هوش مصنوعی

حسین کارشناس

دانشکده ریاضی

ترم اول ۹۴ – ۹۳

حل مسأله با جستجو

حل مسأله با جستجو

- عامل مبتنی بر هدف ← عامل حل مسأله
  - استفاده از نمایش تجزیه‌ناپذیر دانش
- الگوریتم‌های جستجوی همه منظوره (general-purpose)
  - ناآگاه (uninformed): فقط از تعریف مسأله بهره می‌برد
  - آگاه (informed): از دانش خاص مسأله نیز بهره می‌برد
- تعریف مسأله به صورت جستجو
  - تبیین هدف (goal formulation) بر اساس حالت فعلی و معیار کارایی عامل
    - هدف: زیرمجموعه‌ای از حالت‌های محیط
  - تبیین مسأله (problem formulation)
    - حالت‌های محیط و کنش‌ها ممکن در هر حالت برای انتقال به حالت‌های دیگر
    - میزان جزئیات لازم برای حل مسأله

حل مسأله با جستجو

- در محیط‌های ناشناخته، عامل (مبتنی بر هدف) باید نگاه به آینده داشته باشد (کنش‌های آینده را نیز در نظر بگیرد) تا بتواند عملکرد خود را ارزیابی کند.
- فرض می‌کنیم در محیط مشاهده‌پذیر، گسسته، شناخته شده و قطعی به جستجو می‌پردازیم
  - راه‌حل مسأله، دنباله‌ای ثابت از کنش‌هاست
- راه‌برد انشعاب (branching) در محیط‌های پیچیده‌تر
- جستجو: فرآیندی که در پی یافتن دنباله‌ای از کنش‌ها برای رسیدن به هدف است

حل مسأله با جستجو

- مسأله ← الگوریتم جستجو ← راه‌حل (دنباله‌ای از کنش‌ها)
- طراحی مورد نظر برای عامل:
  1) تبیین
     - هدف و مسأله
  2) جستجو
     - پیدا کردن دنباله‌ای از کنش‌ها
  3) اجرا
     - انتخاب یکی از کنش‌ها برای اجرا
- یک سیستم حلقه باز (open-loop)
  - در انتخاب کنش‌ها، ادراک‌های دریافت شده از محیط را در نظر نمی‌گیرد

حل مسأله با جستجو

- طراحی عامل حل مسأله

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

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

تعریف مسأله

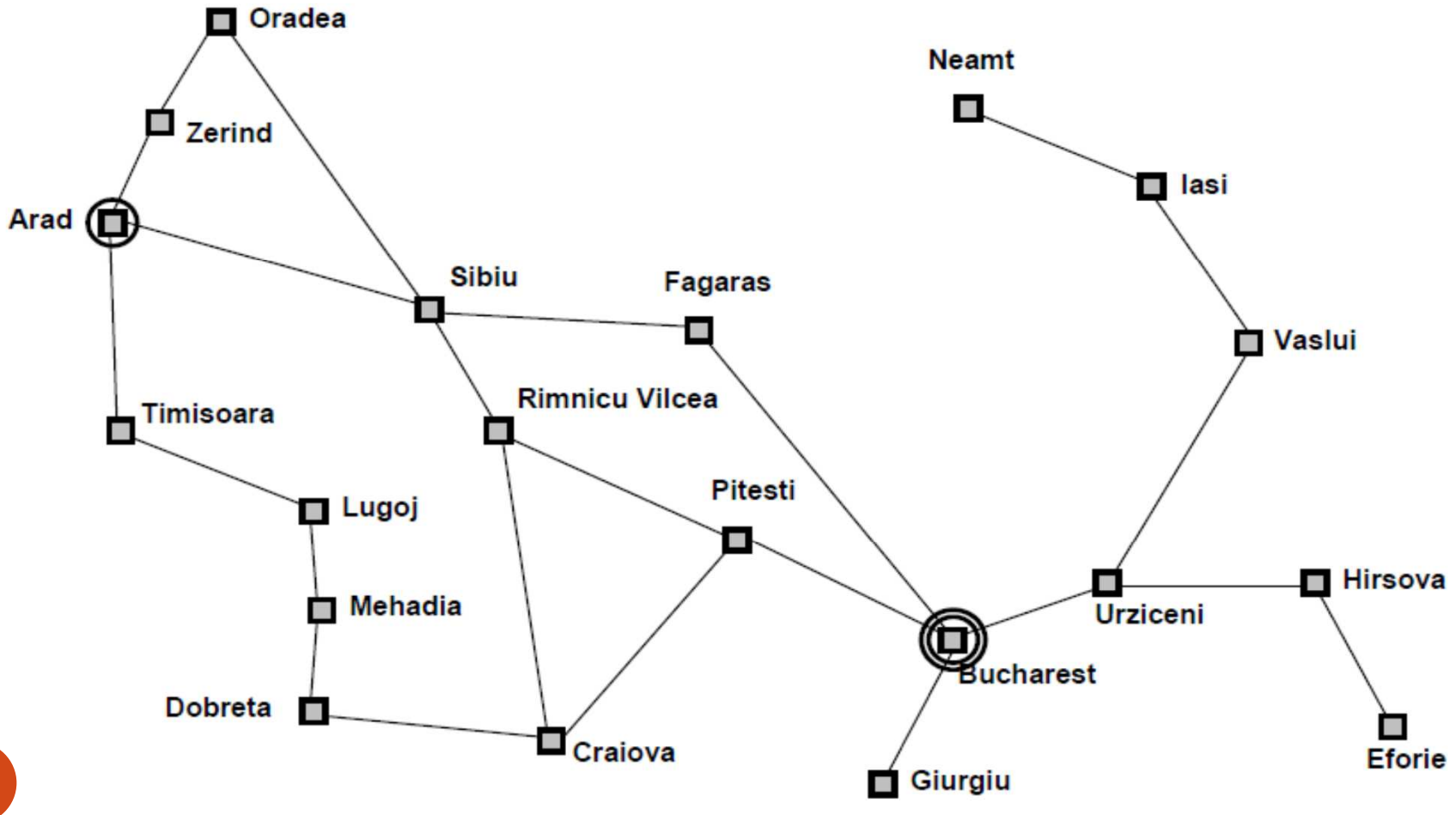
- مؤلفه‌های لازم برای تعریف مسأله
  - حالت اولیه: عامل از کدام حالت شروع به فعالیت می‌کند
    - کلیه‌ی حالت‌های ممکن
  - کنش‌های ممکن: کنش‌هایی که می‌توان در هر حالت اعمال کرد
  - مدل انتقال (transition model): توصیف نتایج کنش‌ها
    - حالت‌هایی بعدی که از اعمال یک کنش در یک حالت بخصوص قابل دسترسی است
  - آزمایش هدف: تعیین هدف بودن یا نبودن یک حالت
  - هزینه مسیر: یک مقدار عددی به هر **مسیر** اختصاص می‌دهد
    - هزینه مسیر برابر مجموع هزینه انجام کنش‌های در طول مسیر است
    - هزینه گام: هزینه انجام کنش a در حالت s برای رسیدن به حالت s' $c(s, a, s')$

