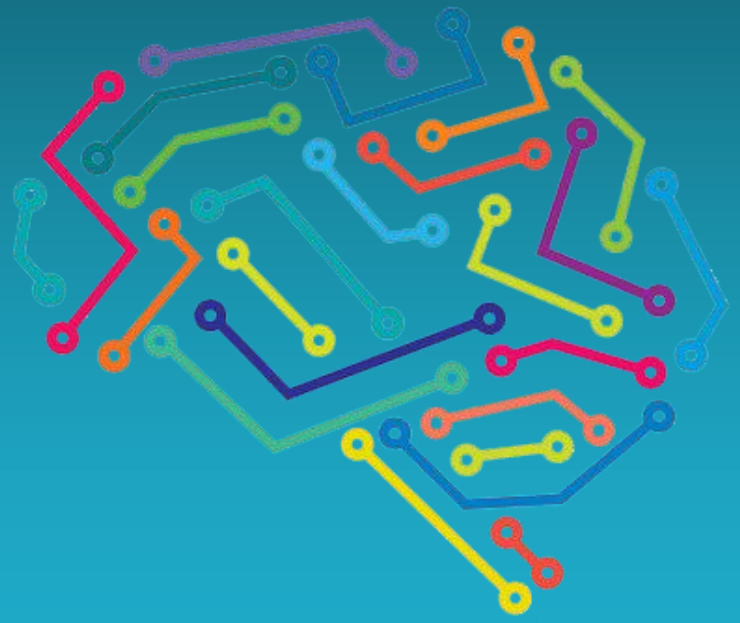
دانشگاه صنعتی شاهرود

حل مساله با جستجو



درس هوش مصنوعی

نیم‌سال اول تحصیلی 98-1397

عامل‌های حل مساله

- در این بخش نوعی از عامل‌های هدف‌گرا با عنوان عامل‌های حل مسئله (problem solving agents) مورد بحث قرار می‌گیرند.
- این عاملها باید قادر باشند در فضای حالت جستجو کرده و حالت هدف را بیابند:
  - ✓ جستجوی نامطلع (Uninformed):
    - الگوریتمهایی که هیچ اطلاعی جز تعریف مسئله از آن ندارند.
  - ✓ جستجوی مطلع (Informed):
    - الگوریتمهایی که اطلاع بیشتری از مسئله دارند و رنخهایی که به آنها نشان می‌دهد در چه ناحیه‌هایی به دنبال جواب بگردند.
- برای حل یک مسئله با الگوریتم‌های جستجو باید ابتدا مسئله را به صورت مناسب تعریف نمود.
- محیط کاملا قابل مشاهده، معین و گسسته است.
- همچنین فرض می‌کنیم، محیط شناخته‌شده (known) است.
  - ✓ یعنی نقشه راه را در اختیار داریم و می‌دانیم هر عمل یا فعالیتی به چه حالتی منتهی خواهد شد.

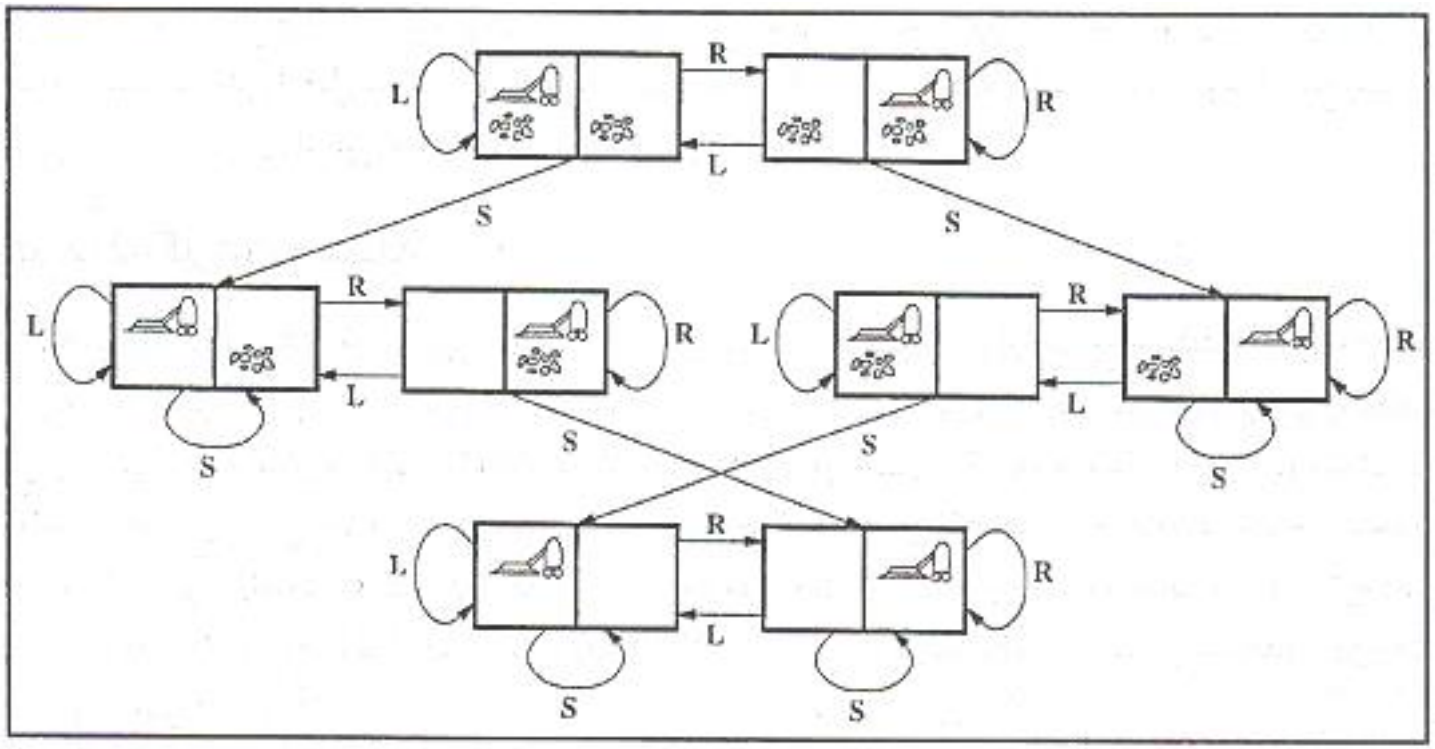
عامل های حل مساله

- تعریف مساله: شامل تعریف فضای حالت مسئله است.
- فضای حالت مساله با توجه به حالت اولیه و تابع مابعد قابل تعریف است:
  - ✓ فضای حالت، مجموعه‌ای از حالت هاست که از حالت اولیه می توان به آنها رسید.
  - ✓ به صورت گراف تصور می‌شود
    - حالت‌ها: گره‌های گراف
    - فعالیت‌ها: یال‌های گراف
    - مسیر فضای حالت: دنباله‌ای از حالت‌ها که در نتیجه دنباله‌ای از فعالیت‌ها طی می‌شوند (به هم متصل می شوند).
  - ✓ ۱) حالت اولیه عامل
  - ✓ ۲) توصیف فعالیت‌های ممکن عامل: استفاده از تابع مابعد
  - ✓ ۳) آزمون هدف: رسیدن به حالت هدف / یک خاصیت انتزاعی مشخص
  - ✓ ۴) تابع هزینه مسیر: برای مشخص کردن معیار کارایی
- راه حل مساله: مسیری از حالت اولیه به حالت هدف
- راه حل بهینه: مسیر با کمترین هزینه
- پس در حل هر مسئله، یک دنباله formulate, search, execute را خواهیم پیمود.

مسائل نمونه

- مسائل ؟؟؟ (toy problems):
  - ✓ دنیای جارو
  - ✓ پازل ۸ (8-Puzzle)
  - ✓ ۸ وزیر (8-Queens)
- مسائل دنیای واقعی:
  - ✓ مسیریابی:
    - مسیریابی در شبکه‌های رایانه‌ای
    - برنامه‌ریزی عملیات نظامی
    - سیستم‌های برنامه‌ریزی مسافرت هوایی

دنیای جارو

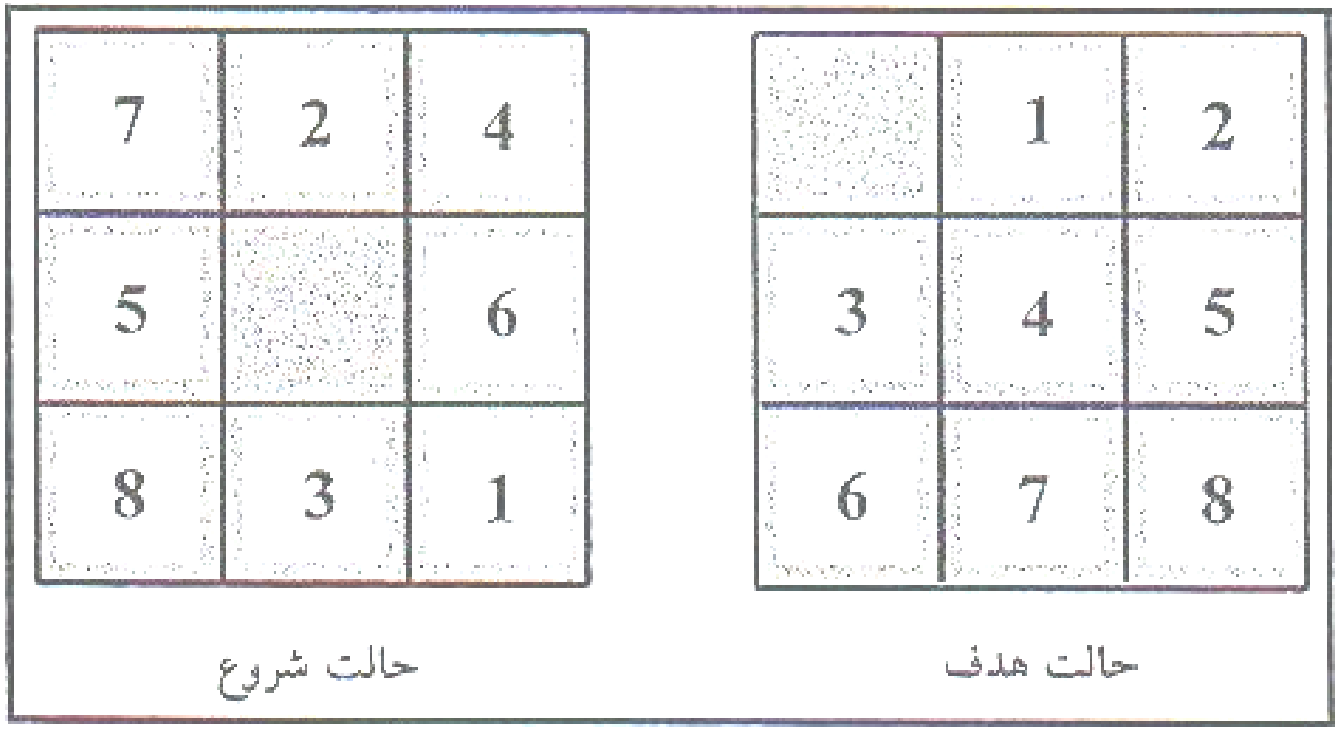


شکل ۳-۳ فضای حالت جهان جارو. یال‌ها فعالیت‌ها را نشان می‌دهند. L = Left، R = Right و S = Suck.

