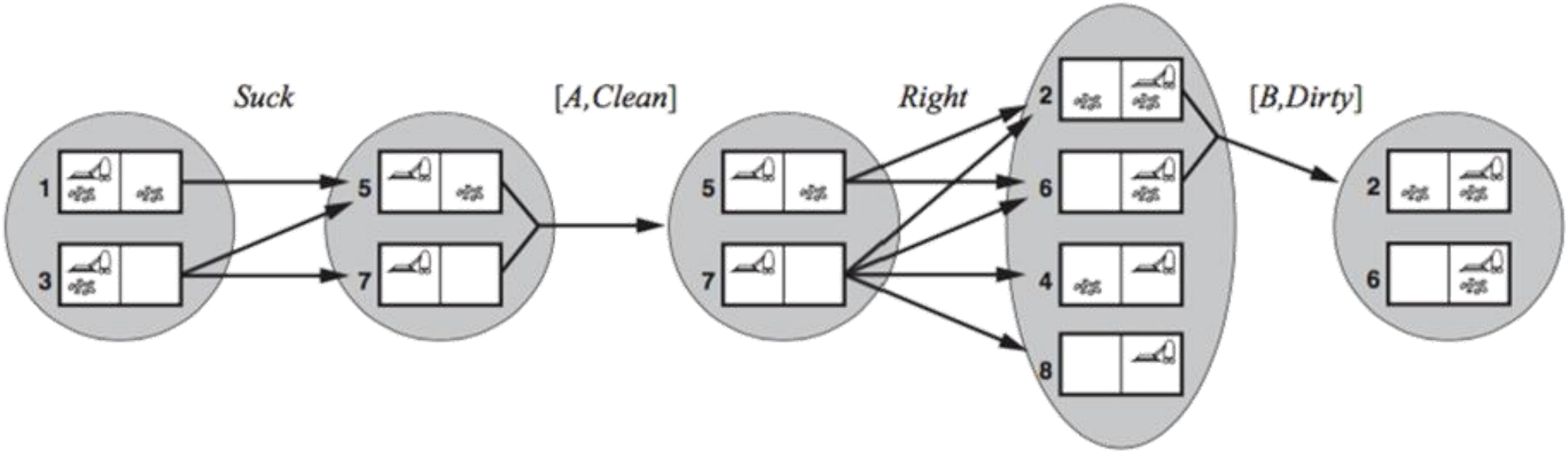
- طراحی عامل حل مسئله برای محیط‌های نیمه مشاهده‌پذیر کاملا مشابه با عامل حل مسئله ساده است
  - فرموله کردن مسئله، فراخوانی الگوریتم جستجو برای به‌دست آوردن راه‌حل و اجرای راه‌حل
- اما دو تفاوت اصلی با آن دارد
  - راه‌حل یک مسئله یک برنامه‌ی شرطی به جای یک دنباله است.
    - براساس ادارک فعلی یا بخش then اجرا می‌شود یا بخش else
  - عامل حالت باورش را هنگامی که اقدامات انجام می‌دهد و ادراکات دریافت می‌کند به‌روز می‌کند.
    - با فرض حالت باور b، عمل a و مشاهده o خواهیم داشت:

$$b' = \text{UPDATE}(\text{PREDICT}(b, a), o)$$

  - نگه‌داری یک حالت باور یک تابع هسته از هر سیستم هوشمند است.

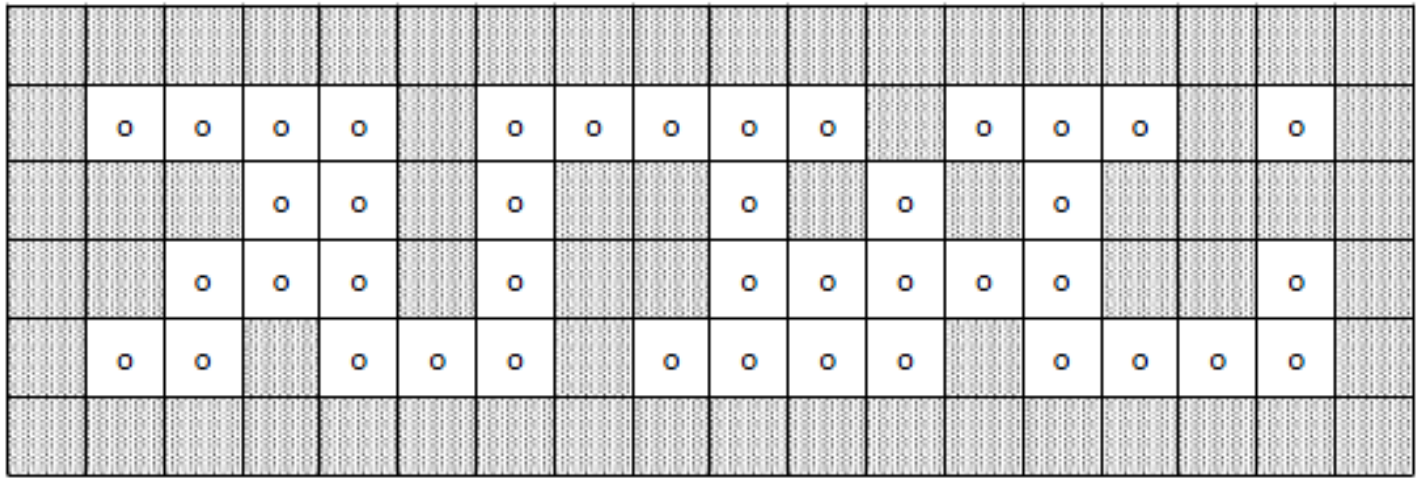
مثال – دنیای جاروبرقی مهدکودک

- دنیای جاروبرقی با حس‌گر محلی را در حالتی در نظر بگیرید که هر مربع آن ممکن است دوباره کثیف شود مگر آن‌که جاروبرقی همیشه در حال تمیز کردن باشد.
  - شکل زیر دو چرخه به‌روزرسانی – پیش‌بینی را نشان می‌دهد.
  - مرحله‌ی به‌روزرسانی – پیش‌بینی باید با سرعت ورود ادارکات انجام شود در غیر این‌صورت عامل عقب خواهد افتاد.



مثال – مکان‌یابی ربات

- با داشتن یک نقشه از محیط و یک دنباله از ادراک و اقدامات، مکان فعلی ربات کجاست؟
  - ربات در هر سمت خود دارای یک سنسور برای تشخیص مانع است.
    - مثال: ادارک NW یعنی مانع‌هایی در شمال و غرب آن وجود دارد.
  - عمل حرکت آن دچار مشکل شده است و به‌طور تصادفی یکی از چهار عمل {راست، چپ، بالا، پایین} را انتخاب می‌کند.



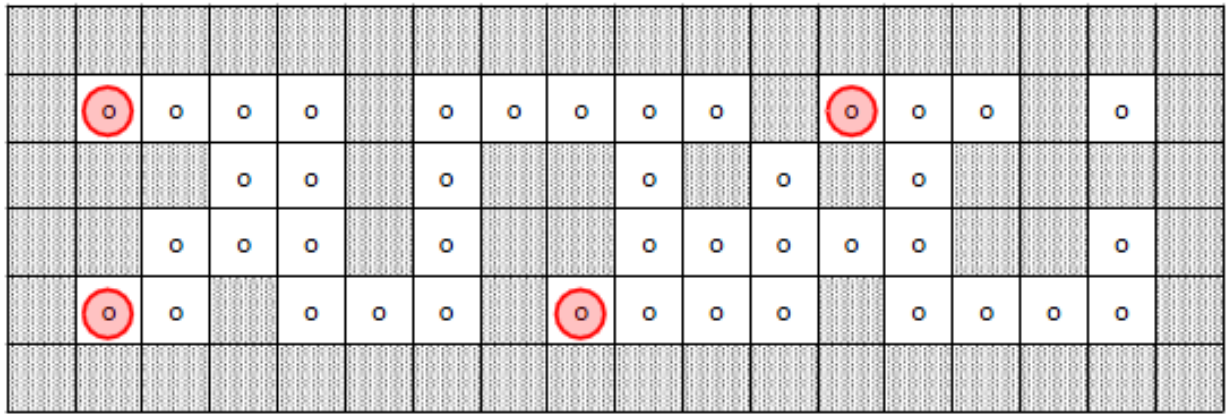
مثال – مکان‌یابی ربات ...

b^0: o squares

Percept: NSW

$b^1 = UPDATE(b^o, NSW)$

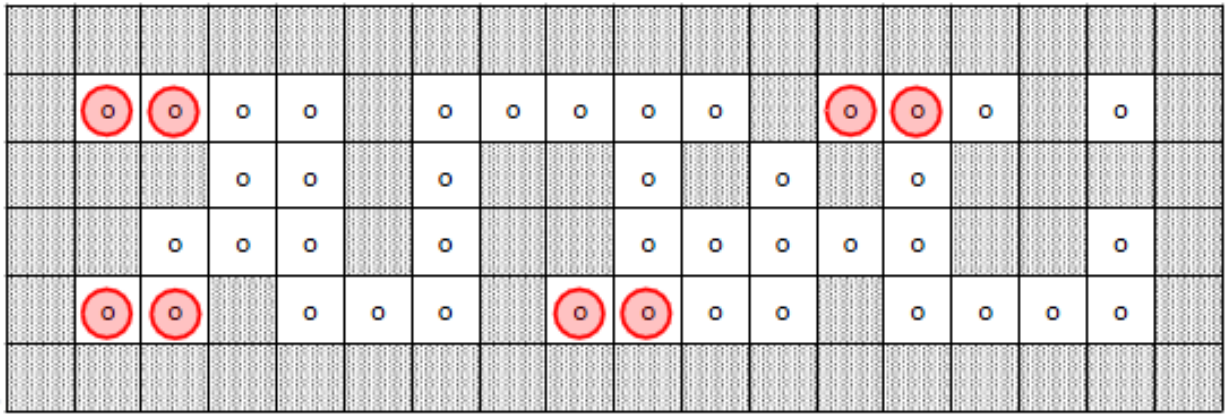
(red circles)



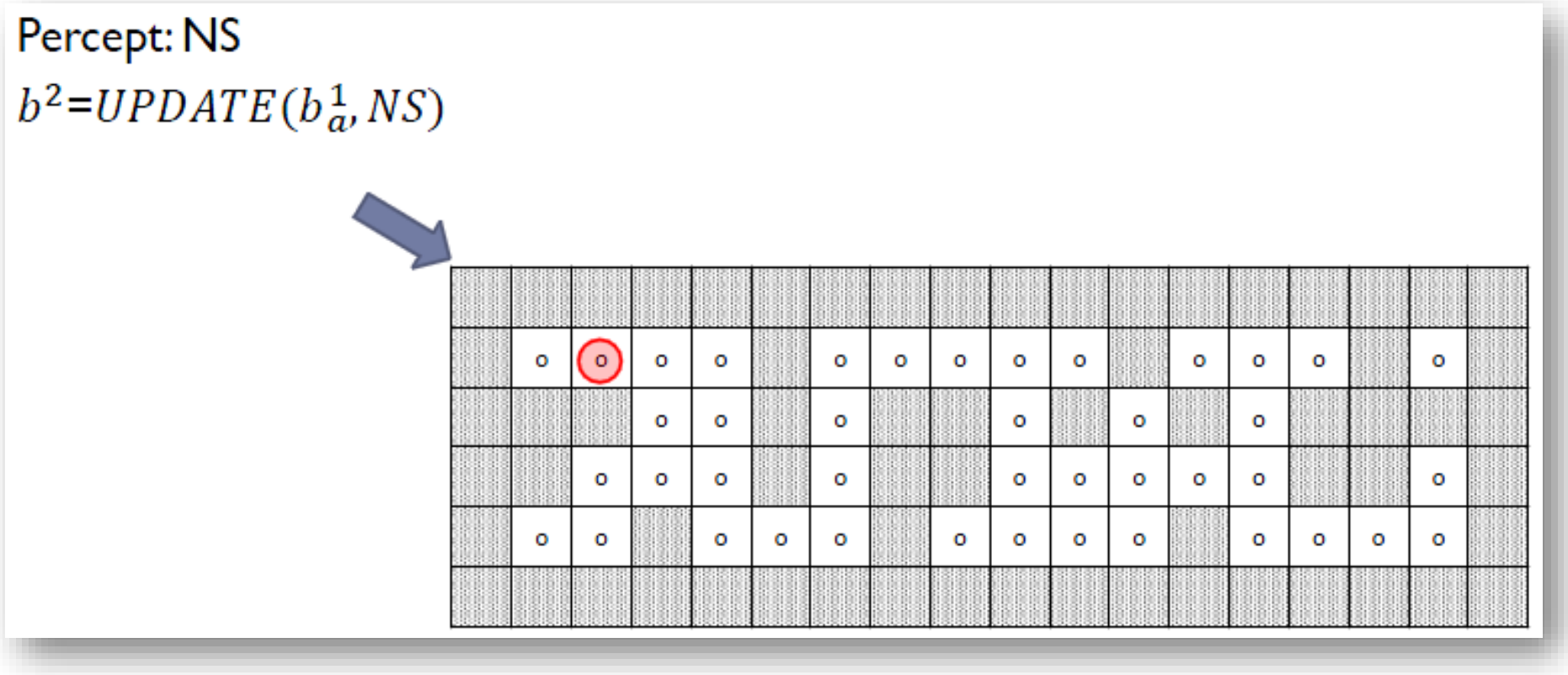
Execute action $a = Move$

$b_a^1 = PREDICT(b^1, a)$

(red circles)



مثال – مکان‌یابی ربات ...



- به‌طور کلی: $\text{UPDATE}(\text{PREDICT}(\text{UPDATE}(b, NSW), Move), NS)$

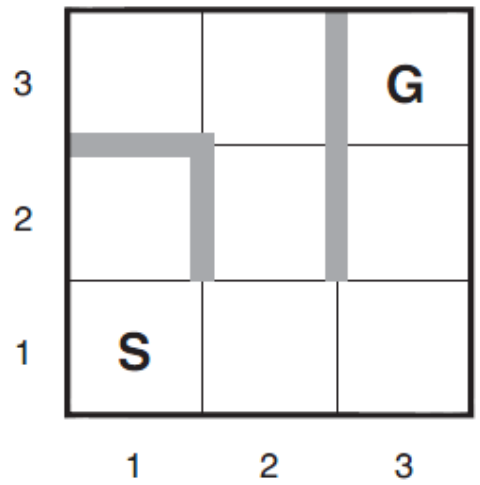
عامل‌های جستجوی برخط و محیط‌های ناشناخته

جستجوی برون خط در مقابل جستجوی برخط

- جستجوی برون خط (Offline search): یافتن راه‌حل قبل از ورود به دنیای واقعی
- جستجوی برخط (Online search): محاسبه و عمل بصورت یک در میان
  - سودمند برای دامنه‌های غیرقطعی: تمرکز توان محاسباتی بر روی آن‌چه که در طول اجرا اتفاق می‌افتد
    - مصالحه میان یک طرح اقتضایی تضمین‌شده (که در یک وضعیت نامطلوب در طول اجرا گیر نکند) و زمان مورد نیاز برای برنامه‌ریزی کامل روبه جلو
  - مناسب برای محیط‌های پویا و نیمه‌پویا
  - ضروری در محیط‌های ناشناخته
    - مواجه شدن با مسئله‌ی اکتشافی: استفاده از اعمال به‌عنوان آزمایشاتی برای یادگیری اعمال آینده
    - مثال‌ها: یک ربات در یک محیط جدید، نوزاد تازه متولدشده

مسائل جستجوی برخط

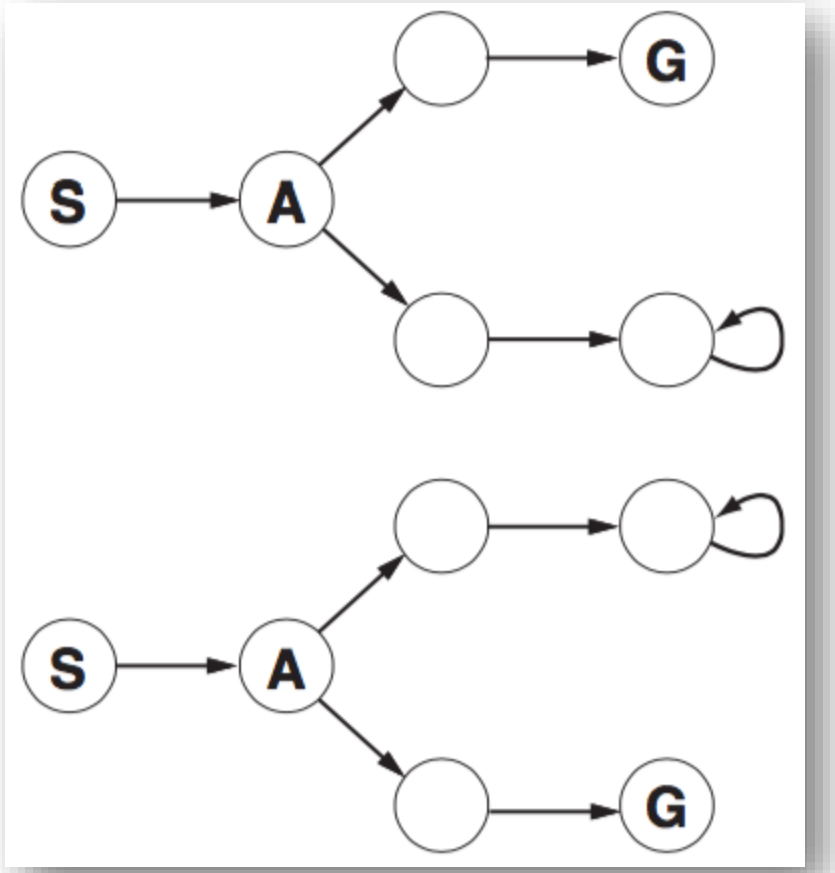
- دانش عامل
  - ACTIONS(s): لیست اعمال مجاز در حالت s
  - c(s, a, s’): تابع هزینه گام تنها پس از تعیین s’
    - عامل نمی‌تواند RESULT(s,a) را مشخص کند مگر با انجام دادن واقعی عمل a در وضعیت s
  - GOAL-TEST(s)
- فرضیات
  - محیط قطعی
  - امکان دستیابی به هیوریستیک قابل قبول h(s)
    - مانند فاصله منهتن برای شکل



نسبت رقابتی (Competitive ratio)

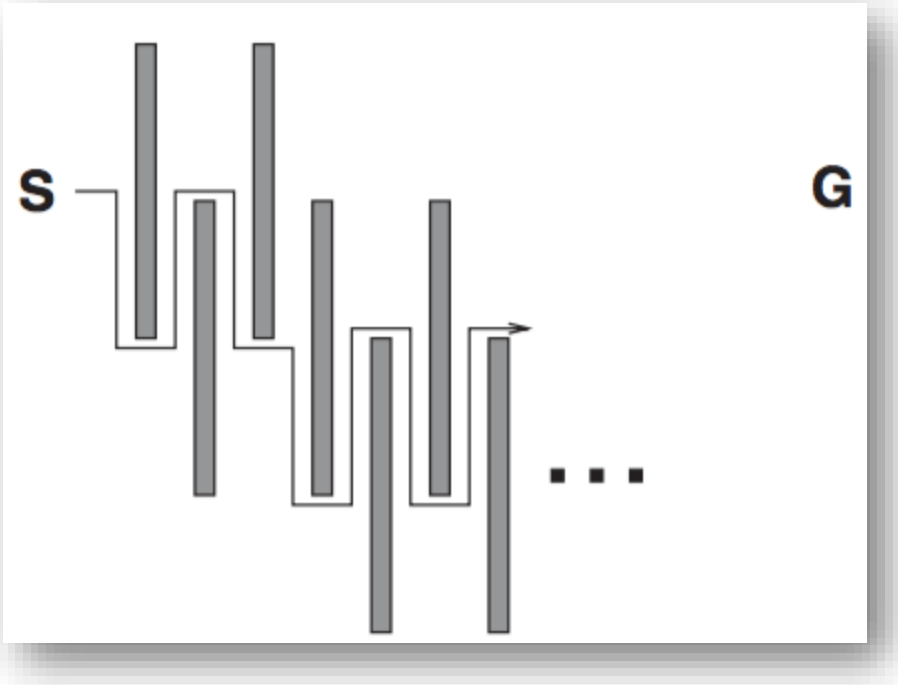
- هدف: رسیدن به حالت هدف با حداقل هزینه
  - هزینه مسیر برخط = کل هزینه مسیر پیموده شده توسط عامل در واقعیت
  - بهترین هزینه = هزینه‌ی کوتاه‌ترین مسیر اگر عامل فضای جستجو را از قبل می‌دانست
- نسبت رقابتی = هزینه مسیر برخط تقسیم بر بهترین هزینه
  - مقادیر کوچک‌تر مطلوب‌تر هستند.
- نسبت رقابتی ممکن است نامحدود باشد
  - رسیدن به یک بن بست (dead-end state) که از آن هیچ حالت هدفی قابل دسترسی نیست.
    - اعمال غیرقابل برگشت می‌توانند منجر به وضعیت‌های بن‌بست شوند.
  - وجود داشتن مسیرهایی با هزینه‌های نامحدود و عدم تضمین نسبت رقابتی کران‌دار

مثال – نسبت رقابتی (بن‌بست)



- فرض کنید حالت های ملاقات شده A و S باشد، با توجه به مشابه بودن فضاهای حالت در هر دو فضا یکی از حالات به‌عنوان حالت بعدی درنظر گرفته می‌شود.
  - در یکی از فضاهای حالت شکست می خورد.
  - هیچ الگوریتمی نمی‌تواند از بن‌بست‌ها در تمامی فضاهای حالت اجتناب کند.
- فرض ساده‌سازی: فضای حالت قابل کاوش امن (safely explorable)
  - از هر حالت دست یافتنی حداقل یک حالت هدف قابل دسترس باشد.
  - مثلا فضاهای حالت با اعمال قابل برگشت

مثال – نسبت رقابتی (هزینه نامحدود)



- یک محیط ۲ بعدی را تصور کنید که حریف قادر است در حین کاوشِ عامل، فضای حالت را بسازد
- باعث می‌شود یک عامل جستجوی برخط، یک مسیر اجبارا ناکارامد را به سمت هدف دنبال کند.

عامل جستجوی برخط

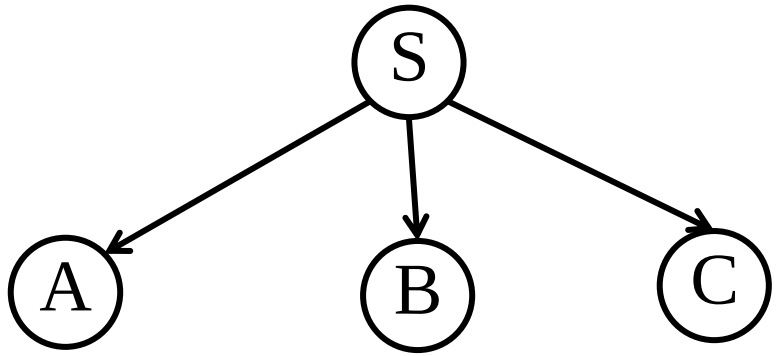
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

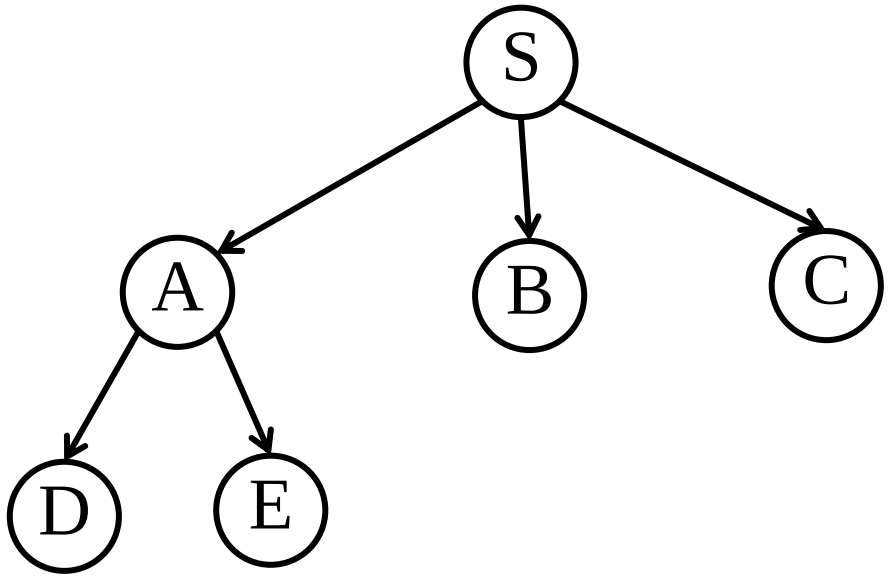
عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی



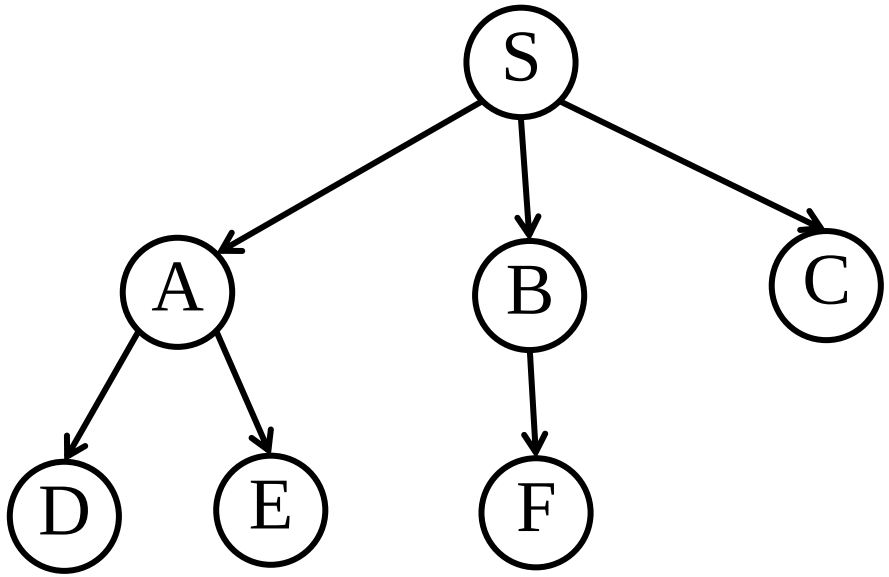
عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی



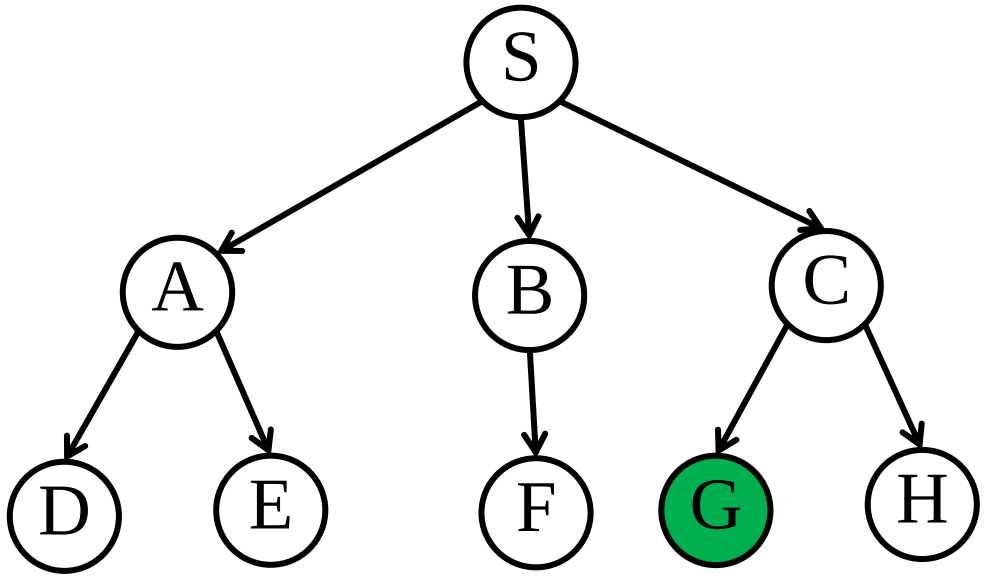
عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی



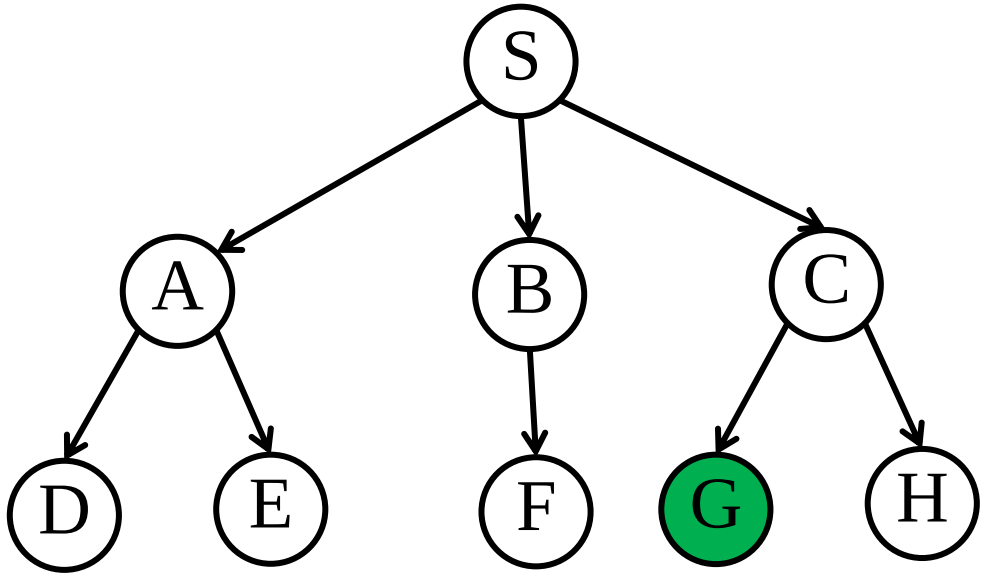
عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی



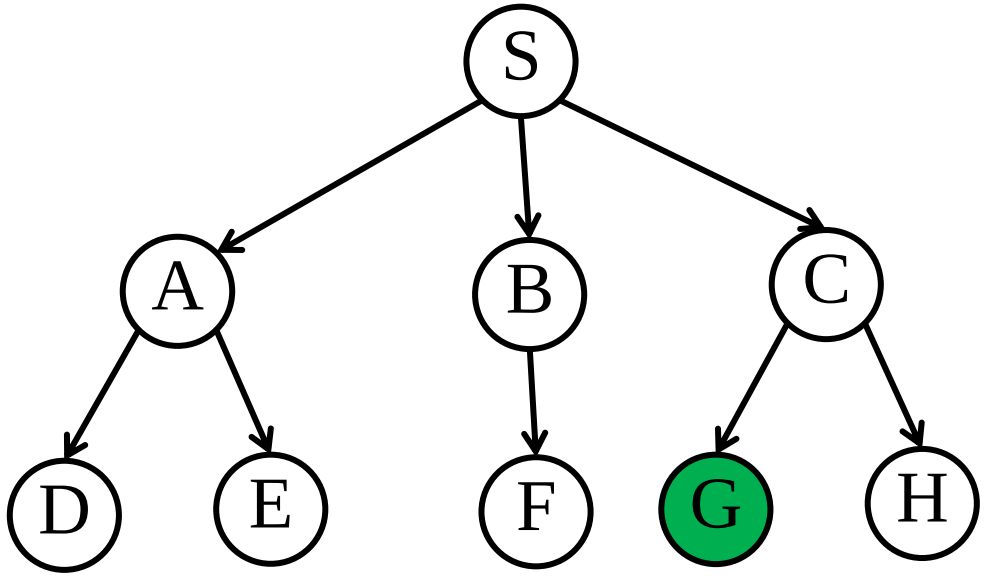
عامل جستجوی برخط

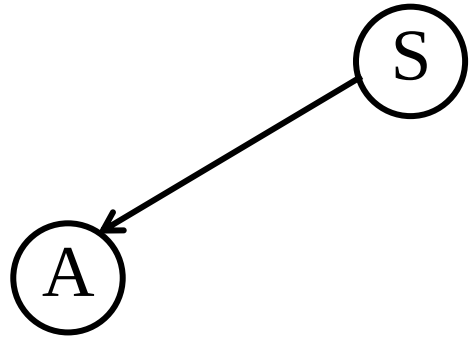
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی



عامل جستجوی برخط

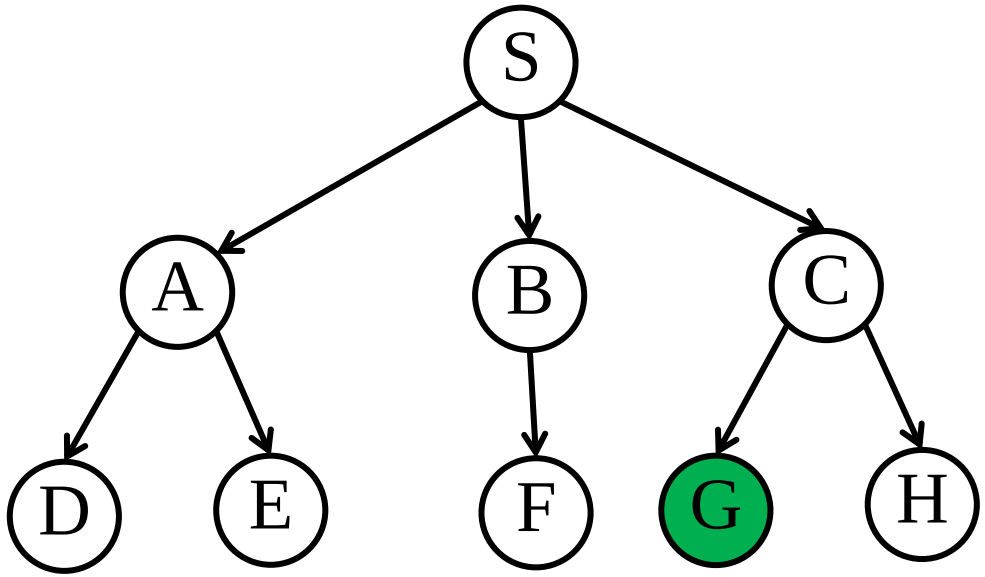
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

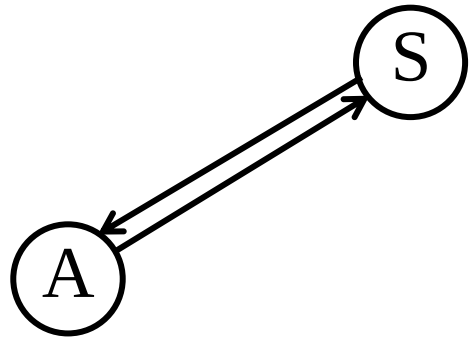




عامل جستجوی برخط

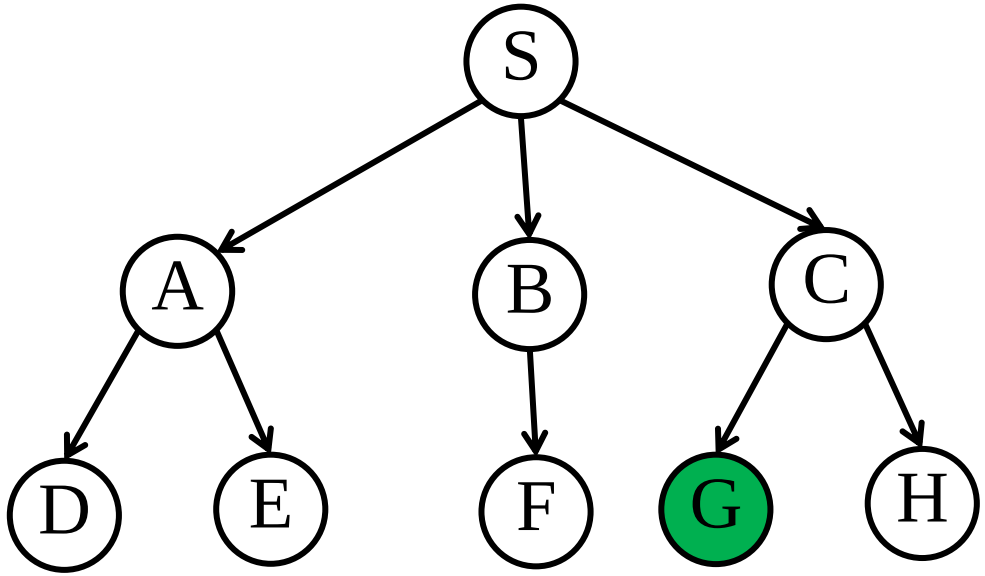
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

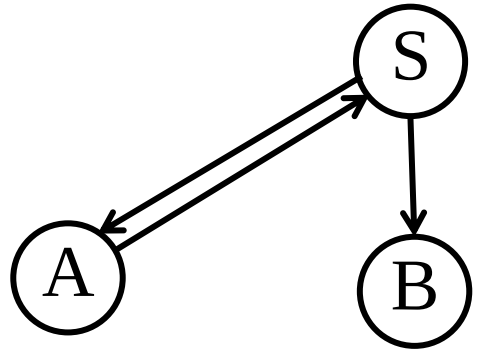




عامل جستجوی برخط

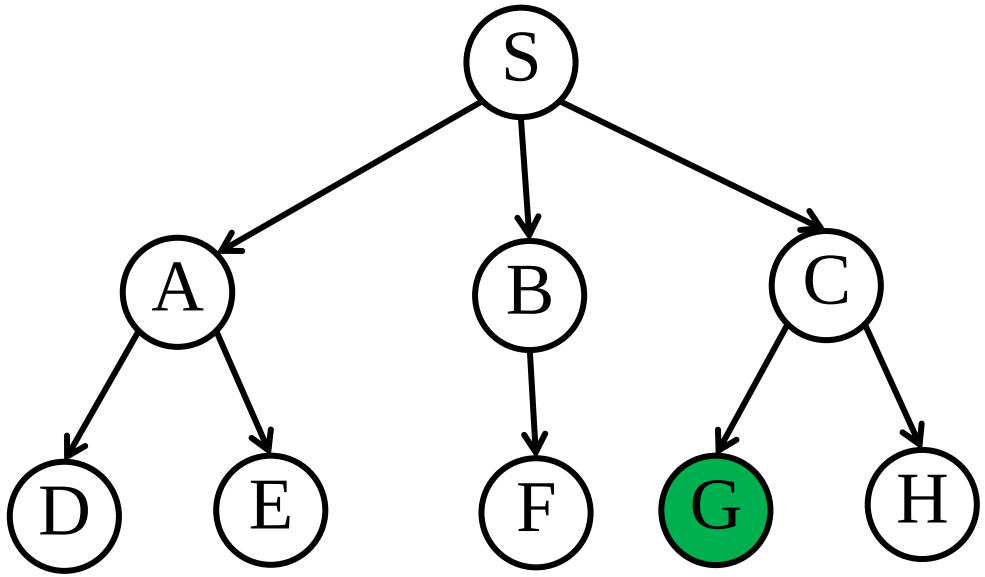
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