تعریف مسأله

- فضای حالت مسأله: مجموعه تمام حالت‌هایی که از حالت اولیه با اعمال دنباله‌ای از کنش‌ها قابل دسترسی است
  - توسط سه مؤلفه اول تعریف می‌شود
  - قابل نمایش با یک گراف جهت‌دار
- مسیر: دنباله‌ای از حالت‌های فضای حالت که توسط دنباله‌ای از کنش‌ها به هم متصل شده‌اند
- راه‌حل مسأله: دنباله‌ای از کنش‌ها که از یک حالت اولیه به یک حالت هدف منجر شود
  - راه‌حل بهینه: راه‌حلی که دارای کمترین هزینه مسیر باشد

تعریف مسأله

- مثال: فضای حالت یک مسأله جستجوی مسیر

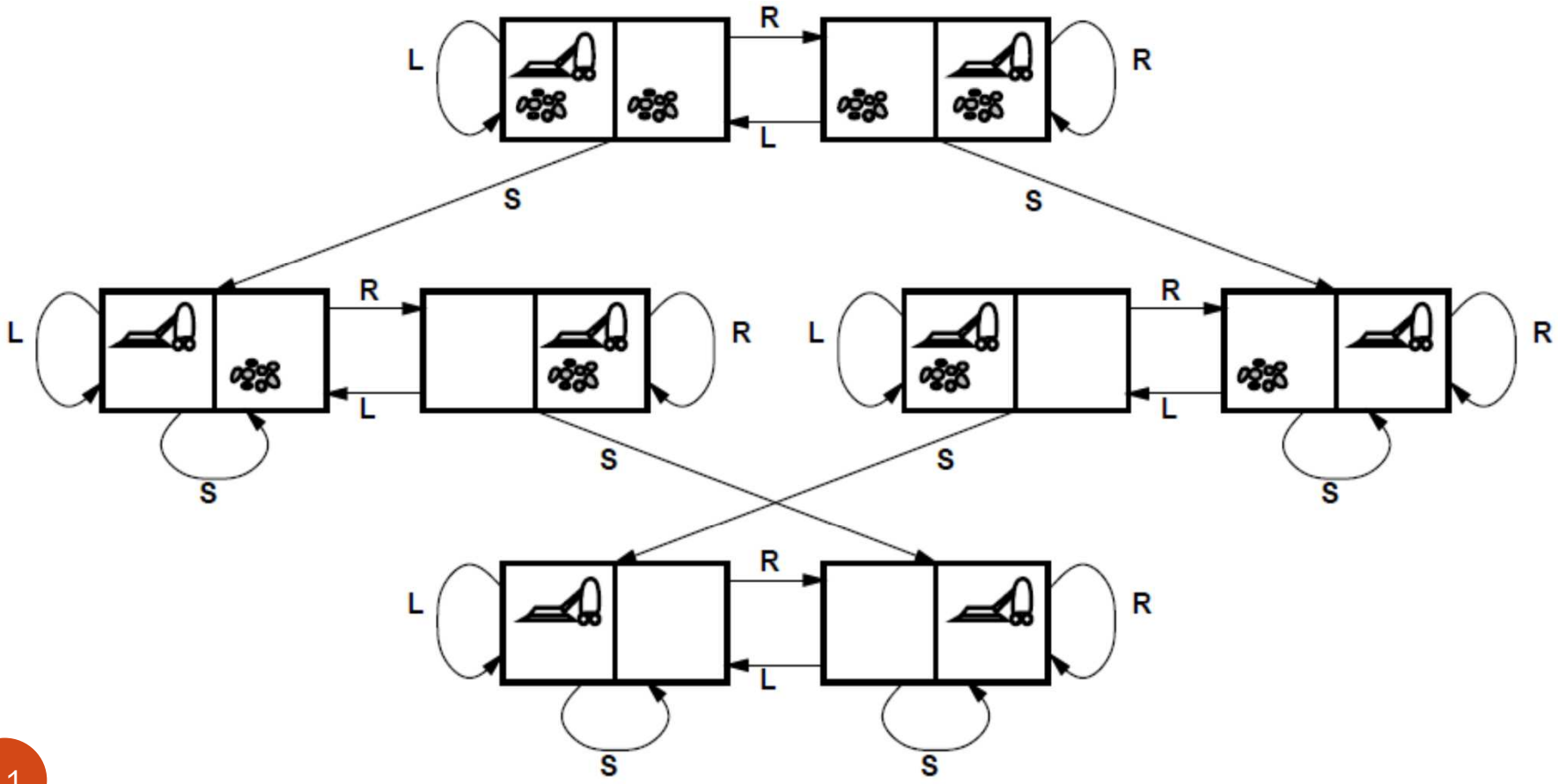


تعریف مسأله

- چه سطحی از جزئیات در تعریف مسأله باید در نظر گرفته شود؟
- انتزاع (abstraction): فرآیند زدودن جزئیات نامرتبط با مسأله از نمایش دانش (توصیف حالت‌ها و کنش‌ها)
  - درجات مختلف انتزاع در سطوح مختلف پیاده‌سازی
  - انتزاع معتبر
    - راه‌حل بدست آمده برای آن قابل گسترش به راه‌حلی برای مسأله با جزئیات بیشتر باشد
  - انتزاع مفید
    - انتزاعی که انجام هر یک از کنش‌های راه‌حل آن از مسأله اصلی ساده‌تر باشد
- راهکار: ساده کردن مسأله تا حدی که انتزاع آن همچنان معتبر باقی بماند

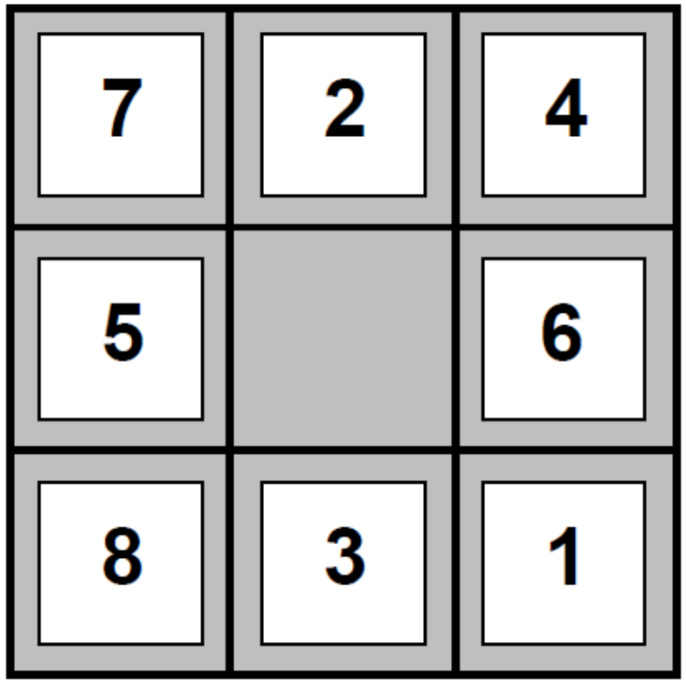
مسائل بازیچه (toy problems)

- دنیای جاروبرقی (vacuum world)