دنیای جارو

- حالتها
  - ✓ عامل در یکی از دو مکان است که هرکدام ممکن است کثیف باشند یا نباشند: ۸ حالت
- حالت اولیه
  - ✓ هر حالتی می‌تواند به عنوان حالت اولیه طراحی شود
- تابع مابعد
  - ✓ حالت‌های معتبری را تولید می‌کند که از ۳ عملیات (Left, Right, Suck) ناشی می‌شود.
- آزمون هدف
  - ✓ تمیز بودن تمام مربع‌ها
- هزینه مسیر
  - ✓ هزینه هر مرحله ۱ است.
  - ✓ هزینه مسیر برابر تعداد مراحل در مسیر است.

پازل (معمای) ۸



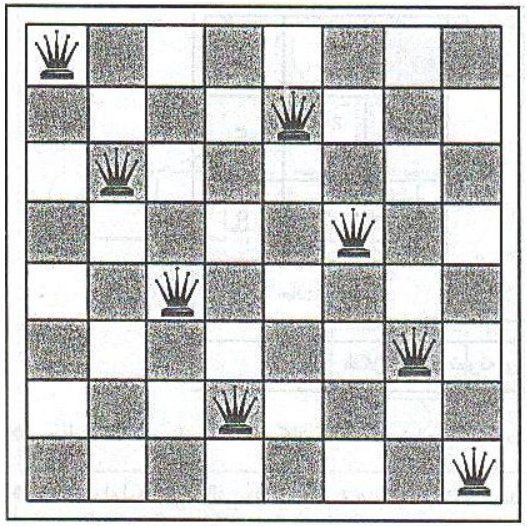
شکل ۴-۳ نمونه‌ای از معمای ۸.

پازل (معمای) ۸

- حالت‌ها:
  - ✓ مکان هر ۸ خانه شماره‌دار
  - ✓ و خانه خالی را در یکی از ۹ خانه مشخص می‌کند.
- حالت اولیه:
  - ✓ هر حالتی می‌تواند به عنوان حالت اولیه طراحی شود.
- تابع جانشین:
  - ✓ حالت‌های معتبری را تولید می‌کند که از چهار عمل به دست می‌آید (انتقال خانه خالی).
- آزمون هدف:
  - ✓ رسیدن به حالت هدف
- هزینه مسیر:
  - ✓ هزینه هر مرحله ۱ است. هزینه مسیر برابر تعداد مراحل در مسیر است.

۸ وزیر

- ۸ وزیر در صفحه شطرنج طوری قرار گیرند که هر وزیری وزیر دیگر را گارد ندهد

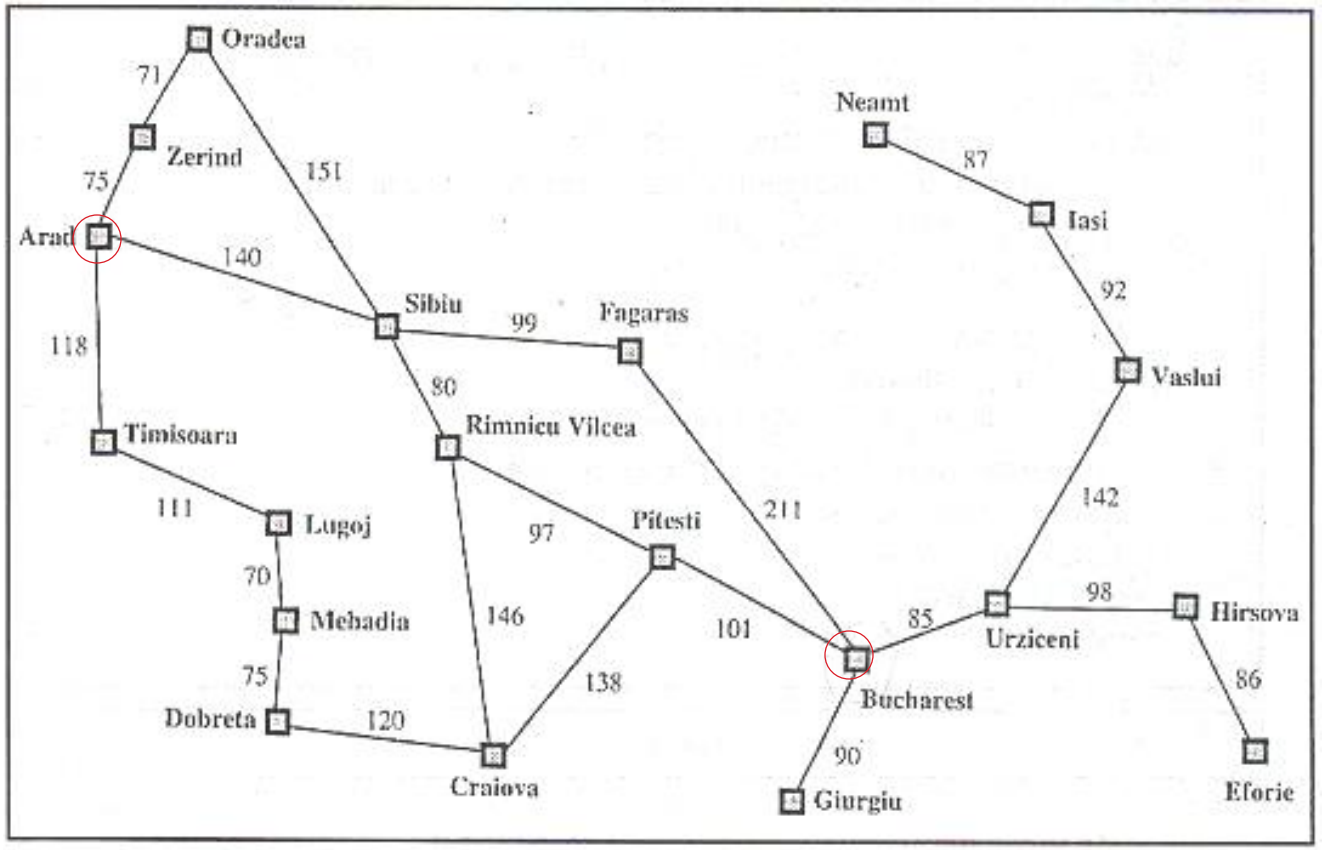


شکل ۵-۳ راه‌حل تقریبی مسئله n وزیر.

۸ وزیر

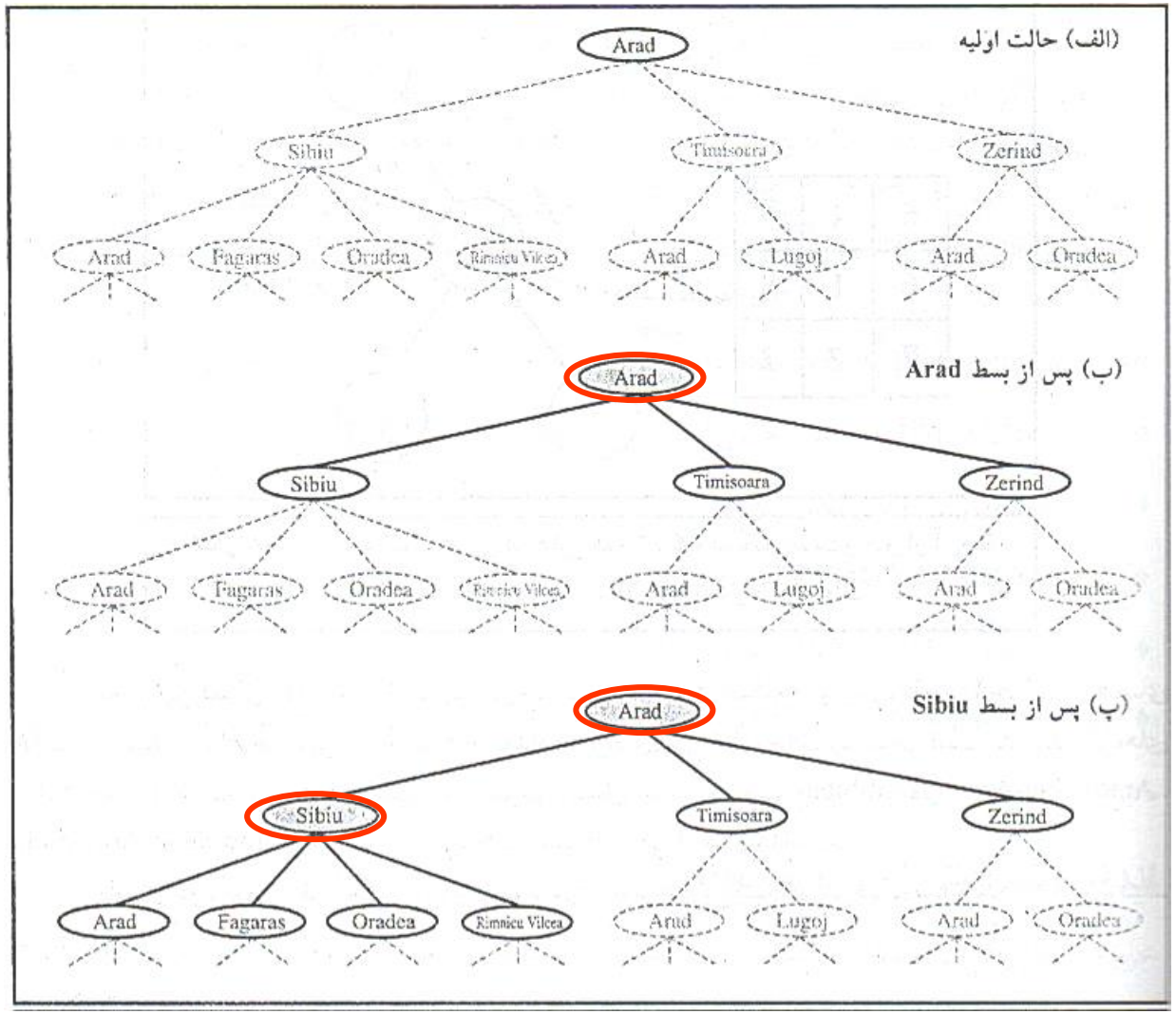
- فرمول‌بندی:
  - ۱- افزایشی
  - ۲- حالت کامل
- حالت‌ها:
  - هر ترتیبی از ۰ تا ۸ وزیر در صفحه
- حالت اولیه:
  - هیچ وزیری در صفحه نیست
- تابع مابعد:
  - وزیری را به خانه خالی اضافه می‌کند
- آزمون هدف
  - ۸ وزیر در صفحه وجود دارند و هیچکدام به دیگری گارد نمی‌دهند

مسیریابی



شکل ۲-۳ نقشه ساده شده‌ای از جاده‌های رومانی.