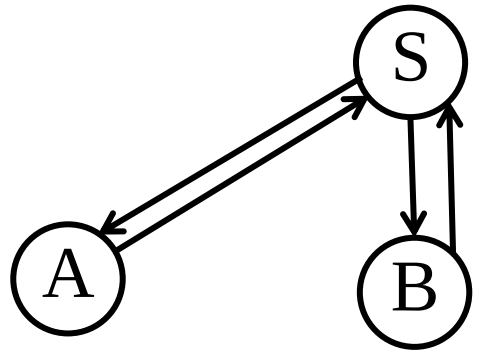




عامل جستجوی برخط

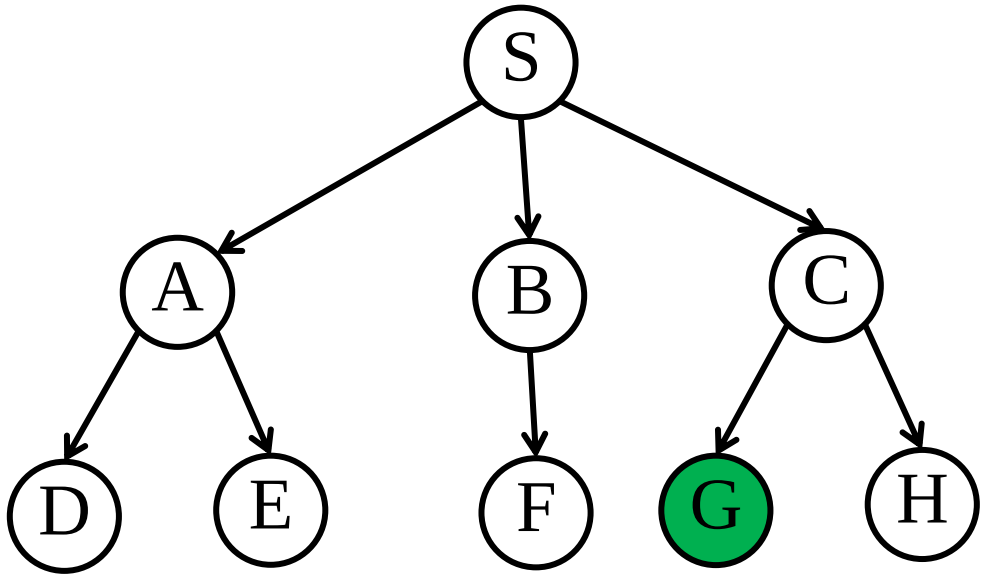
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

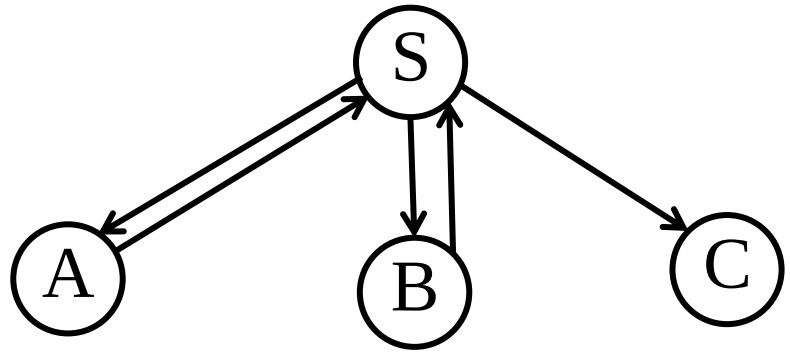




عامل جستجوی برخط

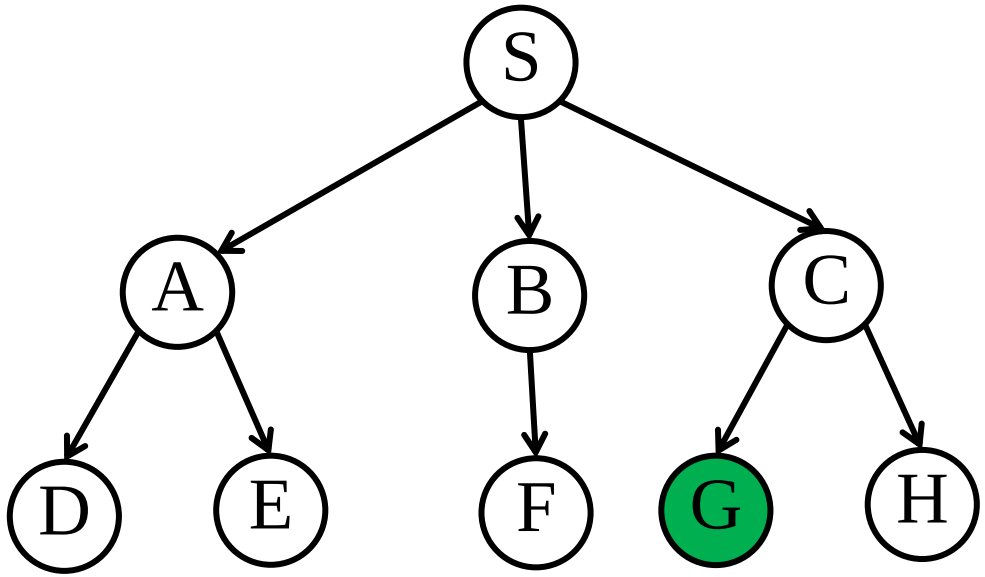
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

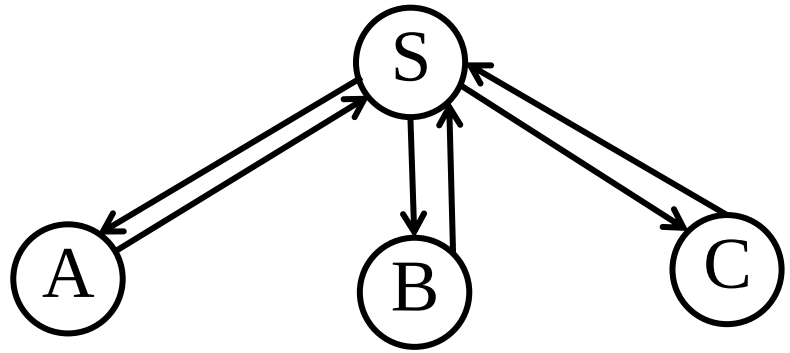




عامل جستجوی برخط

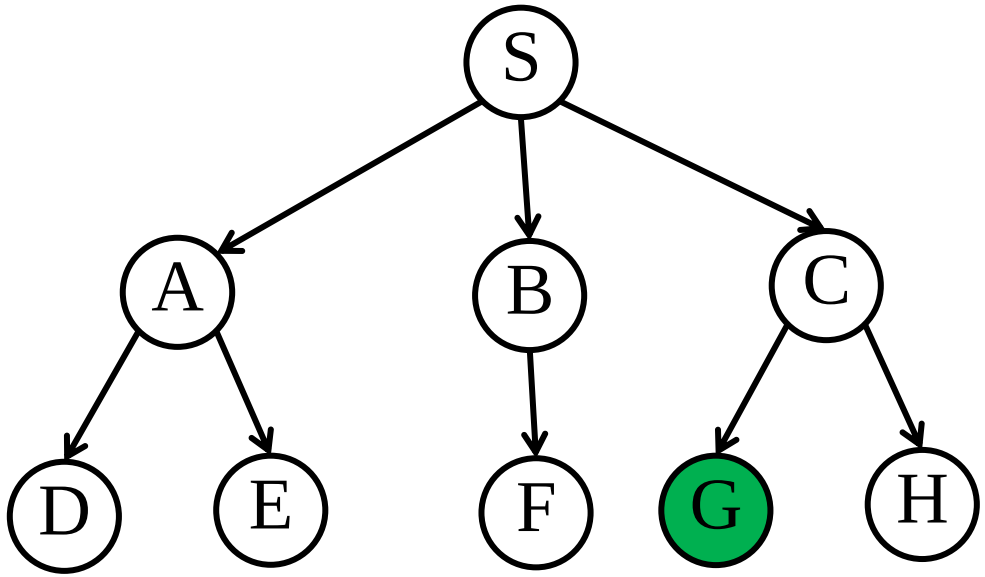
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

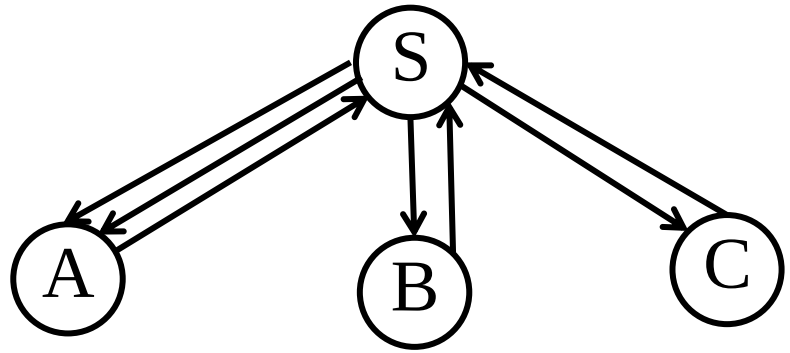




عامل جستجوی برخط

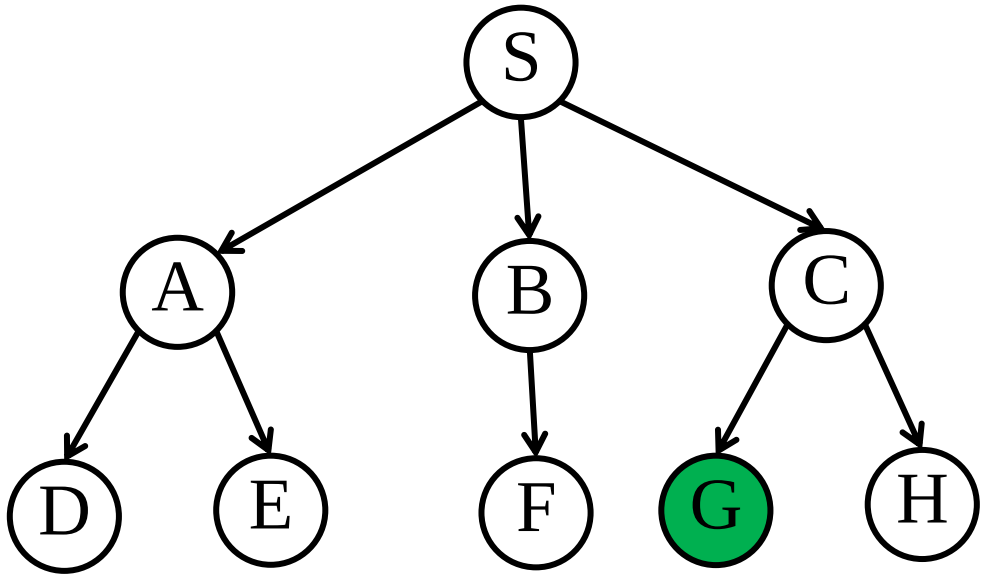
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

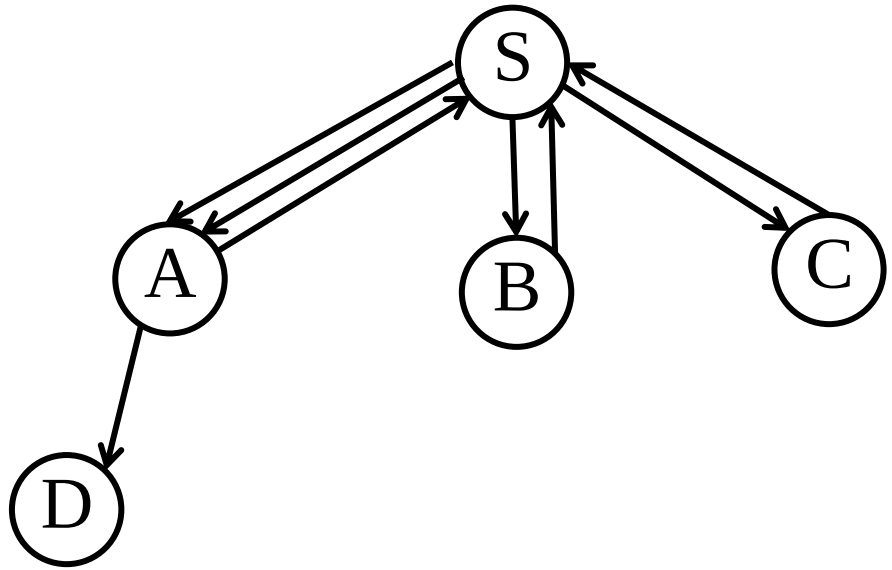




عامل جستجوی برخط

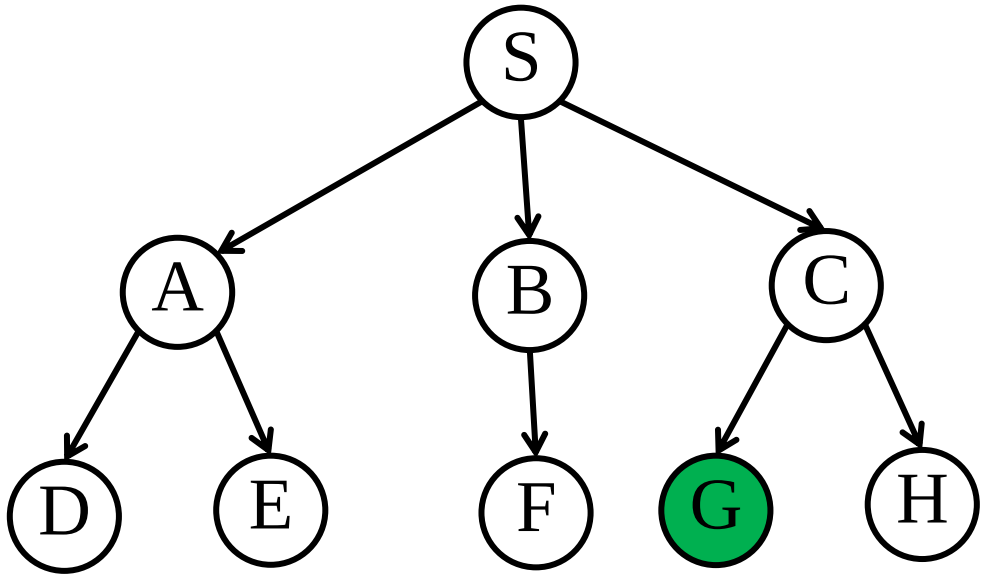
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

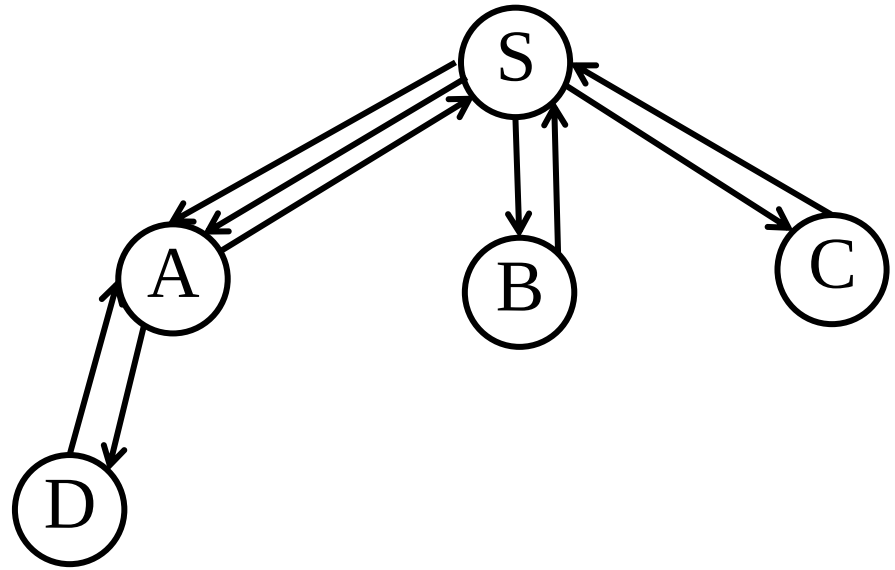




عامل جستجوی برخط

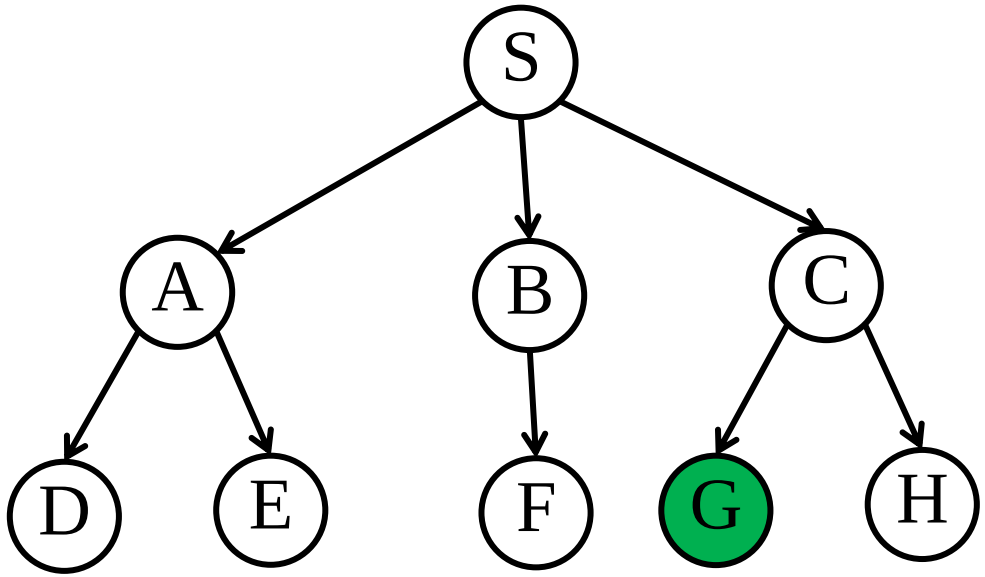
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

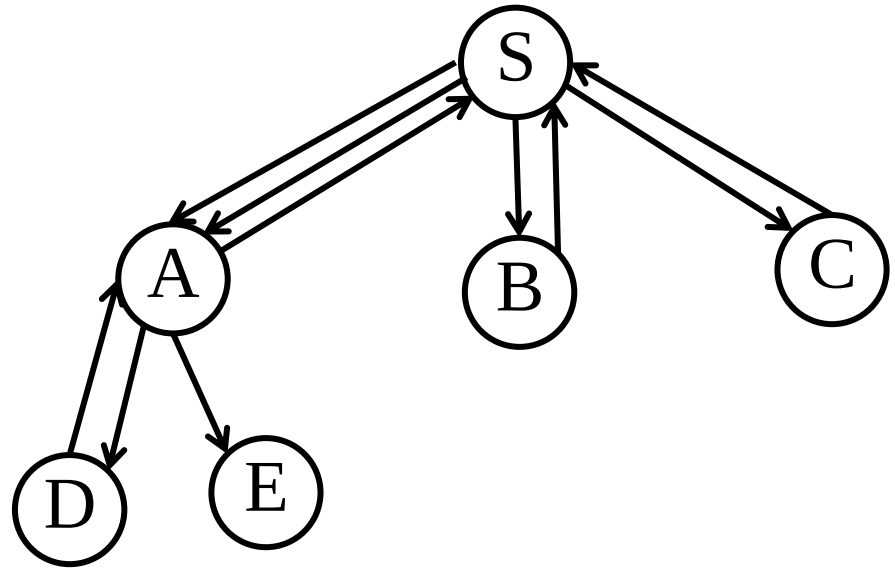




عامل جستجوی برخط

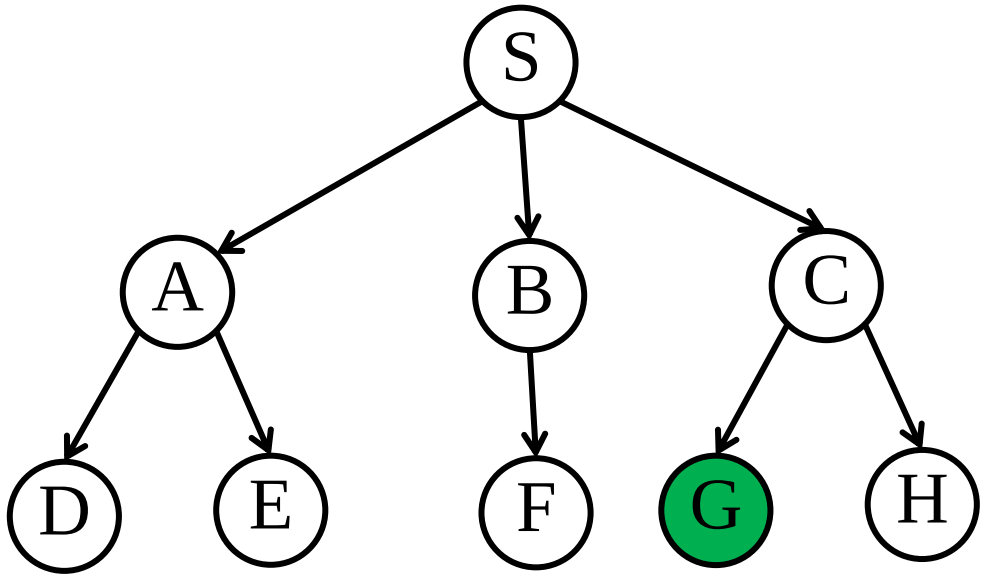
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

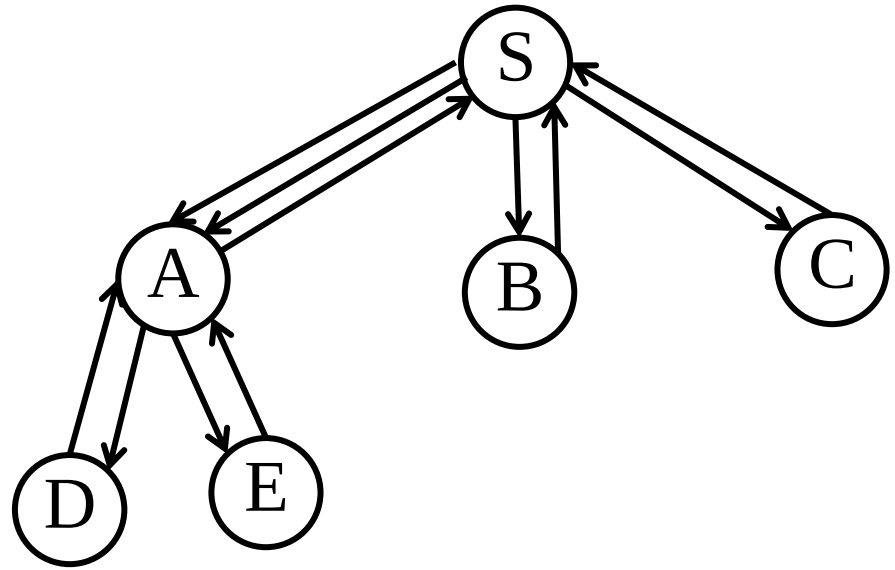




عامل جستجوی برخط

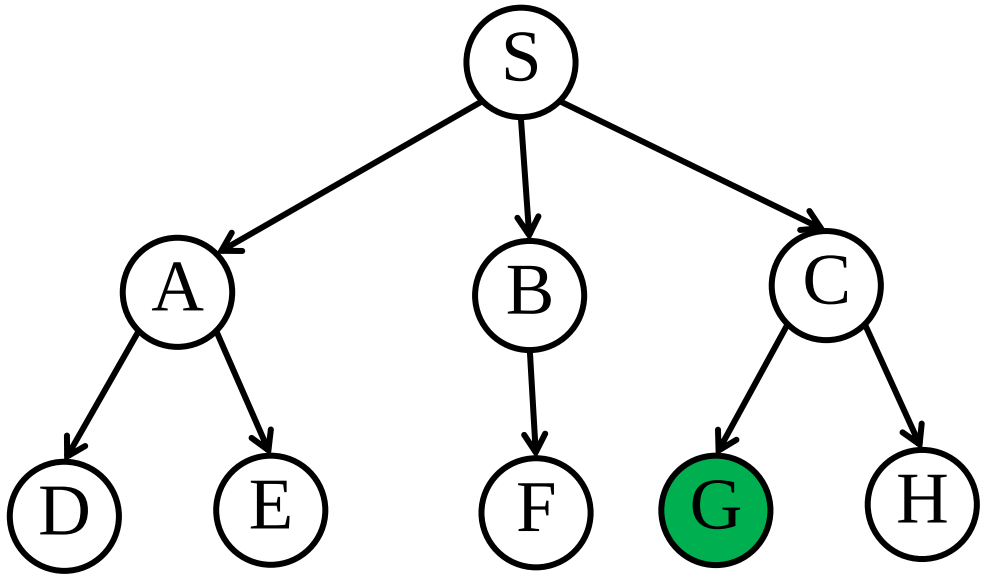
- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی

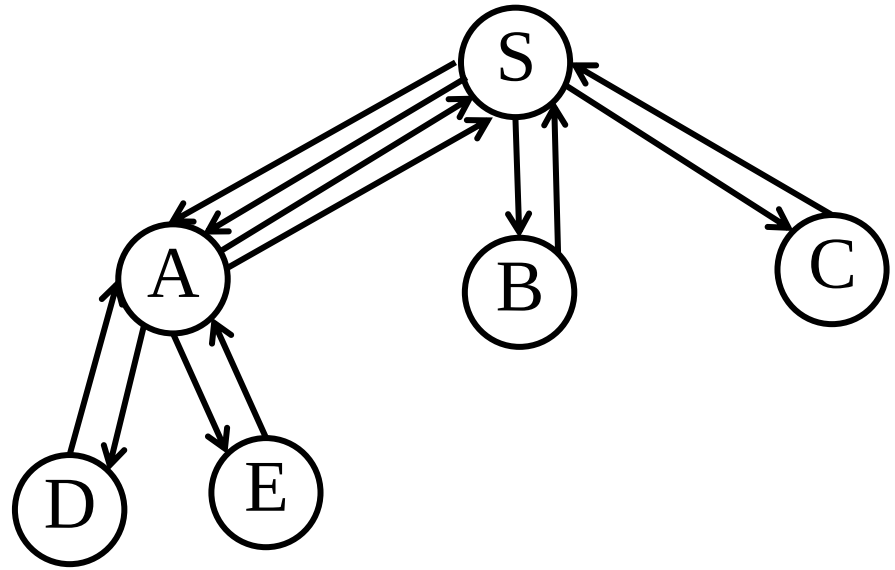




عامل جستجوی برخط

- عامل برخط به‌صورت یکی در میان جستجو و اجرا را انجام می‌دهد که باعث تفاوت الگوریتم‌های آن با الگوریتم‌های جستجوی برون‌خط می‌شود.
  - نگه‌داری یک نقشه از محیط و به‌روز کردن آن براساس ادراک دریافتی و انتخاب عمل بعدی





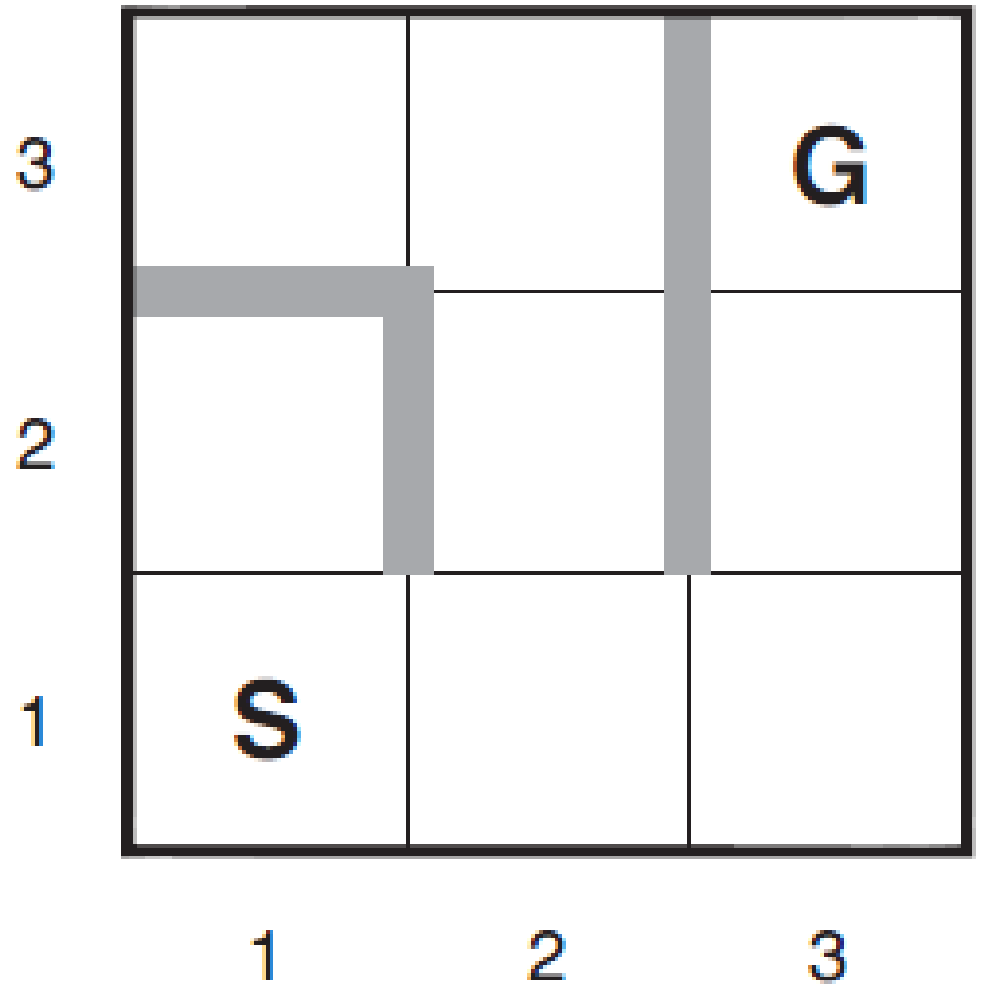
عامل جستجوی برخط

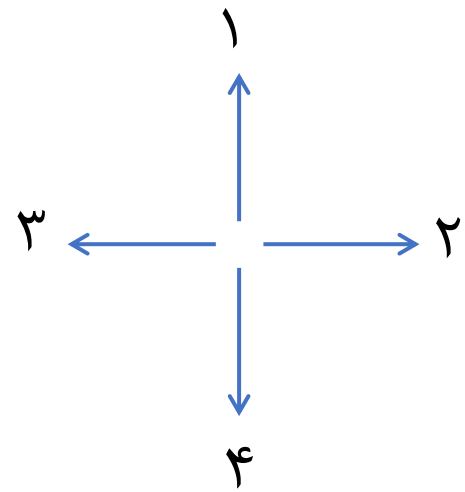
- الگوریتم جستجوی برون‌خط
  - گسترش گره شامل اعمال شبیه‌سازی‌شده است نه اعمال واقعی
    - می‌توانند یک گره را در یک بخش از فضا گسترش دهند و سپس بلافاصله گره‌ای را در بخش دیگری از فضا گسترش دهند.
- الگوریتم جستجوی برخط
  - تنها می‌تواند گره‌ای را گسترش دهد که به طور فیزیکی در آن قرار داشته است.
  - برای اجتناب از جابه‌جایی در عرض درخت بهتر است گره‌ها را به ترتیب محلی گسترش دهیم.
    - مثلا استفاده از جستجوی عمقی و جستجوی محلی

عامل جستجوی برخط اول عمق (Online DFS)

- تا زمانی‌که به هدف نرسیدی یا هنوز وضعیت عقب‌گردی وجود دارد مراحل زیر را تکرار کن
  - اگر برای حالت فعلی، عملی وجود دارد که قبلا انتخاب نشده است آن را انجام بده
  - در غیر این صورت وضعیتی که از حالت فعلی به آن عقب‌گرد نشده پیدا کن و به آن برو
- توجه: عامل به‌صورت فیزیکی به عقب برمی‌گردد.
- تنها در فضاهای حالتی عمل می‌کند که اعمال برگشت‌پذیر باشند.
- در بدترین حالت، هر لینک موجود در فضای حالت را دقیقا دو بار پیمایش می‌کند.
  - در مورد مسائل اکتشاف بهینه است.
  - در مورد مسائل یافتن هدف، نسبت رقابتی عامل ممکن است بسیار زیاد شود.
    - اگر هدف بلافاصله بعد از حالت شروع هم باشد ممکن است ابتدا به بررسی شاخه‌های دیگر بپردازد.