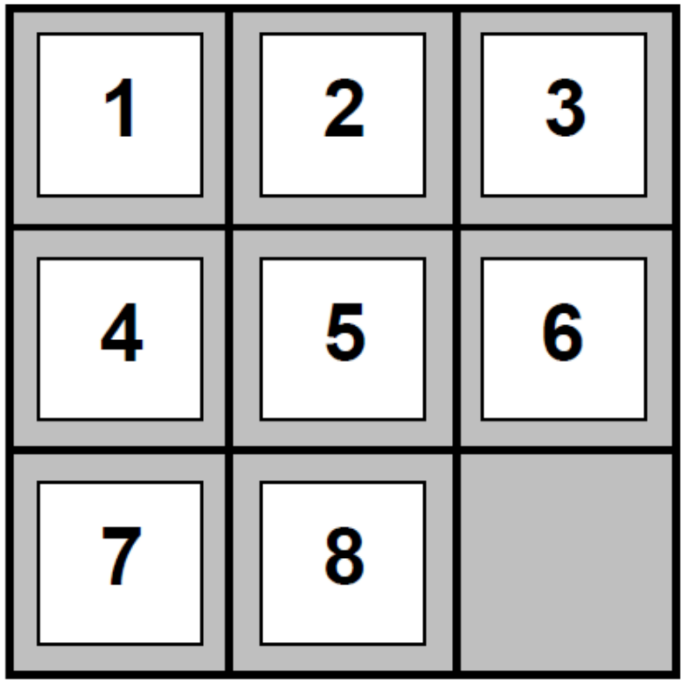


مسائل بازیچه

- معمای ۸ (8-puzzle)



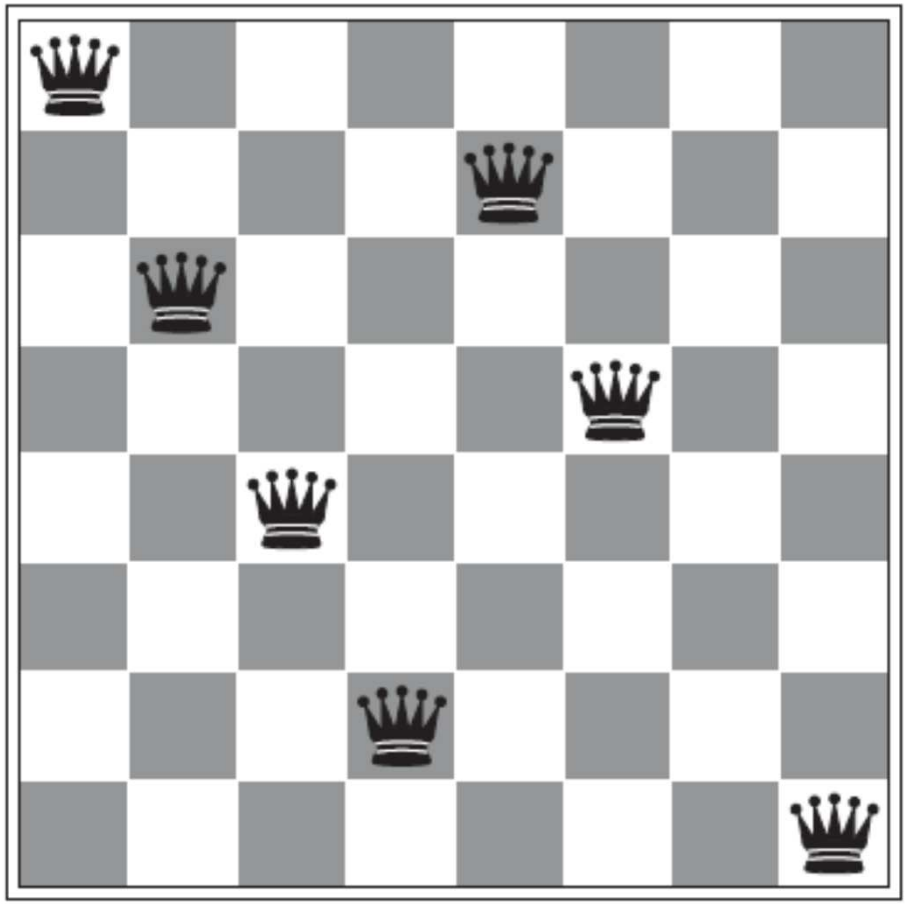
Start State



Goal State

مسائل بازیچه

- ۸ ملکه شطرنج (8-queens)



مسائل واقعی

- مسیریابی (route-finding)
- گشت و گذار (touring)
  - فروشنده دوره‌گرد (traveling salesperson problem - TSP)
- طرح‌بندی (layout) مدارهای مجتمع
  - طرح‌بندی سلول‌های کاری
  - طرح‌بندی مجراهای ارتباطی
- ناوبری (navigation) ربات‌ها
- ترتیب‌بندی همگذاری خودکار (automatic assembly)
  - طراحی پروتئین در تولید دارو

