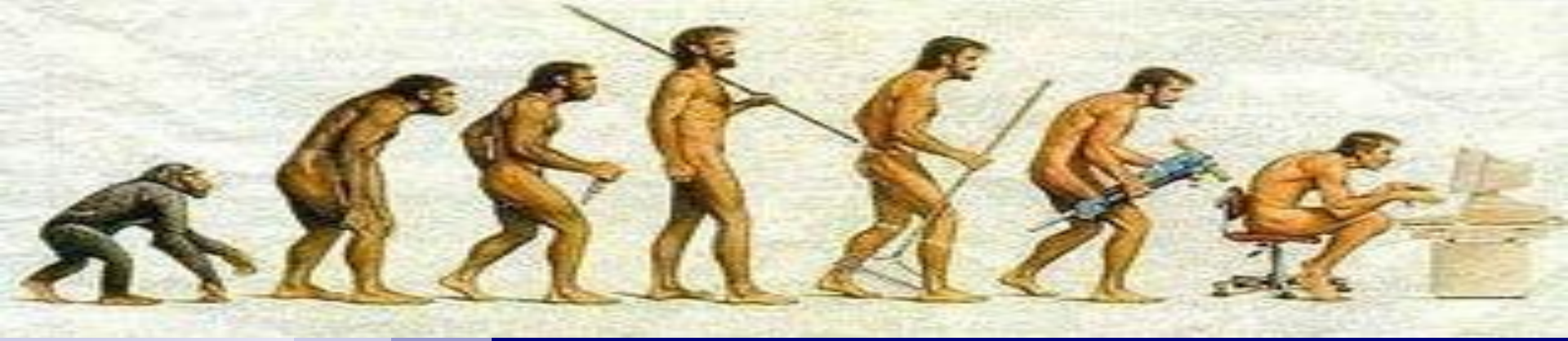


هوش مصنوعی (مقدماتی)

Artificial Intelligence

میلاد سلطانی

فصل سوم

- مقدمه
- بررسی مسأله
- جستجو های ناآگاهانه و انواع آن
- اجتناب از حالت های تکراری
- جستجو با اطلاعات ناقص
- جستجوهای آگاهانه و انواع آن

حل مسئله از طریق جستجو

Solving Problems by Searching

مقدمه

- در این فصل یک نوع از عامل های مبتنی بر هدف، بنام عامل حل مسأله مورد بررسی قرار می گیرد.
- عامل حل مسأله با پیدا کردن رشته ای از اقدامات که به حالت های مطلوب منجر می شود، تصمیم می گیرند که چه عملی را انجام دهند.
- به فرآیند یافتن رشته اقدامات رسیدن به حالت مطلوب، جستجو (Search) گفته می شود.

انواع مسأله

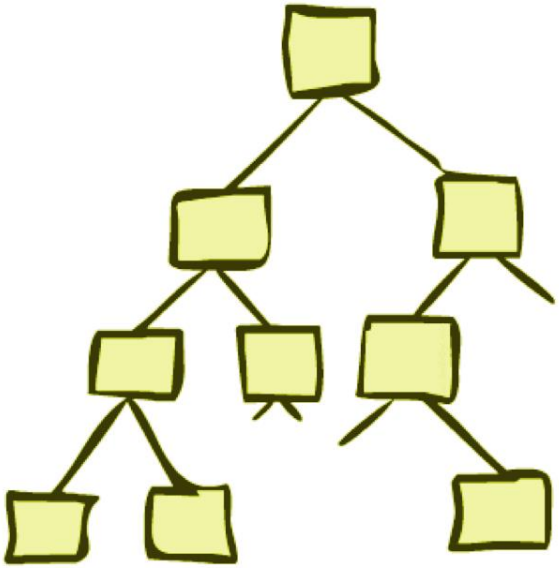
- بطور کلی عامل حل مسأله به فرم « تدوین، جستجو، اجرا » طراحی می شود.
- مسائل به چهار دسته اساسی تقسیم می شوند:
  - مسائل تک حالته (Single-State)
    - دارای محیط رؤیت پذیر و قطعی می باشند.
  - مسائل چند حالته (Multi-State)
    - دارای محیط نیمه رؤیت پذیر و قطعی است.
  - مسائل احتمالی (Contingency)
    - دارای محیط غیر قطعی هستند.
  - مسائل اکتشافی (Exploration)
    - هیچ اطلاعی در مورد اعمال خود ندارد و باید جستجو کند.