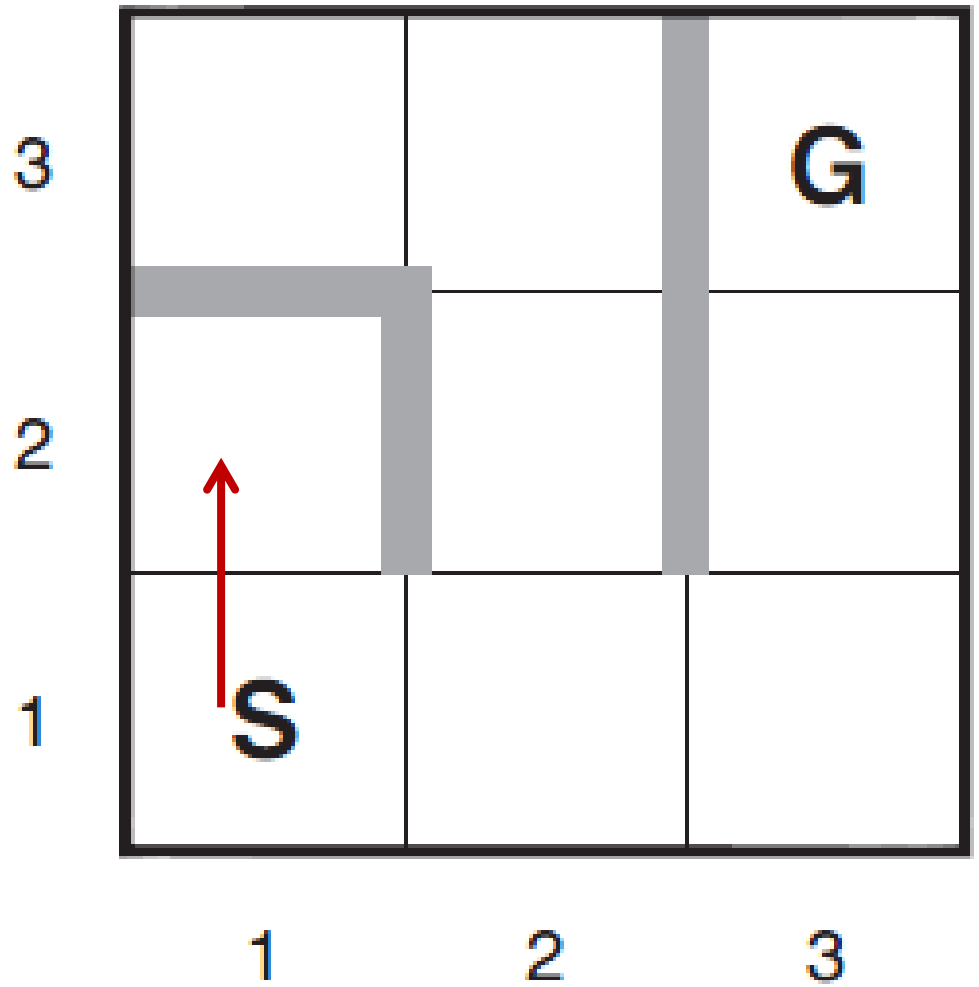
مثال ۱– Online DFS

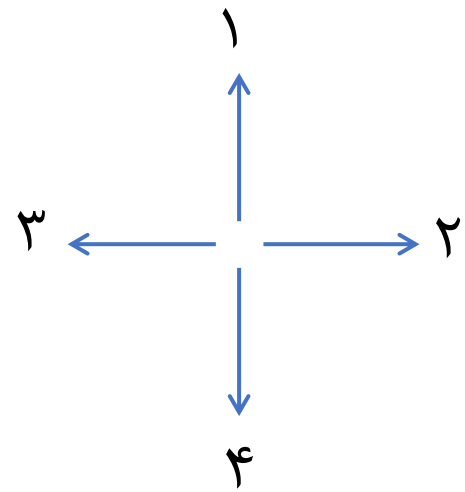




ترتیب انجام عملیات

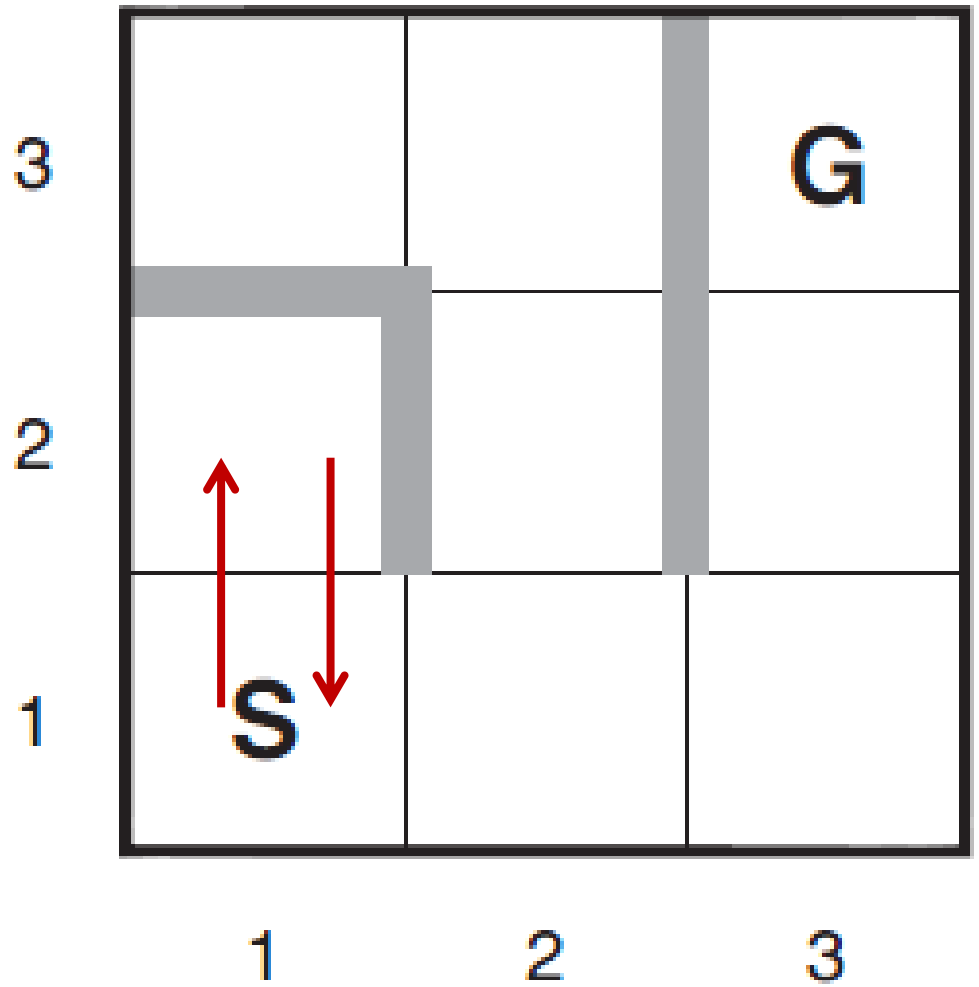
مثال ۱– Online DFS

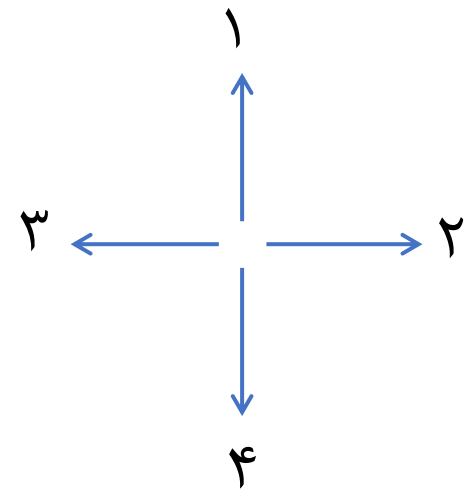




ترتیب انجام عملیات

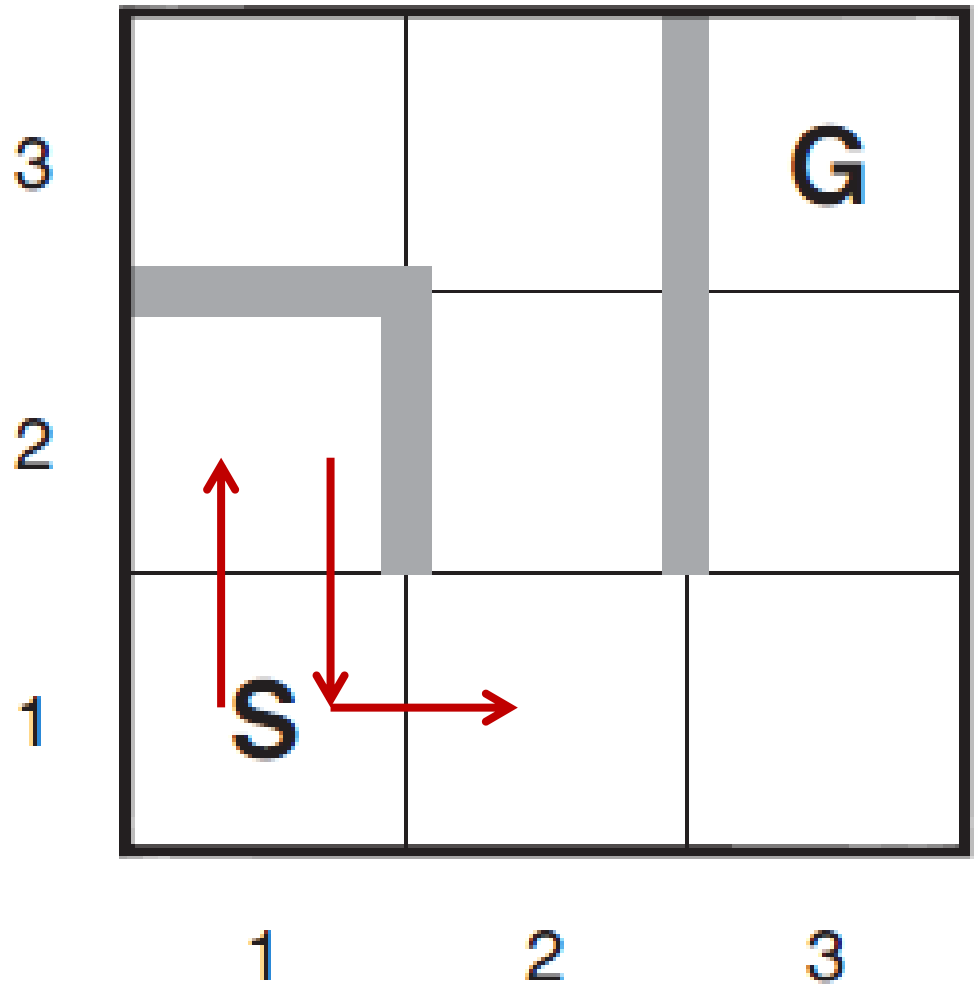
مثال ۱– Online DFS

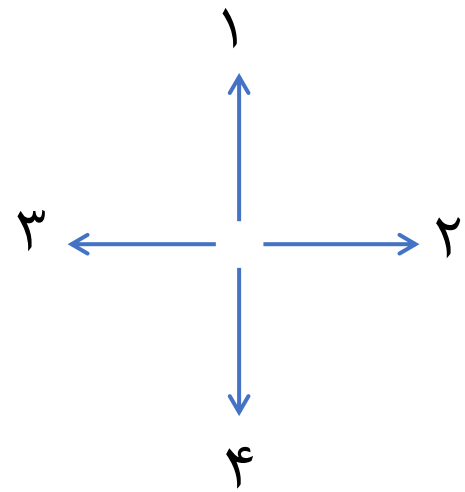




ترتیب انجام عملیات

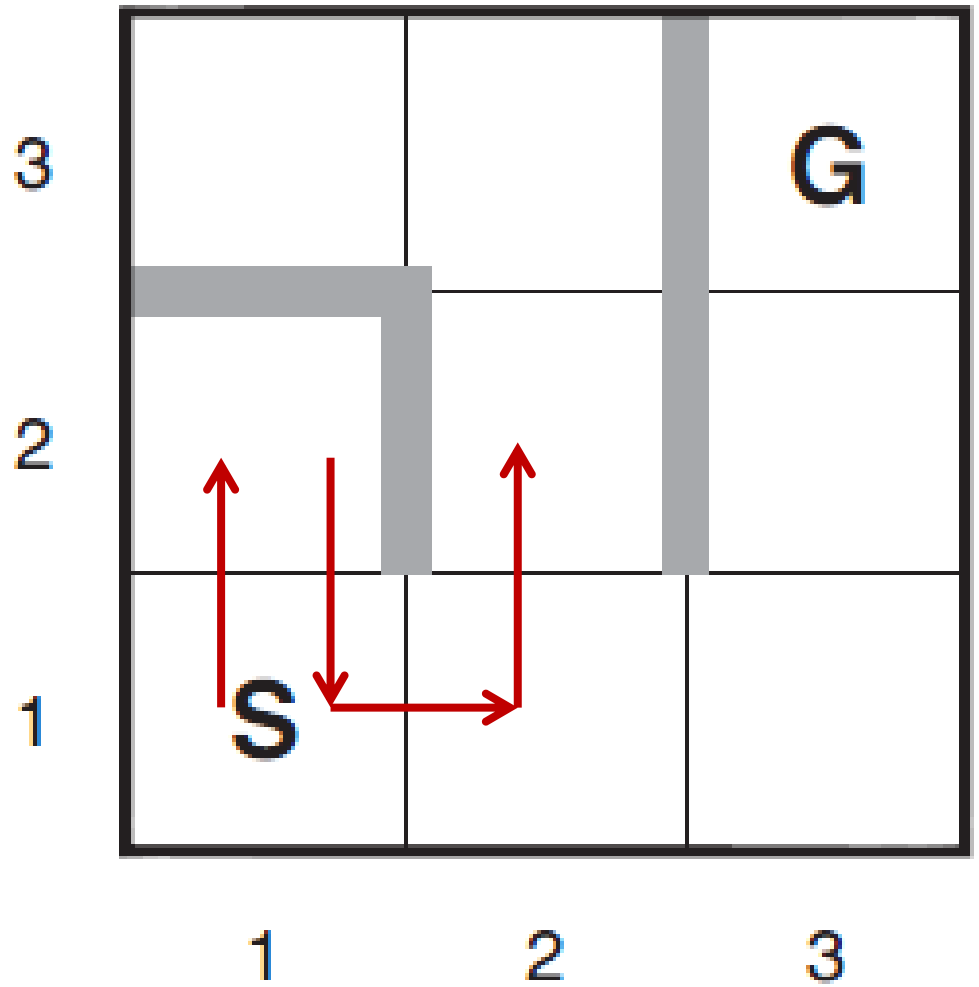
مثال ۱– Online DFS

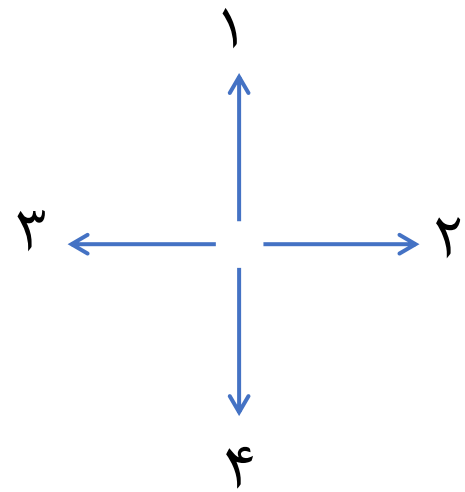




ترتیب انجام عملیات

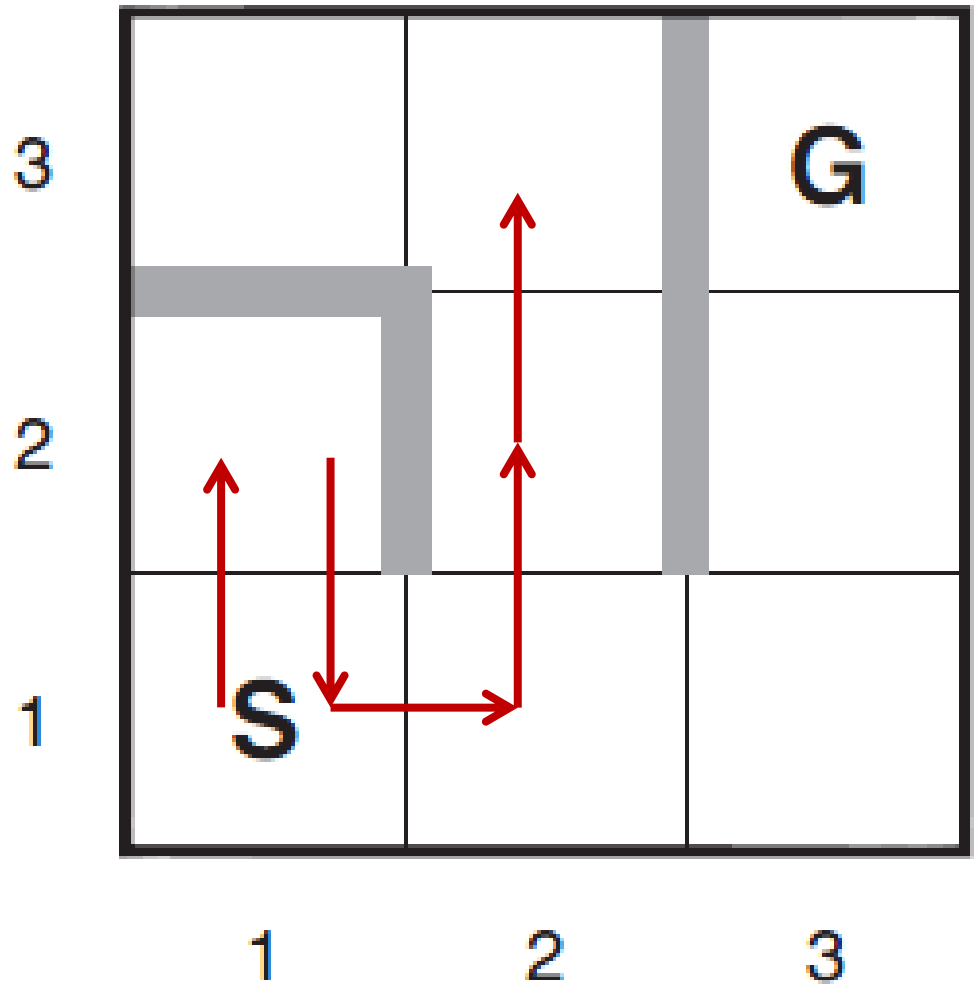
مثال ۱– Online DFS

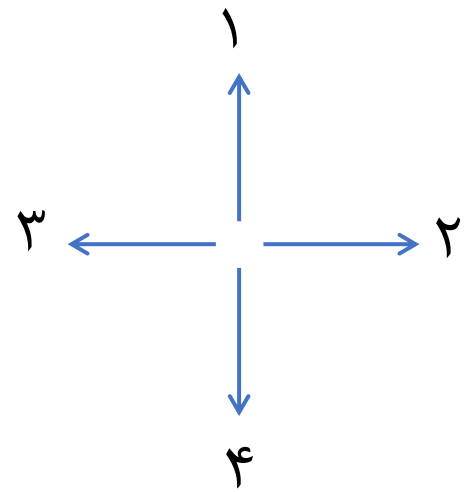




ترتیب انجام عملیات

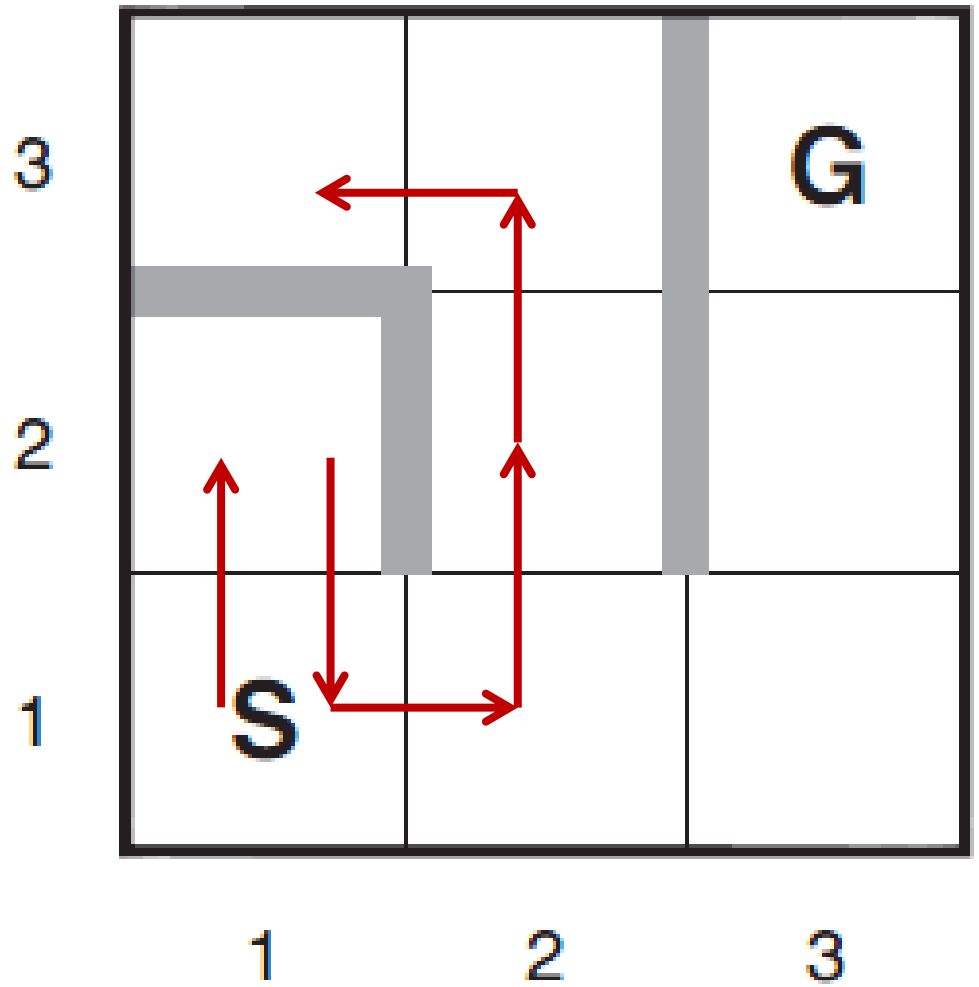
مثال ۱– Online DFS

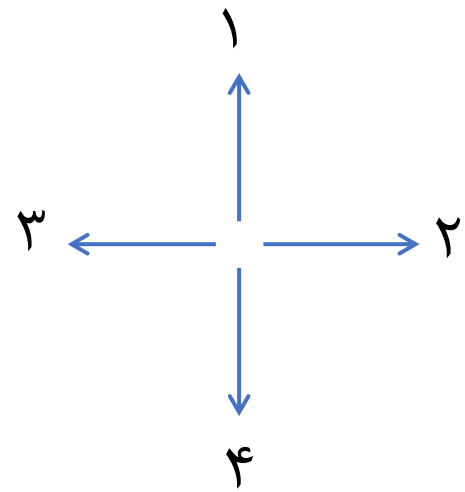




ترتیب انجام عملیات

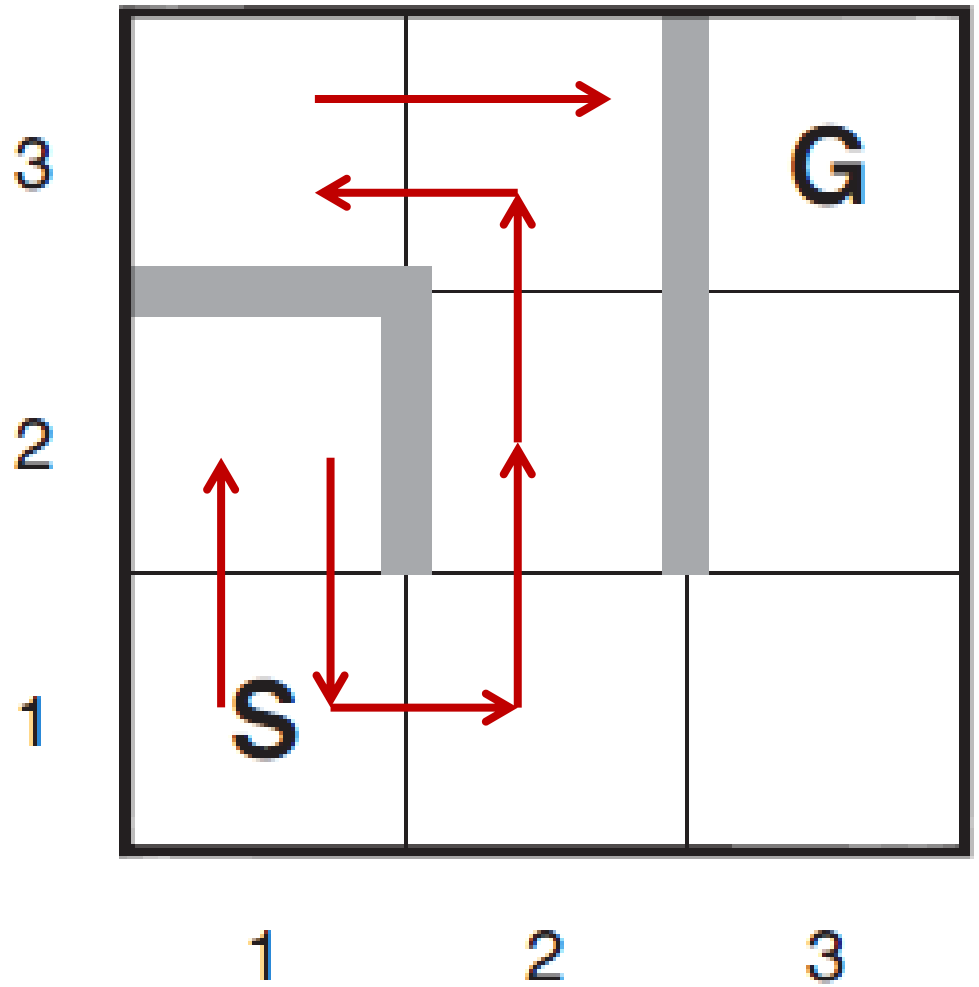
مثال ۱– Online DFS

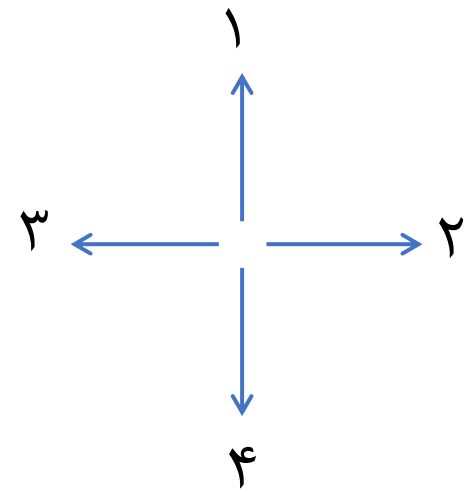




ترتیب انجام عملیات

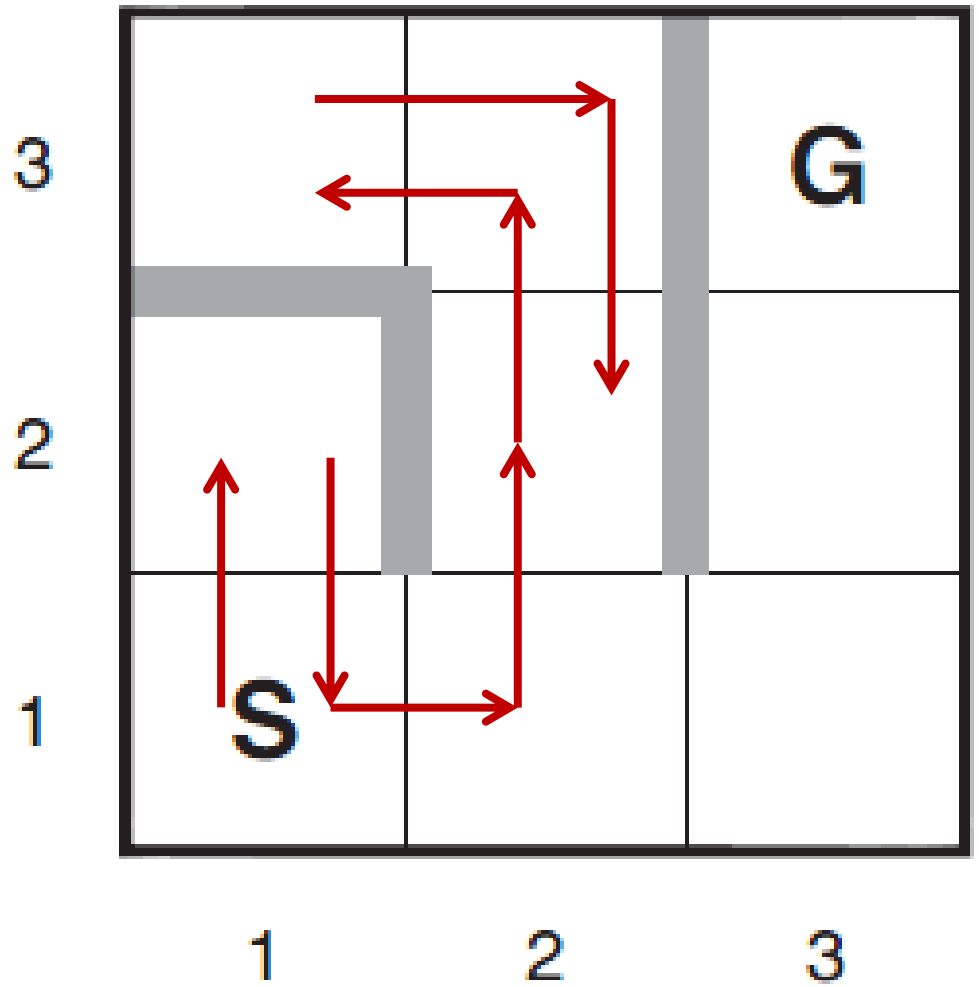
مثال ۱– Online DFS

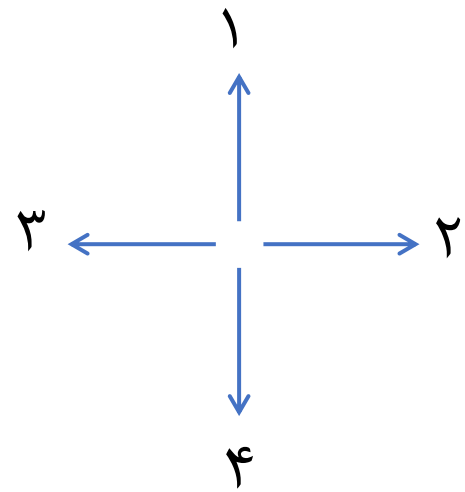




ترتیب انجام عملیات

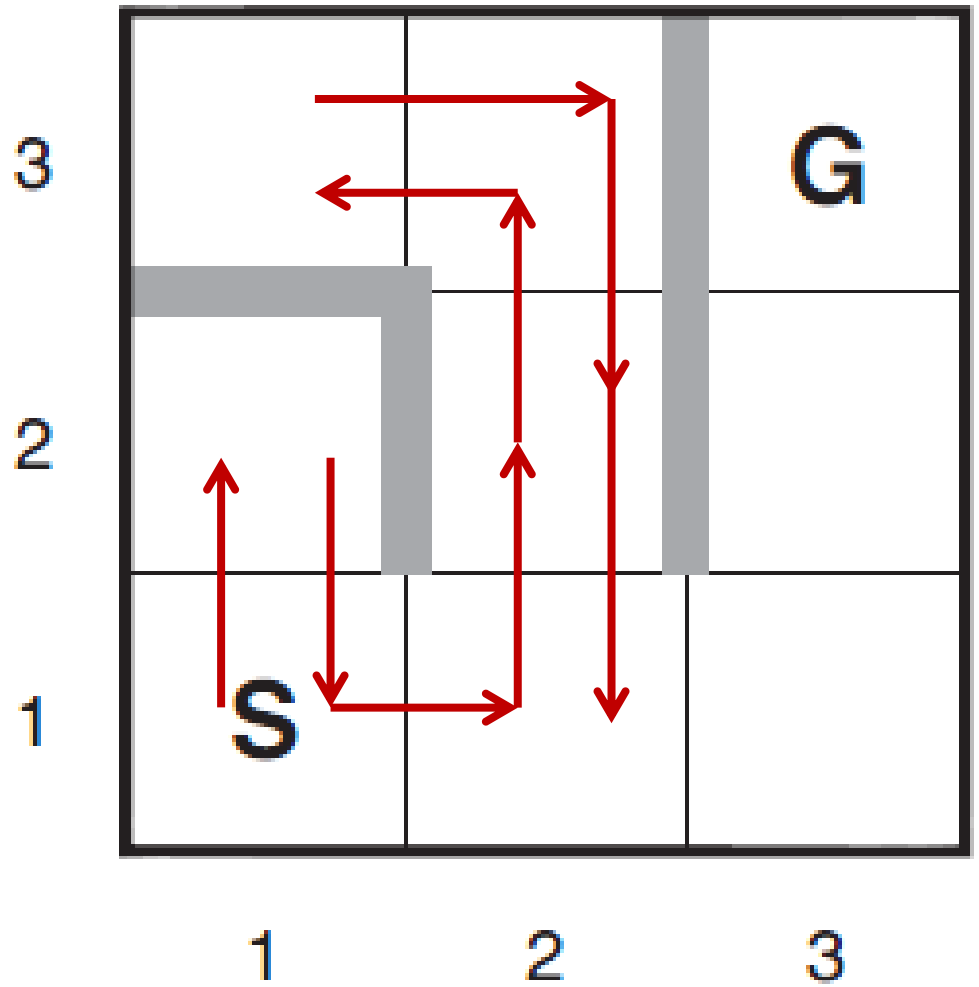
مثال ۱– Online DFS

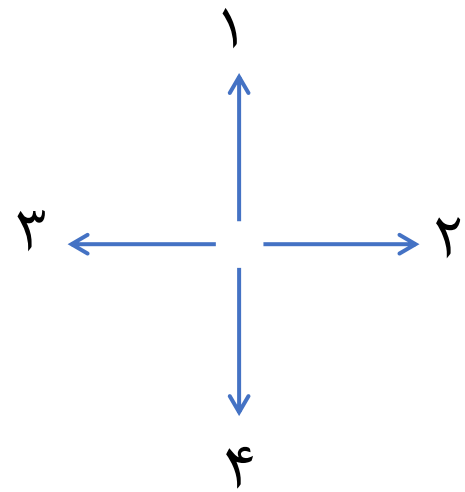




ترتیب انجام عملیات

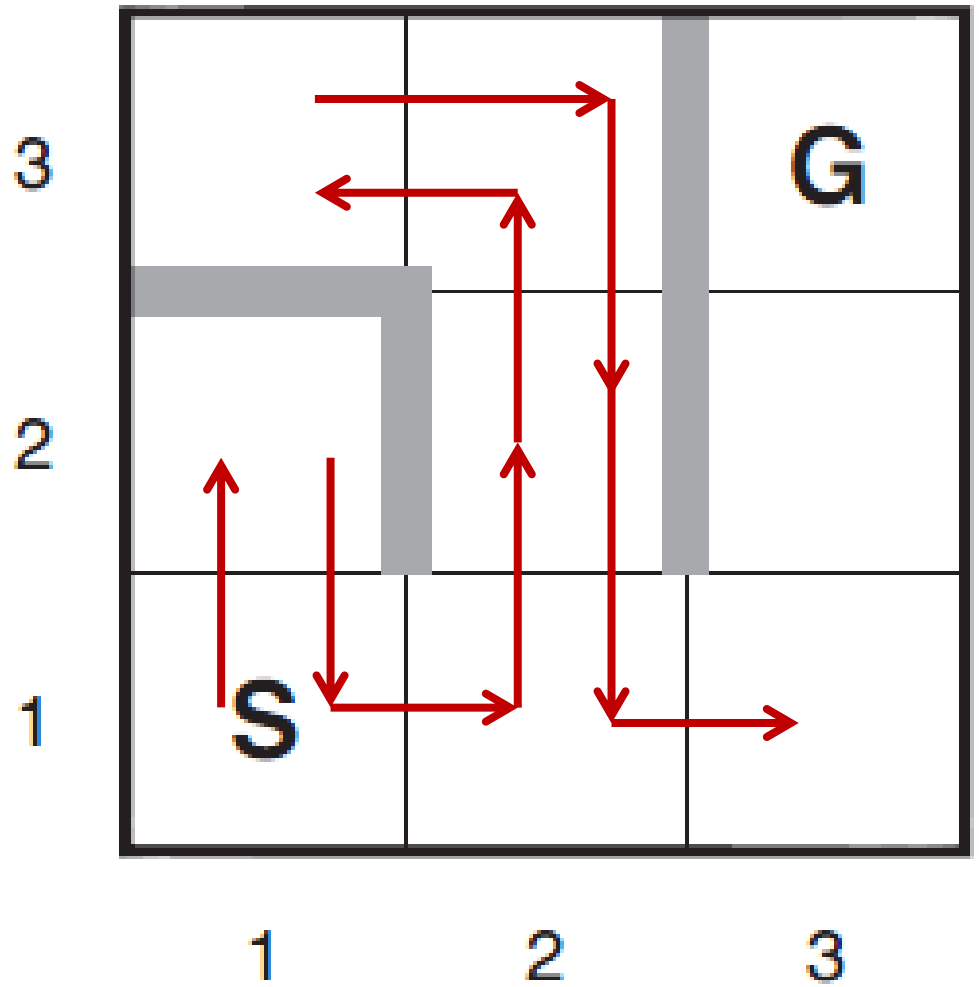
مثال ۱– Online DFS

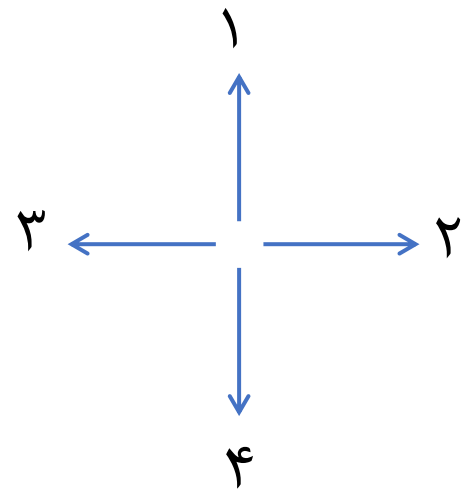




ترتیب انجام عملیات

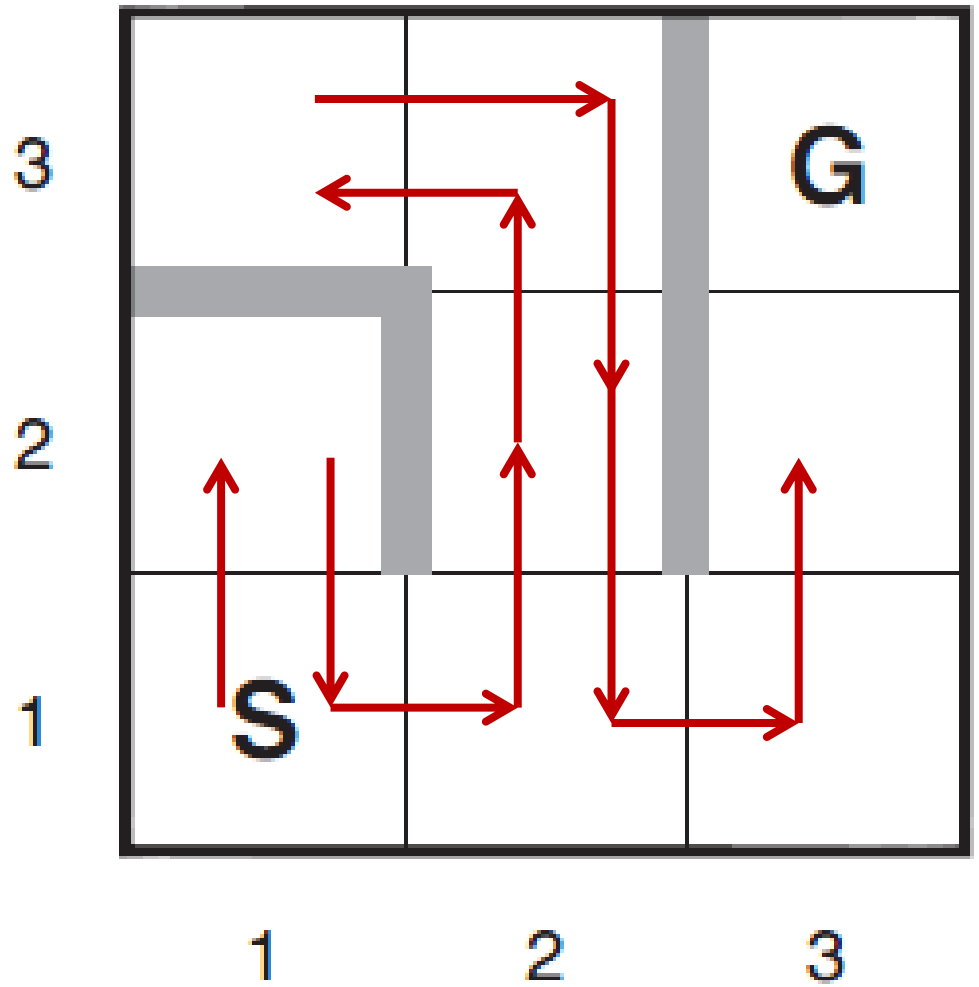
مثال ۱– Online DFS

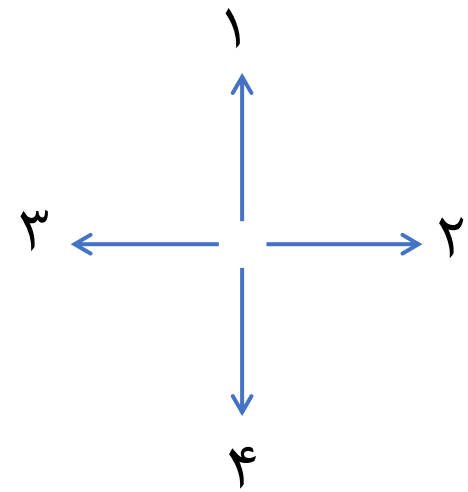




ترتیب انجام عملیات

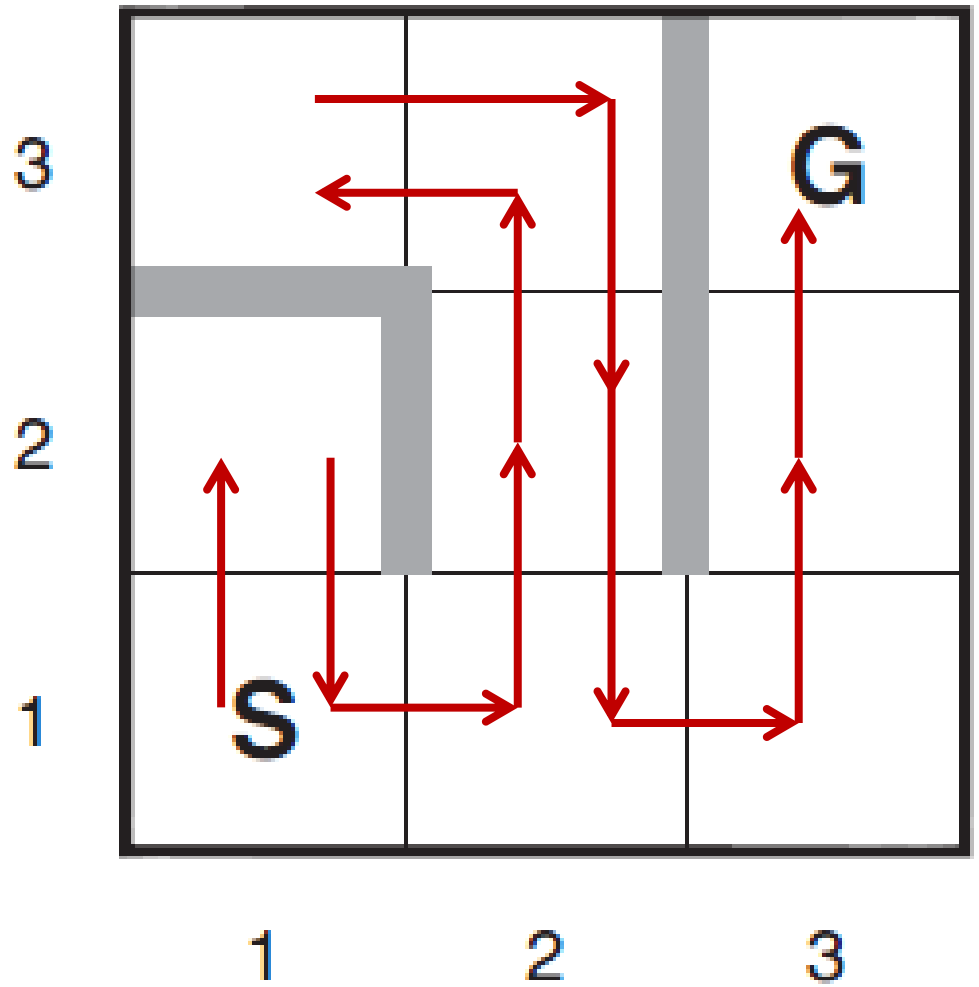
مثال ۱– Online DFS

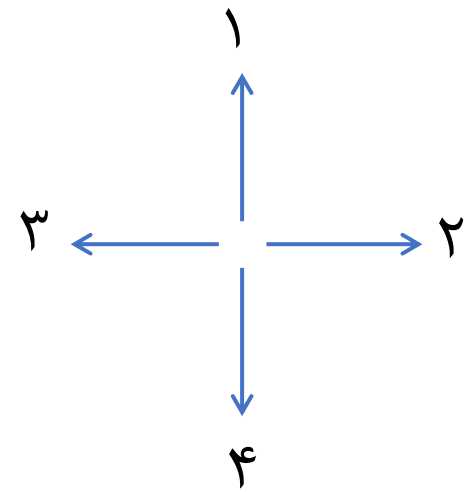




ترتیب انجام عملیات

مثال ۱– Online DFS





ترتیب انجام عملیات

عامل جستجوی برخط با کاوش اول عمق ...