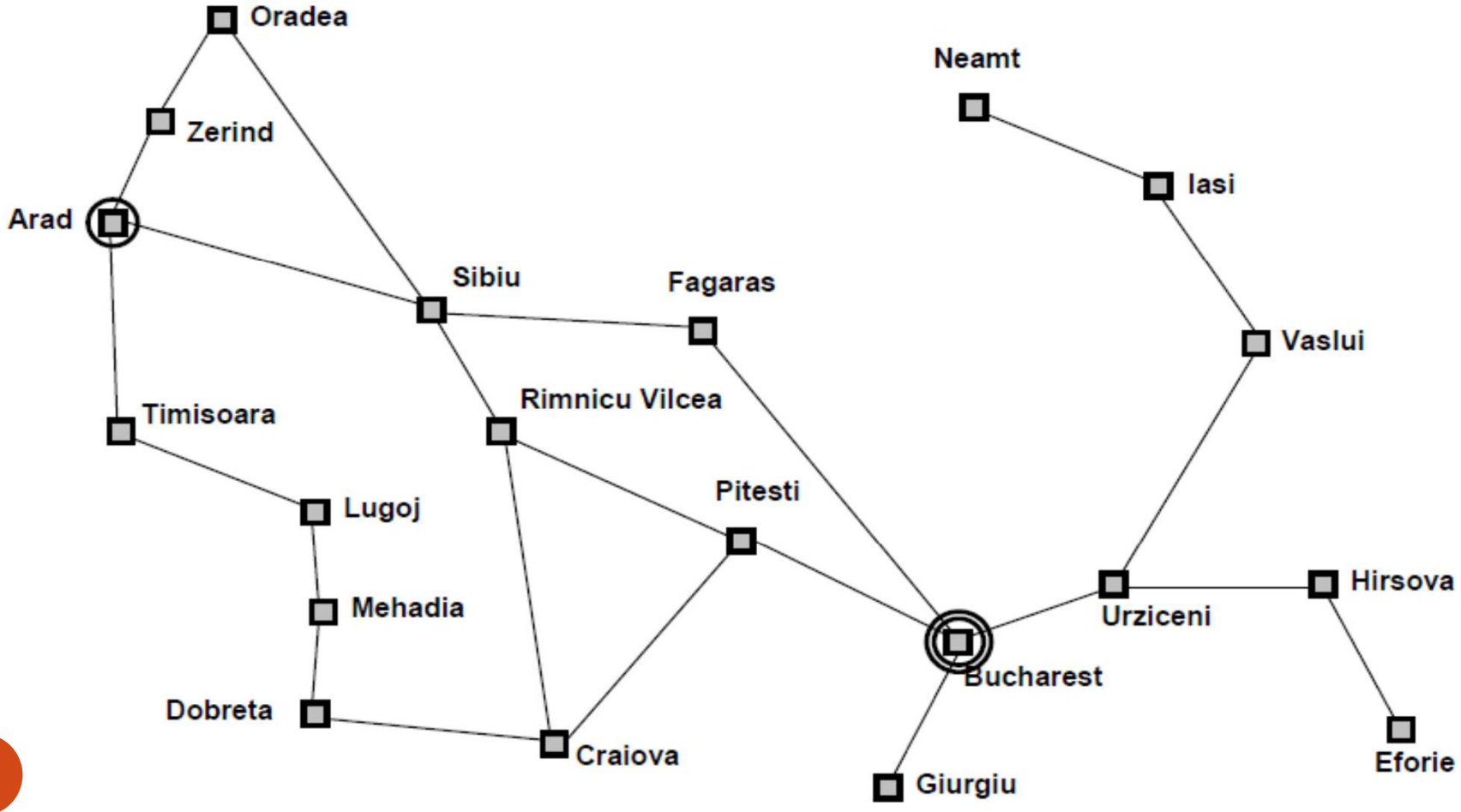
جستجو برای راه‌حل

- الگوریتم‌های جستجو در حقیقت دنباله‌های مختلف کنش‌ها را بررسی می‌کنند
- این دنباله‌های مختلف یک درخت جستجو را تشکیل می‌دهند
  - گره‌ها ⟵ حالت‌ها
  - یال‌ها ⟵ کنش‌ها
- الگوریتم پایه جستجو بر مبنای درخت

1) شروع از حالت اولیه به عنوان ریشه درخت
2) آزمایش هدف: توقف در صورت پیدا کردن هدف
3) بسط حالت فعلی و تولید حالات جدید به عنوان گره‌های فرزند در درخت
4) انتخاب یکی از فرزندان و رفتن به مرحله دوم

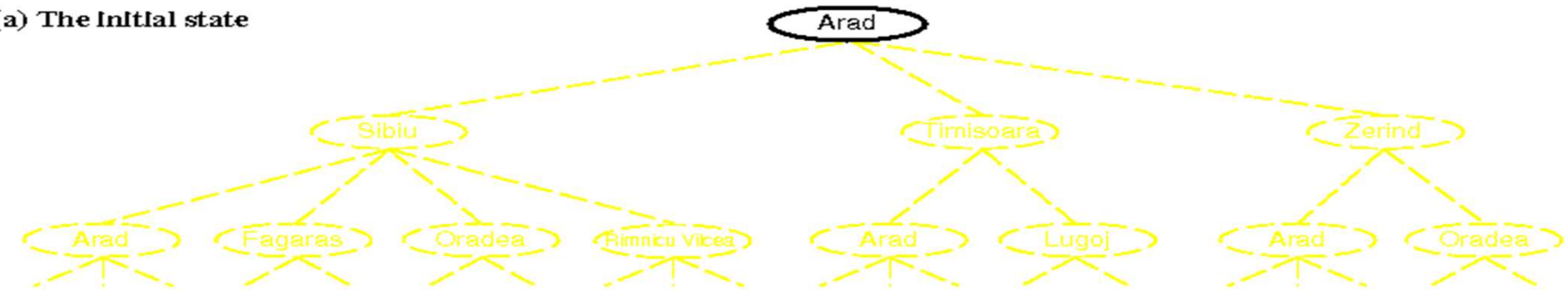
جستجو برای راه‌حل

- مثال: سفر در رومانی

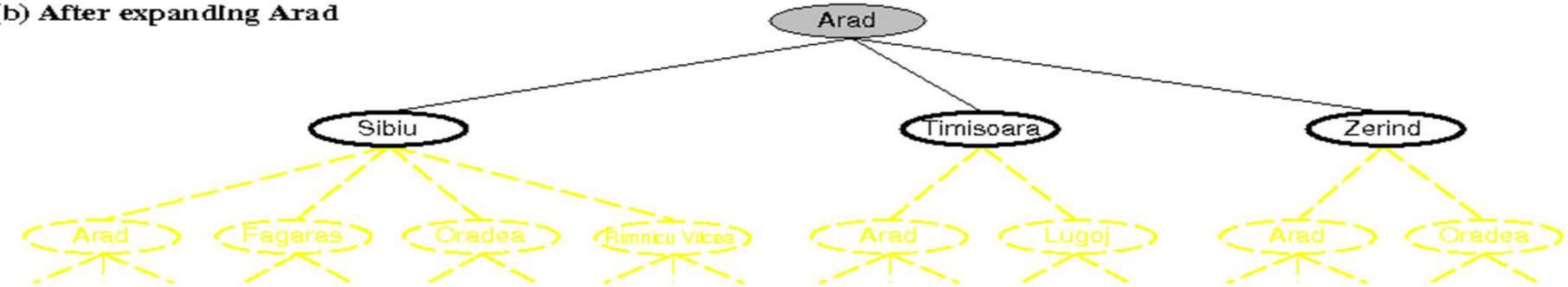


جستجو برای راه‌حل

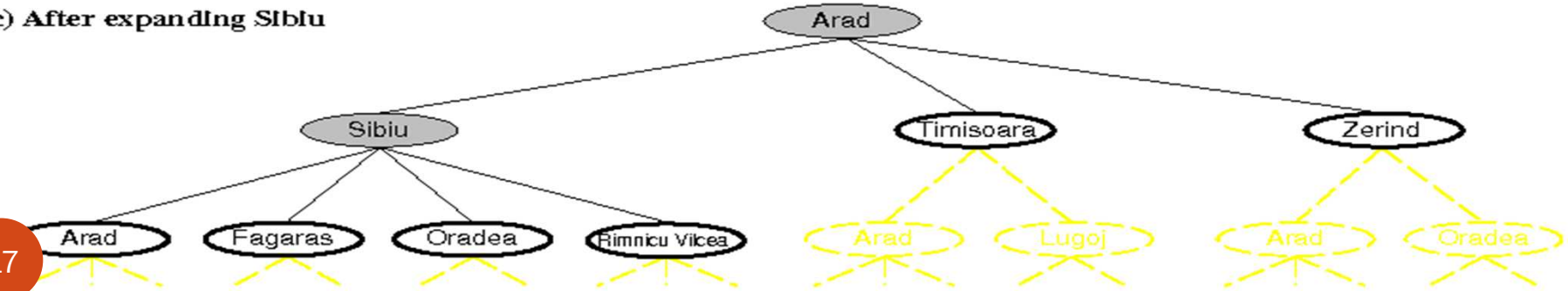
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



جستجو برای راه‌حل

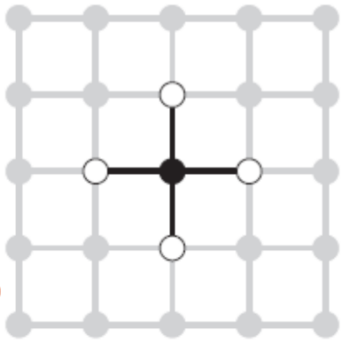
- لیست مقدم (frontier): مجموعه تمام گره‌هایی که در زمان فعلی آماده بسط دادن هستند.
- الگوریتم جستجوی درخت

```
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
```

- تفاوت اصلی الگوریتم‌های متفاوت جستجو در راه‌برد آنها برای انتخاب حالت بعدی است.