جستجو برای راه حل ها



الگوریتم 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
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
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
```

ارزیابی الگوریتم حل مساله

- معیار ارزیابی الگوریتم حل مساله
  - ✓ کامل بودن: تضمین الگوریتم به پیدا کردن راه حل
  - ✓ بهینگی: ارائه راه حل بهینه
  - ✓ پیچیدگی زمانی: زمان لازم برای پیدا کردن راه حل
  - ✓ پیچیدگی فضا: حافظه مورد نیاز برای جستجو

ارزیابی الگوریتم حل مساله

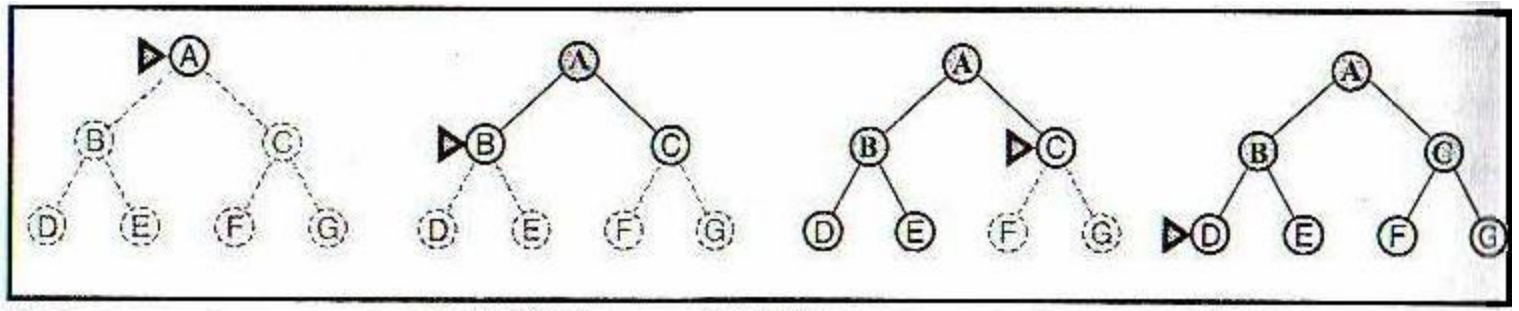
- پیچیدگی مسئله بر حسب سه کمیت بیان می‌شود
  - ✓ فاکتور انشعاب: b
  - ✓ عمق نزدیکترین گره هدف: d
  - ✓ حداکثر عمق ممکن برای نودها در فضای حالت: m
- سنجش زمان بر حسب تعداد گره‌های تولیدشده در حین جستجو
- سنجش فضا بر حسب حداکثر گره‌های ذخیره‌شده در حافظه

جستجوی ناآگاهانه

- جستجوی نا آگاهانه (جستجوی کور)
  - ✓ هیچ اطلاعاتی غیر از تعریف مساله در اختیار الگوریتم نیست.
  - ✓ تولید نود بعدی
  - ✓ تشخیص حالت هدف و غیر هدف
- جستجوی آگاهانه (جستجوی اکتشافی)
  - ✓ توانایی مقایسه میزان امیدبخش بودن حالت غیرهدف نسبت به حالت غیرهدف دیگر

جستجوی اول‌سطح (Breadth First)

- ابتدا ریشه بسط داده می‌شود
- سپس تمام جانشین‌های ریشه بسط داده می‌شوند و به همین ترتیب
- یعنی بسط تمام گره‌های موجود در یک عمق درخت و سپس حرکت در عمق بعدی



شکل ۱۰-۳ جست‌وجوی عرضی در یک درخت دودویی. در هر مرحله، گره‌ای که بعداً باید بسط داده شود، علامت‌دار شده است.

الگوریتم BFS