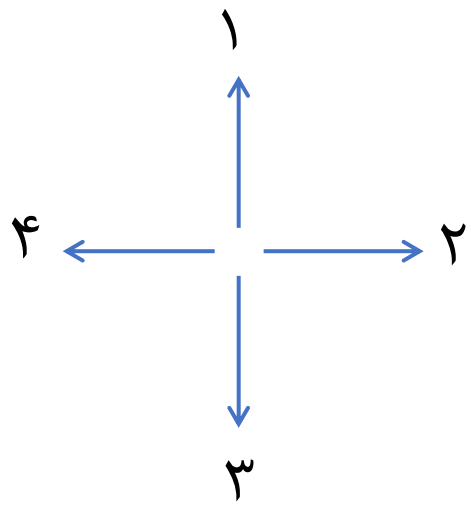
```
function ONLINE-DFS-AGENT(s′) returns an action
  inputs: s′, a percept that identifies the current state
  persistent: result, a table indexed by state and action, initially empty
              untried, a table that lists, for each state, the actions not yet tried
              unbacktracked, a table that lists, for each state, the backtracks not yet tried
              s, a, the previous state and action, initially null

  if GOAL-TEST(s′) then return stop
  if s′ is a new state (not in untried) then untried[s′] ← ACTIONS(s′)
  if s is not null then
      result[s, a] ← s′
      add s to the front of unbacktracked[s′]
  if untried[s′] is empty then
      if unbacktracked[s′] is empty then return stop
      else a ← an action b such that result[s′, b] = POP(unbacktracked[s′])
  else a ← POP(untried[s′])
  s ← s′
  return a
```

مثال ۲– Online DFS

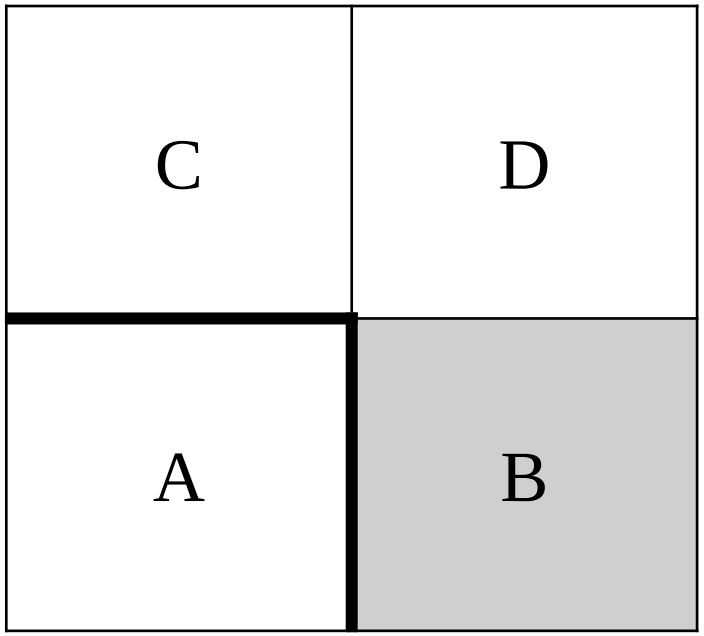
C	D
A	B

محل شروع: B
محل هدف: C



ترتیب انجام عملیات

مثال ۲– Online DFS ...



اجرای اول:

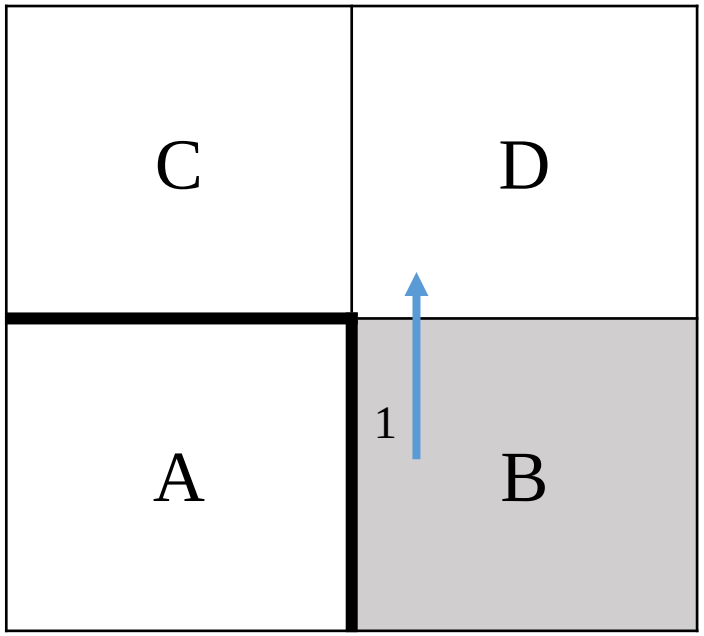
$s' = B \qquad s = null$

$untried[B] = \{u, r, d, l\}$

$a = u \rightarrow untried[B] = \{r, d, l\}$

$s = B$

مثال ۲– Online DFS ...



اجرای اول:

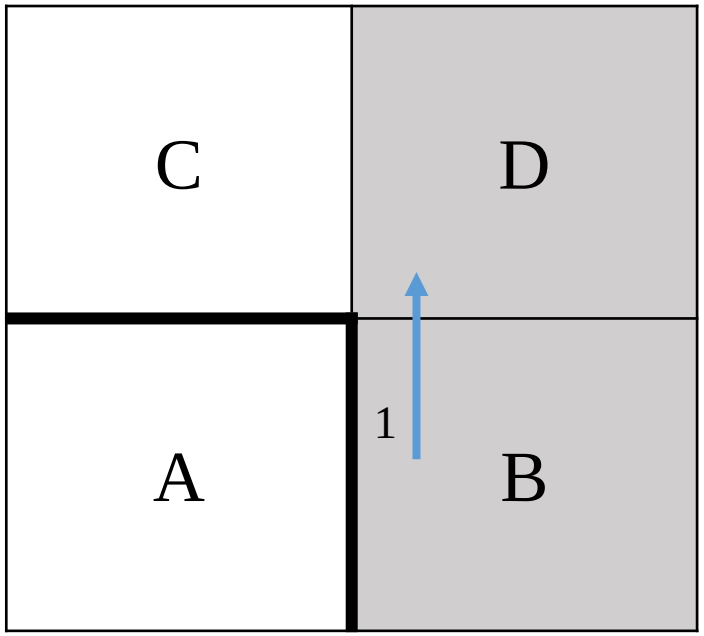
$s' = B \qquad s = null$

$untried[B] = \{u, r, d, l\}$

$a = u \rightarrow untried[B] = \{r, d, l\}$

$s = B$

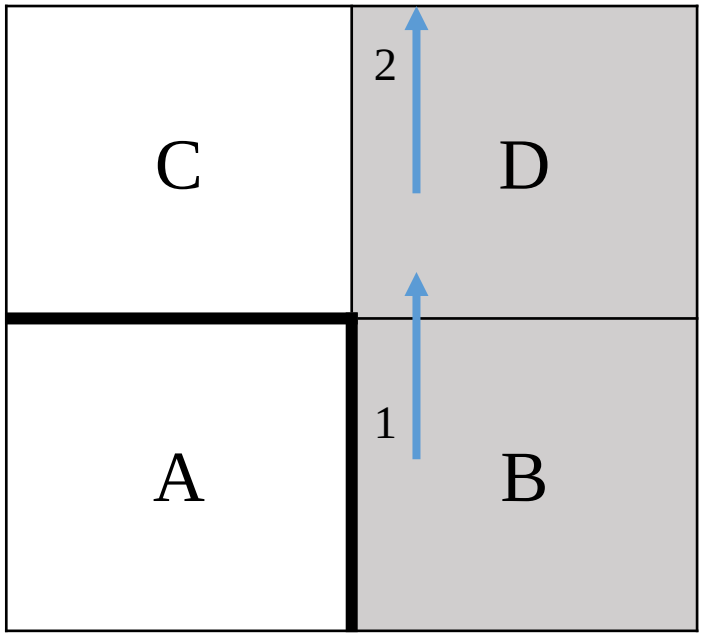
مثال ۲– Online DFS ...



اجرای دوم:

s‘ = D s = B
untried[D] = {u, r, d, l}
result[B,u] = D
unbacktracked[D] = {B}
a = u → untried[D] = {r, d, l}
s = D

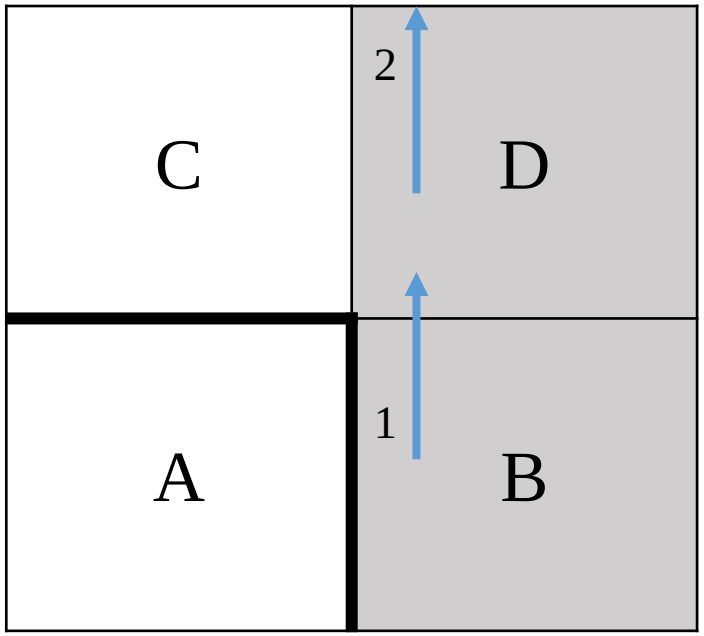
مثال ۲– Online DFS ...



اجرای دوم:

s‘ = D s = B
untried[D] = {u, r, d, l}
result[B,u] = D
unbacktracked[D] = {B}
a = u → untried[D] = {r, d, l}
s = D

مثال ۲– Online DFS ...



اجرای سوم:

$s' = D \qquad s = D$

$untried[D] = \{r, d, l\}$

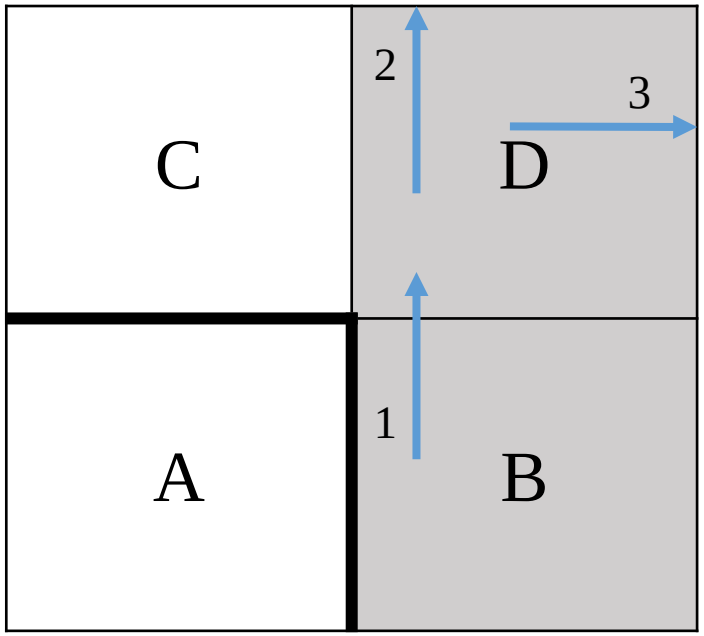
$result[D,u] = D$

$unbacktracked[D] = \{B\}$

$a = r \rightarrow untried[D] = \{d, l\}$

$s = D$

مثال ۲– Online DFS ...



اجرای سوم:

$s' = D \qquad s = D$

untried[D] = {r, d, l}

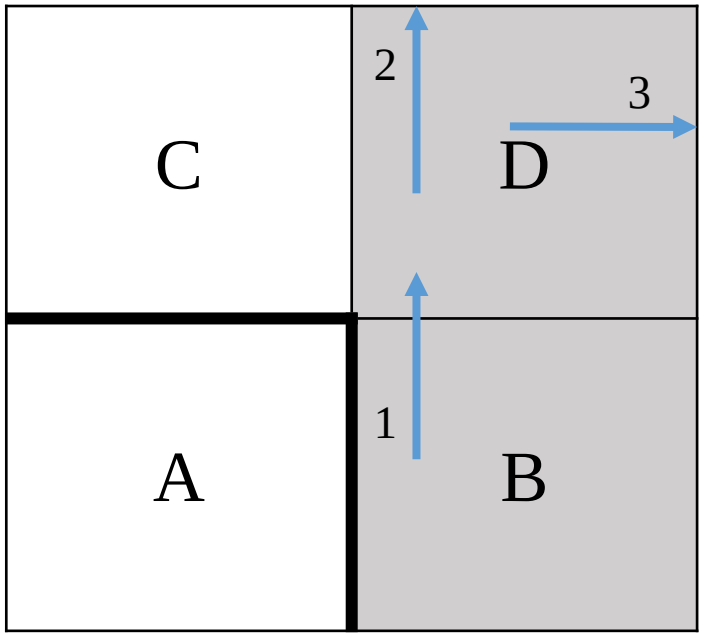
result[D,u] = D

unbacktracked[D] = {B}

$a = r \rightarrow$ untried[D] = {d, l}

s = D

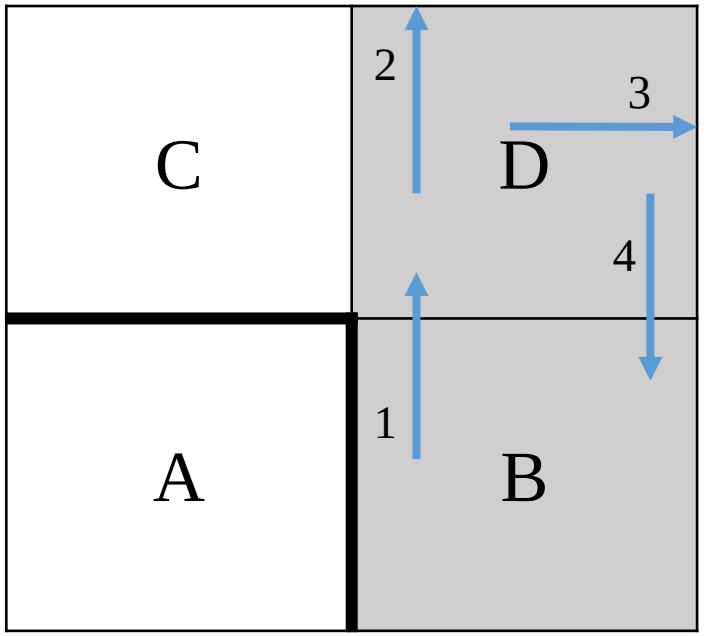
مثال ۲– Online DFS ...



اجرای چهارم:

s‘ = D s = D
untried[D] = {d, l}
result[D,r] = D
unbacktracked[D] = {B}
a = d → untried[D] = {l}
s = D

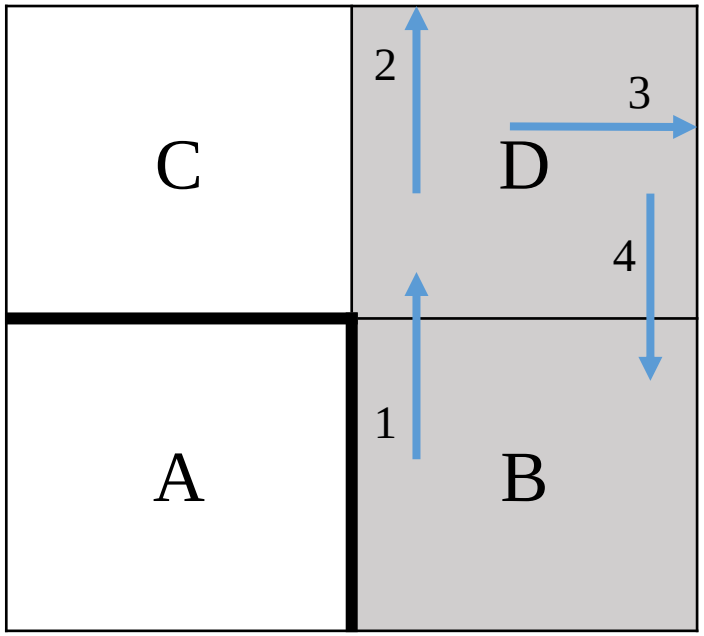
مثال ۲– Online DFS ...



اجرای چهارم:

s‘ = D s = D
untried[D] = {d, l}
result[D,r] = D
unbacktracked[D] = {B}
a = d → untried[D] = {l}
s = D

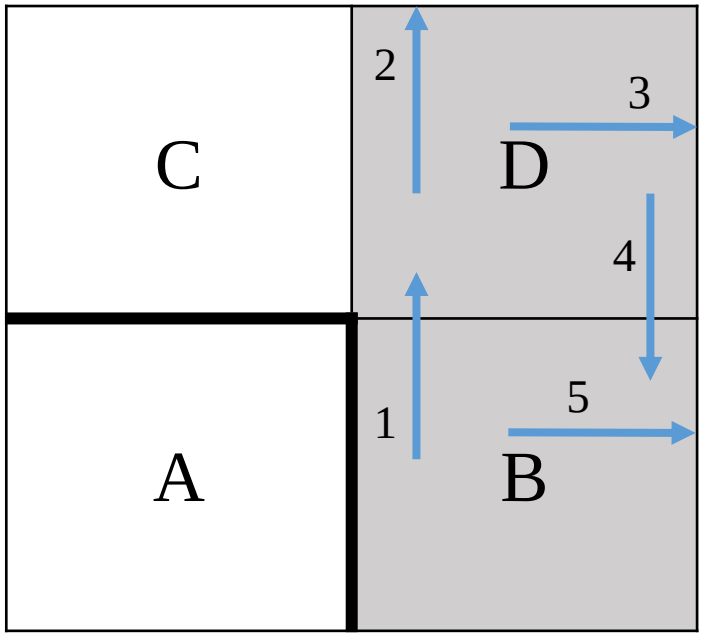
مثال ۲– Online DFS ...



اجرای پنجم:

s‘ = B s = D
untried[B] = {r, d, l}
result[D,d] = B
unbacktracked[B] = {D}
a = r → untried[B] = {d, l}
s = B

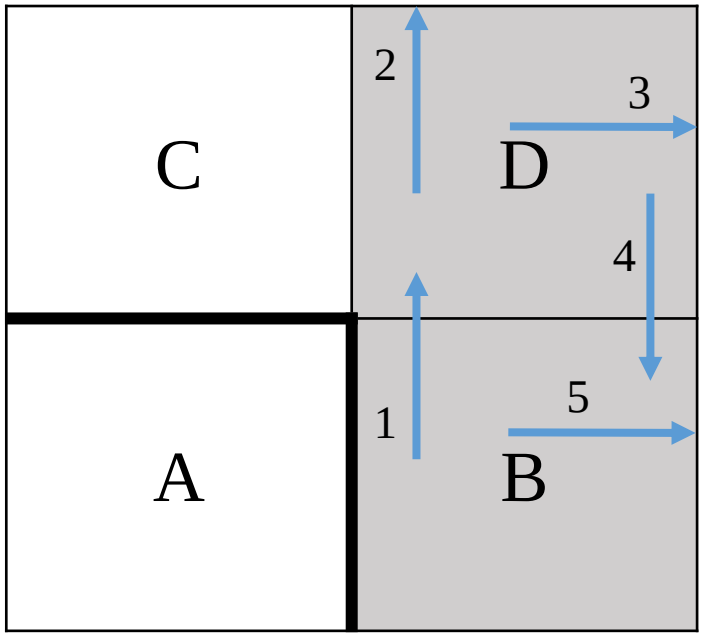
مثال ۲– Online DFS ...



اجرای پنجم:

s‘ = B s = D
untried[B] = {r, d, l}
result[D,d] = B
unbacktracked[B] = {D}
a = r → untried[B] = {d, l}
s = B

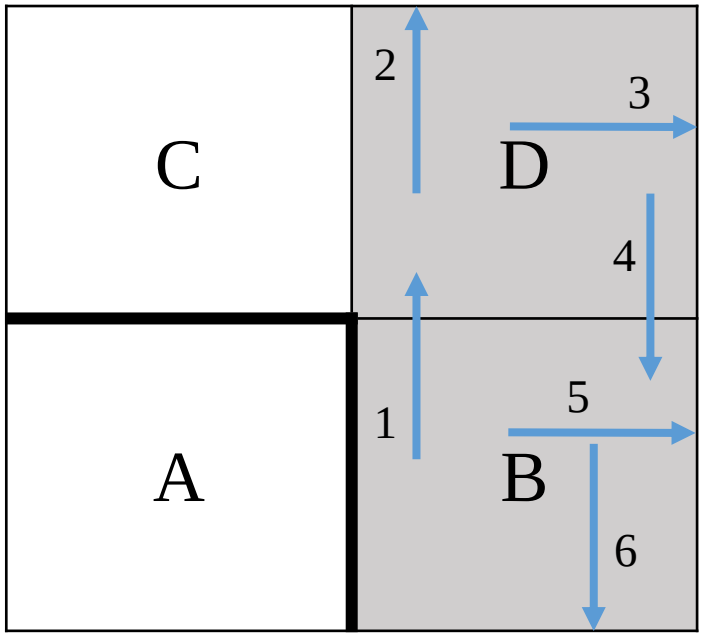
مثال ۲– Online DFS ...



اجرای ششم:

s‘ = B s = B
untried[B] = {d, l}
result[B, r] = B
unbacktracked[B] = {D}
a = d → untried[B] = {l}
s = B

مثال ۲– Online DFS ...



اجرای ششم:

$s' = B \qquad s = B$

$untried[B] = \{d, l\}$

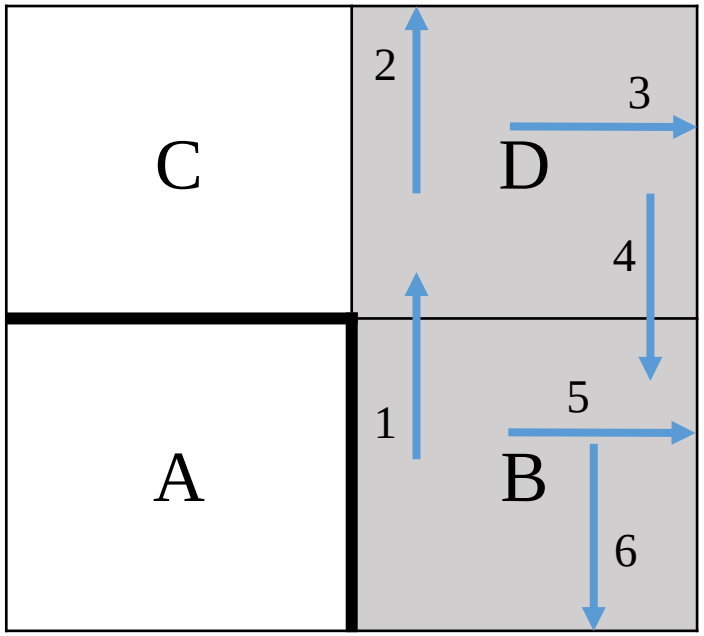
$result[B, r] = B$

$unbacktracked[B] = \{D\}$

$a = d \rightarrow untried[B] = \{l\}$

$s = B$

مثال ۲– Online DFS ...



اجرای هفتم:

$s' = B \qquad s = B$

$untried[B] = \{l\}$

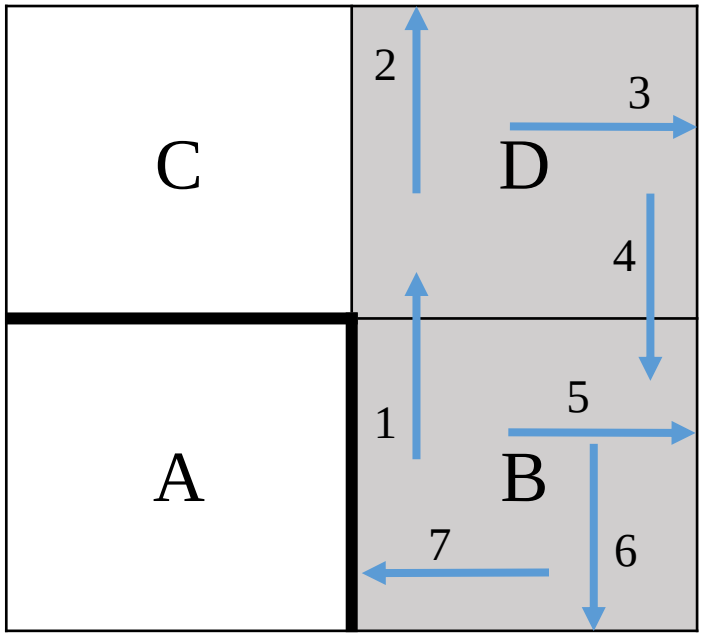
$result[B, d] = B$

$unbacktracked[B] = \{D\}$

$a = l \rightarrow untried[B] = \{\}$

$s = B$

مثال ۲– Online DFS ...



اجرای هفتم:

$s' = B \qquad s = B$

untried[B] = {l}

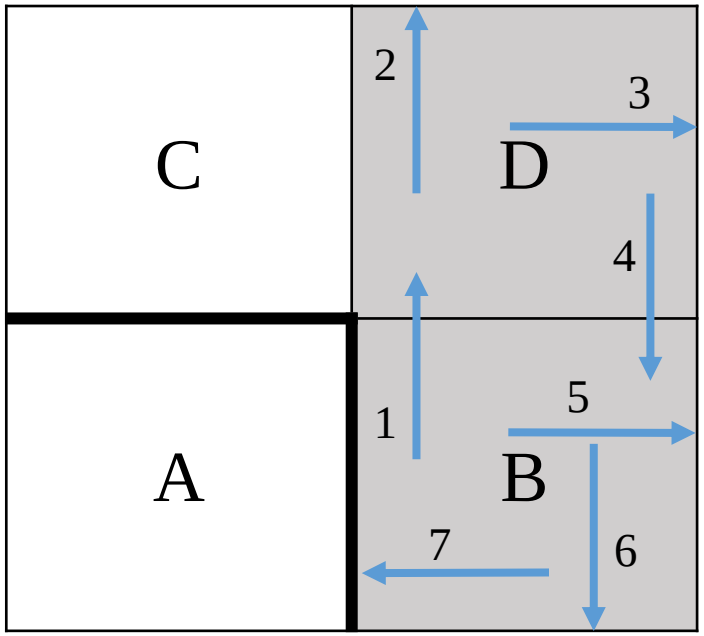
result[B, d] = B

unbacktracked[B] = {D}

$a = l \rightarrow$ untried[B] = {}

$s = B$

مثال ۲– Online DFS ...



اجرای هشتم:

$s' = B \qquad s = B$

untried[B] = {}

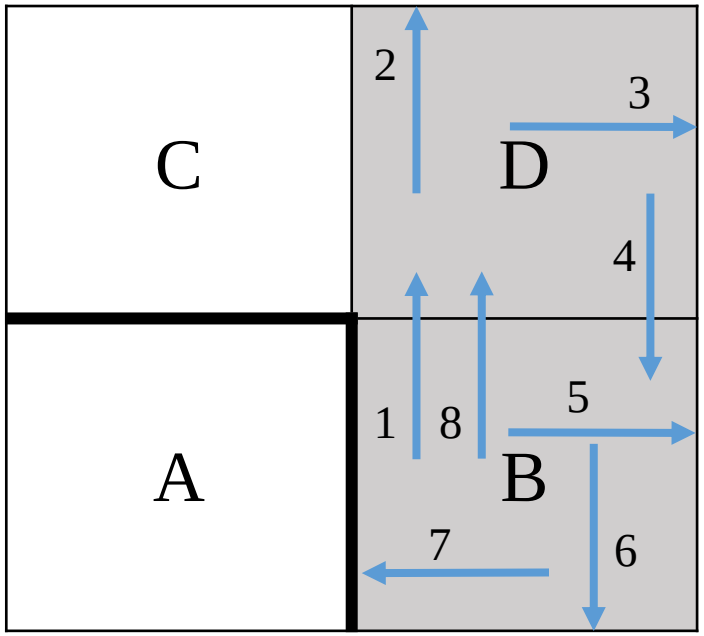
result[B, l] = B

unbacktracked[B] = {D}

a = u $\rightarrow$ unbacktracked[B] = {}

s = B

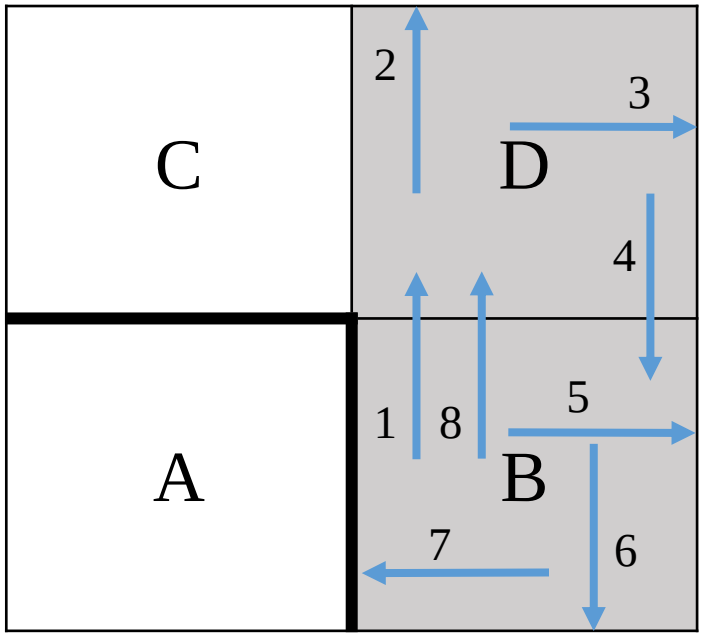
مثال ۲– Online DFS ...



اجرای هشتم:

s‘ = B s = B
untried[B] = {}
result[B, l] = B
unbacktracked[B] = {D}
a = u → unbacktracked[B] = {}
s = B

مثال ۲– Online DFS ...



اجرای نهم:

$s' = D \qquad s = B$

untried[D] = {l}

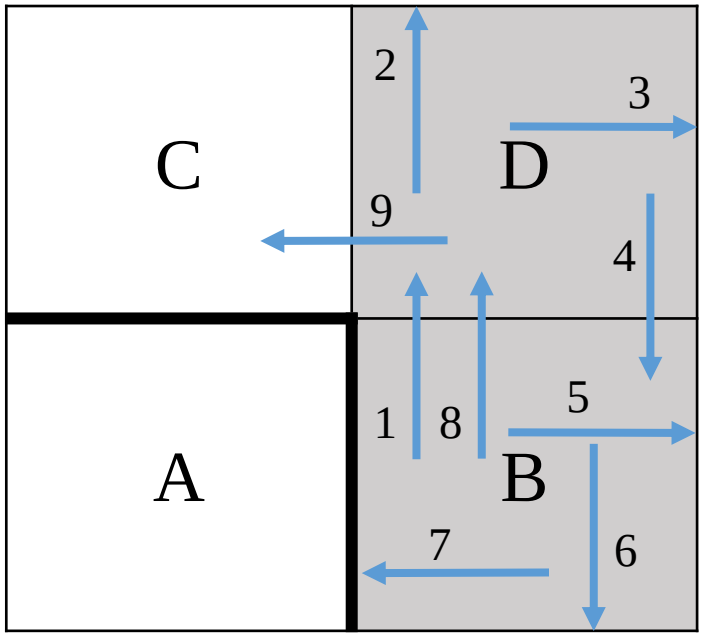
result[B, u] = D

unbacktracked[D] = {B}

a = l → untried[D] = {}

s = D

مثال ۲– Online DFS ...



اجرای نهم:

$s' = D \qquad s = B$

untried[D] = {l}

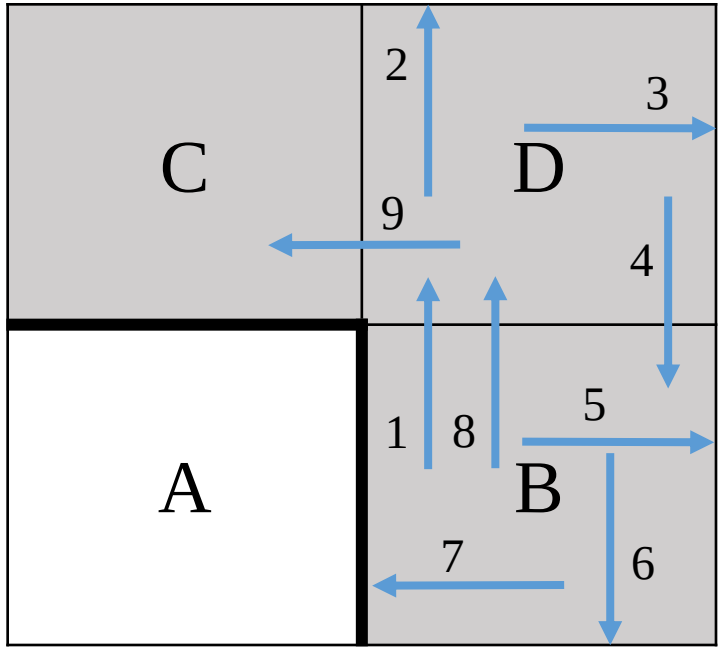
result[B, u] = D

unbacktracked[D] = {B}

$a = l \rightarrow$ untried[D] = {}

s = D

مثال ۲– Online DFS ...