جستجو برای راه‌حل

- حالت‌های تکراری و مشکل مسیرهای حلقه‌دار (loopy paths)
  - منجر به درخت‌های نامتناهی (برای فضاهای حالت متناهی) می‌شود
  - مسیرهای حلقه‌دار همیشه بدتر از مسیرهای معادلشان پس از حذف حلقه
    - با فرض هزینه قدم‌های مثبت
- مسیرها افزونه (redundant): وجود بیش از یک مسیر بین دو حالت
  - چند راه برای رسیدن به هدف
  - تغییر تعریف مسأله برای حذف مسیرهای افزونه
  - برای بعضی از مسأله‌ها امکان چنین تغییری وجود ندارد
    - مسائلی با کنش‌های برگشت‌پذیر (reversible)
    - مثال: مسیریابی روی شبکه‌های مستطیلی (rectangular grids)



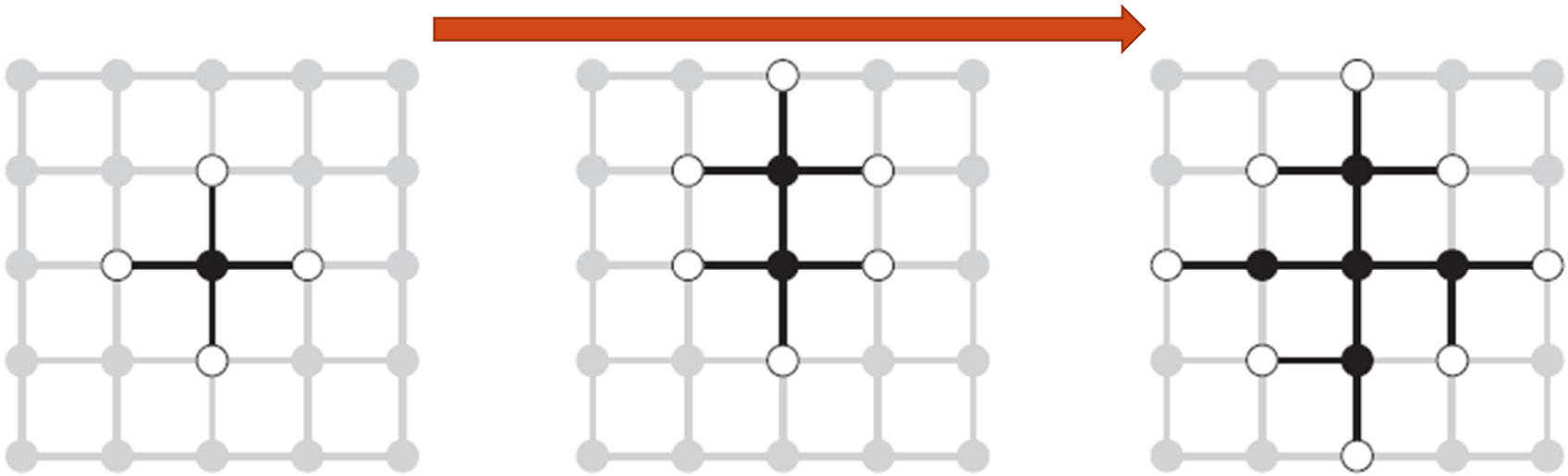
جستجو برای راه‌حل

- استفاده از مجموعه گره‌های مشاهده شده (explored set) برای پیشگیری از حالت‌های تکراری و مسیرهای افزونه
- الگوریتم جستجوی گراف

```
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
```

جستجو برای راه‌حل

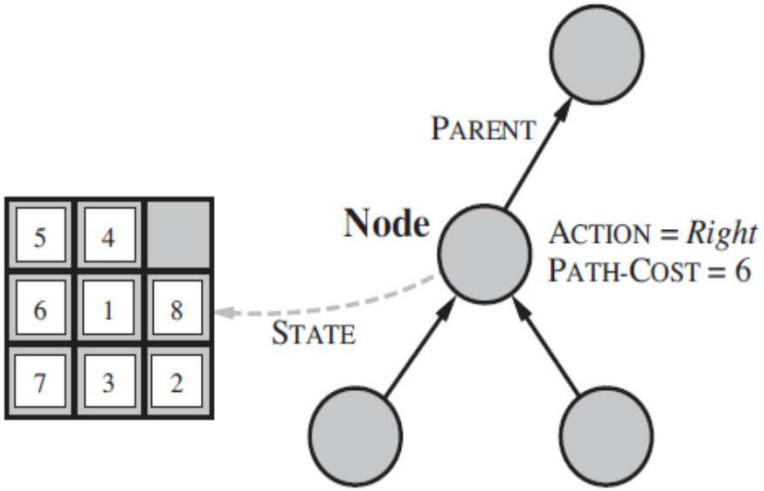
- در الگوریتم جستجوی گراف، لیست مقدم فضای حالت را به دو ناحیه مشاهده شده و مشاهده نشده تفکیک می‌کند



- بررسی قاعده‌مند (systematic) کلیه‌ی حالت‌های فضای حالت تا پیدا کردن راه‌حل

مقدمات طراحی الگوریتم‌های جستجو

- ساختمان داده برای نشان دادن درخت ساخته شده در حین جستجو
- هر گره شامل چهار مؤلفه است:
  - حالت (state)
  - والد (Parent)
  - کنش تولید کننده گره
  - هزینه مسیر (g(n)) از حالت اولیه تا گره



- هزینه مسیر تا فرزند برابر است با هزینه مسیر تا والد به اضافه هزینه قدم از والد تا فرزند
- استفاده از لیست پیوندی معکوس برای نشان دادن درخت
  - تولید راه‌حل بعد از یافتن هدف با دنبال کردن نشانه‌گرهای والد تا ریشه

مقدمات طراحی الگوریتم‌های جستجو

- ساختمان داده برای نشان دادن لیست مقدم: صف (Queue)
  - صف FIFO
  - صف LIFO (استک – stack)
  - صف اولویتی (بر اساس یک مرتب‌سازی)
- امکان پیاده‌سازی صف با زمان ذخیره و بازیابی تقریباً ثابت
- ساختمان داده برای مجموعه گره‌های مشاهده شده
  - یک جدول درهم (hash-table) برای جستجوی سریع محتویات

مقدمات طراحی الگوریتم‌های جستجو

- نحوه ارزیابی الگوریتم‌های جستجو
  - کمال (completeness): تضمین در پیدا کردن یک راه حل
  - بهینگی (optimality): تضمین در پیدا کردن راه‌حل بهینه
  - پیچیدگی زمانی: تعداد گره‌های تولید شده برای یافتن راه‌حل
  - پیچیدگی فضایی: حداکثر تعداد گره‌های ذخیره شده در حافظه در حین جستجو
- پارامترهای مورد استفاده برای محاسبه پیچیدگی
  - ضریب انشعاب (b): حداکثر تعداد فرزندان هر گره از درخت
  - عمق (d): سطح قرارگیری کم‌عمق‌ترین گره حاوی حالت هدف در درخت
  - حداکثر طول (m): بیشترین طول ممکن برای هر مسیر در فضای حالت‌ها
- از دیدگاه بیرونی، کارآمدی عامل جستجوگر با هزینه کل آن، شامل هزینه جستجوی راه‌حل به اضافه هزینه مسیر راه‌حل ارزیابی، می‌شود