```
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)
```

Figure 3.11 Breadth-first search on a graph.

- برای هر نود فرزند که در نتیجه بسط نود فعلی تولید می‌گردد، بررسی می‌شود که آیا نود هدف هست یا نه و اگر نبود، در صف قرار می‌گیرد.
- به همین دلیل هر نودی که در صف یا مجموعه مشاهده‌شده‌ها قرار بگیرد، قبلا آزمون هدف را گذرانده است.

مثال

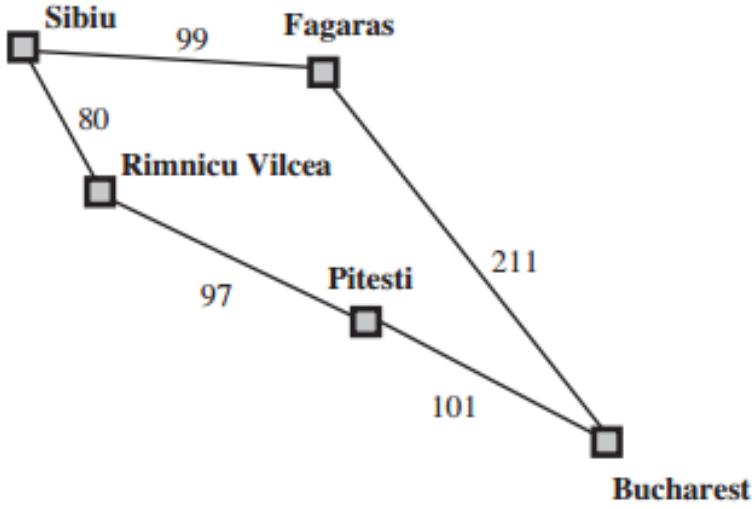


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

مثال

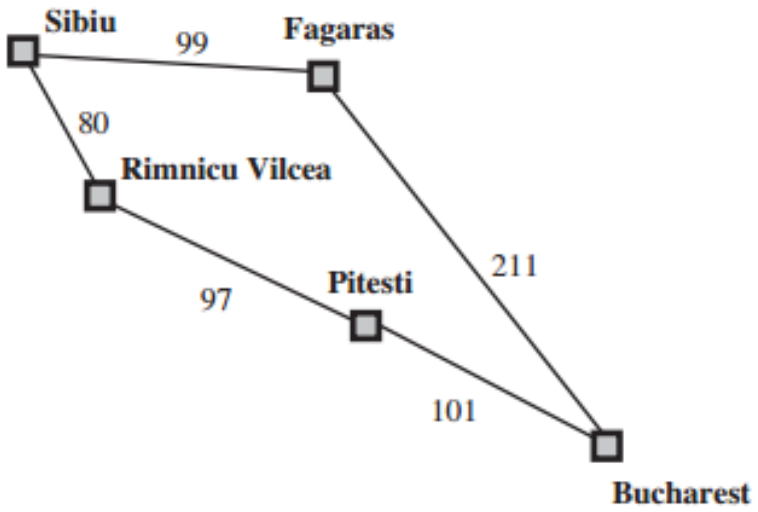


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

(S) → (F, R) → (P, F) → return SFB

پیچیدگی BFS

- فرض کنید هر گره دارای b گره مابعد است:
  - ✓ b
  - ✓ b^2
  - ✓ b^3
  - ✓ ... تا سطح d، b^d:
- پیچیدگی زمانی این الگوریتم $O(b^d)$ می‌باشد (در بدترین حالت آخرین نود تولیدشده در سطح d جواب است و بنابراین تمام گره‌های سطح d تولید می‌شوند)

$$b + b^2 + b^3 + \cdots + b^{d+1} = b\sum_{i=0}^{d-1} b^i = b\frac{b^d - 1}{b - 1} = O(b^d)$$

- سوال اگر در زمان بسط (خروج از صف) آزمون هدف انجام می‌شد چطور؟

پیچیدگی BFS

- سوال اگر در زمان بسط (خروج از صف) آزمون هدف انجام می‌شد چطور؟
  - ✓ در این صورت، برای بدترین حالت فرض‌شده در اسلاید قبل، تمامی نودهای سطح $d+1$ هم تولید می‌شدند پس مرتبه زمانی $O(b^{d+1})$ می‌شد.

پیچیدگی فضایی

- پیچیدگی فضایی این الگوریتم نیز برای جستجوی گرافی $O(b^d)$ می‌باشد.
- چرا؟
  - ✓ در عمق d در مجموعه frontier حداکثر چند نود موجود است؟