اجرای دهم:

$s' = C \qquad s = D$

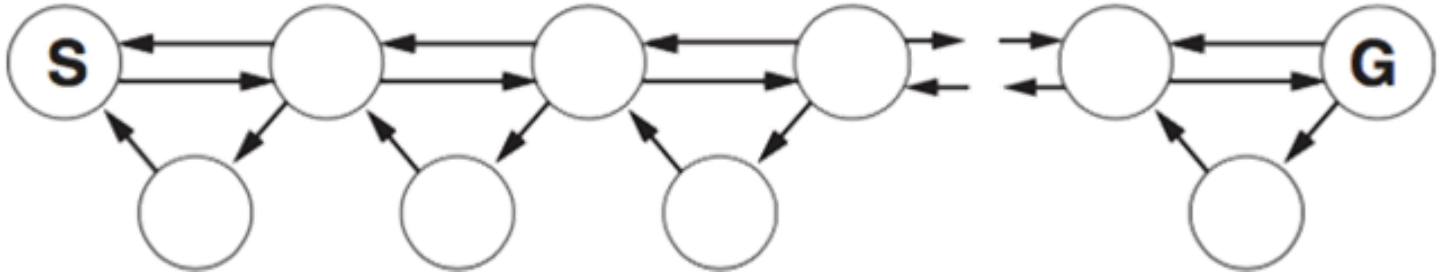
Goal-Test(C) = true → STOP

جستجوی محلی برخط

- جستجوی تپه نوردی تنها یک حالت فعلی را در حافظه نگه‌داری می‌کند و از این‌رو یک جستجوی برخط است.
  - عملکرد بد به دلیل وجود ماکزیمم های محلی
  - شروع مجدد تصادفی غیر ممکن است زیرا عامل نمی‌تواند خود را به یک حالت تصادفی جدید انتقال دهد.
- دو راهکار پیشنهادی
  - گام برداشتن تصادفی (random walk)
  - یادگیرنده‌ی بلادرنگ A* (Learning Real Time A*)

گام برداشتن تصادفی

- یکی از اعمال موجود از حالت فعلی را به صورت تصادفی انتخاب می‌کند.
  - می توان به اقداماتی که تا کنون امتحان نشده‌اند اولویت بالاتری داد.
- اگر فضا متناهی بود در نهایت یک هدف را پیدا می‌کند یا اکتشافش را کامل می‌کند.
  - ممکن است خیلی کند باشد (می‌تواند حالات بسیاری را به‌صورت نمایی تولید کند)



یادگیرنده‌ی بلادرنگ A* (LRTA*)

- این الگوریتم به تپه‌نوردی یک حافظه اضافه می‌کند.
  - H(s)= هزینه‌ی بهترین تخمین فعلی (current best estimate) رفتن از وضعیت s به وضعیت هدف
- H(s) با برآورد هیوریستیک h(s) آغاز می‌شود و به‌تدریج که عامل در فضای حالت تجربه کسب می‌کند، به‌روز می‌شود.
- عامل هزینه‌ی تخمینی حالتی را که هم‌اکنون از آن خارج شده است را به‌روز می‌کند. سپس براساس تخمین‌های هزینه‌ی فعلی‌اش، حرکتی را که به ظاهر بهترین است انتخاب می‌کند.
- نکته: اعمالی که تا این لحظه در حالت s امتحان نشده‌اند، همیشه فرض می‌شوند که با کم‌ترین هزینه‌ی ممکن h(s) به هدف منتهی می‌شوند.
  - این موضوع عامل را تشویق می‌کند تا همواره مسیر جدیدی را انتخاب کند و به‌جای عقب‌گرد کردن، مسیر خود را ادامه دهد.

عامل یادگیرنده‌ی بلادرنگ A* (LRTA*)

function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table, indexed by state and action, initially empty
    H, a table of cost estimates indexed by state, initially empty
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

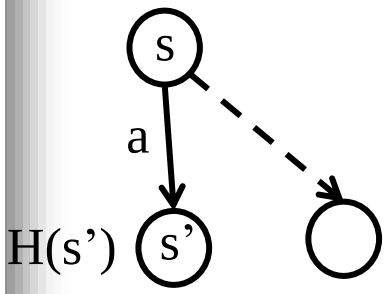
function LRTA*-COST(s, a, s', H) **returns** a cost estimate
  if s' is undefined **then return** $h(s)$
  else return $c(s, a, s') + H[s']$

عامل یادگیرنده‌ی بلادرنگ A* (LRTA*)

function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table, indexed by state and action, initially empty
    H, a table of cost estimates indexed by state, initially empty
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

function LRTA*-COST(s, a, s', H) **returns** a cost estimate
  if s' is undefined **then return** $h(s)$
  else return $c(s, a, s') + H[s']$

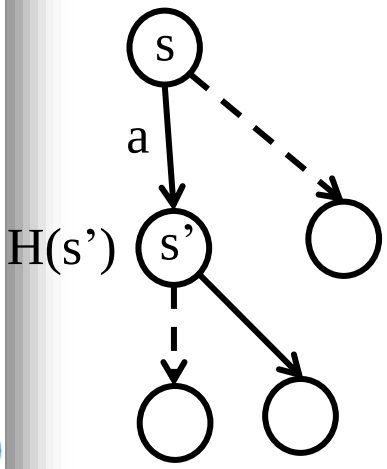


عامل یادگیرنده‌ی بلادرنگ A* (LRTA*)

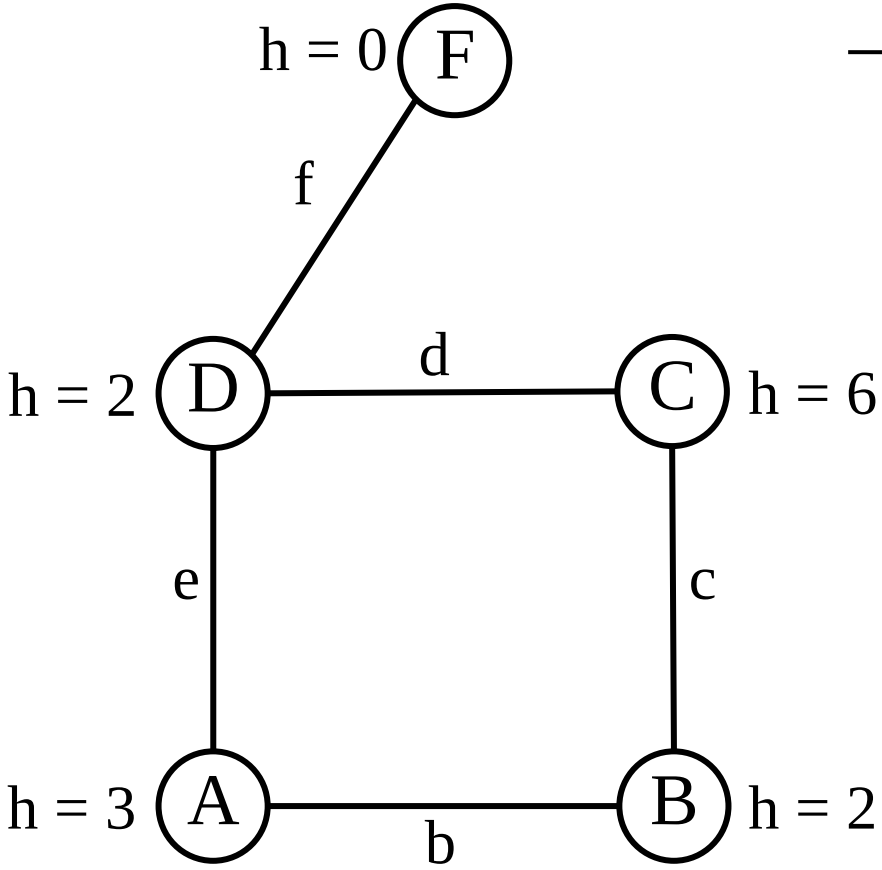
function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table, indexed by state and action, initially empty
    H, a table of cost estimates indexed by state, initially empty
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

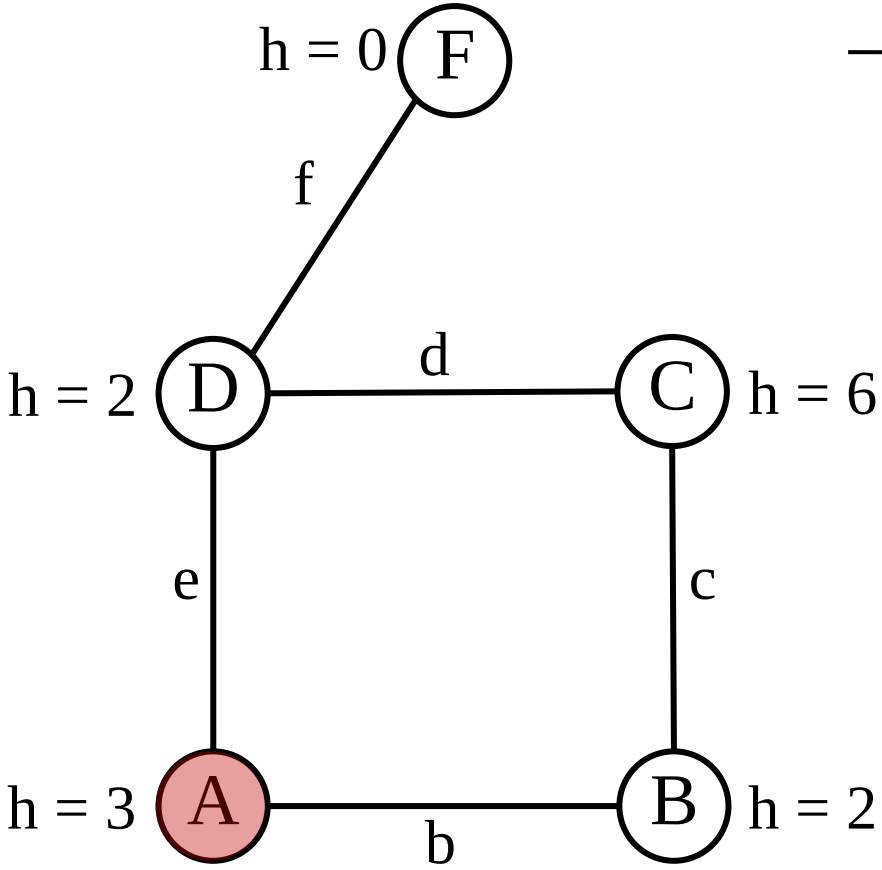
function LRTA*-COST(s, a, s', H) **returns** a cost estimate
  if s' is undefined **then return** $h(s)$
  else return $c(s, a, s') + H[s']$