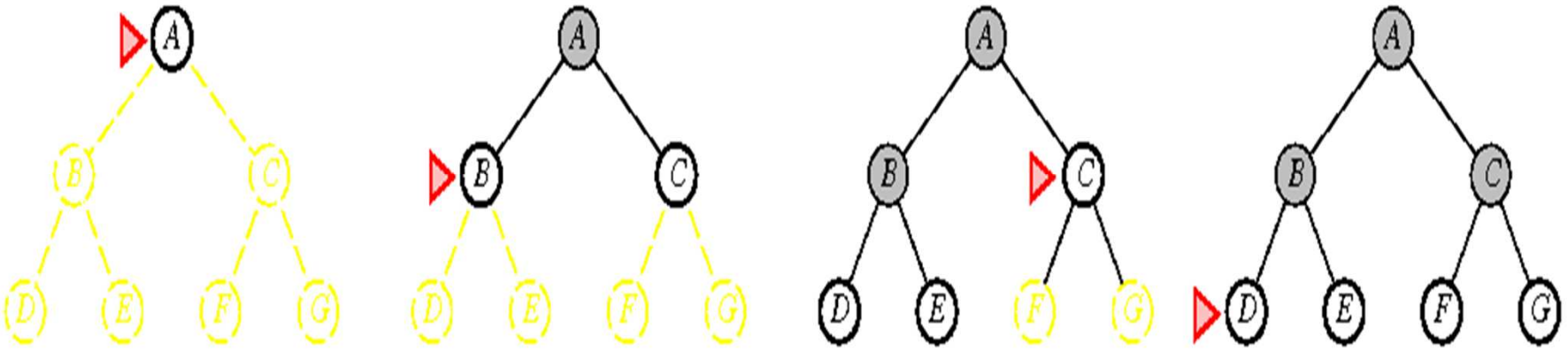
جستجوی کورکورانه

جستجوی کورکورانه (ناآگاه)

- بجز تعریف مسأله از اطلاعات دیگر خاص مسأله استفاده نمی‌کنند
  - تنها می‌توانند حالت‌های بعدی را تولید کنند و بین حالت‌های هدف و غیرهدف تمایز قائل شوند
- تفاوت این الگوریتم‌ها در ترتیبی (order) است که گره‌ها را بسط می‌دهند
- اگر الگوریتم بداند که کدام حالت‌های غیرهدف امیدبخش‌تر (promising) از بقیه هستند به یک الگوریتم آگاه تبدیل می‌شود

جستجوی عرض اول (Breadth-First)

- تمام گره‌های یک سطح از درخت قبل از گره‌های سطح بعد بسط داده می‌شوند.
  - در هر زمان کم‌عمق‌ترین گره بسط داده می‌شود
- استفاده از یک صف FIFO برای نگهداری لیست مقدم



جستجوی عرض اول

```
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)
```

جستجوی عرض اول

- همیشه کم‌عمق‌ترین مسیر به هر گره در لیست مقدم نگهداری می‌شود.
- ویژگی‌ها:
  - کامل است اگر ضریب انشعاب متناهی باشد
  - بهینه است اگر هزینه مسیر تابع غیرنزولی از عمق باشد (برای مثال تمام کنش‌ها دارای هزینه یکسان باشند)
    - کم‌عمق‌ترین گره هدف لزوماً بهینه نیست
  - پیچیدگی زمانی: $O(b^d)$
  - پیچیدگی فضایی: $O(b^d)$

جستجوی عرض اول

- با فرض b=10، تولید یک میلیون گره در ثانیه و اندازه گره 1000B

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

- نیاز به حافظه به مراتب بیشتر از نیازمندی زمانی الگوریتم است
- الگوریتم‌های ناآگاه نمی‌توانند مسائل جستجو با پیچیدگی نمایی را در حالت کلی حل کنند.

جستجو با هزینه یکنواخت (uniform-cost)

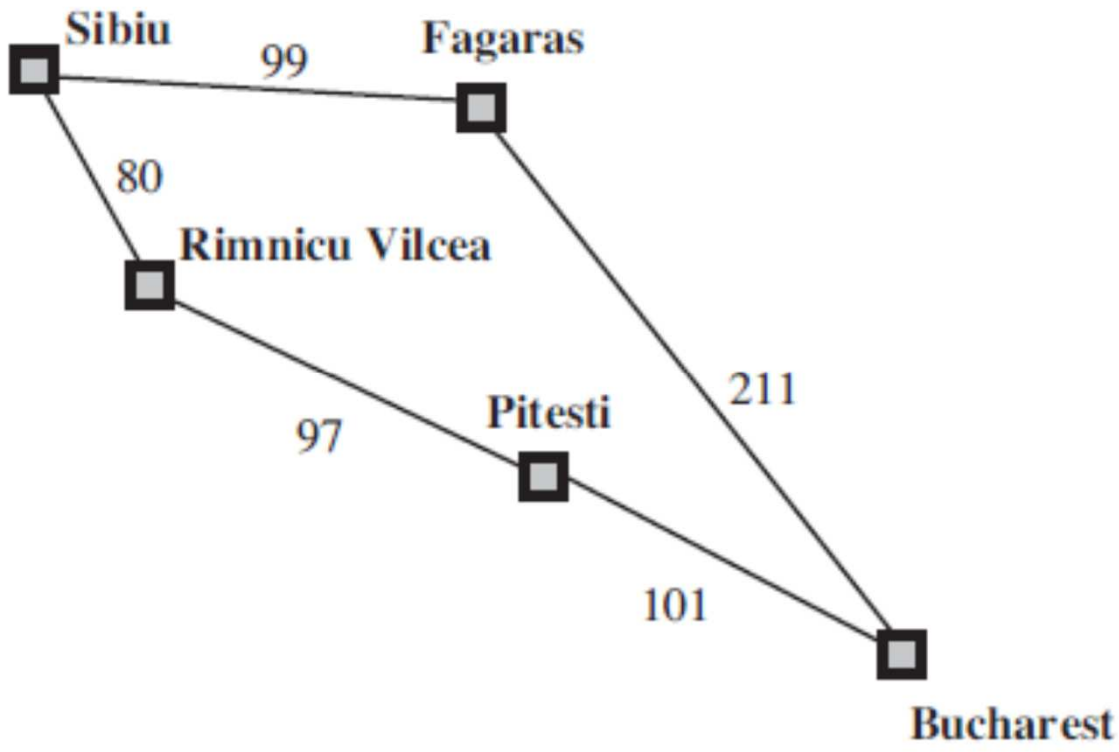
- بهبود الگوریتم جستجوی عرض اول برای یافتن راه‌حل بهینه با هر هزینه قدمی
  - هزینه قدم‌های غیر یکسان
- بسط گره‌ی با کمترین هزینه مسیر ($g(n)$) بجای کمترین عمق
- استفاده از صف اولویتی برای نگهداری لیست مقدم مرتب شده بر اساس تابع هزینه مسیر $g(n)$