  - ✓ در مجموعه explored چند نود داریم؟

پیچیدگی فضایی

- پیچیدگی فضایی این الگوریتم نیز برای جستجوی گرافی $O(b^d)$ می‌باشد.
- چرا؟
  - ✓ در عمق d در مجموعه frontier حداکثر چند نود موجود است؟ $O(b^d)$
  - ✓ در مجموعه explored چند نود داریم؟ $O(b^{d-1})$
- برای جستجوی درختی با توجه به حذف مجموعه explored آیا پیچیدگی کمتر می‌شود؟

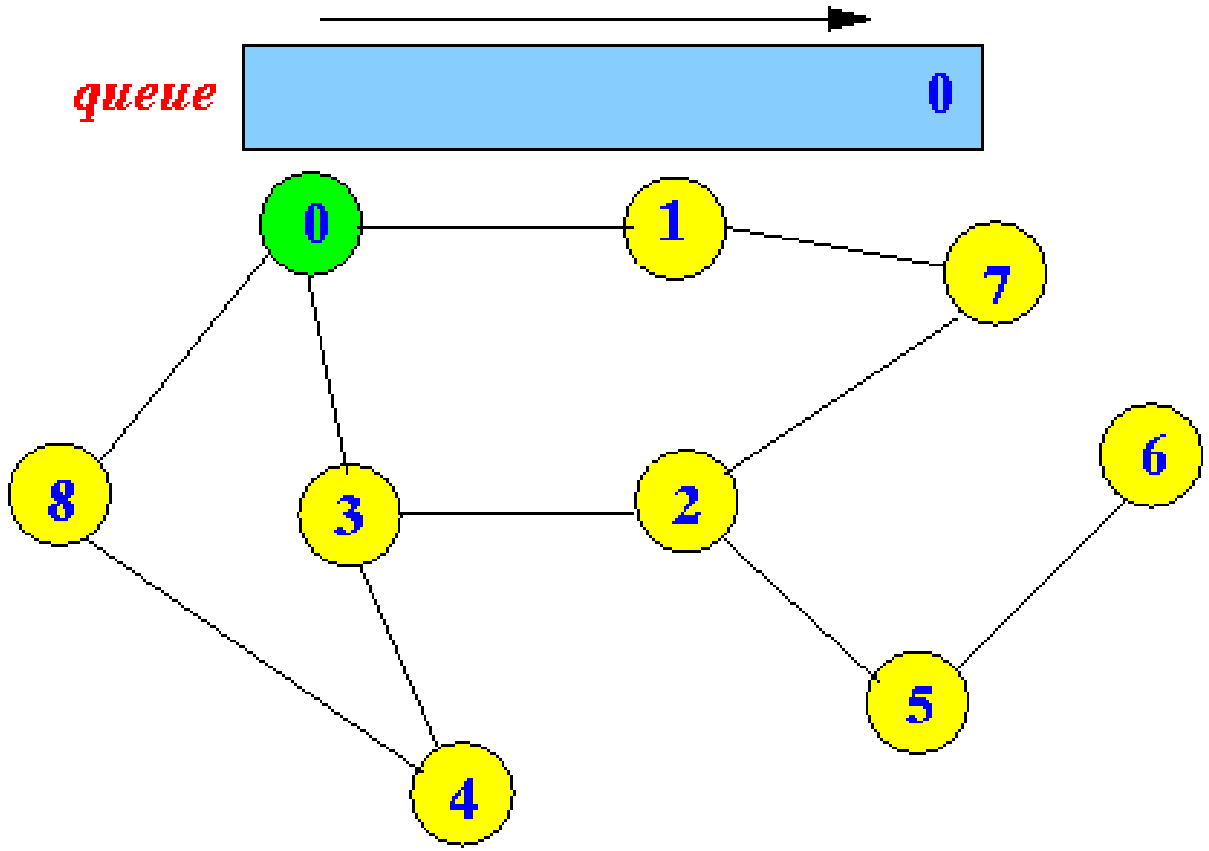
پیچیدگی BFS

- با فرض $b = 10$، و اینکه ذخیره هر نود نیازمند ۱۰۰۰ بایت است، برای سیستمی با توانایی تولید یک میلیون نود درثانیه:

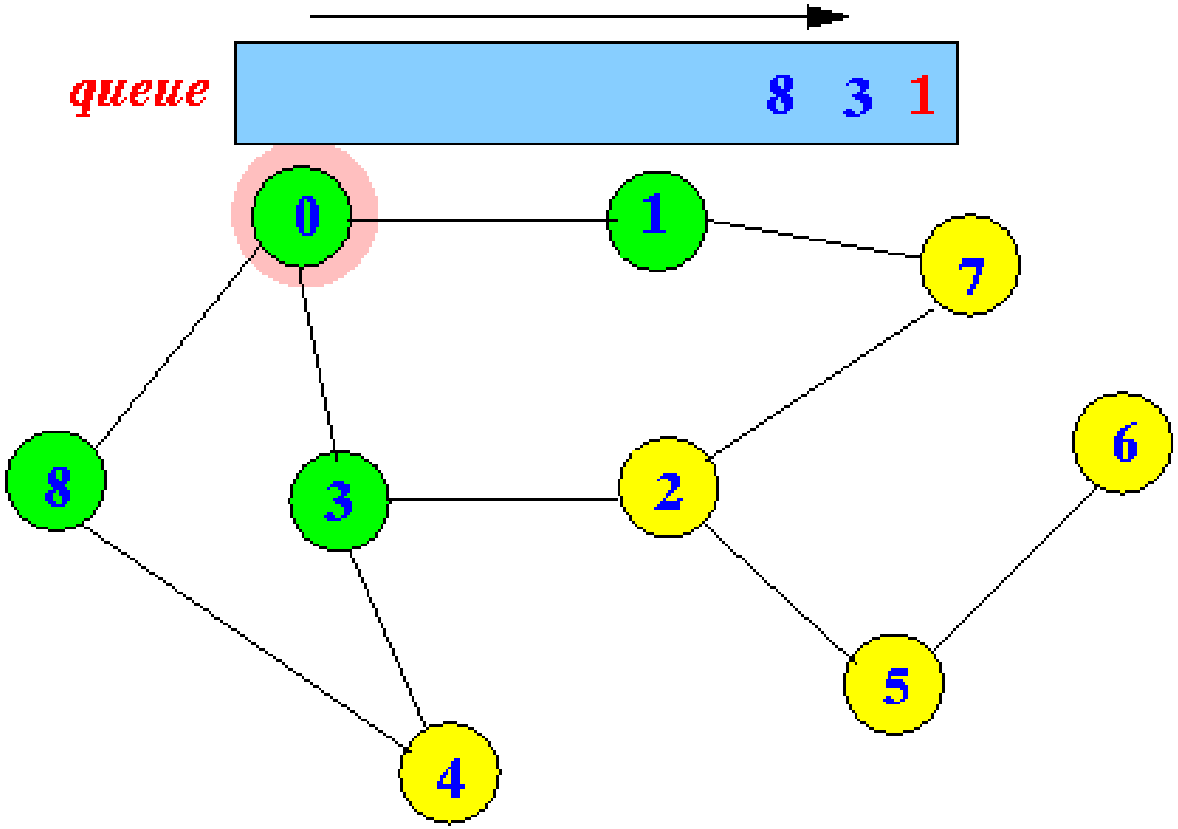
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

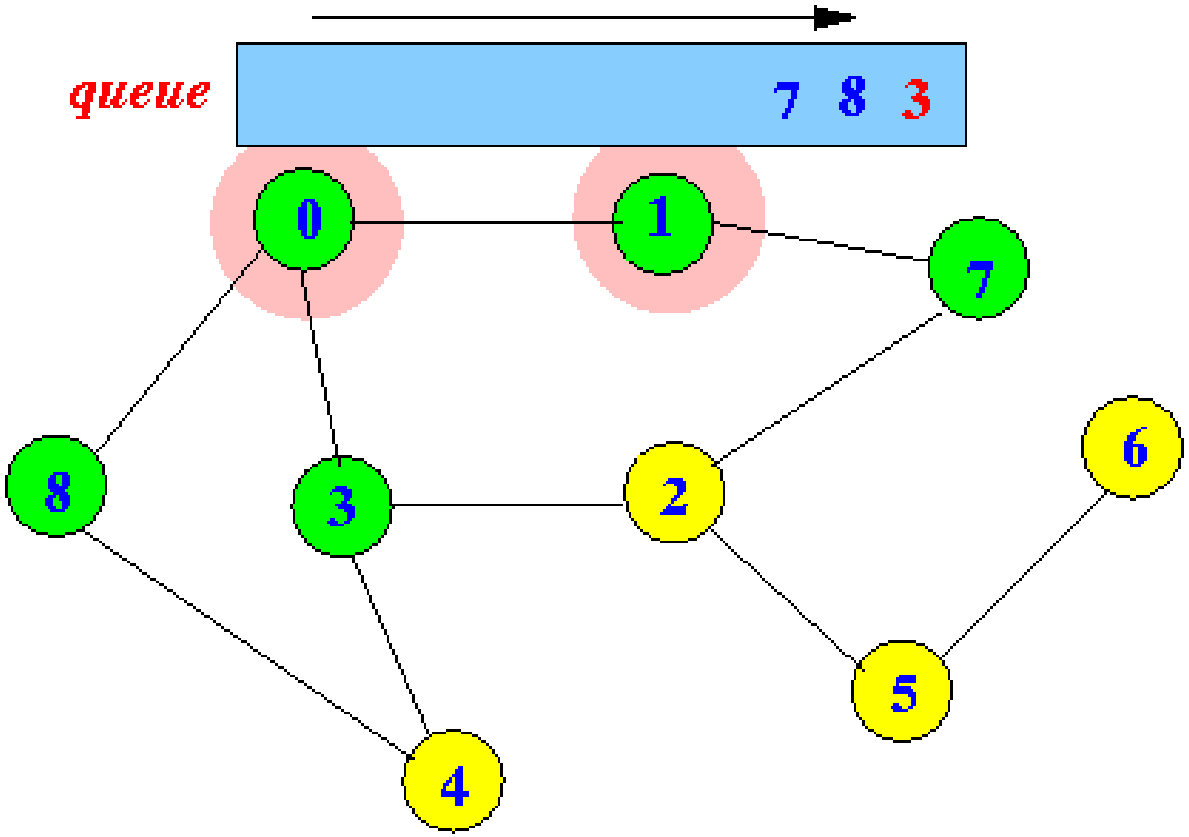
BFS



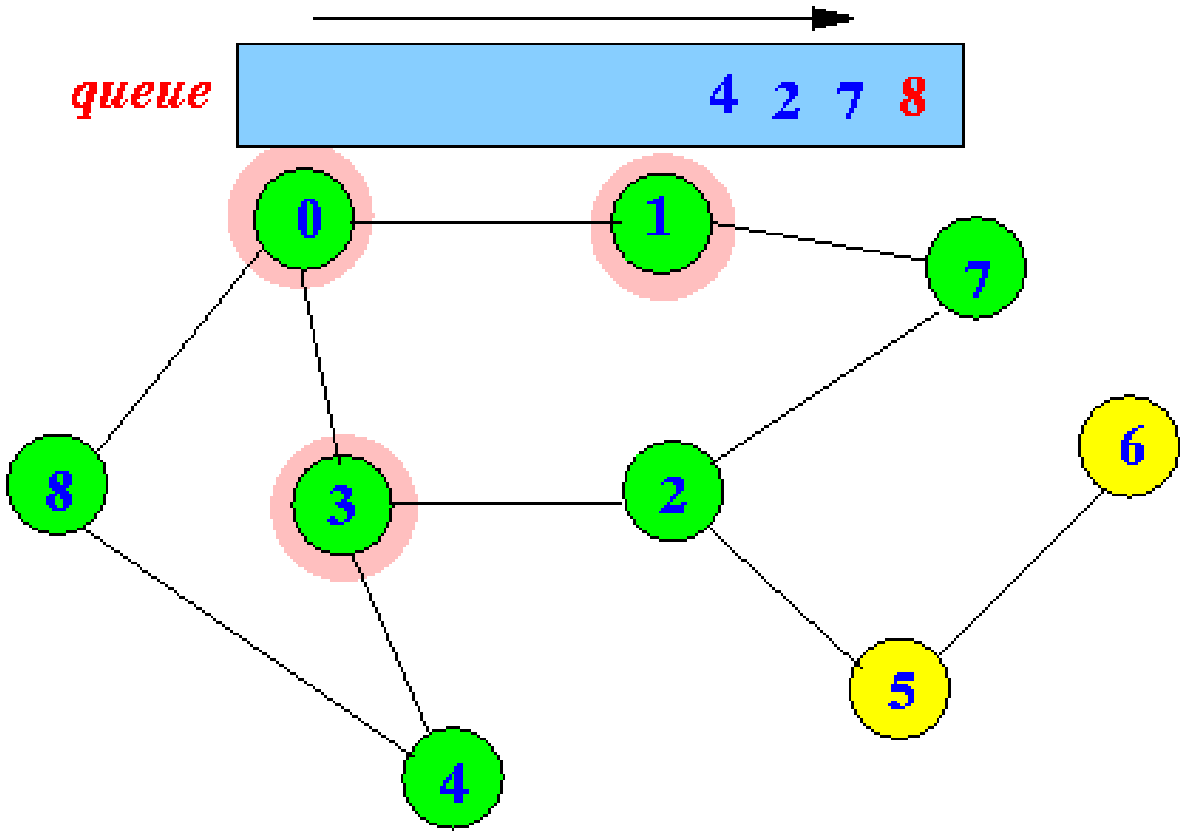
BFS



BFS



BFS



ادامه تحلیل BFS

- پیچیدگی زمانی (بررسی شد)
- پیچیدگی فضایی(بررسی شد)
- کامل است: یعنی اگر جوابی باشد، بالاخره به آن می‌رسد.
- بهینه است اگر:
  - ✓ هزینه مسیر یک تابع غیرنزولی از عمق نود باشد.
  - ✓ مثال رایج: هزینه همه اعمال (یالها) یکسان باشد
- با کمی تغییر می‌تواند همیشه بهینه باشد:
  - ✓ Uniform-cost BFS

Uniform-cost BFS

- همان جستجوی اول-سطح یا عرضی است که به جای بسط سطحی‌ترین گره، گرهی با کمترین هزینه مسیر را بسط می‌دهد.
- اهمیت در کل هزینه مراحل است نه فقط تعداد مراحل

جستجوی هزینه یکنواخت

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

تفاوت‌های BFS و UC-BFS

- سه تفاوت عمده دارند:
  - ✓ بجای کم‌عمق‌ترین نود، نودی با کمترین هزینه مسیر را بسط می‌دهیم.