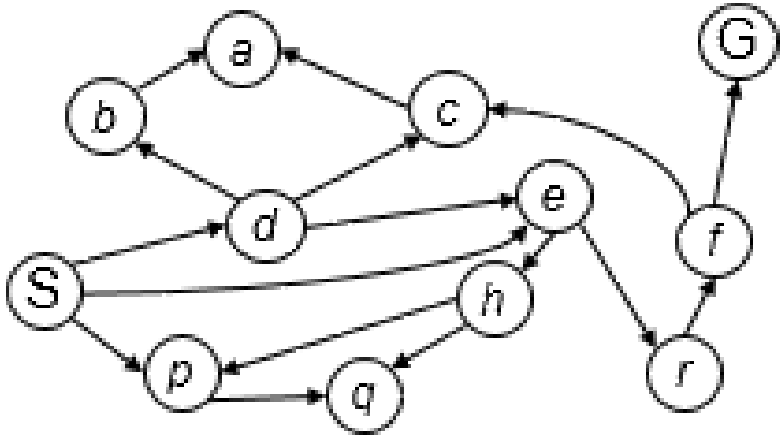
بررسی مسأله

- یک مسأله بصورت رسمی با چهار مؤلفه زیر تعریف می شود:
  - حالت ابتدایی شروع به کار عامل
  - تعریف اقدامات ممکن که در دسترس عامل هستند.
  - آزمون هدف، تعیین کننده اینکه یک حالت خاص هدف است یا خیر؟
  - تابع هزینه مسیر، که به هر مسیر یک هزینهٔ عددی نسبت می دهد.
- معیارهای ارزیابی انواع حل مسأله
  - کامل بودن (Completeness): در صورت وجود راه حل، جستجو آنرا پیدا می کند یا نه؟
  - پیچیدگی زمانی (Time Complexity): زمان لازم جستجو تا اینکه راه حل را پیدا کند.
  - پیچیدگی مکانی (Space Complexity): حافظه لازم برای انجام جستجو.
  - بهینگی (Optimality): آیا جستجو بهترین راه حل را پیدا می کند؟

جستجو

- فضای حالت حل یک مسأله یک گراف بوده، در حالیکه برای فرآیند جستجو و حل مسأله از درخت جستجو استفاده می شود.