جستجو با 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
```

جستجو با هزینه یکنواخت

- مثال: مسیریابی در رومانی برای رفتن از Sibiu به Bucharest
  - 80+97+101<99+211



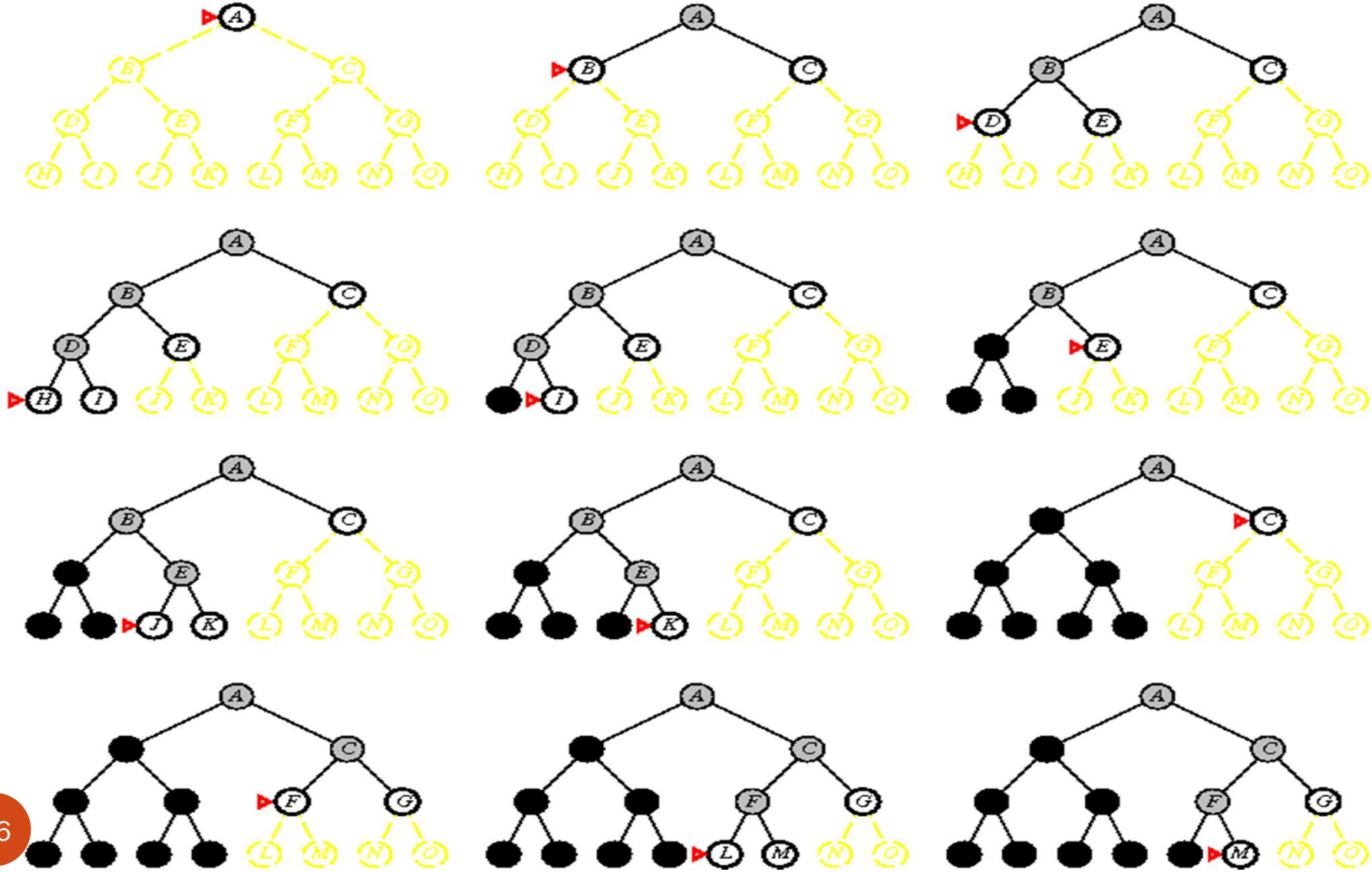
جستجو با هزینه یکنواخت

- ویژگی‌ها
  - بهینه است چون هزینه قدم‌ها منفی نیست
    - با اضافه شدن گره‌ها به مسیر، طول آن کوتاه‌تر نمی‌شود
    - همیشه مسیر بهینه برای گره‌های انتخاب شده برای بسط پیدا شده است
  - کامل است اگر هزینه هر قدم یک مقدار مثبت باشد (برای مثال بزرگتر از ε)
  - پیچیدگی زمانی: $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$
    - C^* هزینه راه‌حل بهینه است
  - پیچیدگی فضایی: $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$
- اگر هزینه تمام قدم‌ها برابر باشد، جستجو با هزینه یکنواخت برابر با جستجوی عرض اول خواهد بود: $b^{1+\lfloor C^*/\epsilon \rfloor} = b^{1+d}$

جستجوی عمق اول (depth-first)

- عمیق‌ترین گره در لیست مقدم را بسط می‌دهد
  - یک شاخه از درخت را دنبال می‌کند تا به برگ برسد
- استفاده از صف LIFO برای پیاده‌سازی لیست مقدم
- پیاده‌سازی بر اساس الگوریتم جستجوی درخت
  - عدم نگهداری گره‌های مشاهده شده
  - امکان پیاده‌سازی به صورت بازگشتی (recursive)

جستجوی عمق اول



جستجوی عمق اول

- ویژگی‌ها
  - کامل است اگر فضای حالت‌ها متناهی باشد
    - مسأله Knuth با فضای حالت نامتناهی
    - نسخه مبتنی بر جستجوی درخت (با مسیرهای تکراری و افزونه) کامل نیست
  - بهینه نیست
  - نسخه مبتنی بر جستجوی درخت
    - پیچیدگی زمانی: $O(b^m)$
    - پیچیدگی فضایی: $O(bm)$
    - در جستجوی برگشت کننده (backtracking) پیچیدگی فضایی به $O(m)$ کاهش پیدا می‌کند

جستجوی عمق محدود (depth-limited)

- پیشگیری از افتادن الگوریتم جستجوی عمق اول در شاخه‌های نامتناهی با در نظر گرفتن یک حد (l) برای عمق درخت
  - گره‌های در عمق l برگ در نظر گرفته می‌شوند

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

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

جستجوی عمق محدود

- ویژگی‌ها:
  - کامل است اگر $l \geq d$ باشد
  - بهینه است اگر $l > d$ **نباشد**
  - پیچیدگی زمانی: $O(b^l)$
  - پیچیدگی فضایی: $O(bl)$