  - ✓ آزمون هدف در زمان انتخاب برای بسط (خروج از صف) انجام می‌شود.
  - ✓ یک تست دیگر به الگوریتم اضافه شده است: اگر نود تولیدشده فعلی، در صف وجود دارد و هزینه مسیر برای آن بیشتر است، آن نود با مقدار فعلی اصلاح می‌شود (یعنی نسخه قبلی نود، جایگزین می‌شود).

مثال

- دقت کنید که در مرحله‌ای که با رنگ قرمز مشخص شده، گره هدف تولید شده است، ولی از آنجا که آزمون هدف را در این محله انجام نمی‌دهیم جستجو ادامه می‌یابد تا زمانی که نود هدف از صف خارج شود.
- دلیل آن این است که می‌خواهیم مطمئن باشیم، سایر مسیرها به این نود که ممکن است بهتر از مسیر فعلی باشند هم بررسی شده‌اند. یا به عبارتی نود در مسیر suboptimal قرار ندارد.

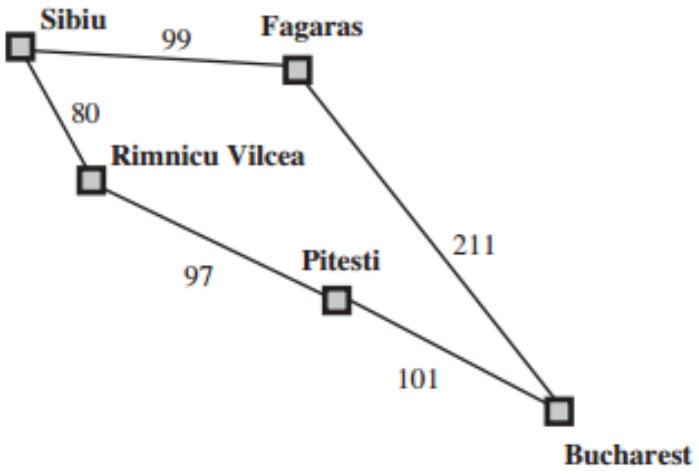


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

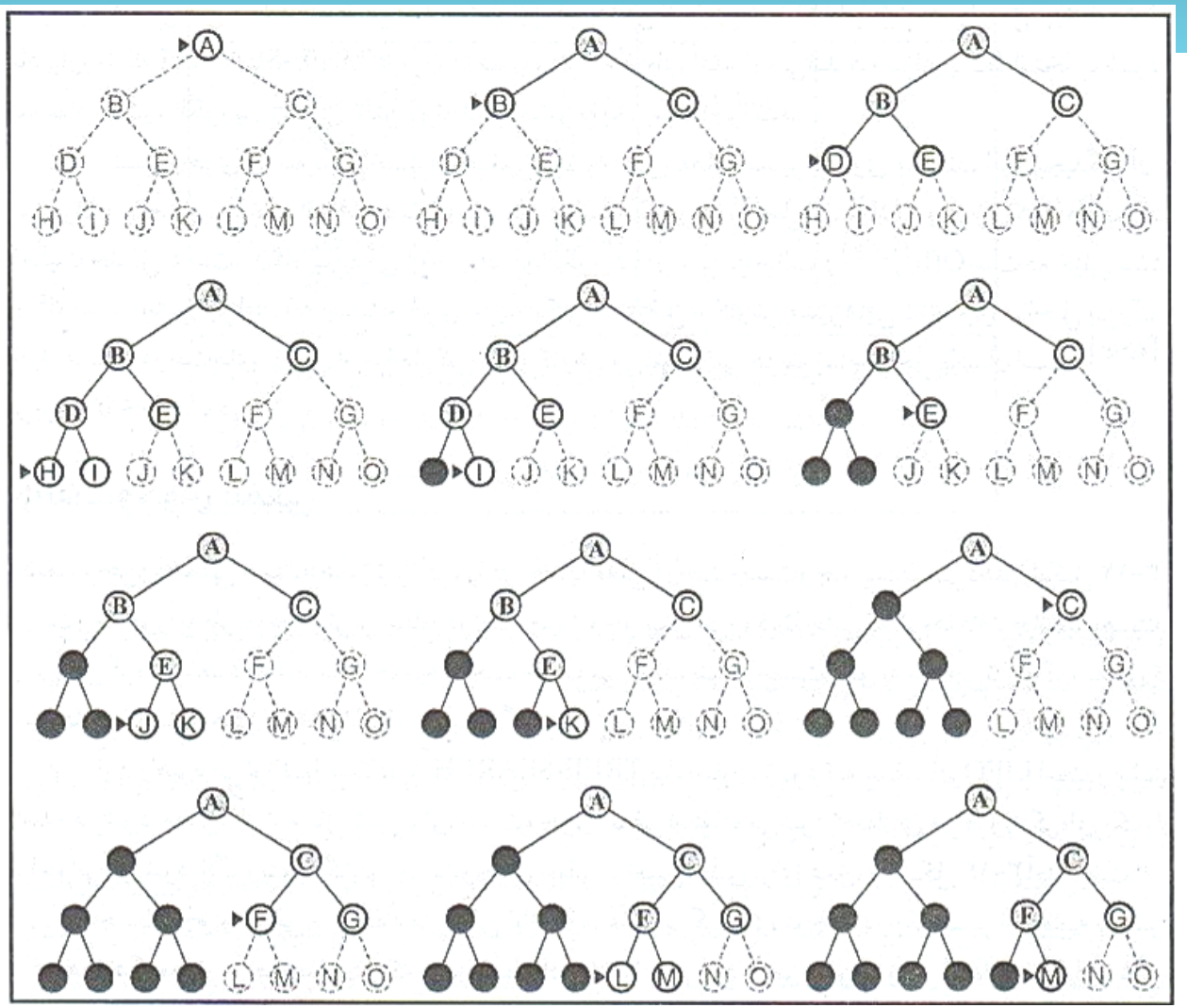
$$(S_0) \rightarrow (F_{99}, R_{80}) \rightarrow (P_{177}, F_{99}) \rightarrow (B_{310}, P_{177}) \rightarrow (B_{278}) \rightarrow returns\ SRPB$$

تحلیل UC-BFS

- پیچیدگی زمان و فضا اگر دیگر برمبنای تعداد نود در هر سطح و d قابل بیان نیست. چون نودها به ترتیب سطح بسط داده نمی‌شوند.
- فرض کنیم
  - ✓ C^* هزینه راه‌حل بهینه باشد،
  - ✓ هزینه هر عملی حداقل ε باشد،
  - ✓ در بدترین حالت:

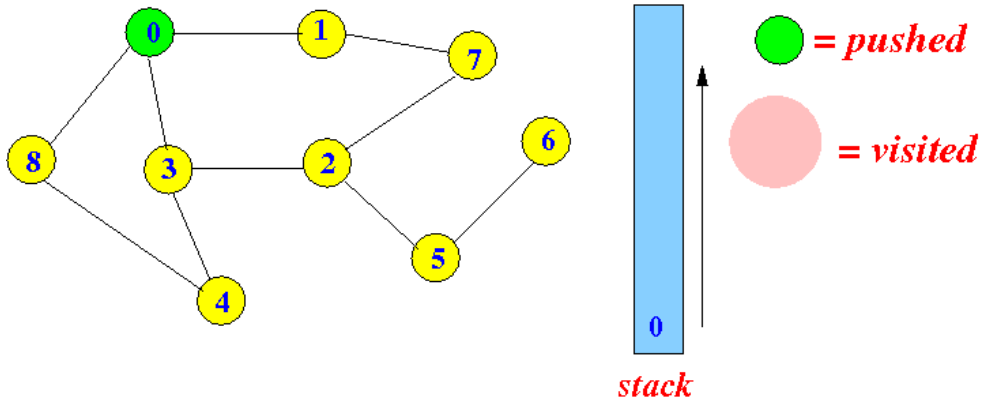
$$b^{1+\lfloor C^*/\varepsilon \rfloor}$$

جستجوی اول‌عمق (DFS)

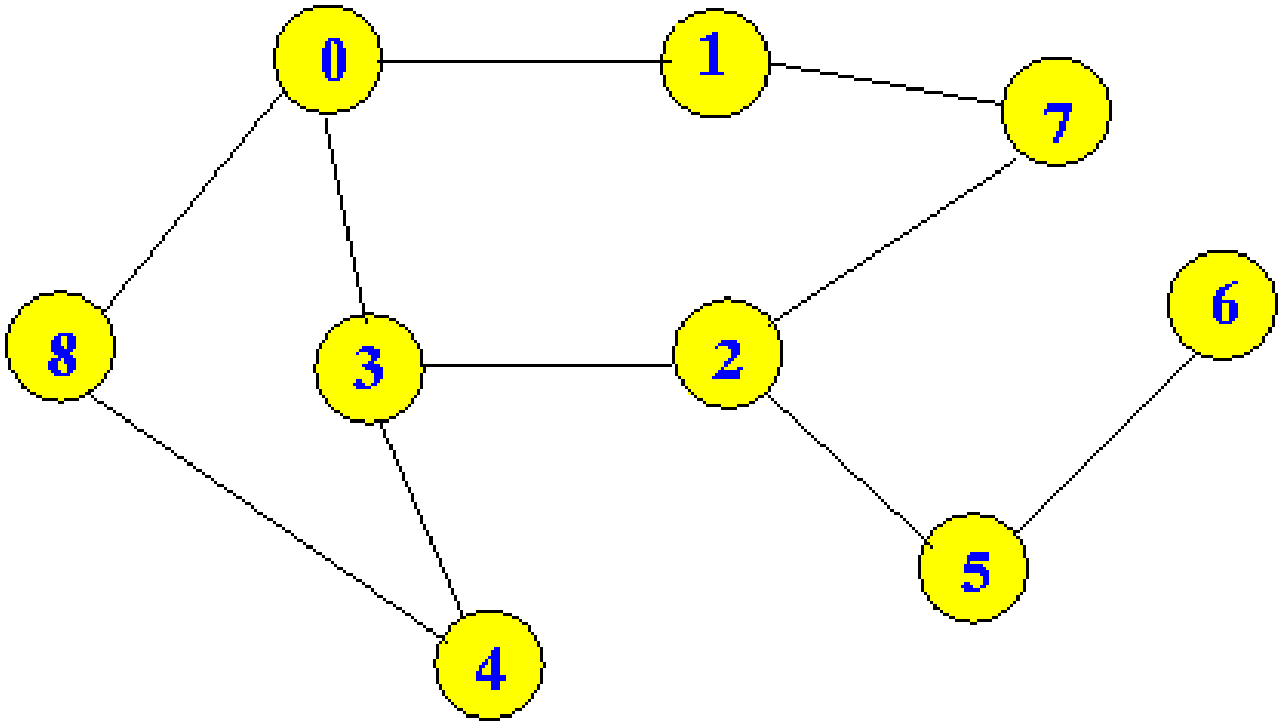


جستجوی اول‌عمق

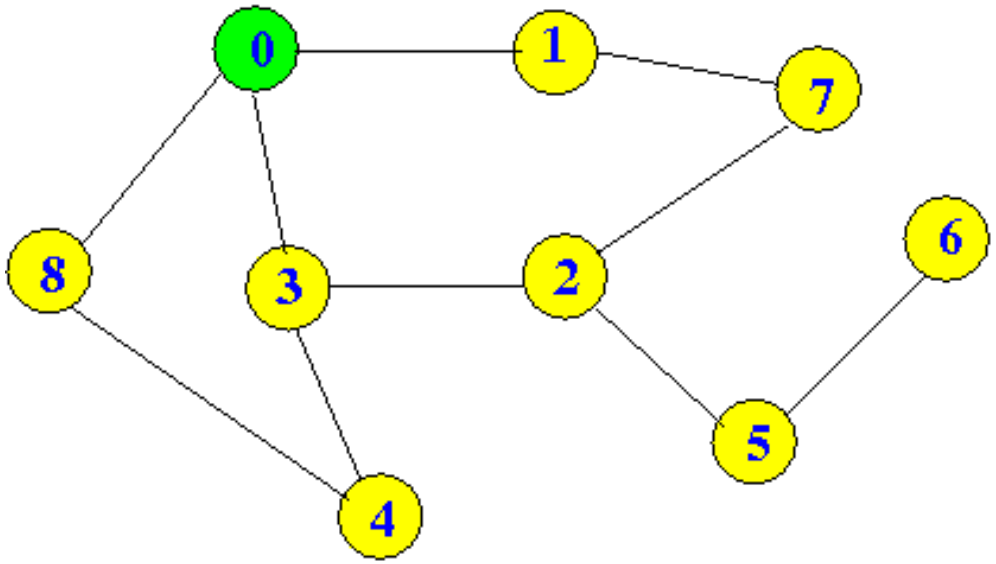