مثال ۱ – LRTA*

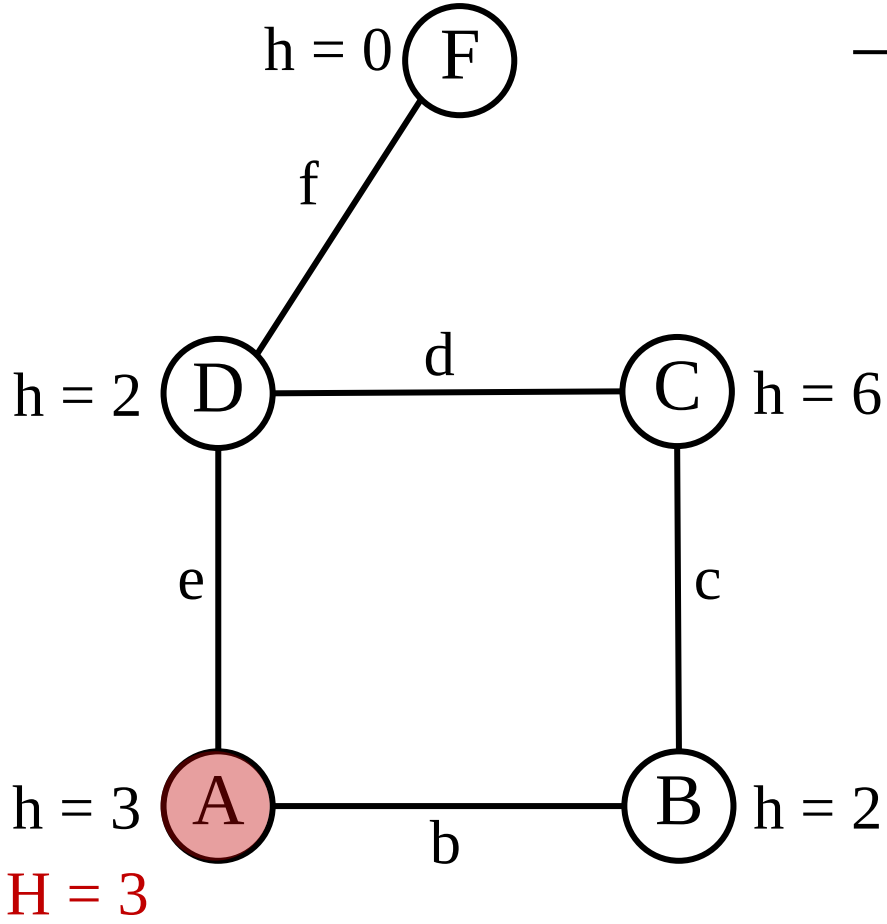


مثال ۱ – LRTA*



s‘ = A s = null

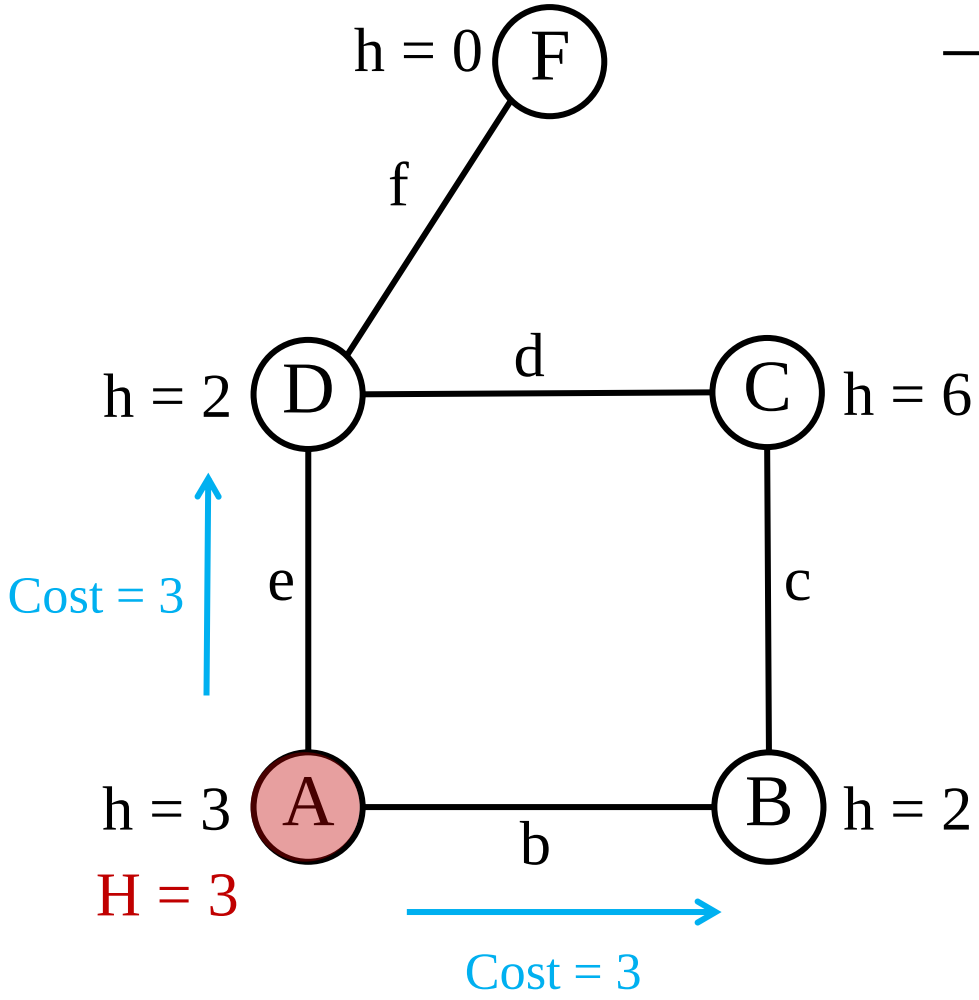
مثال ۱ – LRTA*



s‘ = A s = null

H(A) = h(A) = 3

مثال ۱ – LRTA*

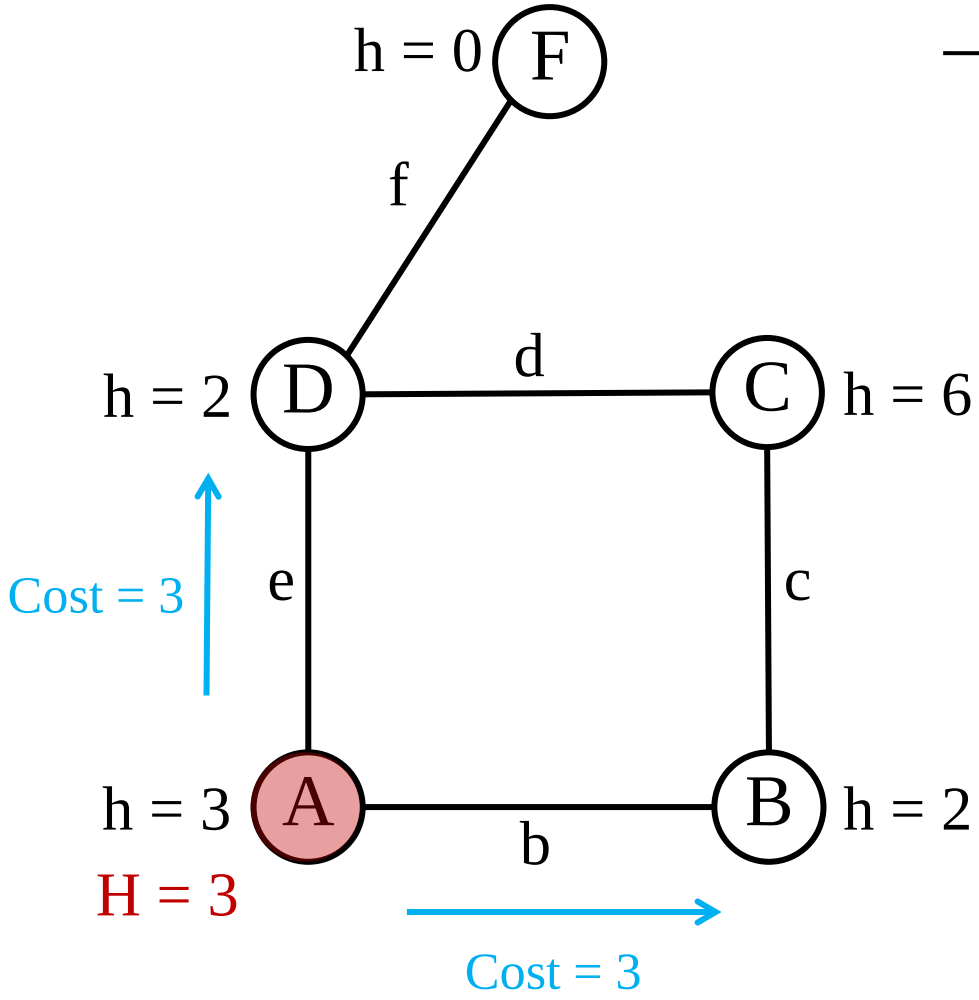


s‘ = A s = null

H(A) = h(A) = 3

a = b

مثال ۱ – LRTA*



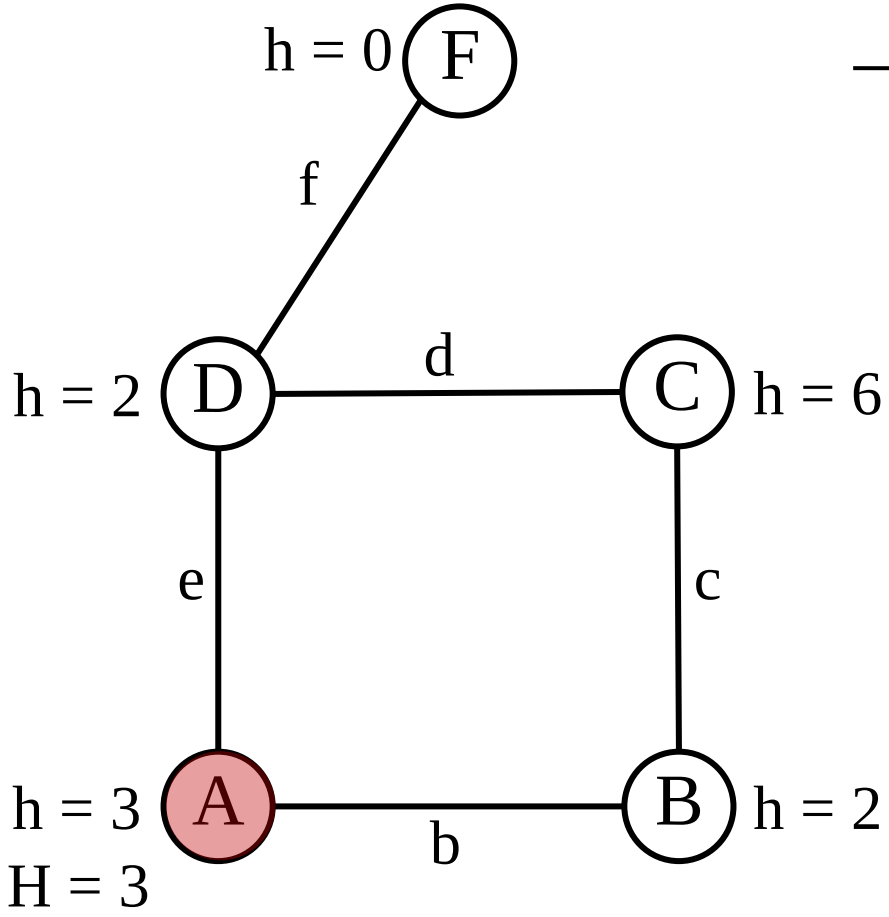
s‘ = A s = null

H(A) = h(A) = 3

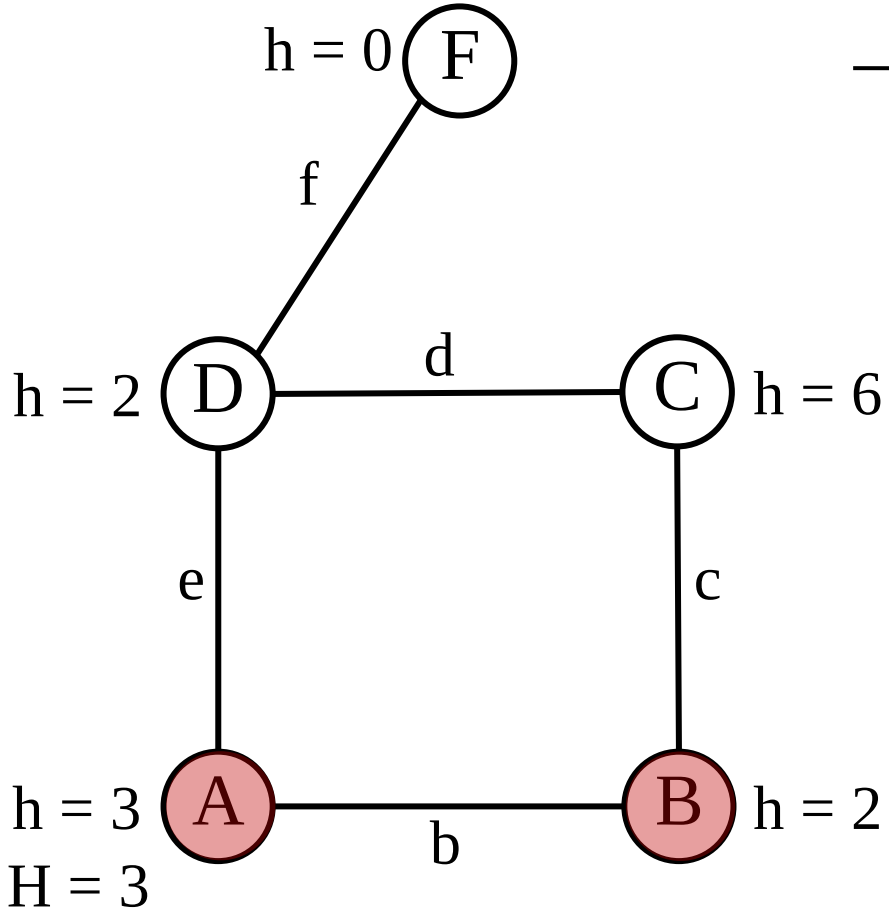
a = b

s = A

مثال ۱ – LRTA* …

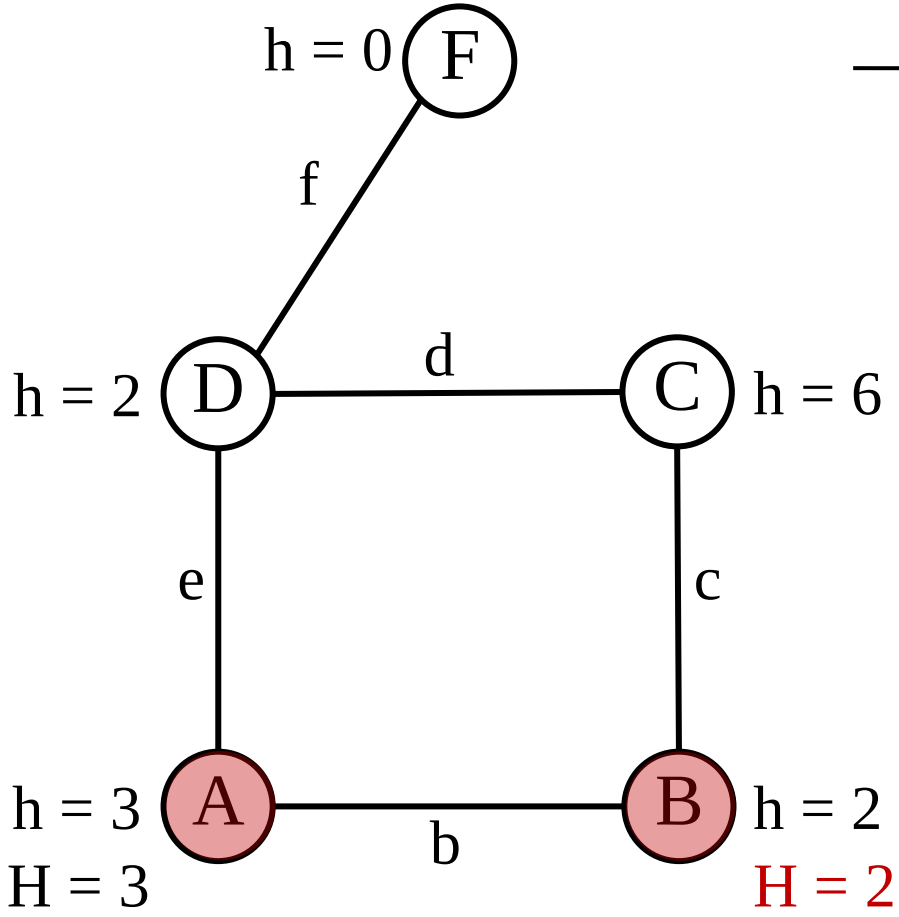


مثال ۱ – LRTA* …



$s' = B$ $s = A$

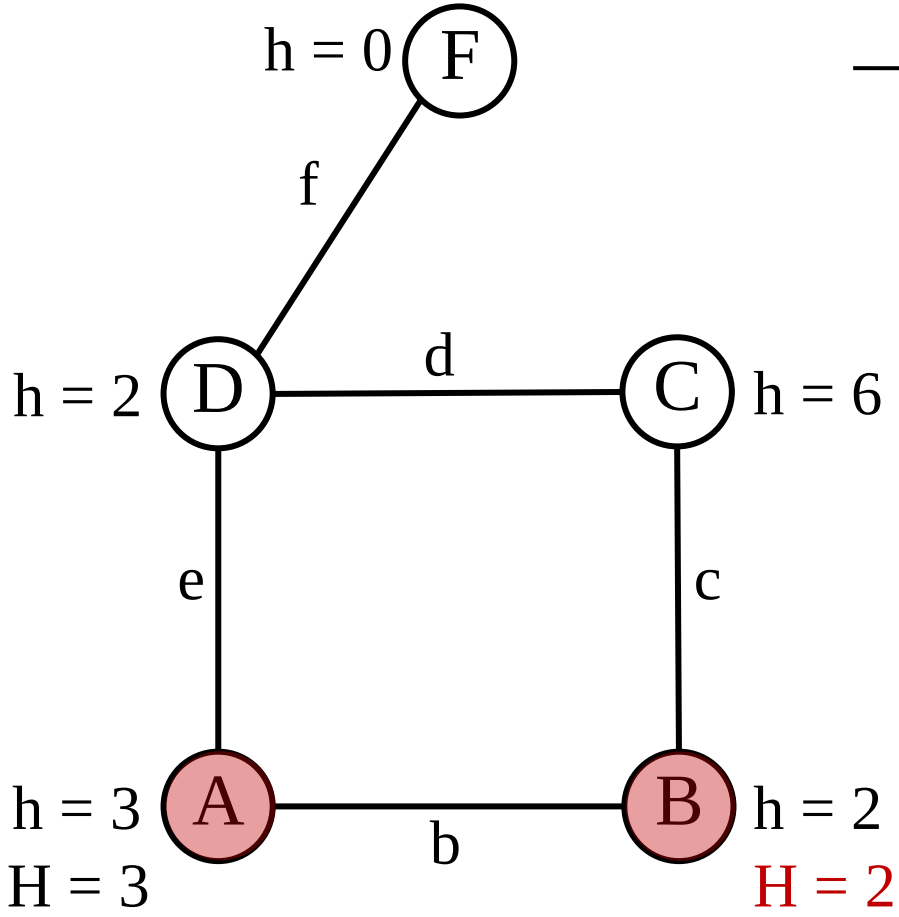
مثال ۱ – LRTA* …



$s' = B$ $s = A$

$H(B) = h(B) = 2$

مثال ۱ – LRTA* …

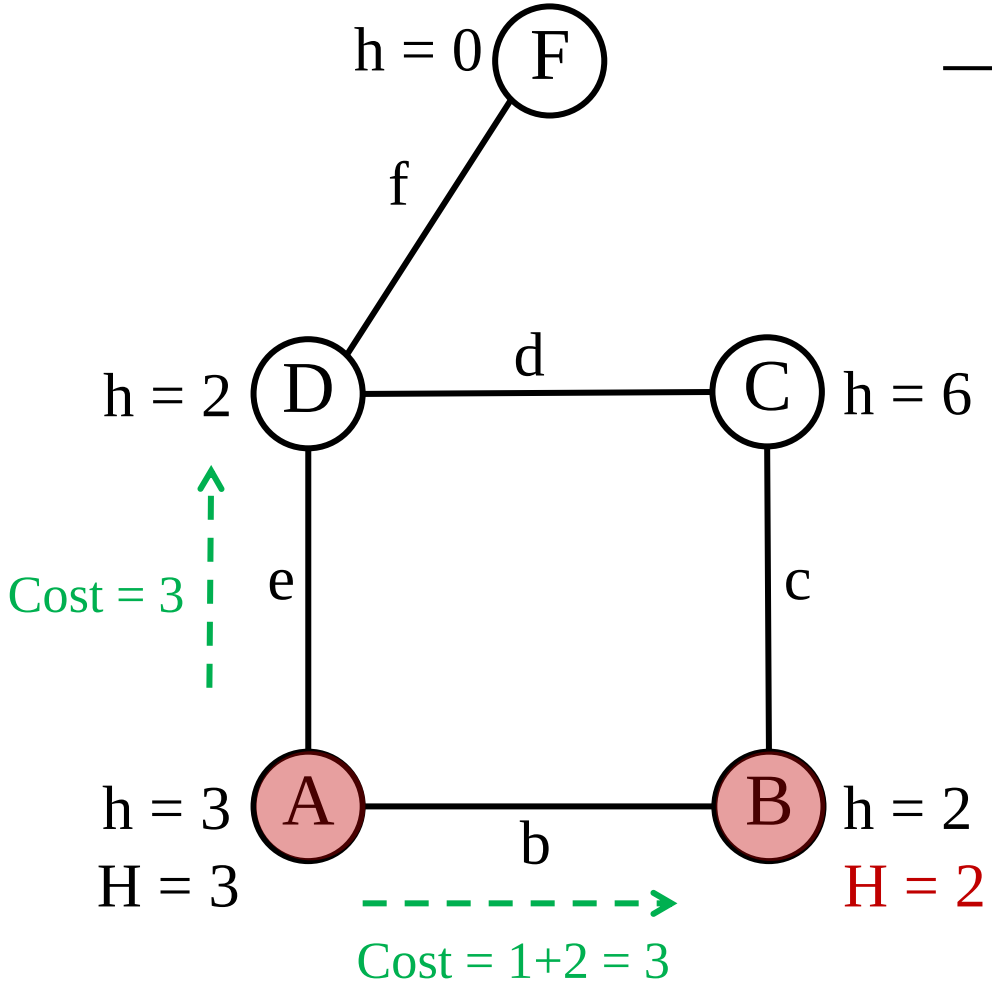


$s' = B$ $s = A$

$H(B) = h(B) = 2$

$result(A, b) = B$

مثال ۱ – LRTA* …

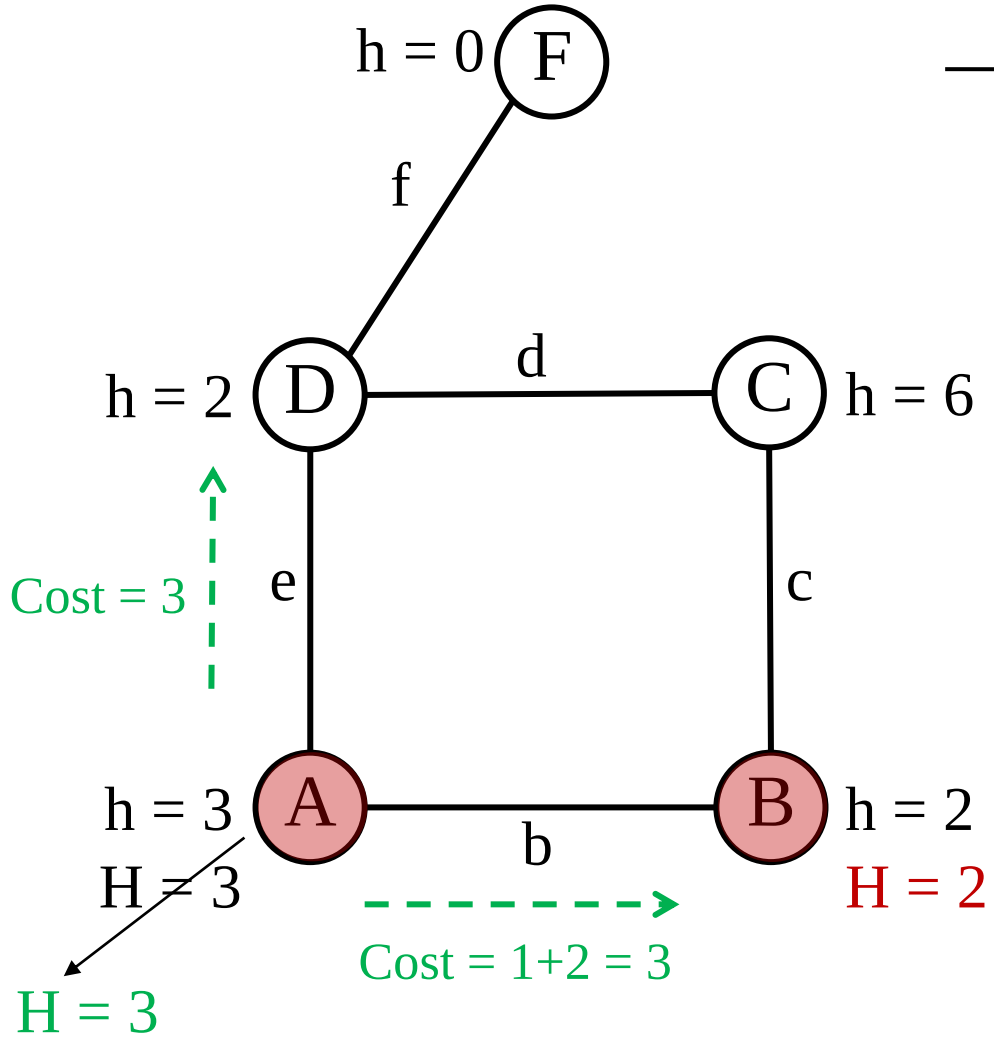


$s' = B$ $s = A$

$H(B) = h(B) = 2$

$result(A, b) = B$

مثال ۱ – LRTA* …



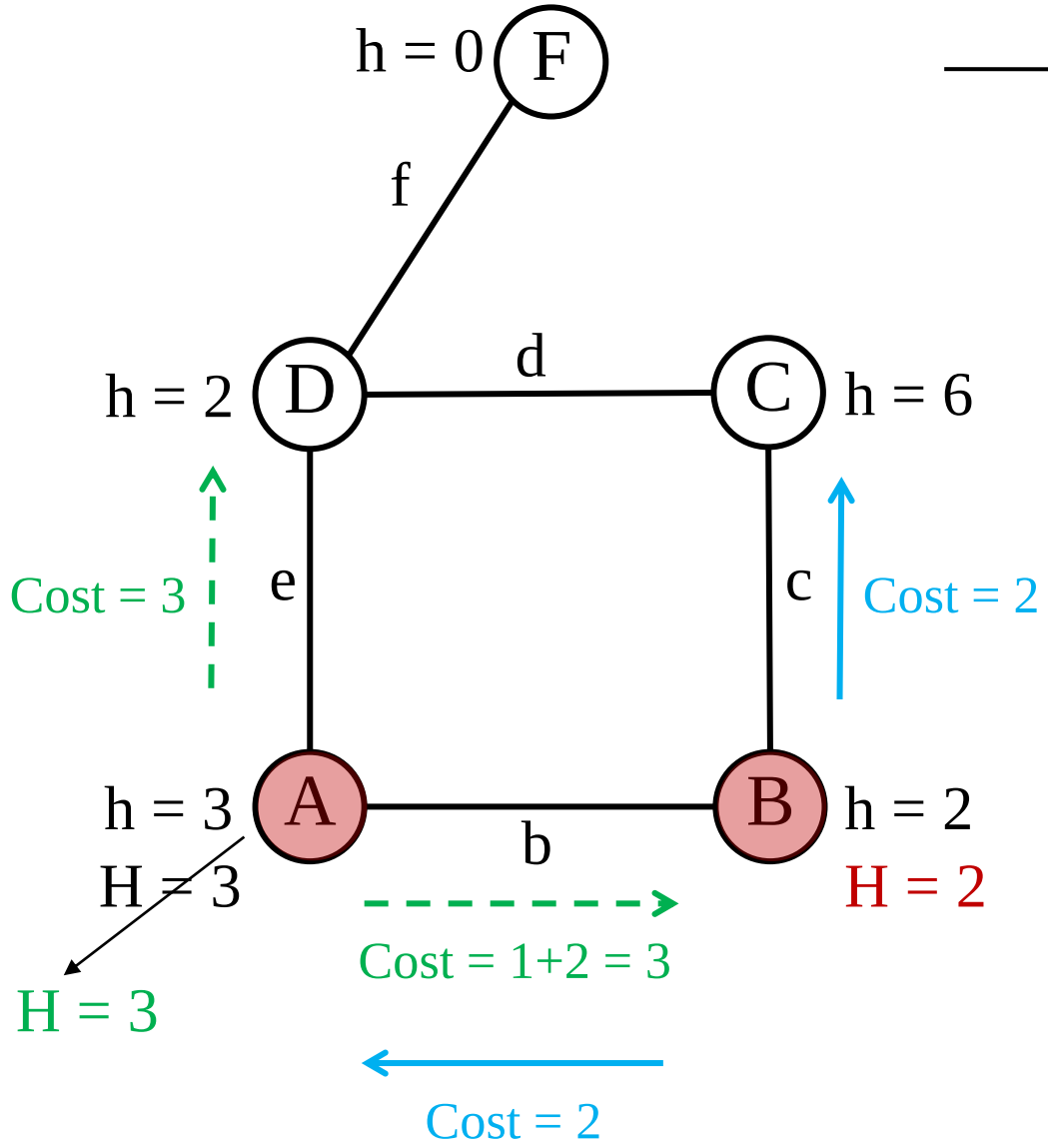
$s' = B$ $s = A$

$H(B) = h(B) = 2$

$result(A, b) = B$

$H(A) = 3$

مثال ۱ – LRTA* …



$s' = B$ $s = A$

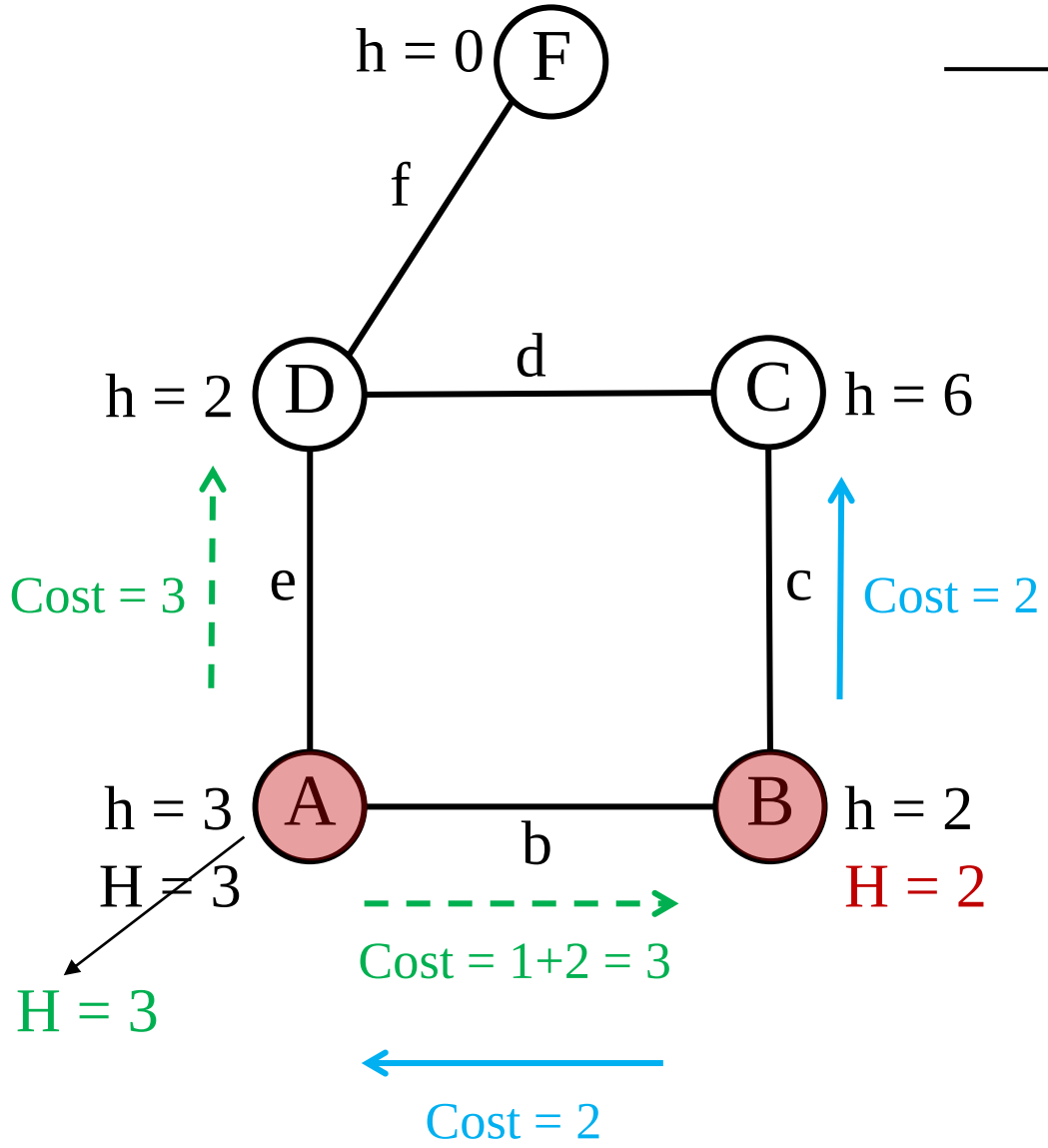
$H(B) = h(B) = 2$

result(A, b) = B

$H(A) = 3$

$a = c$

مثال ۱ – LRTA* …



$s' = B$ $s = A$

$H(B) = h(B) = 2$

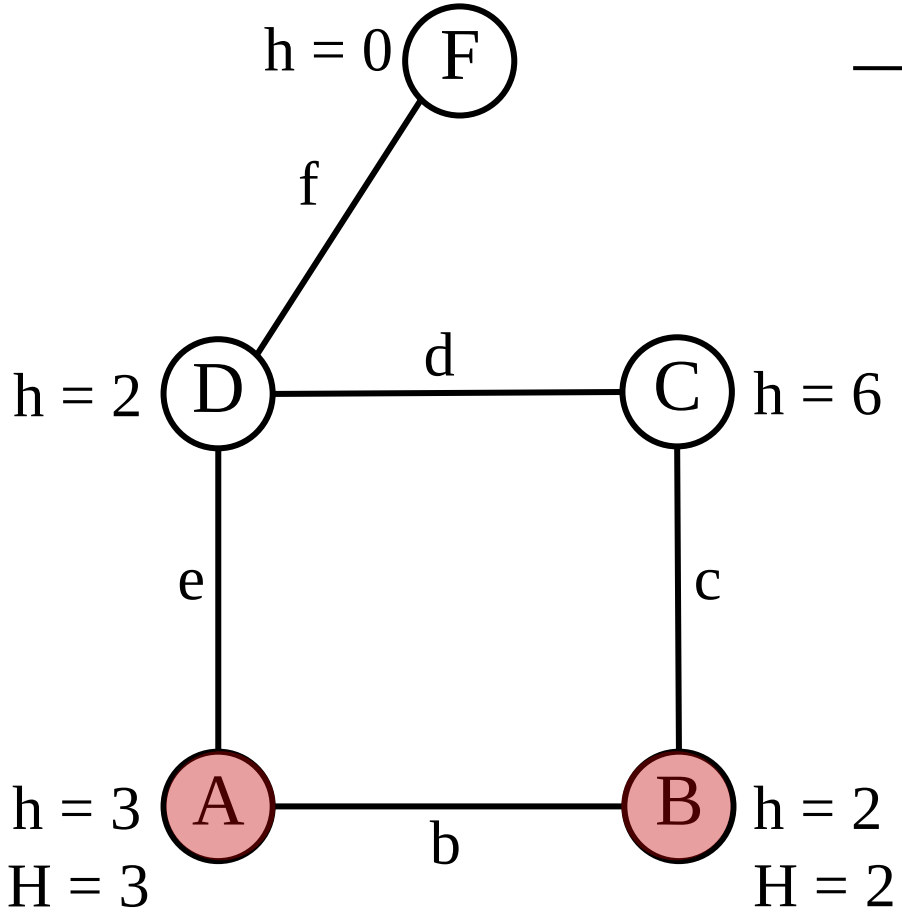
$result(A, b) = B$

$H(A) = 3$

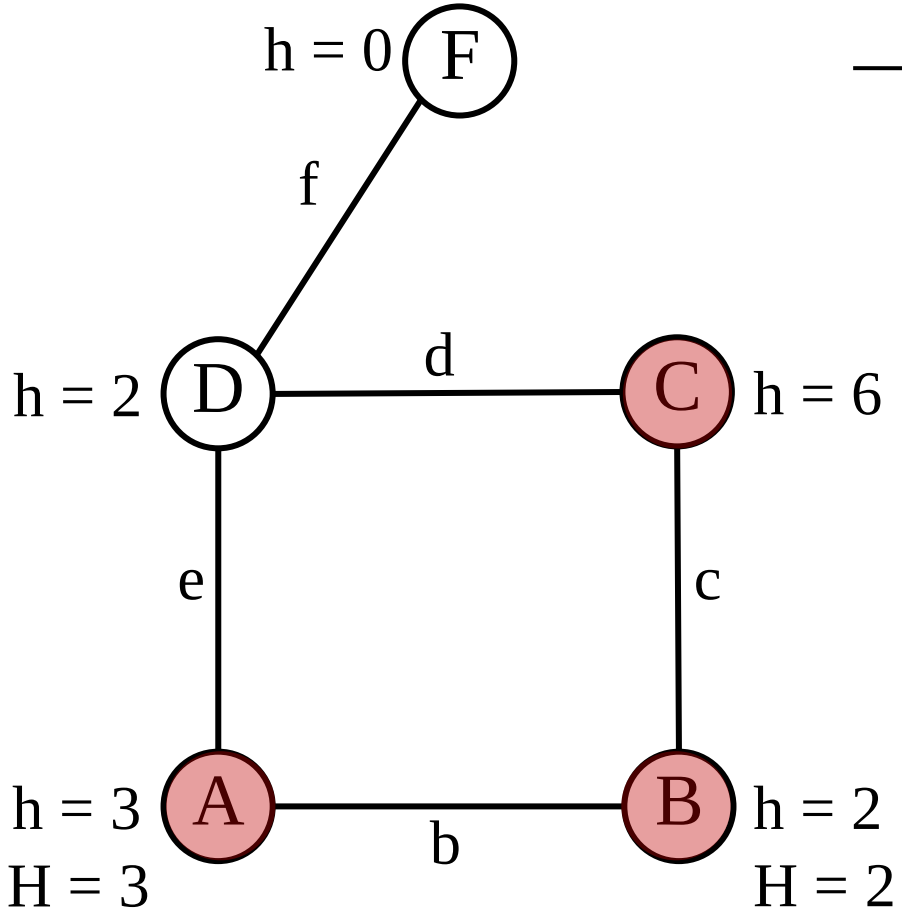
$a = c$

$s = B$

مثال ۱ – LRTA* …

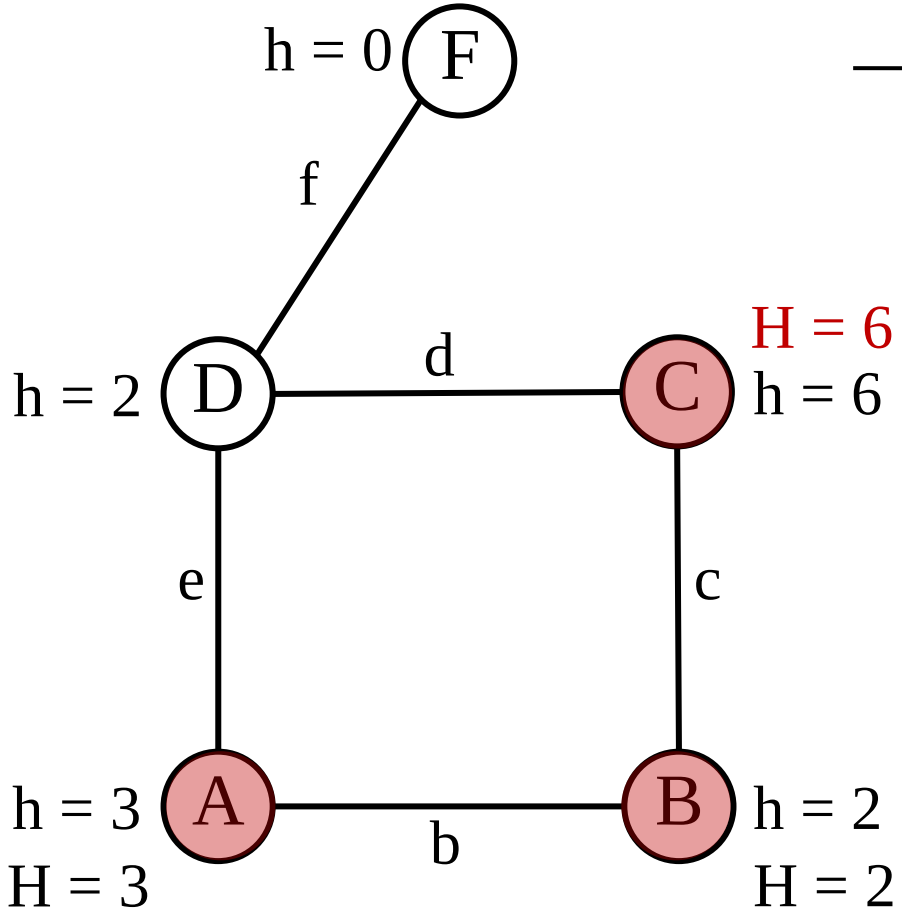


مثال ۱ – LRTA* …



$s' = C$ $s = B$

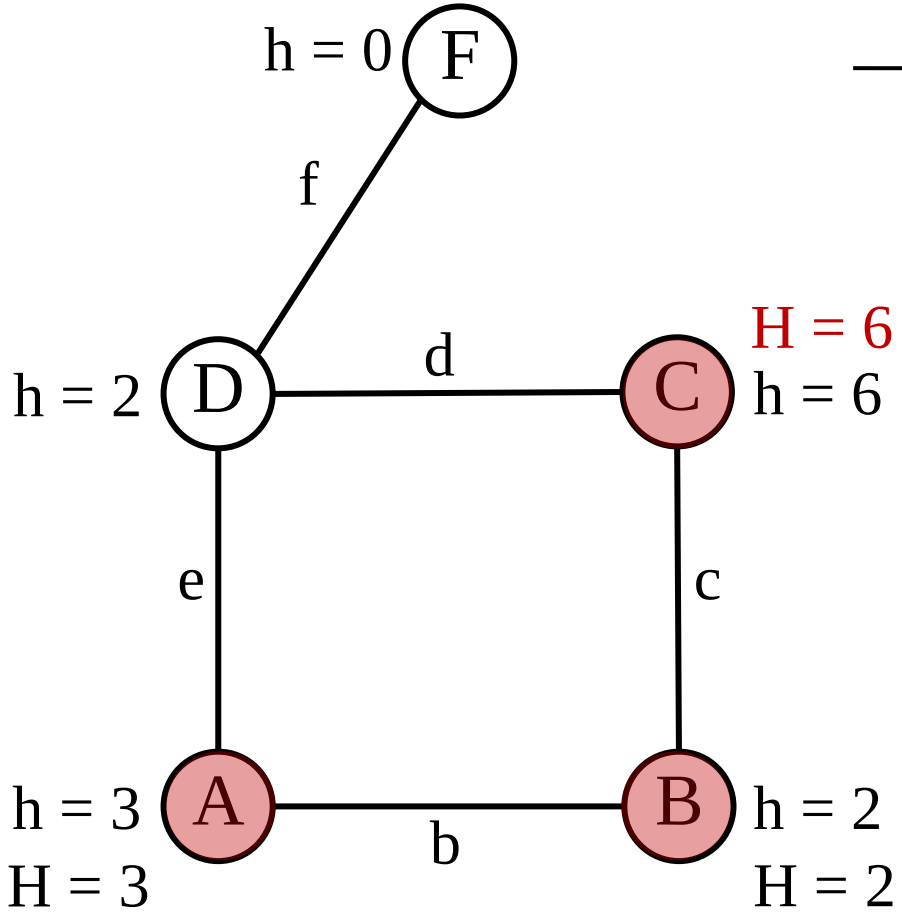
مثال ۱ – LRTA* …



$s' = C$ $s = B$

$H(C) = h(C) = 6$

مثال ۱ – LRTA* …

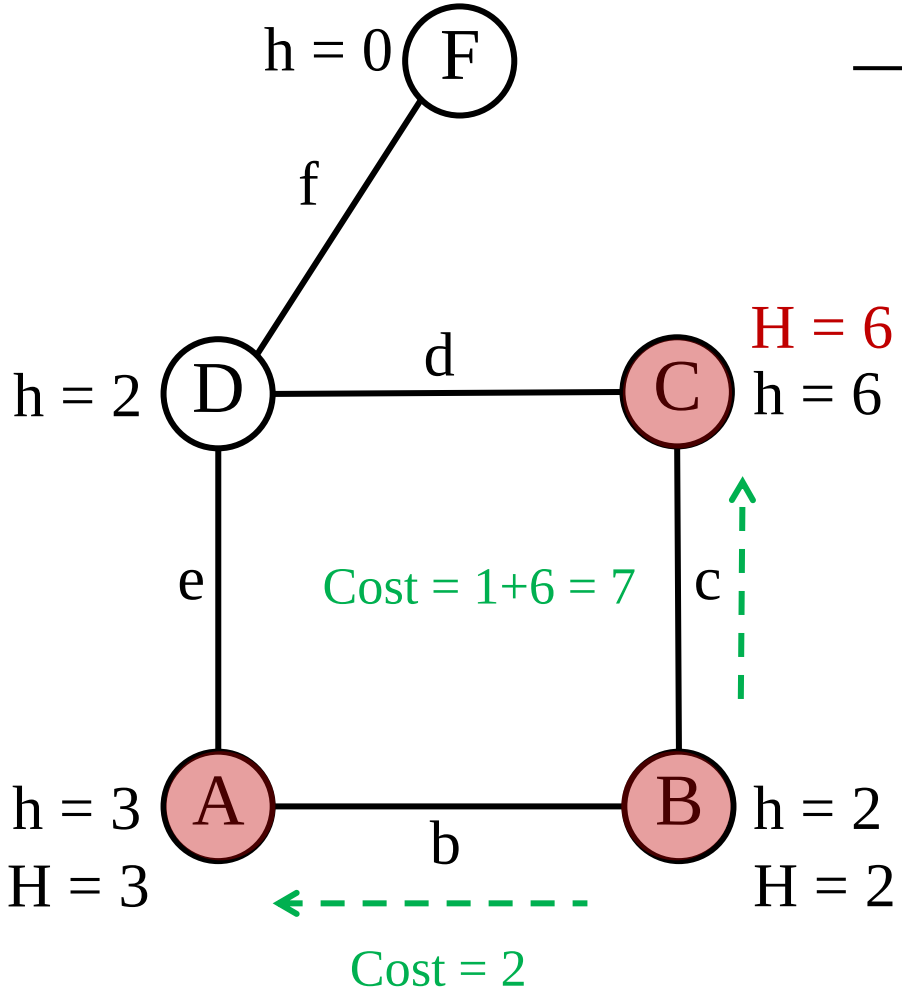


$s' = C$ $s = B$

$H(C) = h(C) = 6$

$result(B, c) = C$

مثال ۱ – LRTA* ...