- جستجوی عمق اول یک حالت خاص از جستجوی عمق محدود با $l = \infty$ است.
- تعیین مقدار درست l برای موفقیت الگوریتم ضروری است
  - یک تقریب مناسب برای l، قطر (diameter) فضای حالت است
    - حداکثر تعداد گام برای رفتن از یک حالت به هر حالت دیگر از فضای حالت

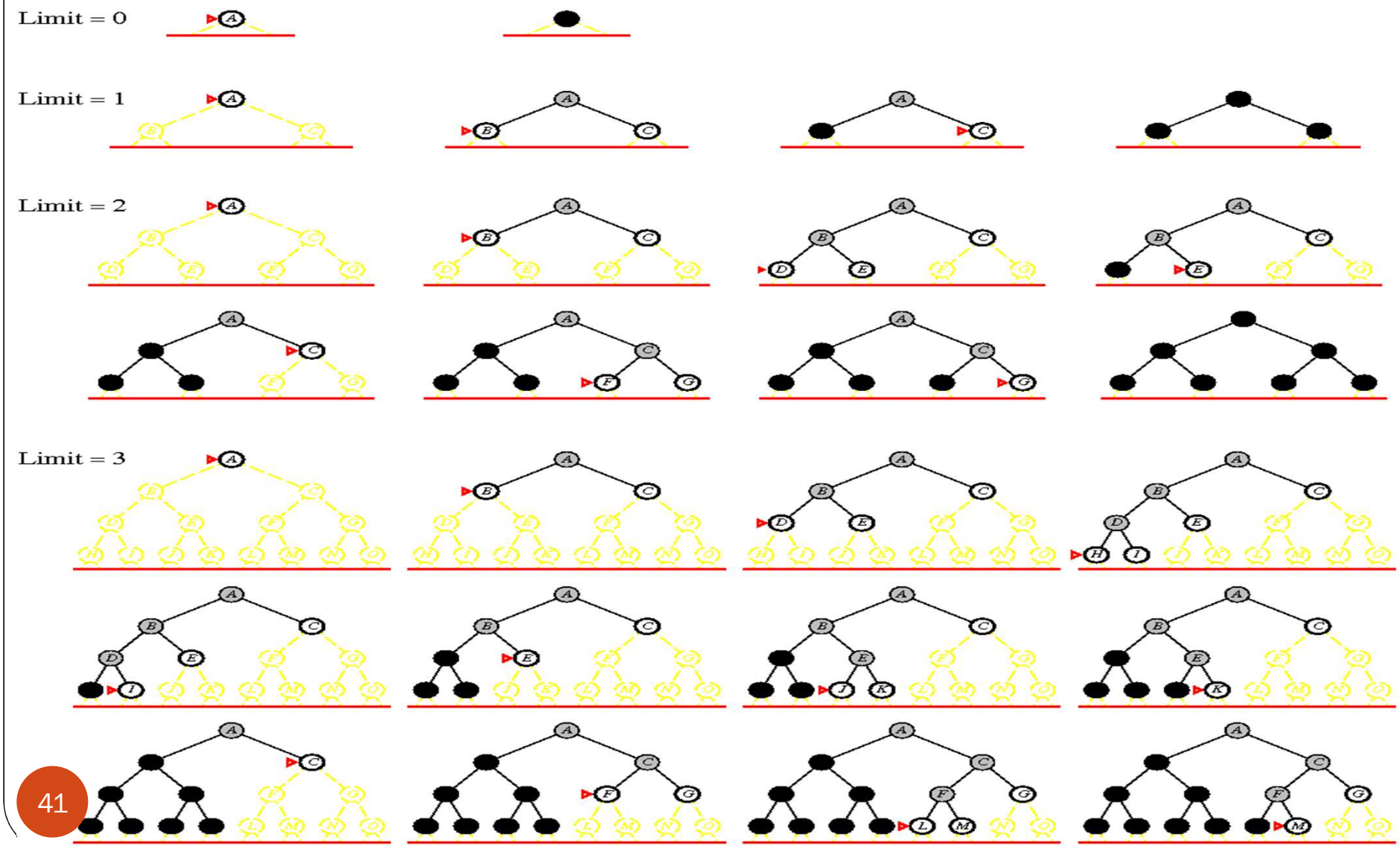
جستجوی عمق اول با افزایش مکرر عمق (iterative deepening)

- اعمال الگوریتم جستجوی عمق محدود با افزایش تدریجی عمق

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

- تولید چندباره‌ی گره‌های درخت چندان هزینه‌بر نیست
  - اکثر گره‌های یک درخت در پایین‌ترین سطح آن هستند
- ترکیب مزایای الگوریتم‌های جستجوی عرض اول و عمق اول با هم

جستجوی عمق اول با افزایش مکرر عمق

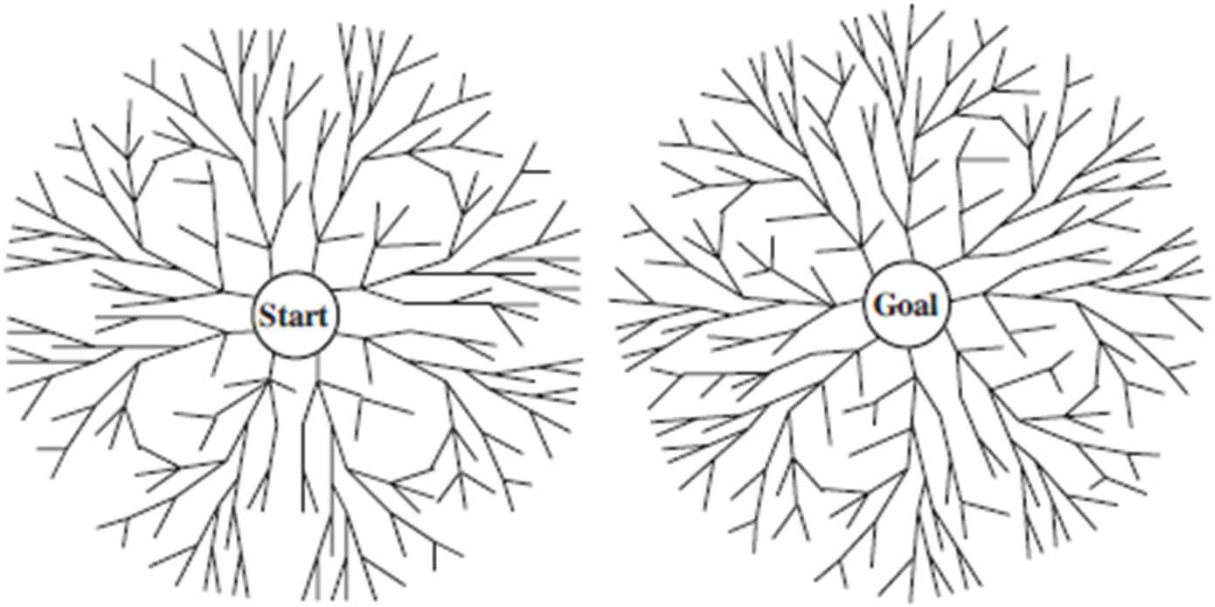


جستجوی عمق اول با افزایش مکرر عمق

- ویژگی‌ها
  - کامل است با شرایط الگوریتم جستجوی عرض اول
  - بهینه است با شرایط الگوریتم جستجوی عرض اول
  - پیچیدگی زمانی: $O(b^d)$
  - پیچیدگی فضایی: $O(bd)$
- روش جستجوی ناآگاه برگزیده
  - برای مسائلی که دارای فضای حالت بزرگ هستند و عمق هدف آنها در درخت جستجو از قبل مشخص نیست
- جستجو با افزایش مکرر طول
  - استفاده از حد بر روی هزینه مسیر بجای عمق

جستجوی دوجهتی (bidirectional)

- بکارگیری دو جستجوی همزمان
  - از حالت اولیه به سمت حالت هدف
  - از حالت هدف به سمت حالت اولیه
- انگیزه: $b^{d/2} + b^{d/2} < b^{d}$



جستجوی دوجهتی

- بجای آزمایش هدف از آزمایش اشتراک لیست مقدم دو جستجو استفاده می‌شود
- اگر دو جستجوی استفاده شده عرض اول باشند
  - کامل است (با شرایط جستجوی عرض اول)
  - بهینه است با بعضی شرایط
  - پیچیدگی زمانی: $O(b^{d/2})$
  - پیچیدگی فضایی: $O(b^{d/2})$
- نیاز به محاسبه حالت قبلی و جستجوی عقب‌رو (backward) دارد
- شروع جستجو از حالت هدف
  - معمای ۸، مسیریابی، دنیای جاروبرقی، ...
  - ۸ ملکه شطرنج

مقایسه الگوریتم‌های جستجوی ناآگاه

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]

- a: کامل است اگر ضریب انشعاب متناهی باشد
- b: کامل است اگر تمام قدم‌ها دارای هزینه‌ی مثبت باشند
- c: بهینه است اگر تمام قدم‌ها دارای هزینه برابر باشند
- d: هنگامی که در هر دو جهت از جستجوی عرض اول استفاده شود