- بسط عمیق‌ترین گره در frontier فعلی درخت جستجو، تاجایی که گره‌ها فاقد فرزند باشند.
- حذف این گره‌ها از frontier درخت.
  - ✓ استفاده از صف LIFO (پشته)

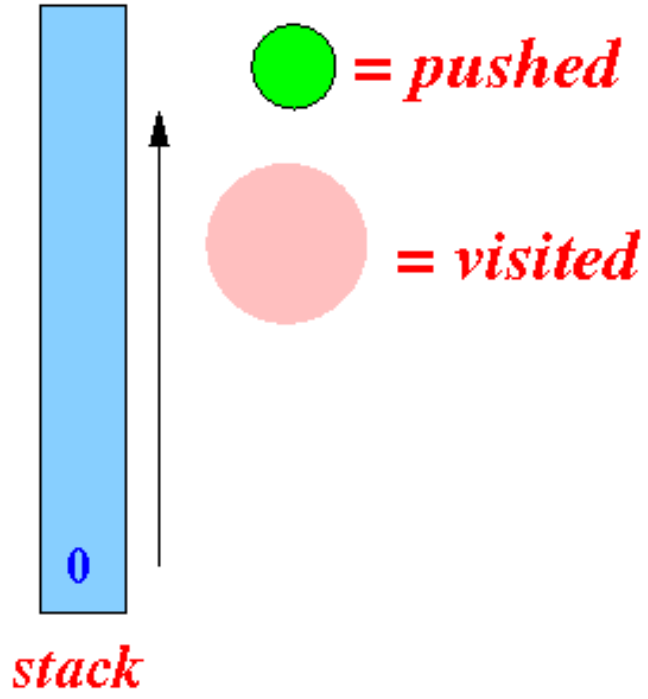


DFS

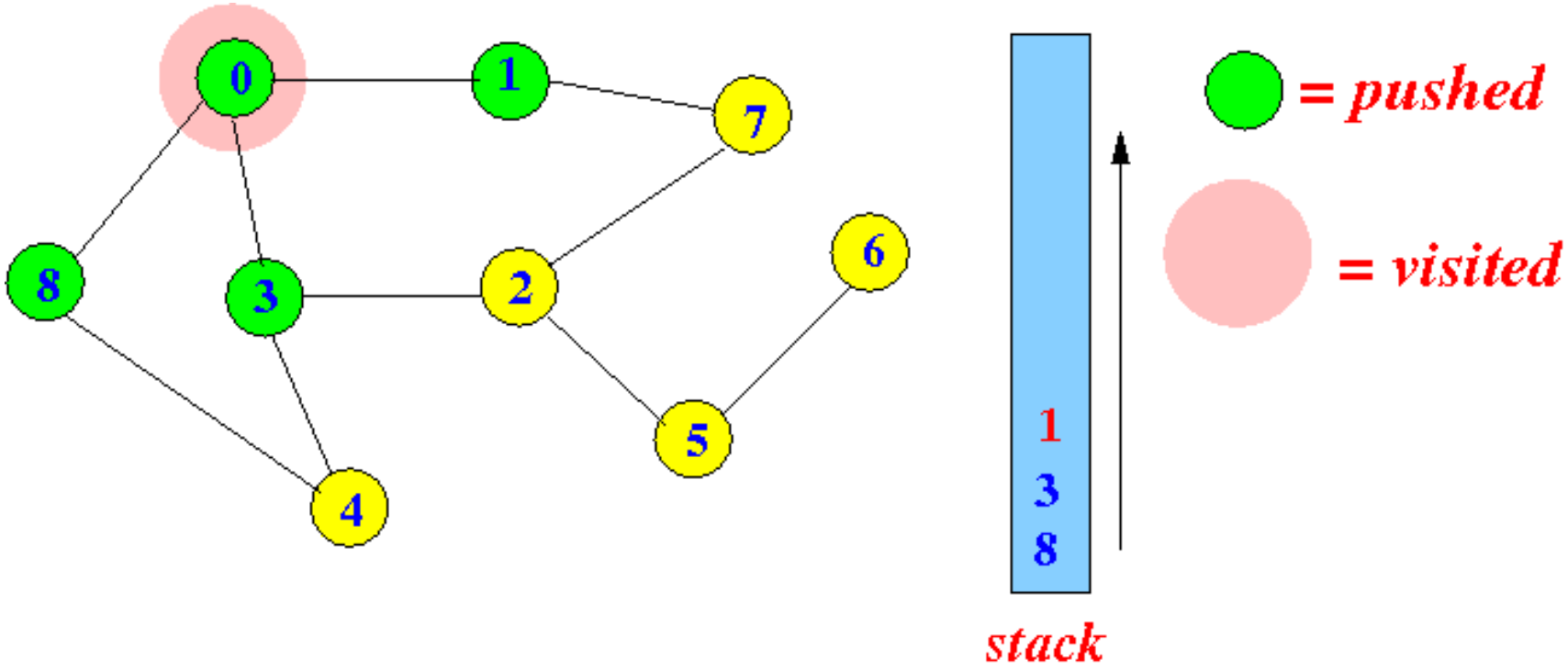


DFS

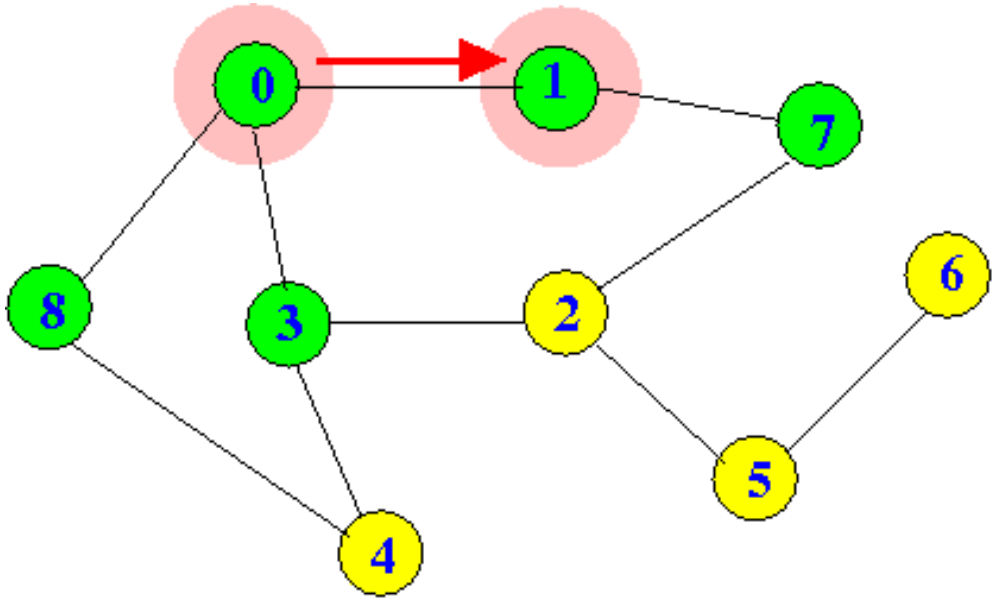


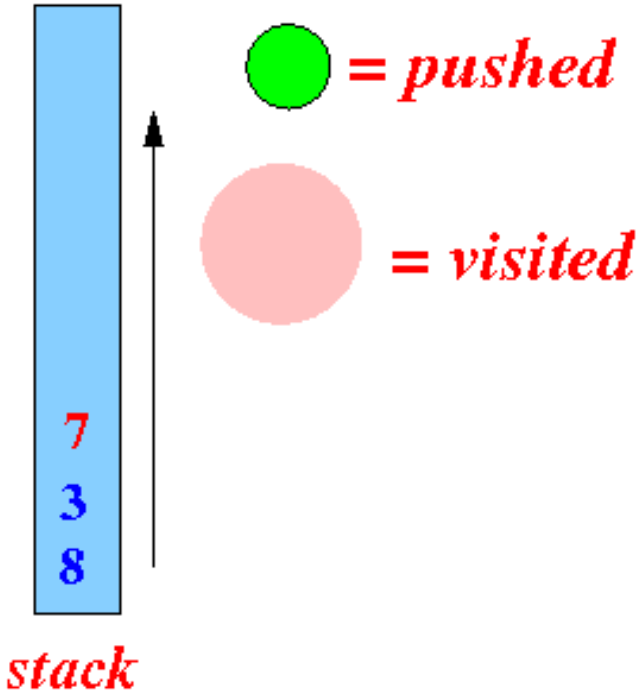


DFS

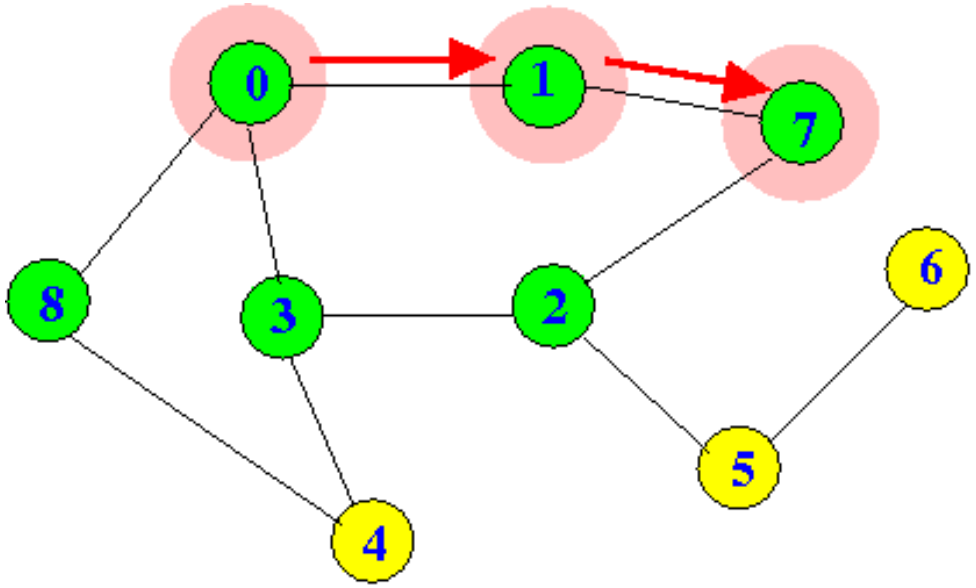


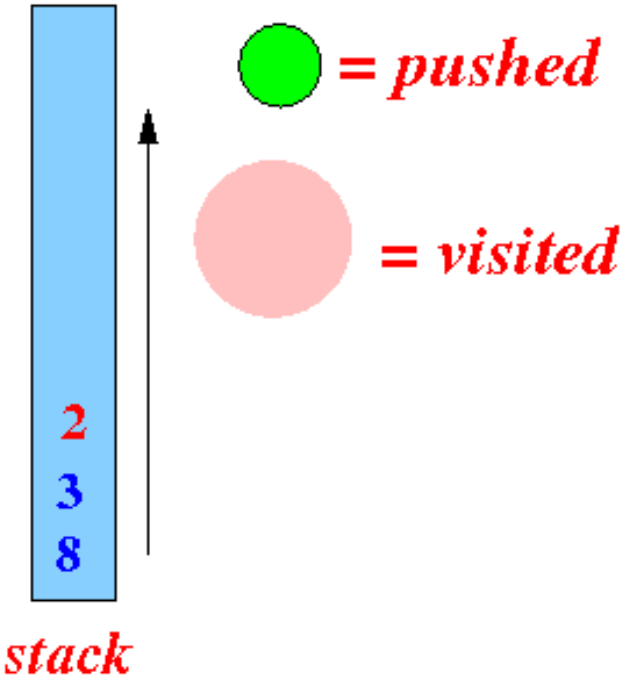
DFS



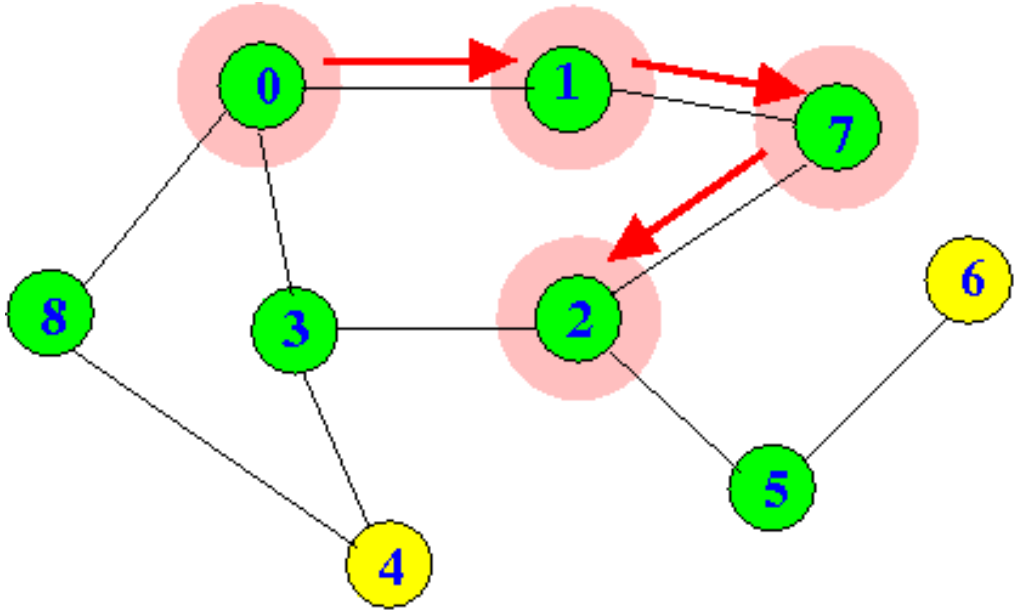


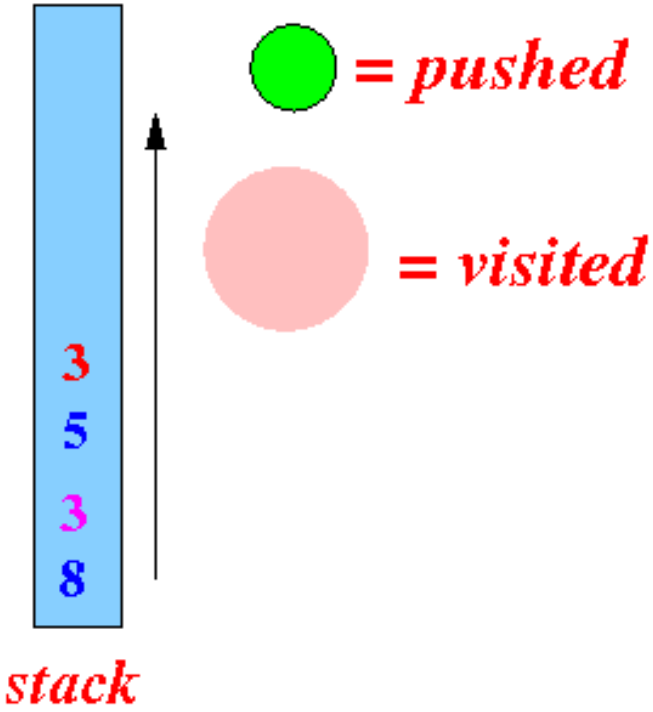
DFS





DFS





DFS

- در فضای حالت نامحدود (infinite) ناکامل است.
- بهینه نیست.
- پیچیدگی زمانی $O(b^m)$، یعنی DFS در بدترین حالت ممکن است تمام نودهای ممکن در درخت جستجو را تولید نماید. این تعداد، در صورت استفاده از جستجوی درختی، حتی می‌تواند بسیار بزرگتر از اندازه فضای حالت باشد.

- که می‌تواند خیلی بزرگتر از $O(b^d)$یعنی پیچیدگی BFS باشد.
- پس چرا استفاده می‌شود؟

DFS

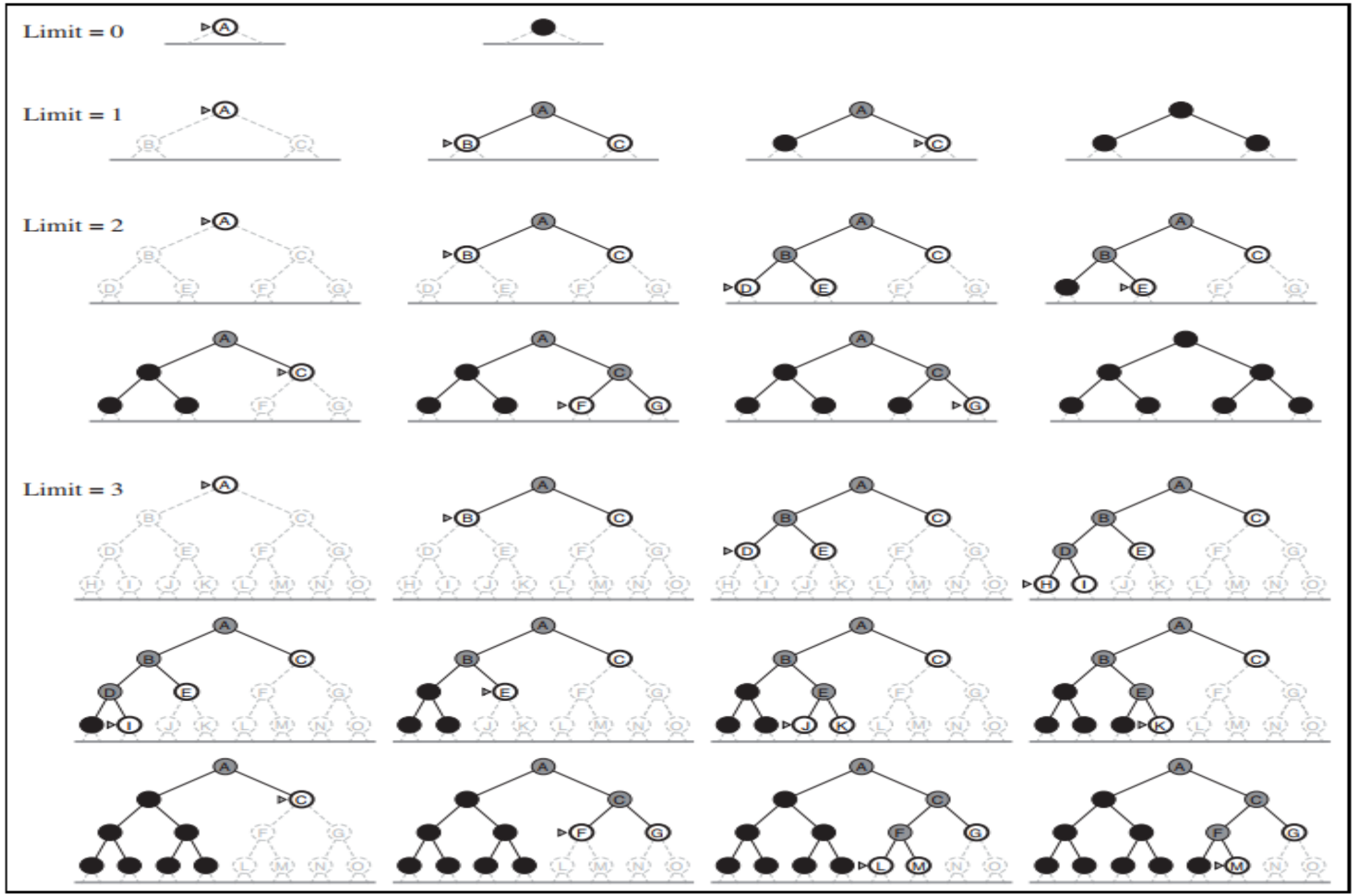
- دلیل آن، میزان مصرف حافظه است:
  - ✓ بعد از بسط کامل هر نود، آن نود قابل حذف از پشته (frontier) است.
  - ✓ که البته اگر از جستجوی گرافی استفاده کنیم، بهبود چشمگیری نسبت به BFS ایجاد نمی‌کند.
  - ✓ ولی اگر از جستجوی درختی استفاده نماییم، باتوجه به اینکه مجموعه Explored نگهداری نمی‌شود، می‌تواند پیچیدگی فضا را به $O(b \times m)$ یعنی تنها تعداد گره‌های تولیدشده در مسیر ریشه به عمیق‌ترین برگ، کاهش دهد.

جستجوی اول‌عمق محدود