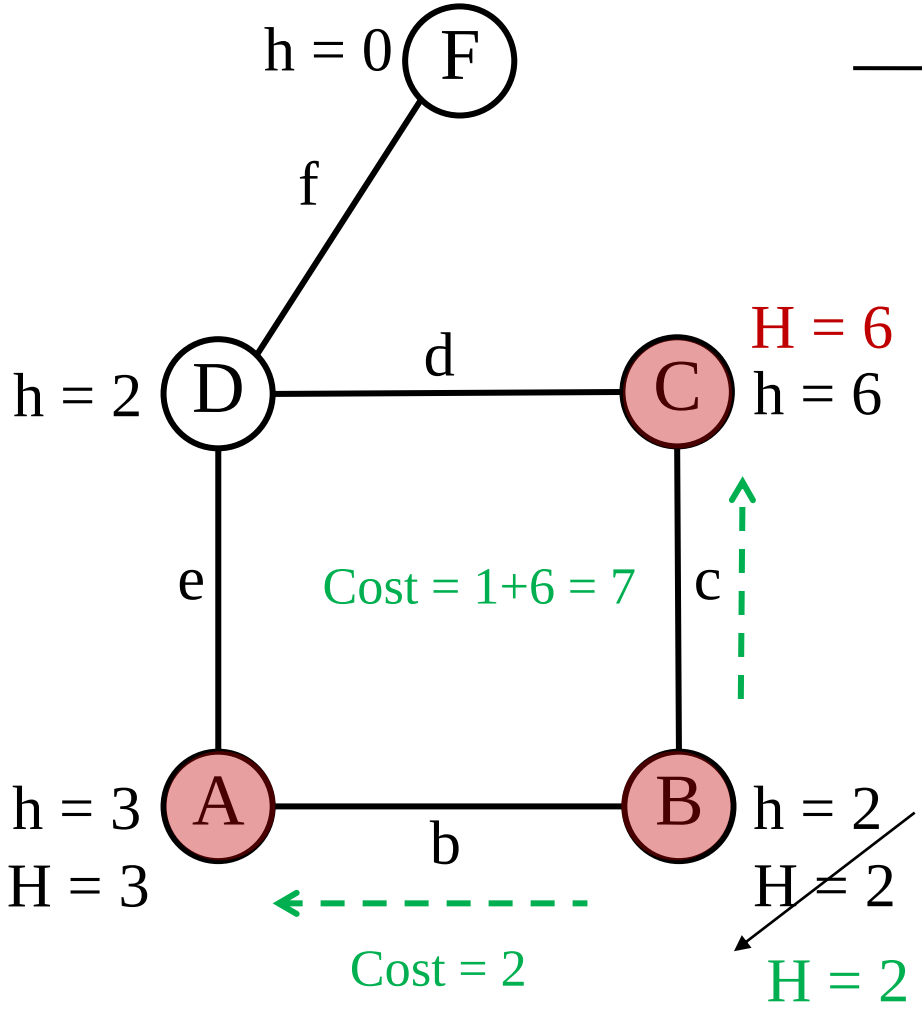


$s' = C$ $s = B$

$H(C) = h(C) = 6$

$result(B, c) = C$

مثال ۱ – LRTA* …



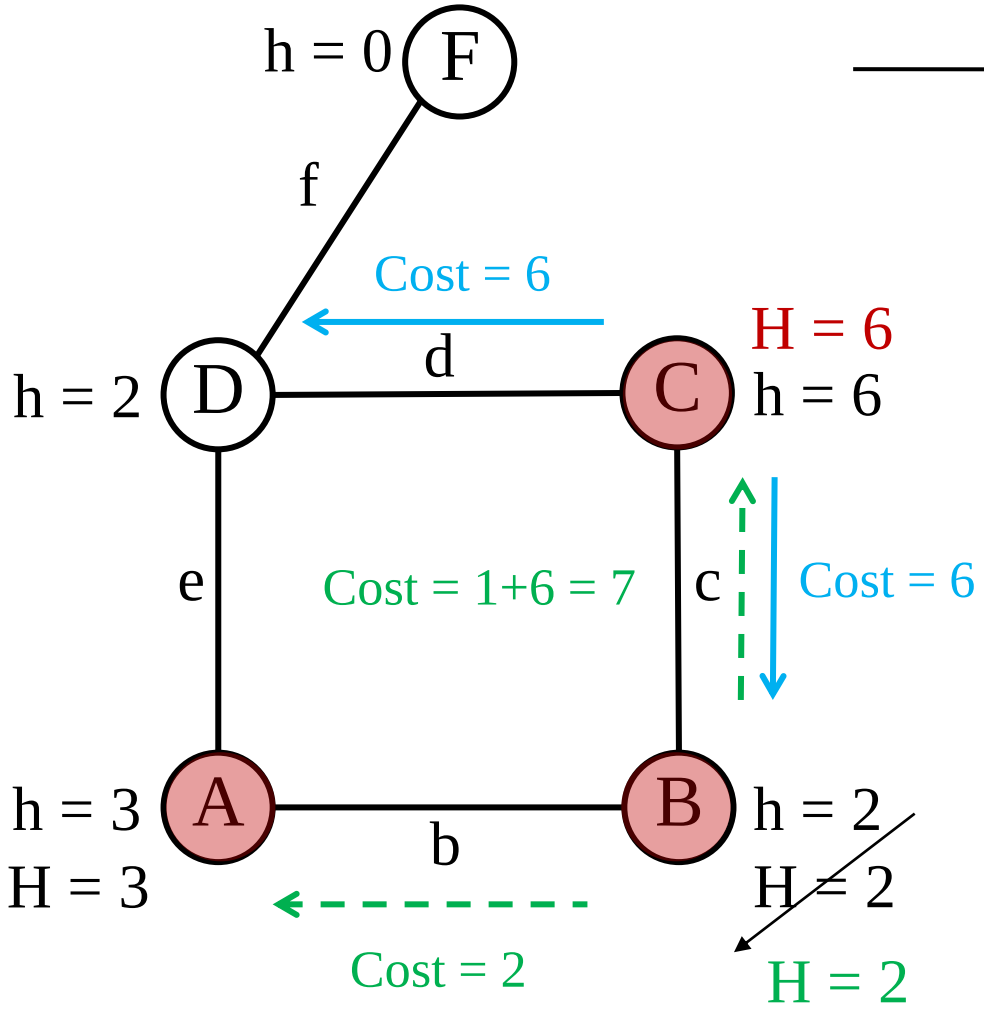
$s' = C$ $s = B$

$H(C) = h(C) = 6$

$result(B, c) = C$

$H(B) = 2$

مثال ۱ – LRTA* …



$s' = C$ $s = B$

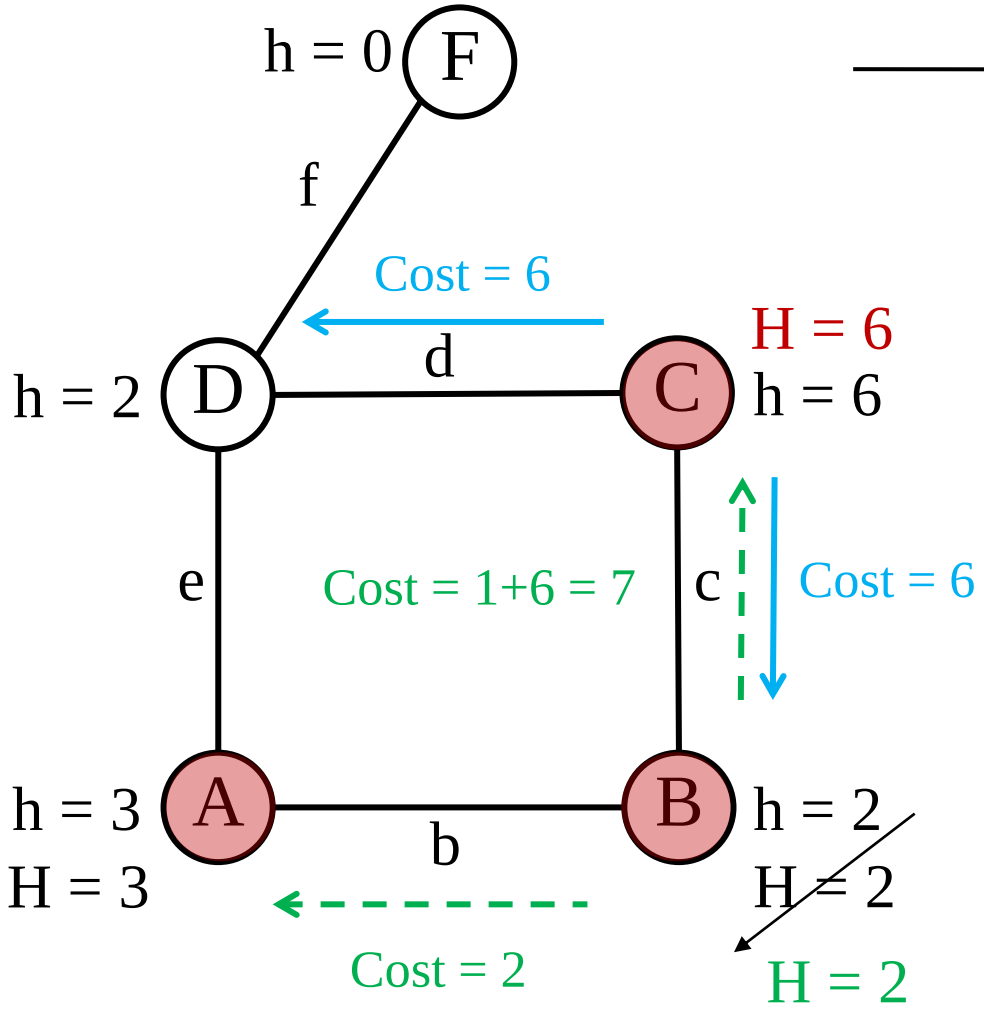
$H(C) = h(C) = 6$

$result(B, c) = C$

$H(B) = 2$

$a = c$

مثال ۱ – LRTA* …



$s' = C$ $s = B$

$H(C) = h(C) = 6$

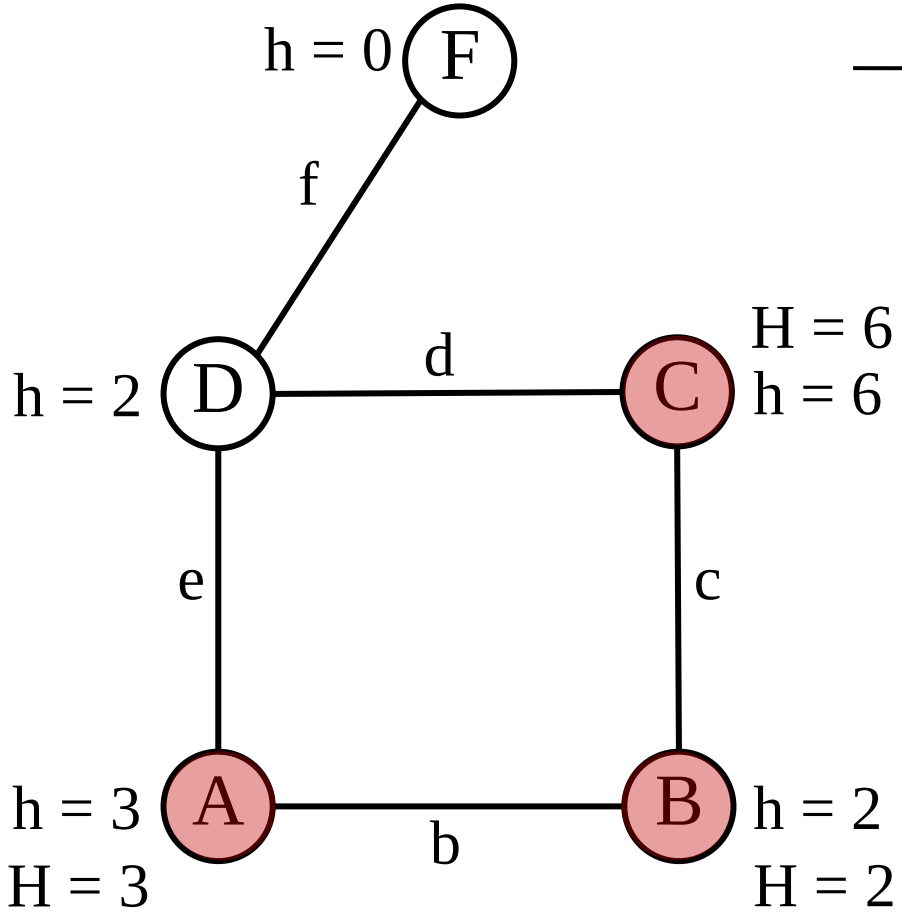
$result(B, c) = C$

$H(B) = 2$

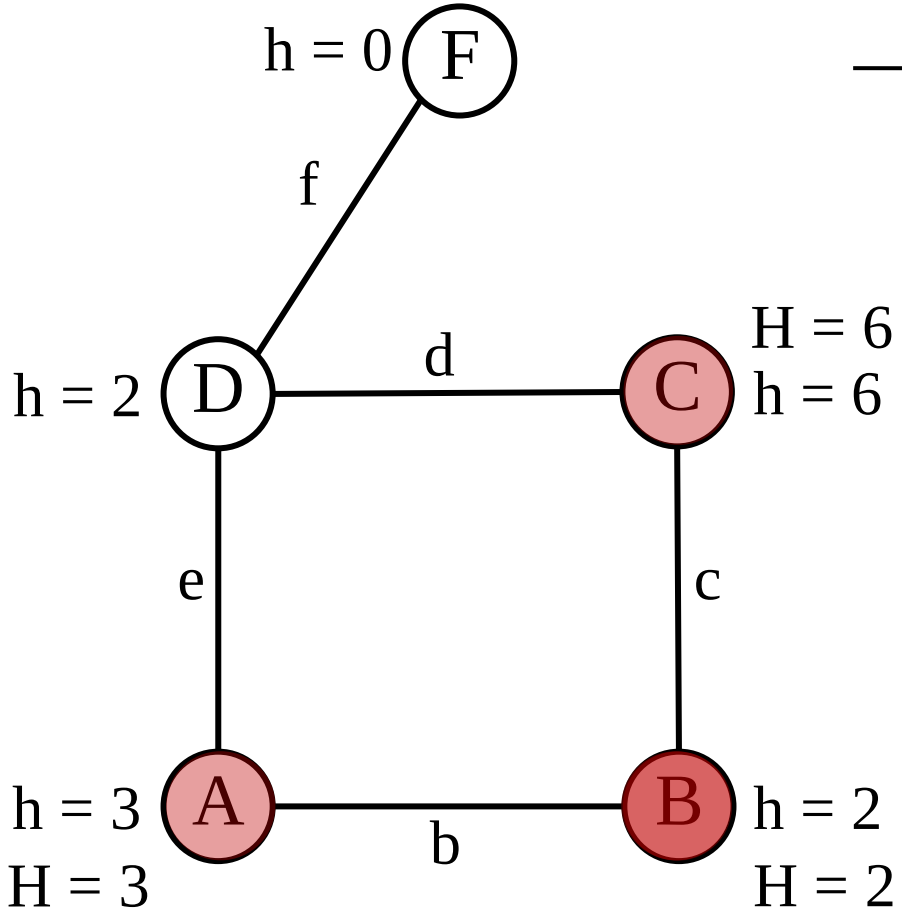
$a = c$

$s = C$

مثال ۱ – LRTA* …

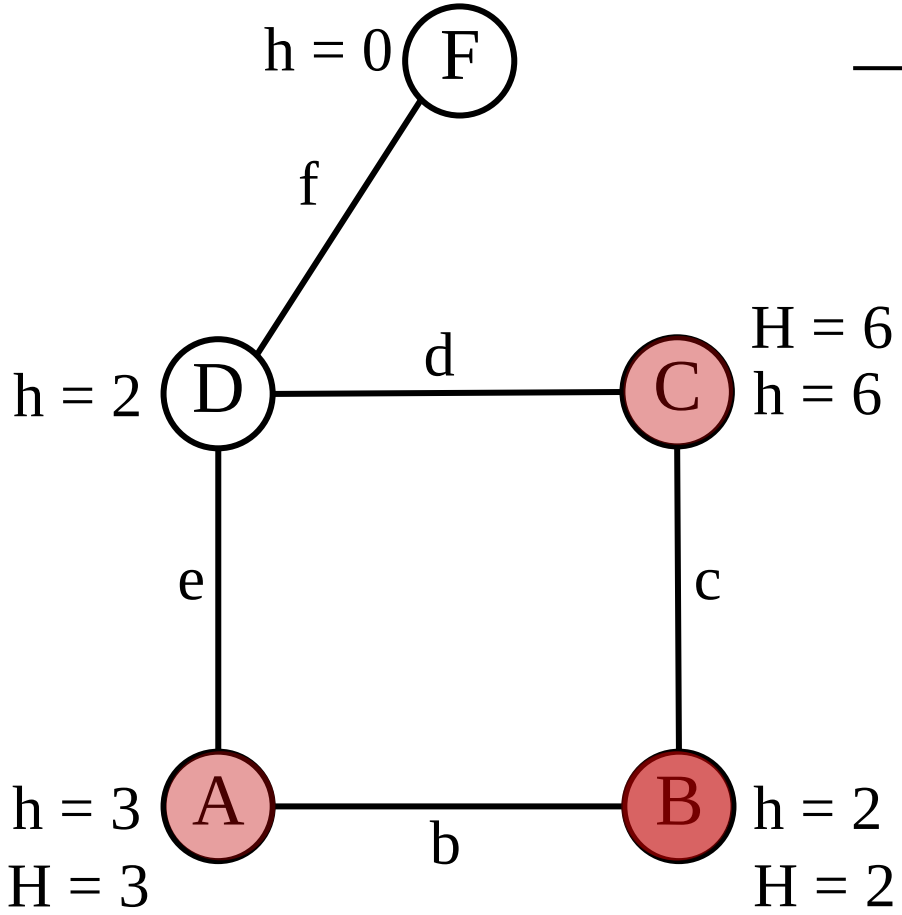


مثال ۱ – LRTA* …



$s' = B$ $s = C$

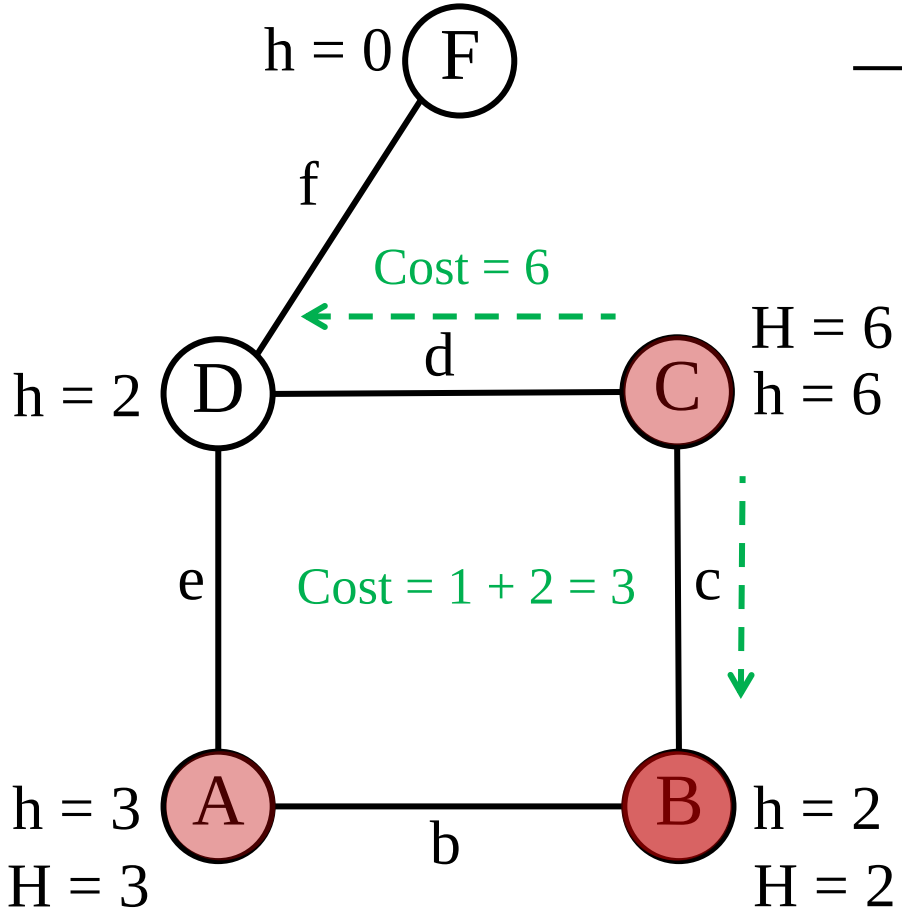
مثال ۱ – LRTA* …



$s' = B$ $s = C$

$result(C, c) = B$

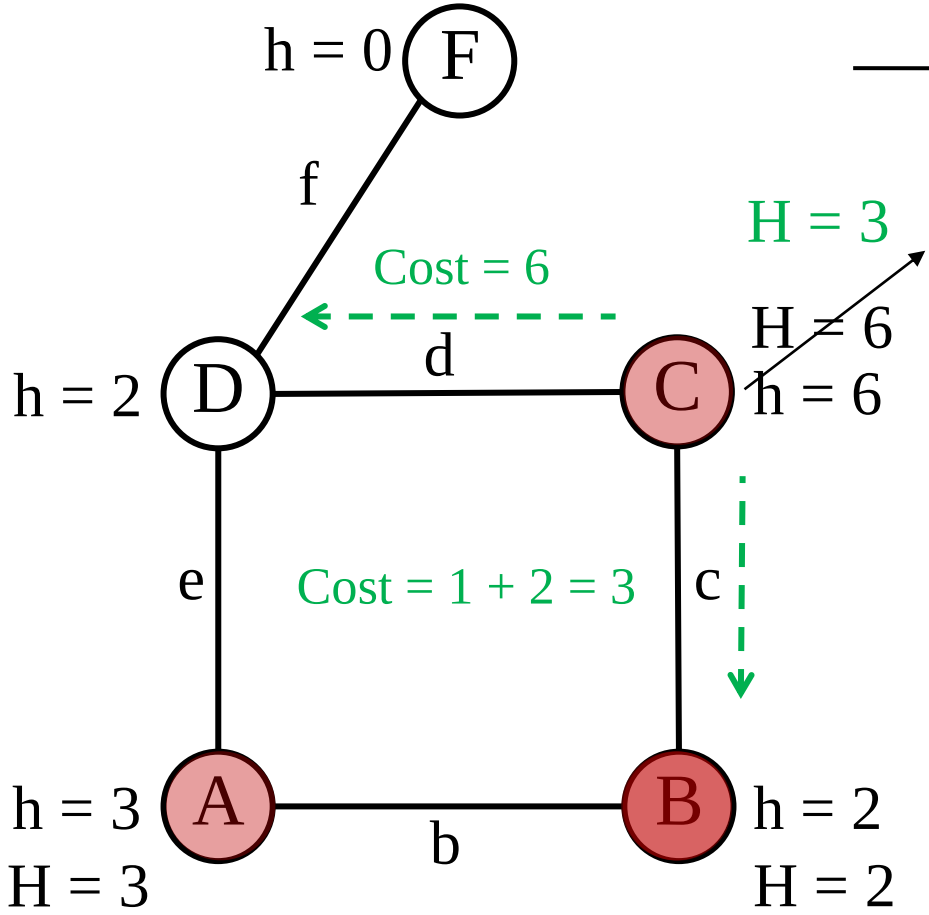
مثال ۱ – LRTA* …



$s' = B$ $s = C$

result(C, c) = B

مثال ۱ – LRTA* …

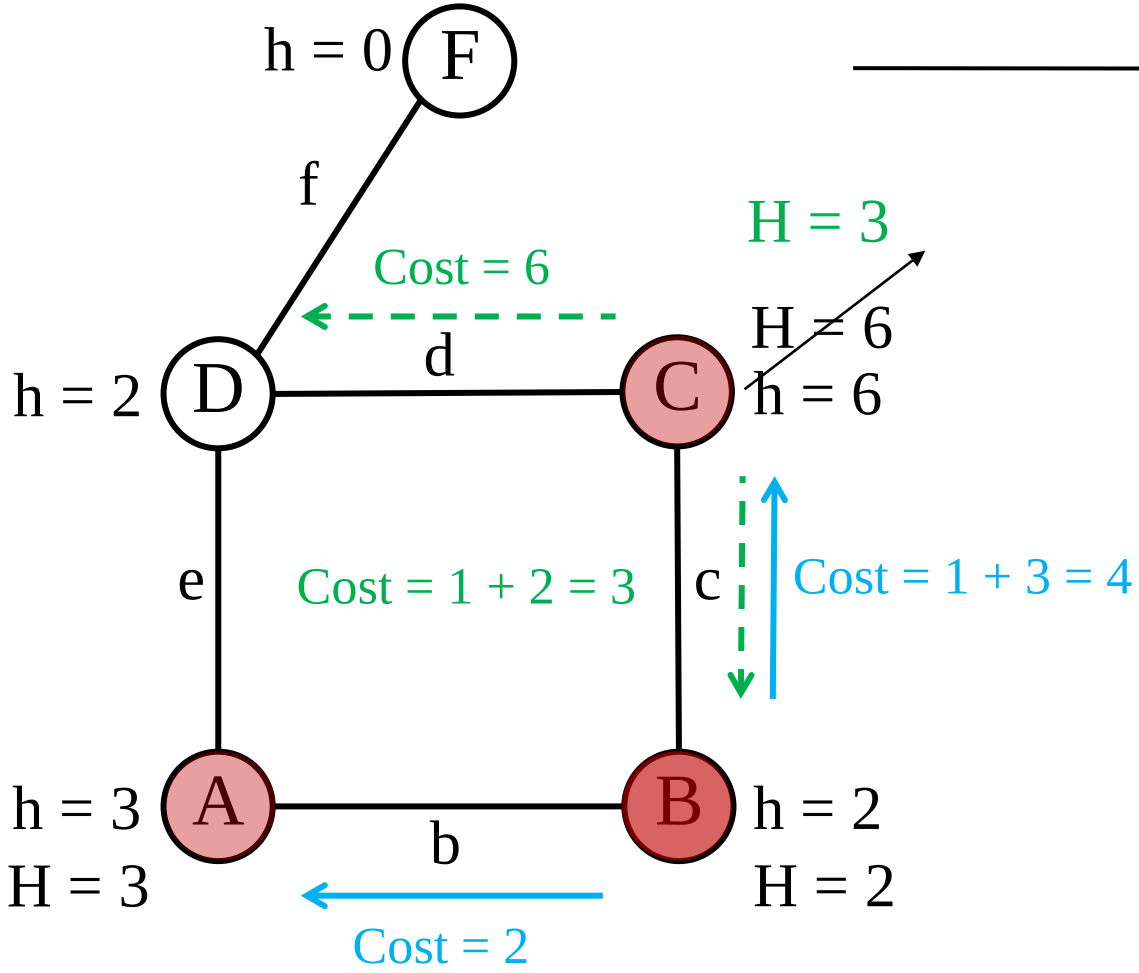


$s' = B$ $s = C$

result(C, c) = B

H(C) = 3

مثال ۱ – LRTA* …



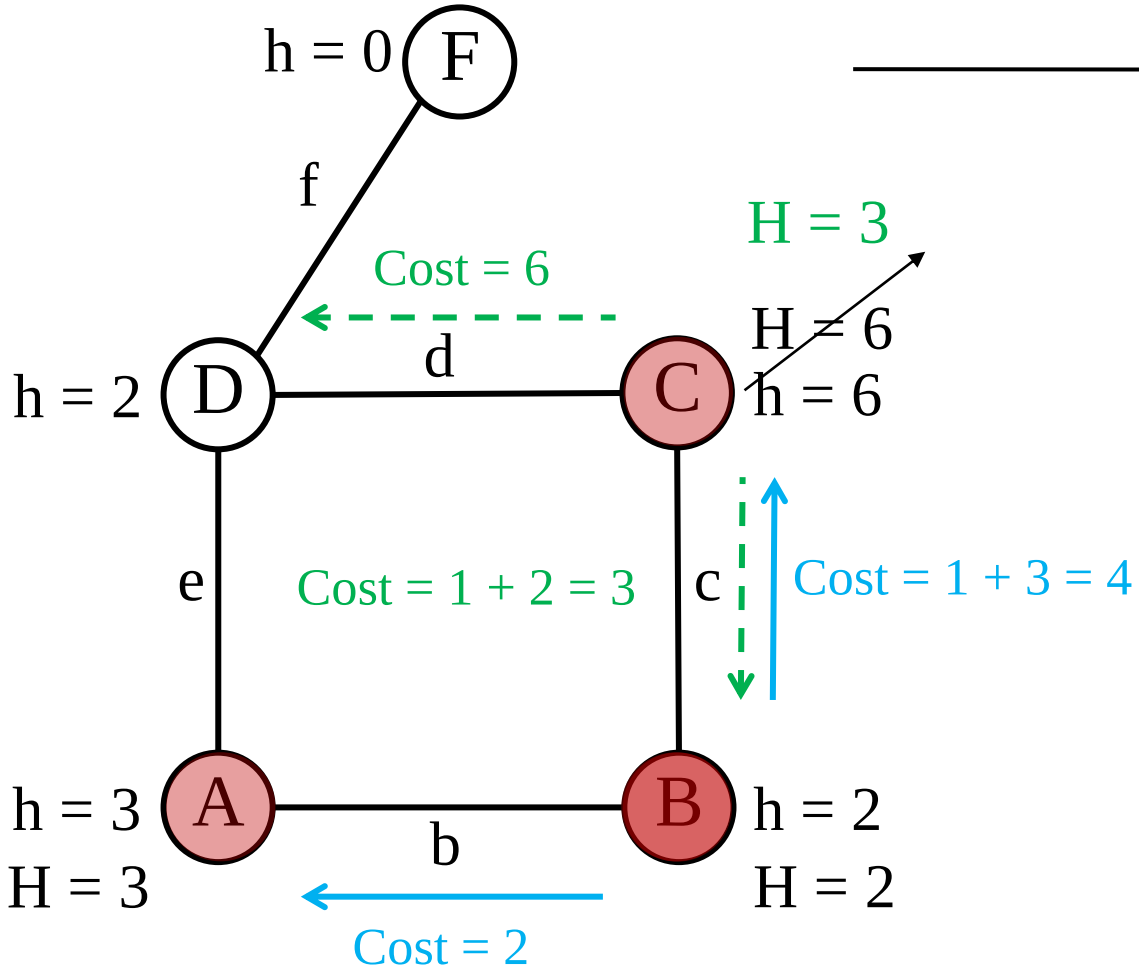
$s' = B$ $s = C$

result(C, c) = B

H(C) = 3

a = b

مثال ۱ – LRTA* …



$s' = B$ $s = C$

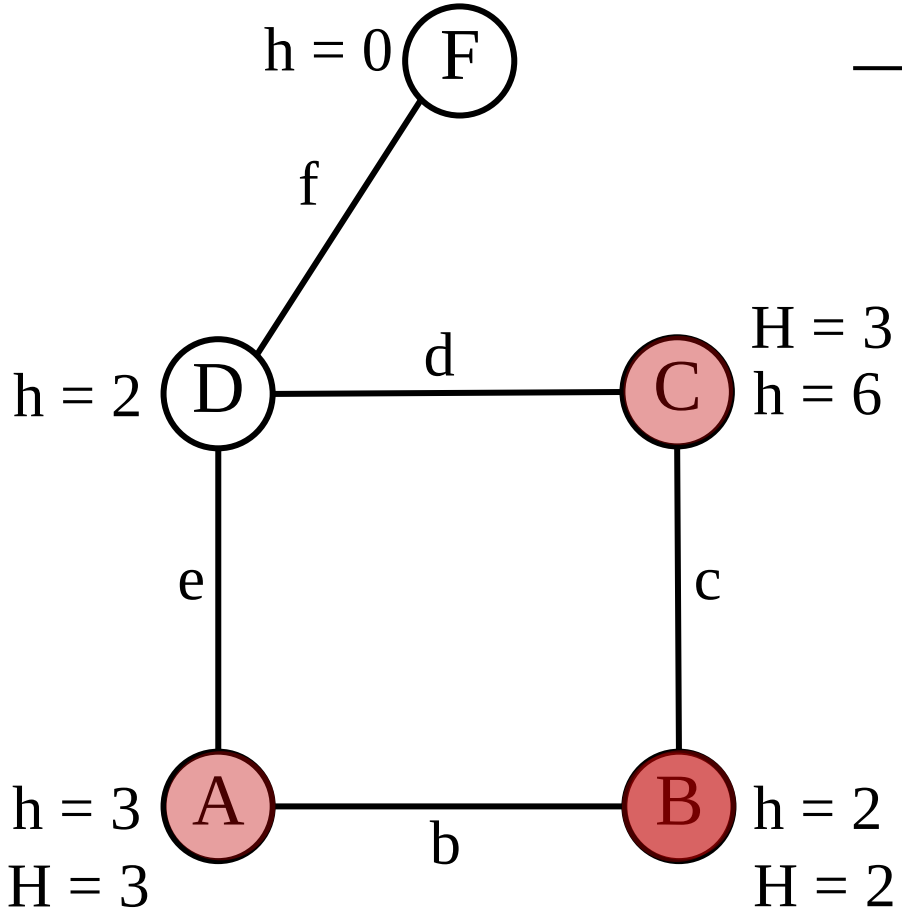
result(C, c) = B

H(C) = 3

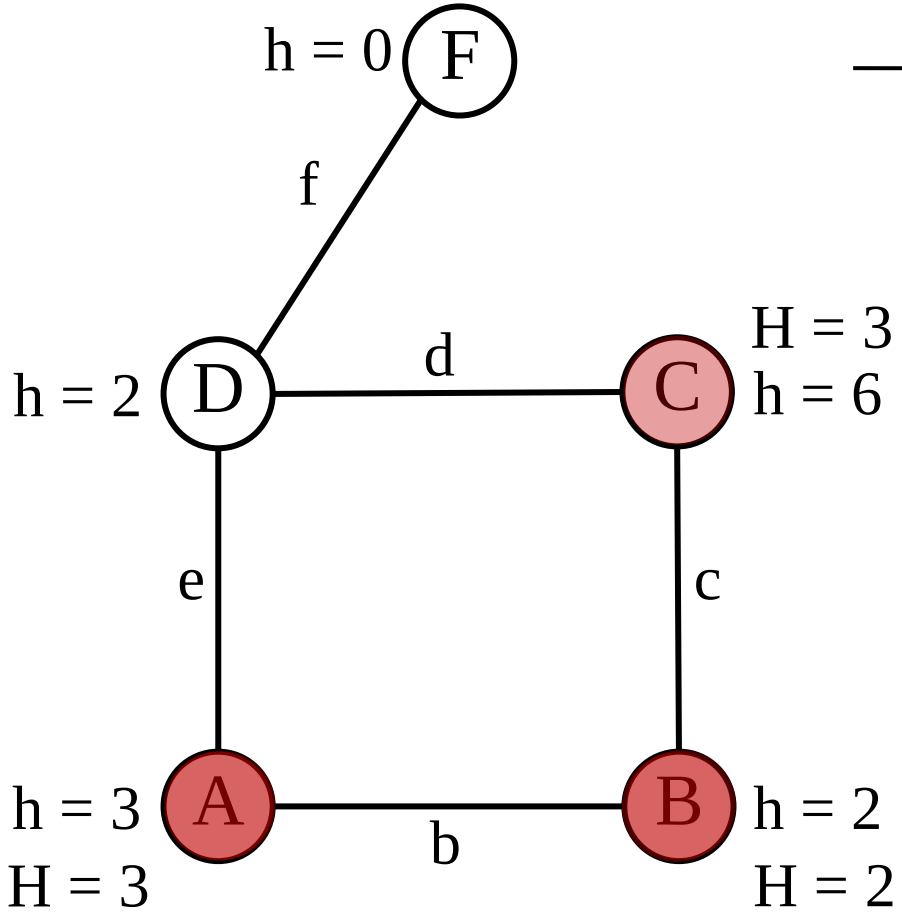
a = b

s = B

مثال ۱ – LRTA* …

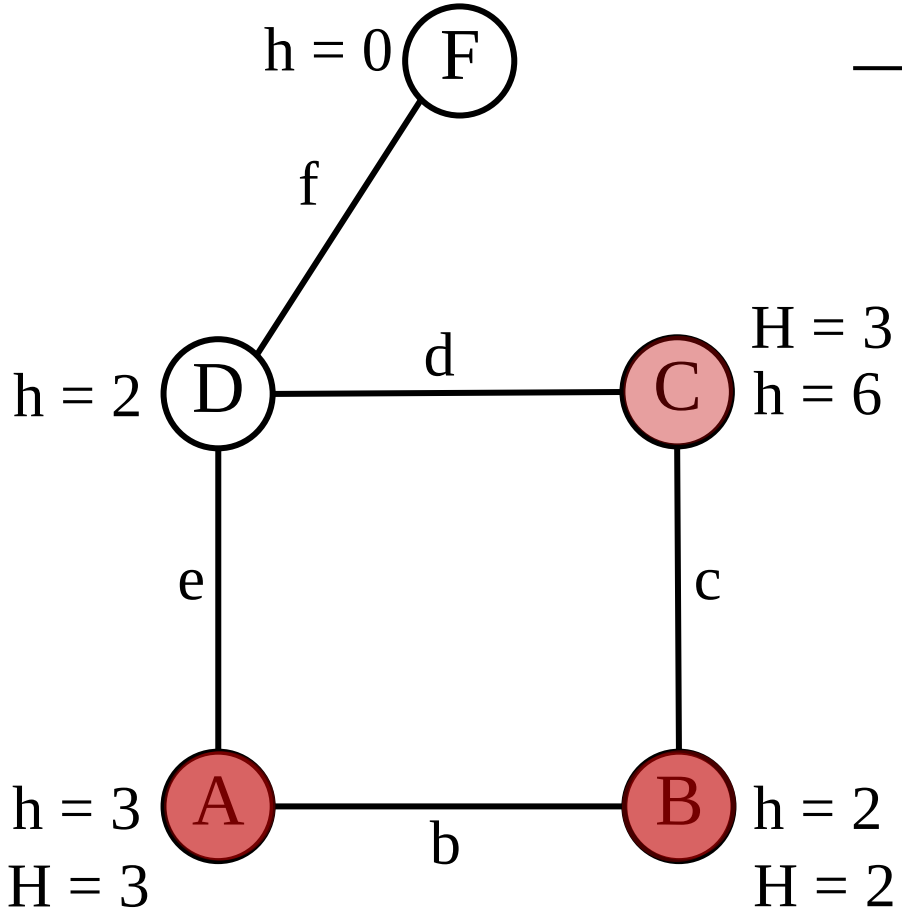


مثال ۱ – LRTA* …



$s' = A$ $s = B$

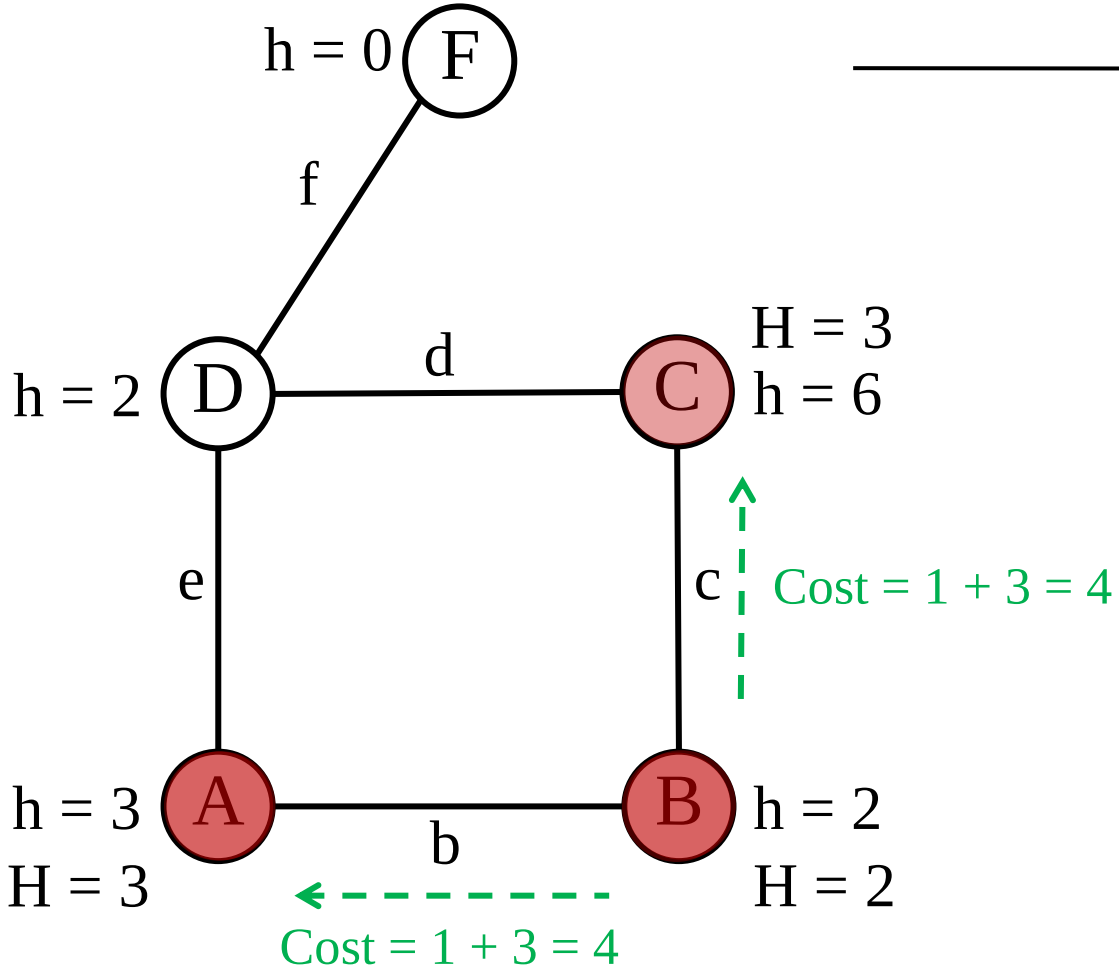
مثال ۱ – LRTA* …



$s' = A$ $s = B$

result(B, b) = A

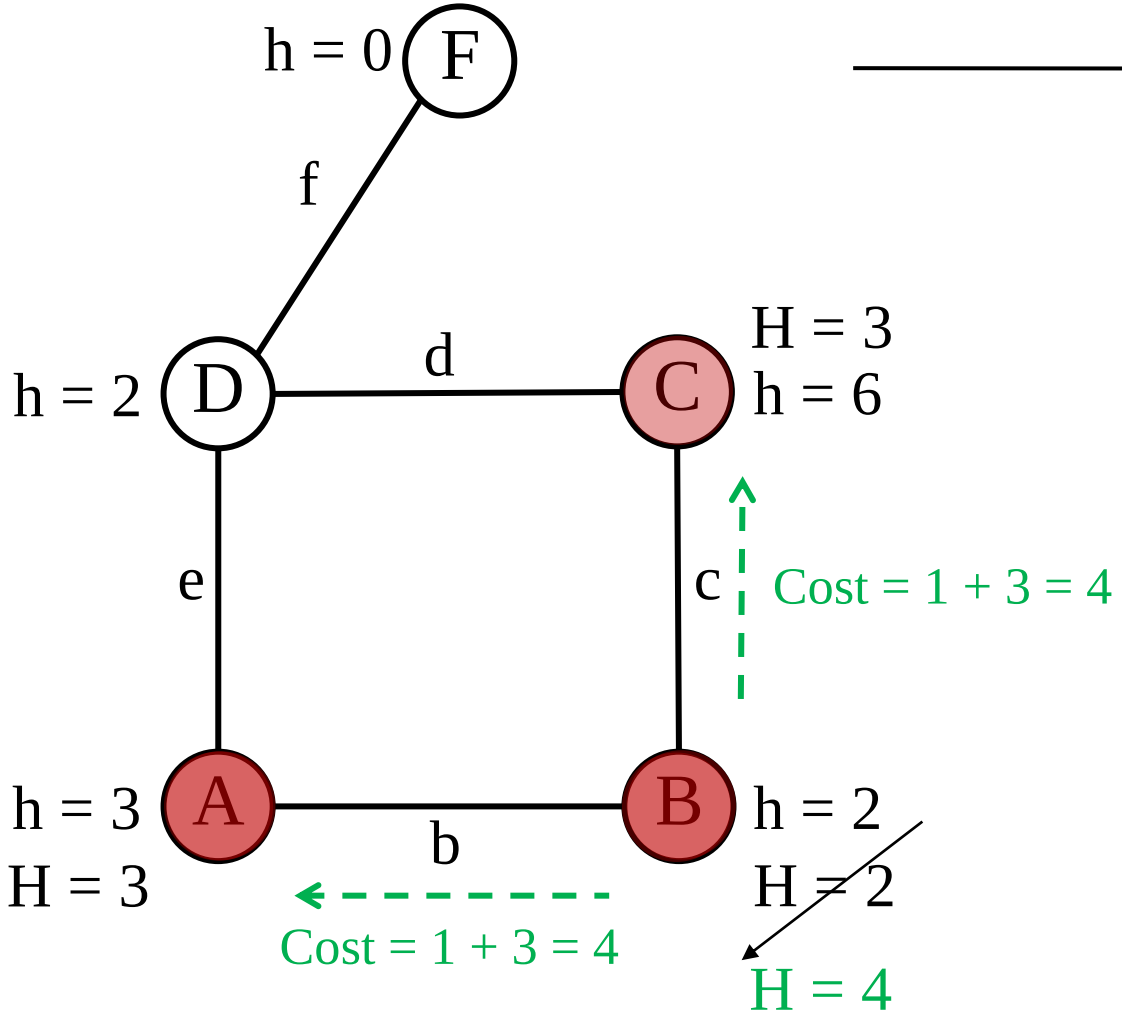
مثال ۱ – LRTA* ...



$s' = A$ $s = B$

result(B, b) = A

مثال ۱ – LRTA* ...

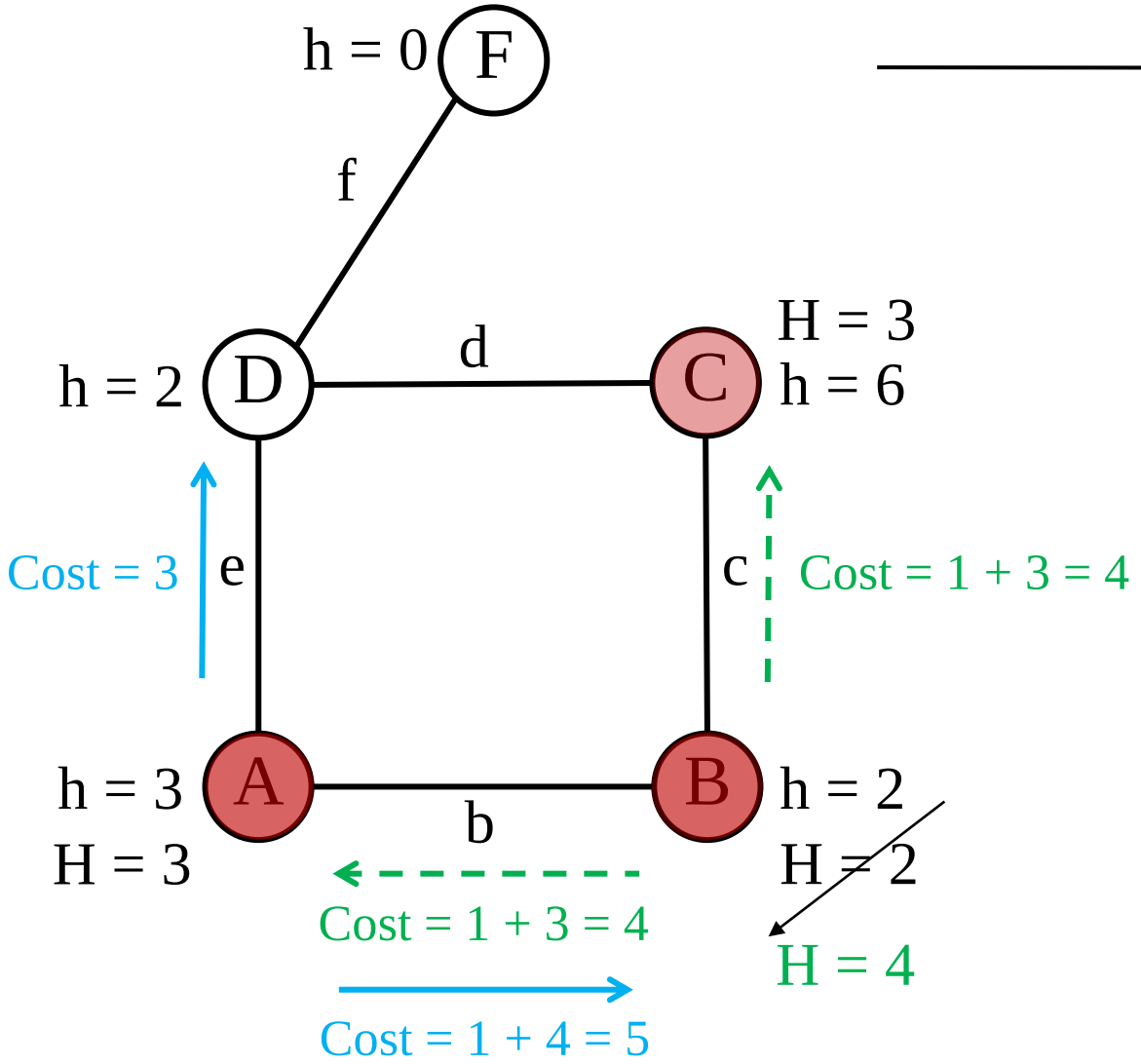


$s' = A$ $s = B$

result(B, b) = A

H(B) = 4

مثال ۱ – LRTA* …



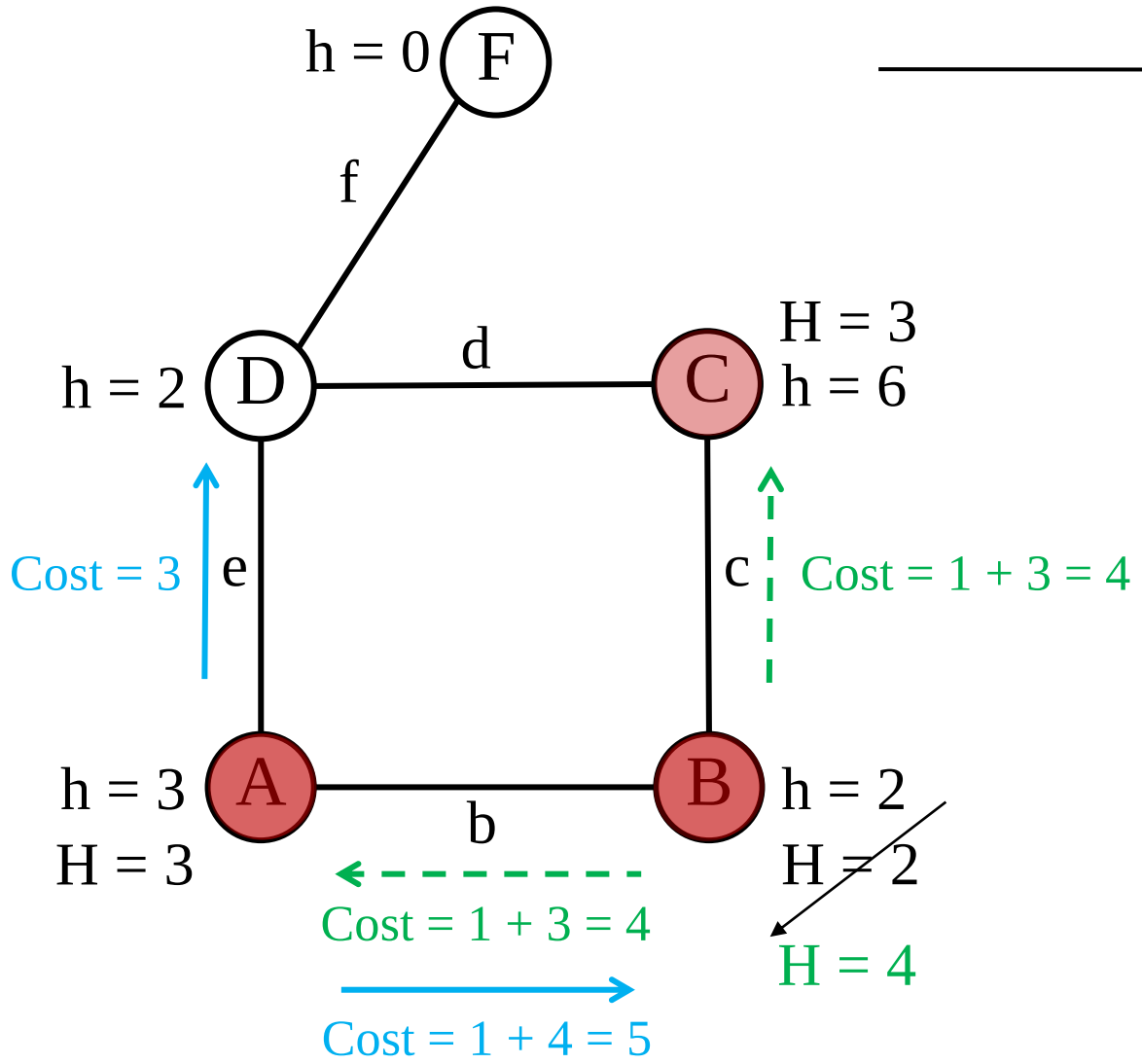
$s' = A$ $s = B$

result(B, b) = A

H(B) = 4

a = e

مثال ۱ – LRTA* …



$s' = A$ $s = B$

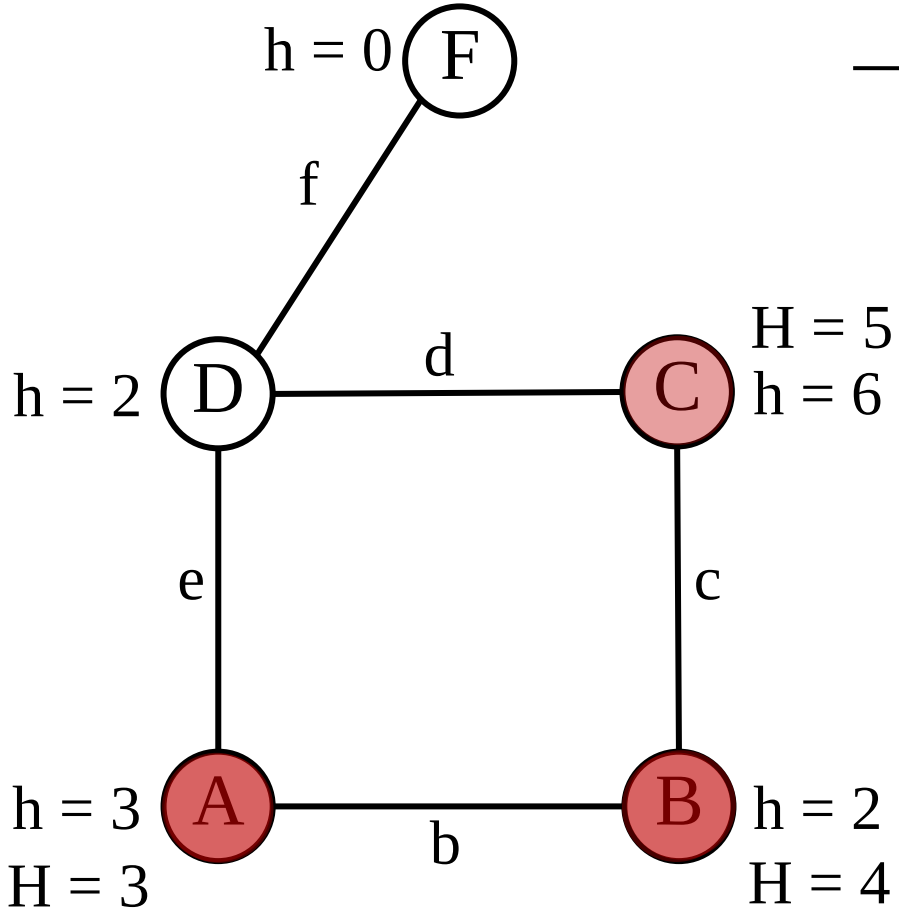
result(B, b) = A

H(B) = 4

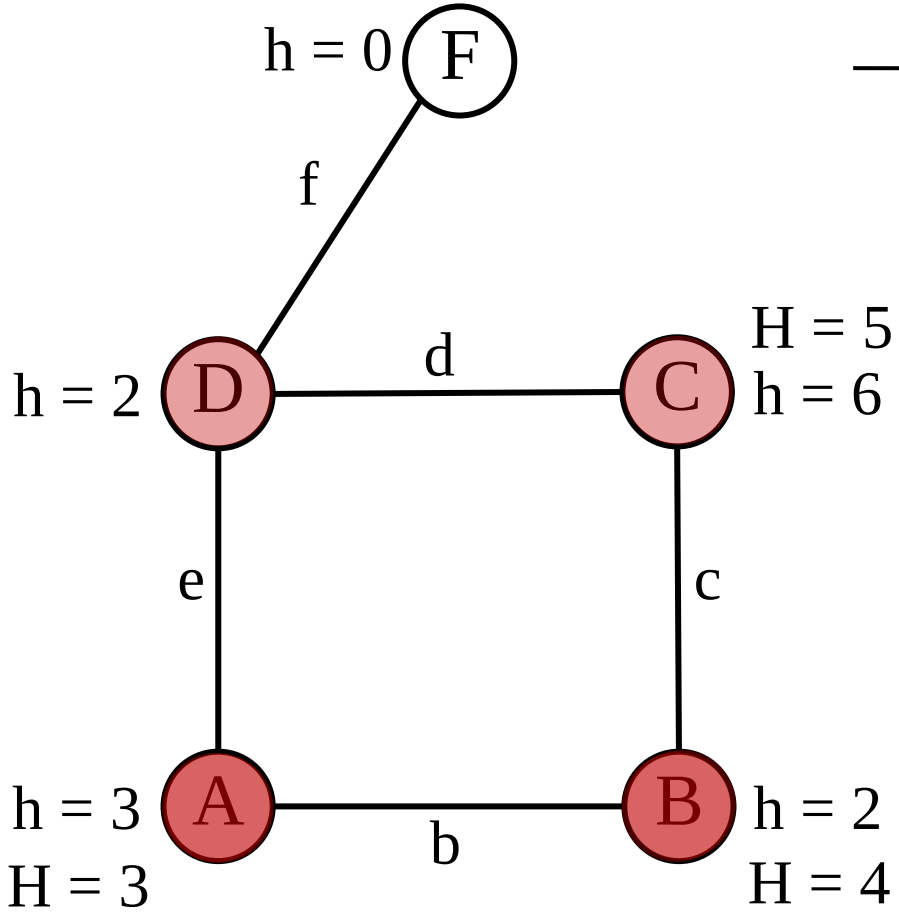
a = e

s = A

مثال ۱ – LRTA* …

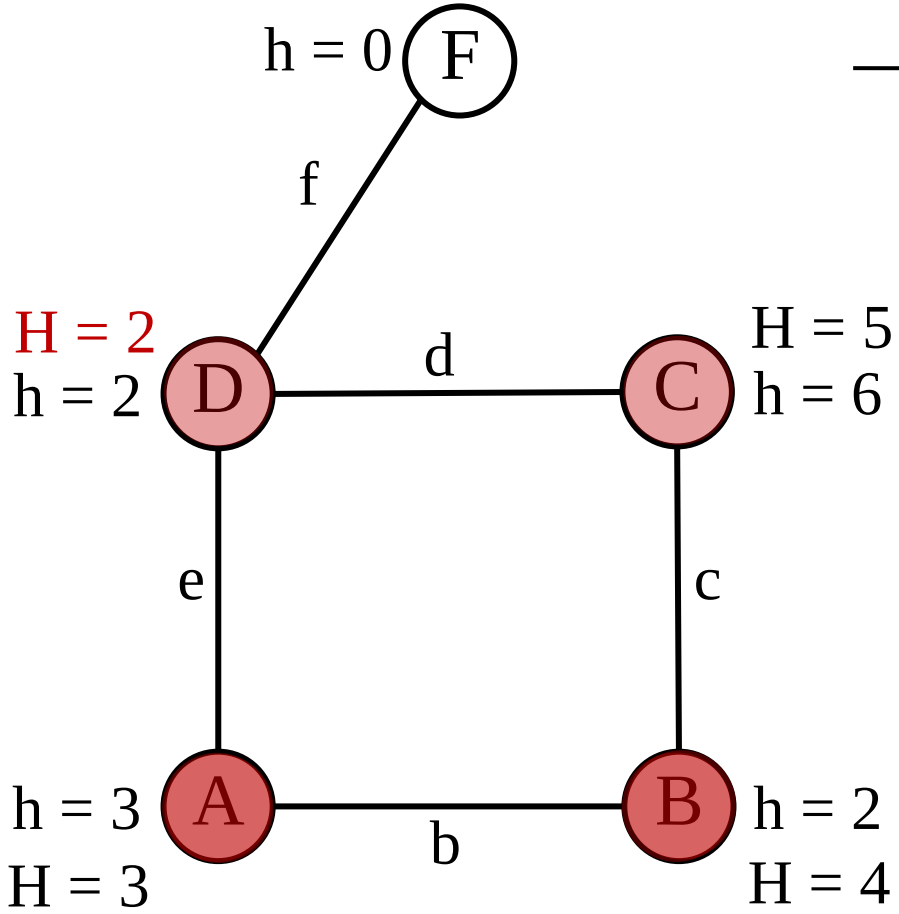


مثال ۱ – LRTA* …



$s' = D$ $s = A$

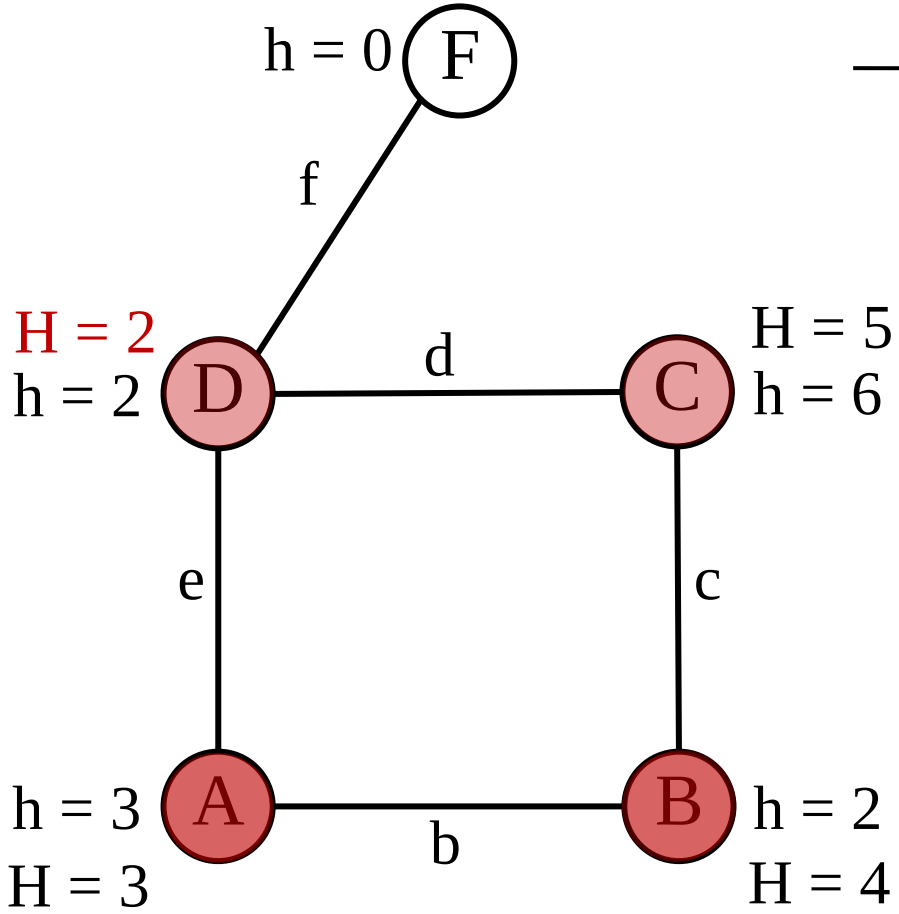
مثال ۱ – LRTA* …



$s' = D$ $s = A$

$H(D) = h(D) = 2$

مثال ۱ – LRTA* …

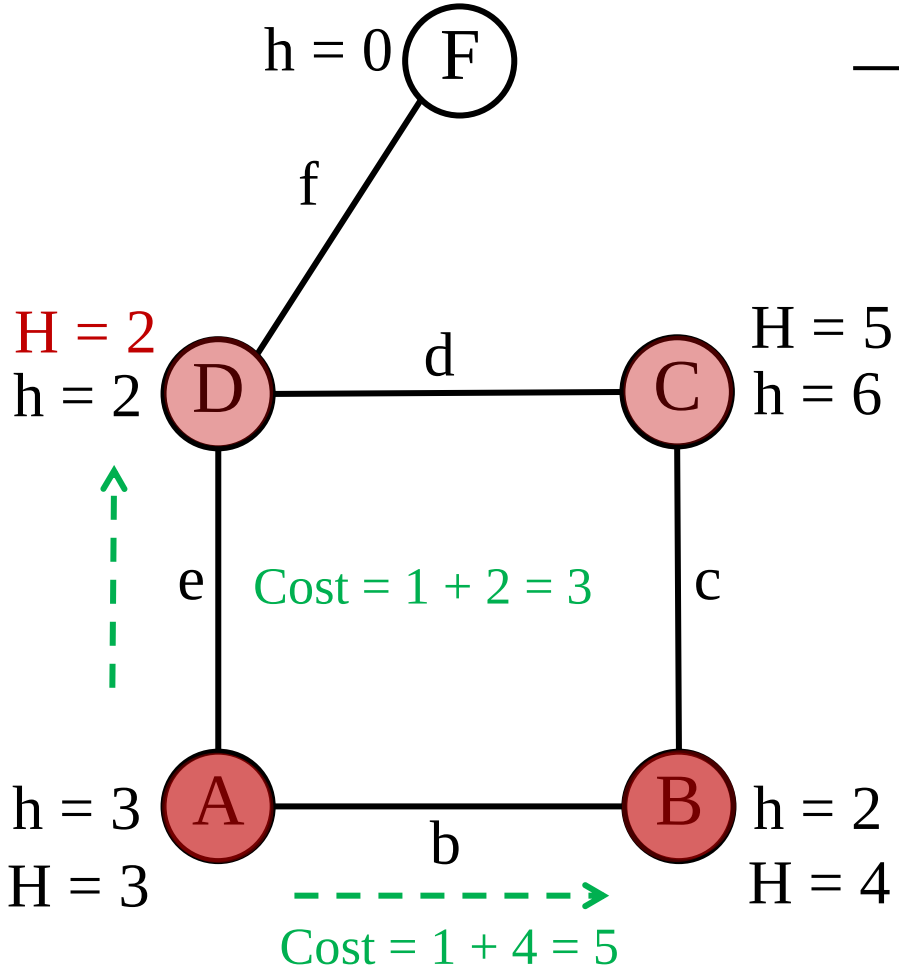


$s' = D$ $s = A$

$H(D) = h(D) = 2$

$result(A, e) = D$

مثال ۱ – LRTA* …

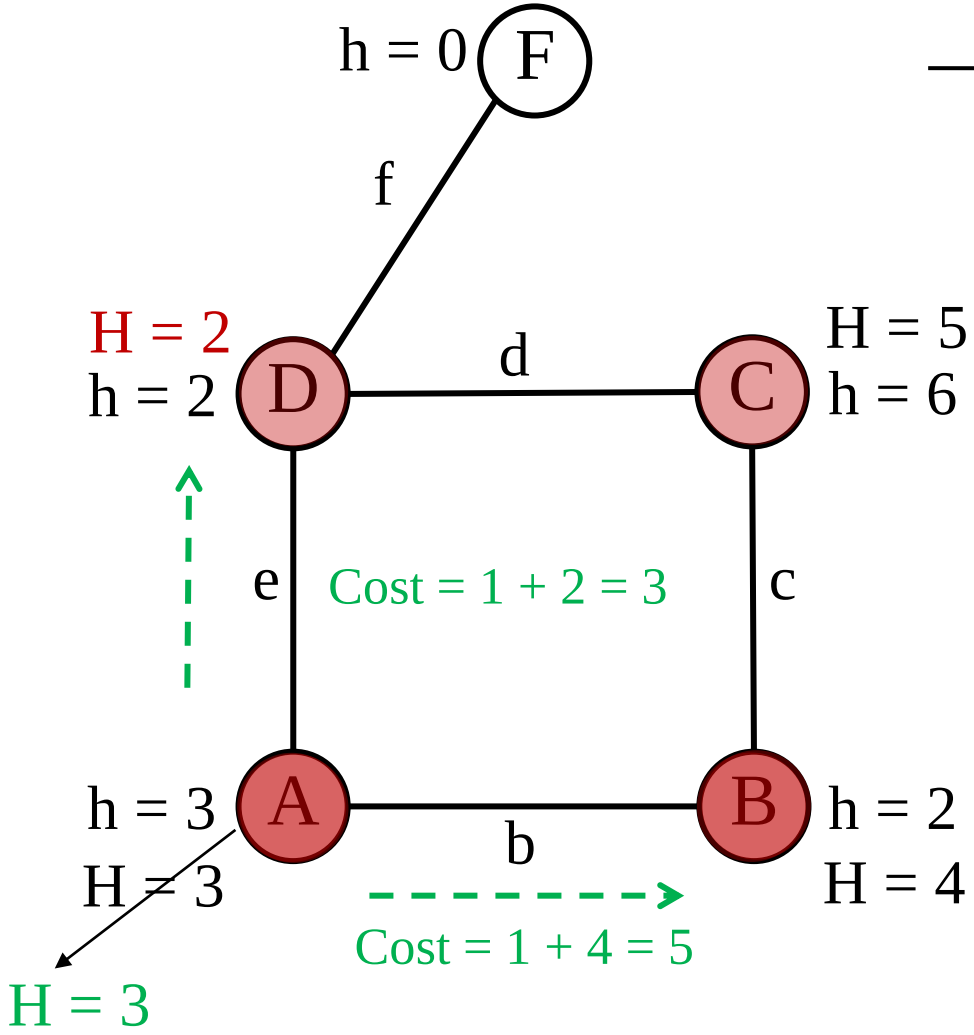


$s' = D$ $s = A$

$H(D) = h(D) = 2$

$result(A, e) = D$

مثال ۱ – LRTA* …



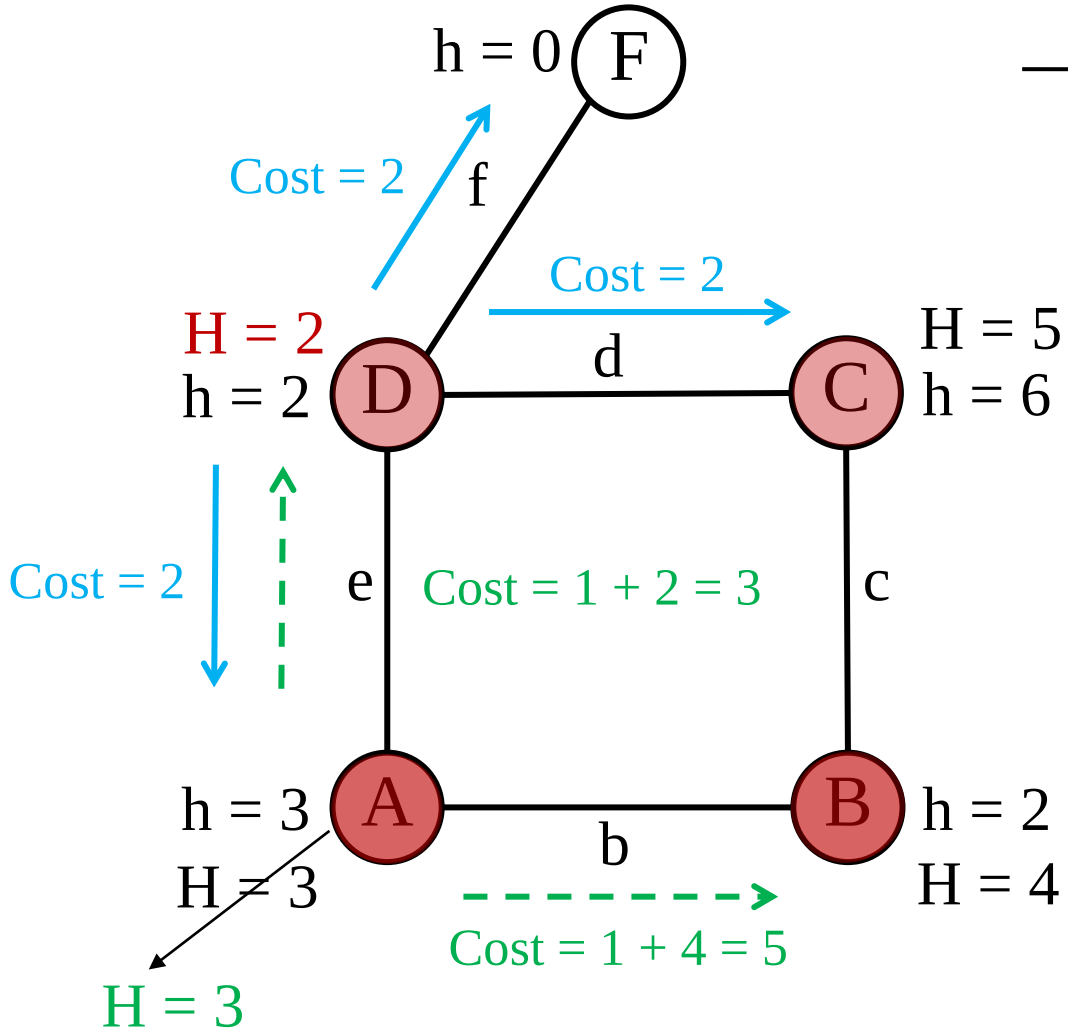
$s' = D$ $s = A$

$H(D) = h(D) = 2$

result(A, e) = D

$H(A) = 3$

مثال ۱ – LRTA* …



$s' = D$ $s = A$

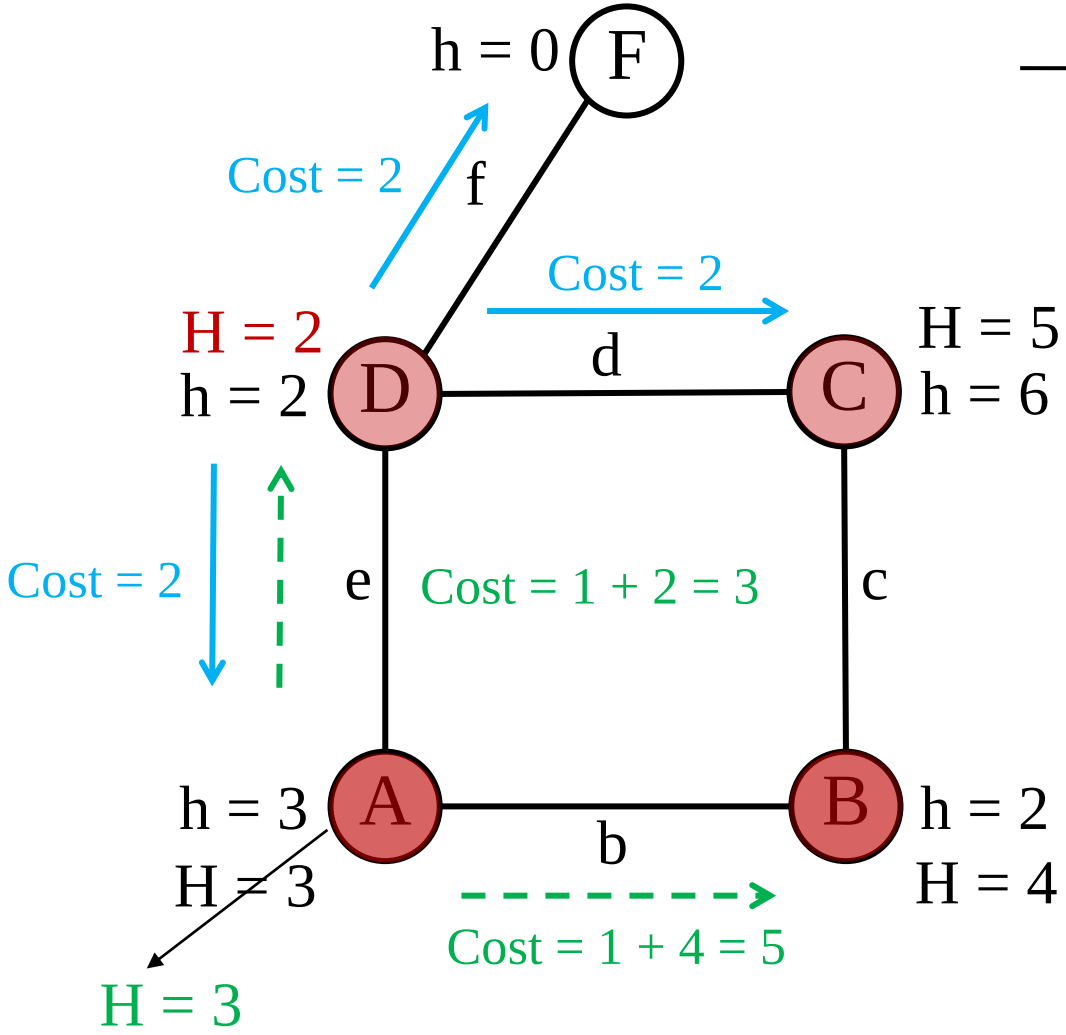
$H(D) = h(D) = 2$

result(A, e) = D

$H(A) = 3$

$a = f$

مثال ۱ – LRTA* …



$s' = D$ $s = A$

$H(D) = h(D) = 2$

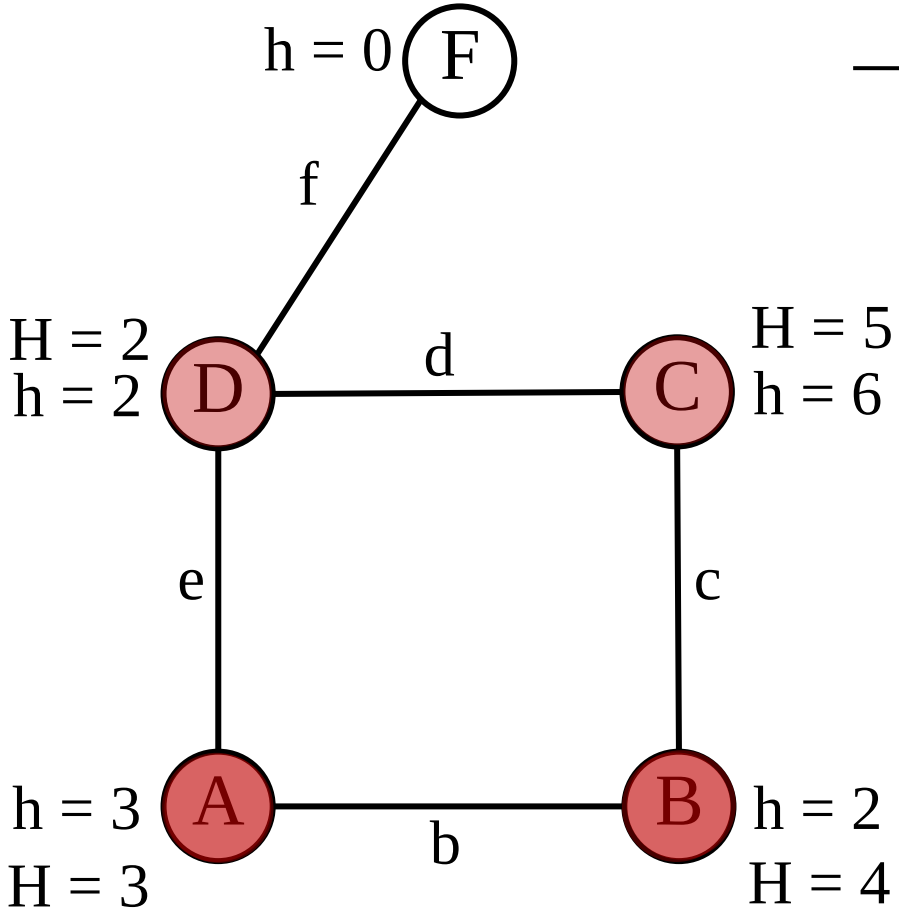
result(A, e) = D

$H(A) = 3$

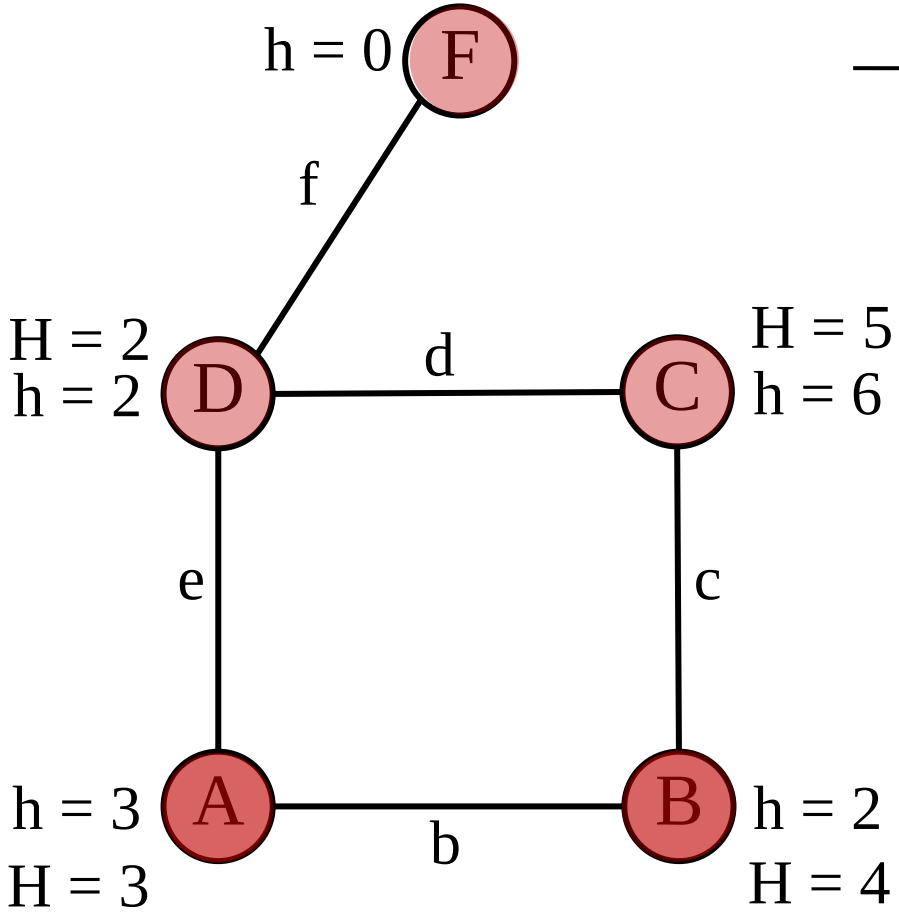
$a = f$

$s = D$

مثال ۱ – LRTA* …

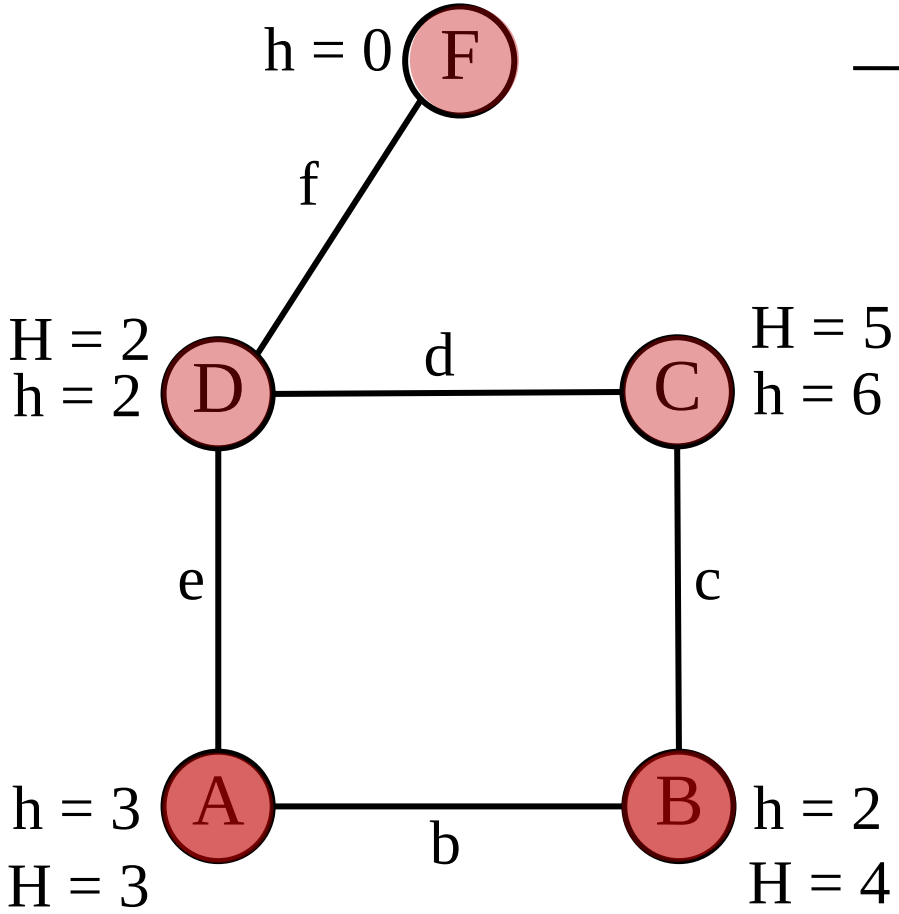


مثال ۱ – LRTA* …



$s' = F$ $s = D$

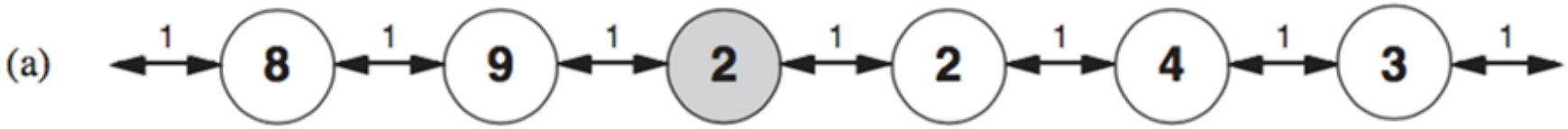
مثال ۱ – LRTA* …



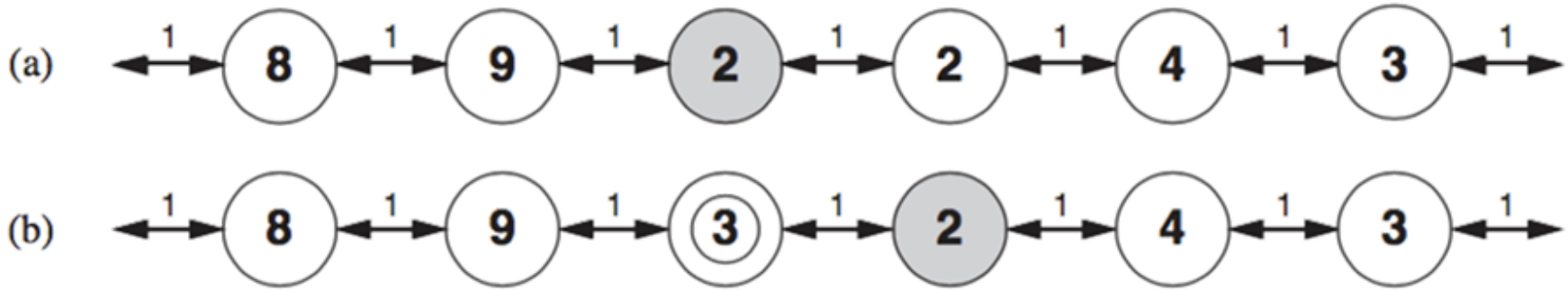
s‘ = F s = D

F is Goal return STOP

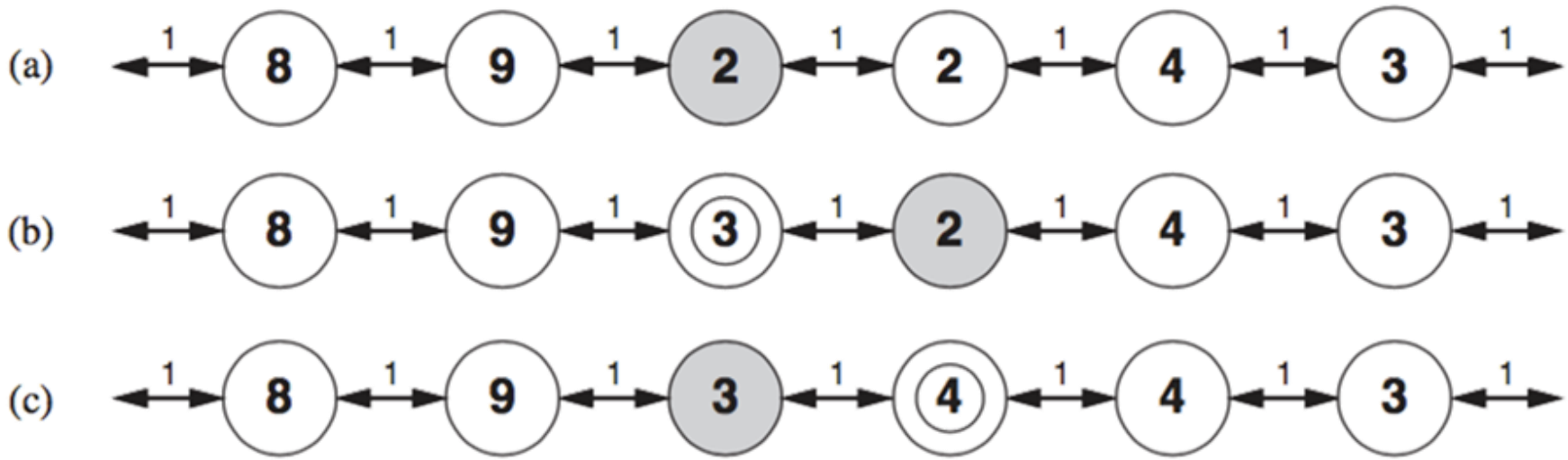
مثال ۲ – LRTA*



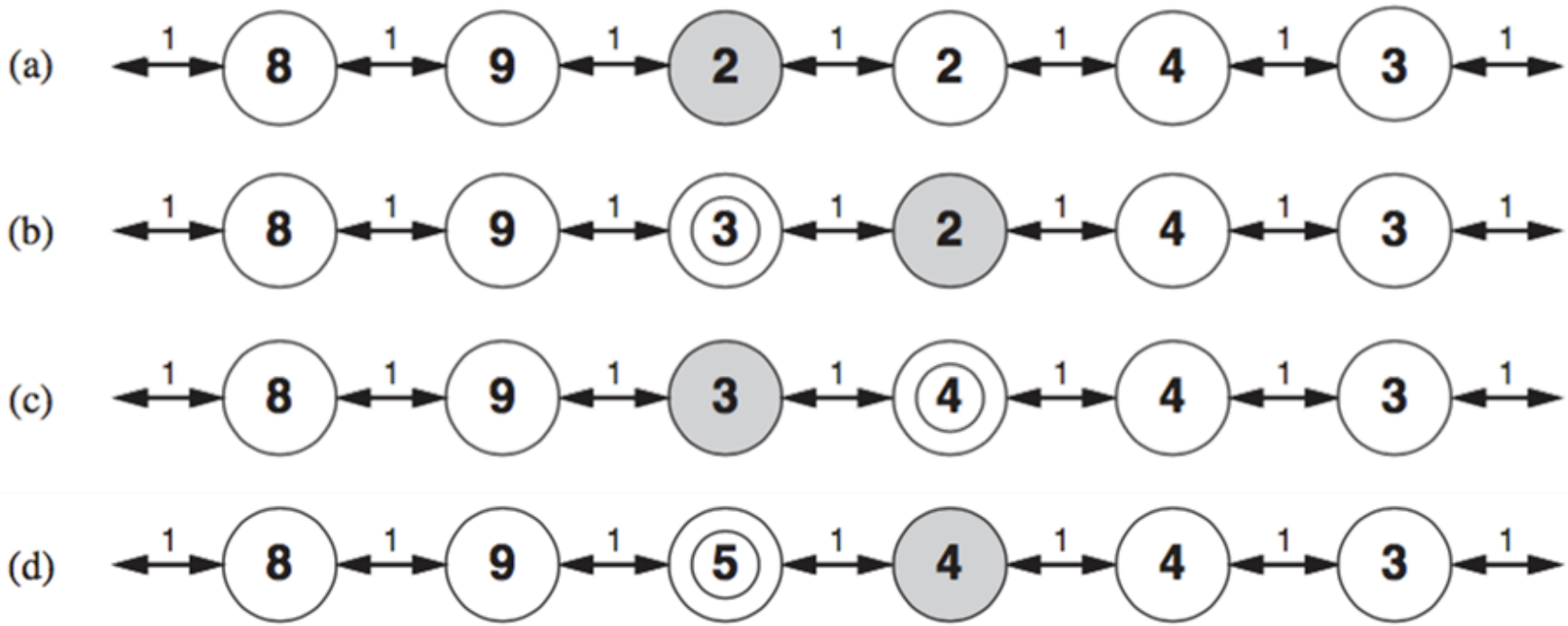
مثال ۲ – LRTA*



مثال ۲ – LRTA*



مثال ۲ – LRTA*



مثال ۲ – LRTA*