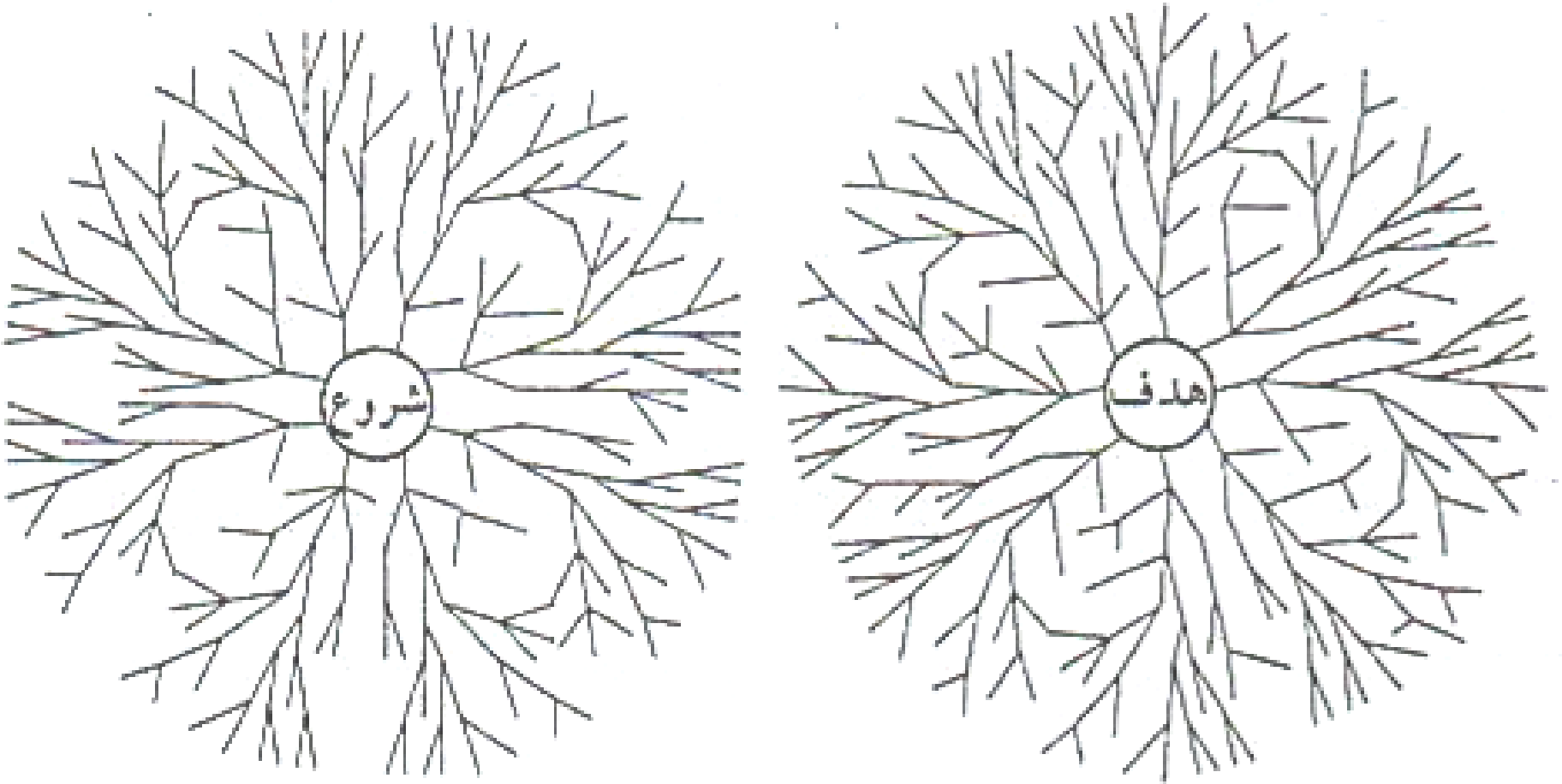
- دیدیم که الگوریتم DFS در پیچیدگی زمانی به شدت با مشکل روبرو است.
- برای حل این مشکل، می‌توان عمق جستجو را به l محدود کرد.
- مقدار l را می‌توان برمبنای دانش مجود در مسئله محاسبه نمود
- بهینگی و کامل بودن:
  - ✓ اگر $l < d$ باشد کامل نیست.
  - ✓ اگر $l > d$ باشد بهینه نیست.
- پیچیدگی زمانی: $O(b^l)$
- پیچیدگی فضا: $O(bl)$

جستجوی عمیق‌شونده تکراری

- بهترین عمق محدود را می‌یابد.
- همانند جستجوی اول‌سطح وقتی کامل است که فاکتور انشعاب محدود باشد.
- وقتی بهینه است که هزینه مسیر تابعی غیر نزولی از عمق گره باشد.
- پیچیدگی زمانی: $O(b^d)$
- همانند جستجوی عمقی پیچیدگی فضایی آن کم است: $O(bd)$


Limit = 0
Limit = 1
Limit = 2
Limit = 3

جستجوی دوطرفه



جستجوی دوطرفه

- اجرای همزمان دو جستجو از حالت اولیه و حالت هدف است.
  - ✓ هدف آن کاهش پیچیدگی الگوریتم از b^d به $b^{d/2} + b^{d/2}$ است.
  - ✓ وجود گره مشترک در frontierهای دو جهت جستجو به معنی یافته شدن یک جواب است.
  - ✓ لزوما جواب بهینه را نمی‌یابد.
  - ✓ جستجو از هدف به سمت عقب سخت و پیچیده است.
  - ✓ اگر در هر دو سمت از BFS استفاده شود کامل است.
  - ✓ پیچیدگی فضایی بالا به دلیل استفاده از BFS

مقایسه راهبردهای جست و جوی نا آگاهانه

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	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

- ضریب انشعاب محدود باشد.
- هزینه گام یک مقدار مثبت باشد.
- همه هزینه گام‌ها مثبت باشند.
- در هر دو طرف، از الگوریتم BFS استفاده شود.