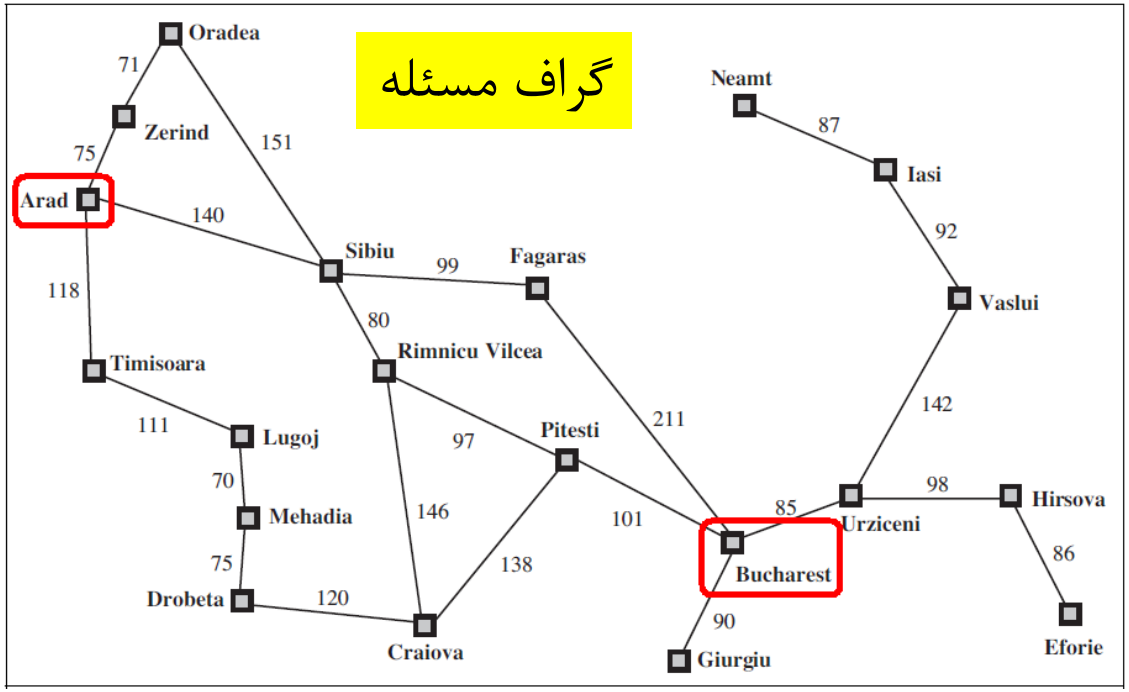


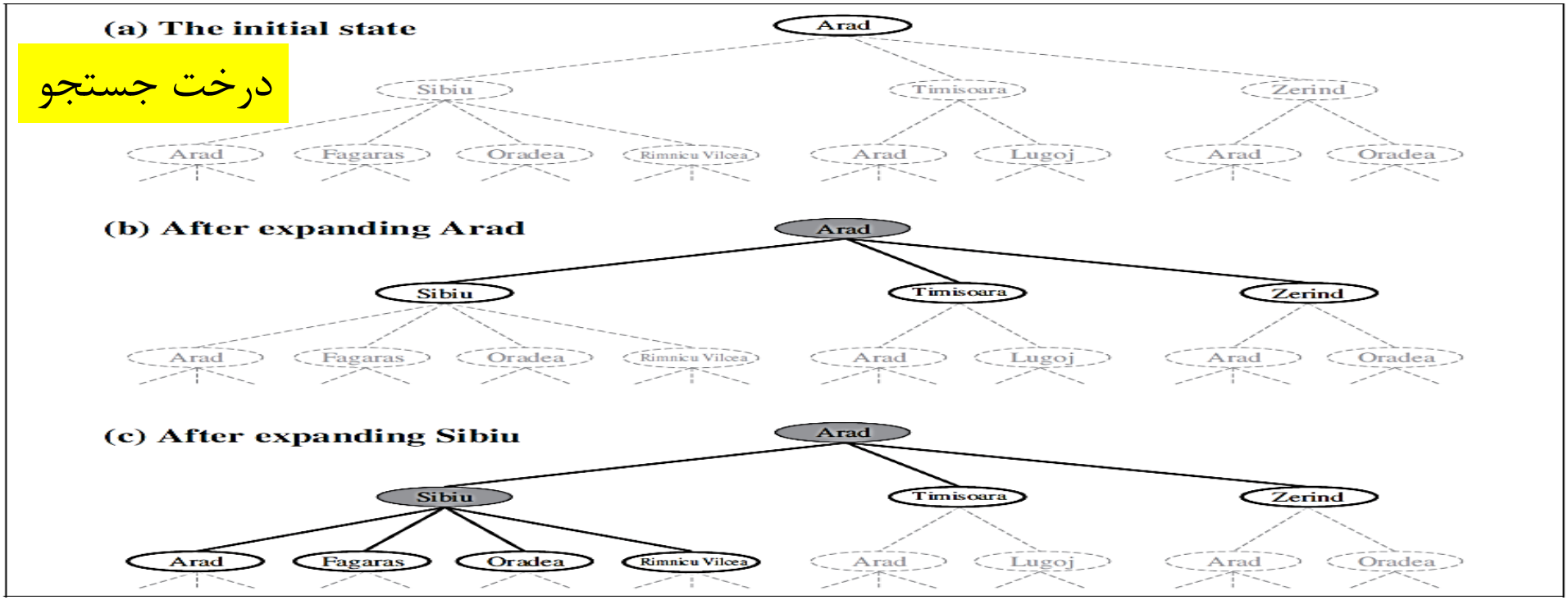


نمونه مسائل جستجو

- تعیین مسیر از شهر آراد به بخارست در کشور رومانی
- مسأله دنیای جارو برقی
- پازل هشت تایی
- مسأله هشت وزیر
- فروشنده دوره گرد
- جهت یابی روبات ها
- تعیین خودکار ترتیب مونتاژ
- ...

تعیین مسیر از شهر آراد به بخارست در کشور رومانی به همراه درخت جستجوی حل مسأله



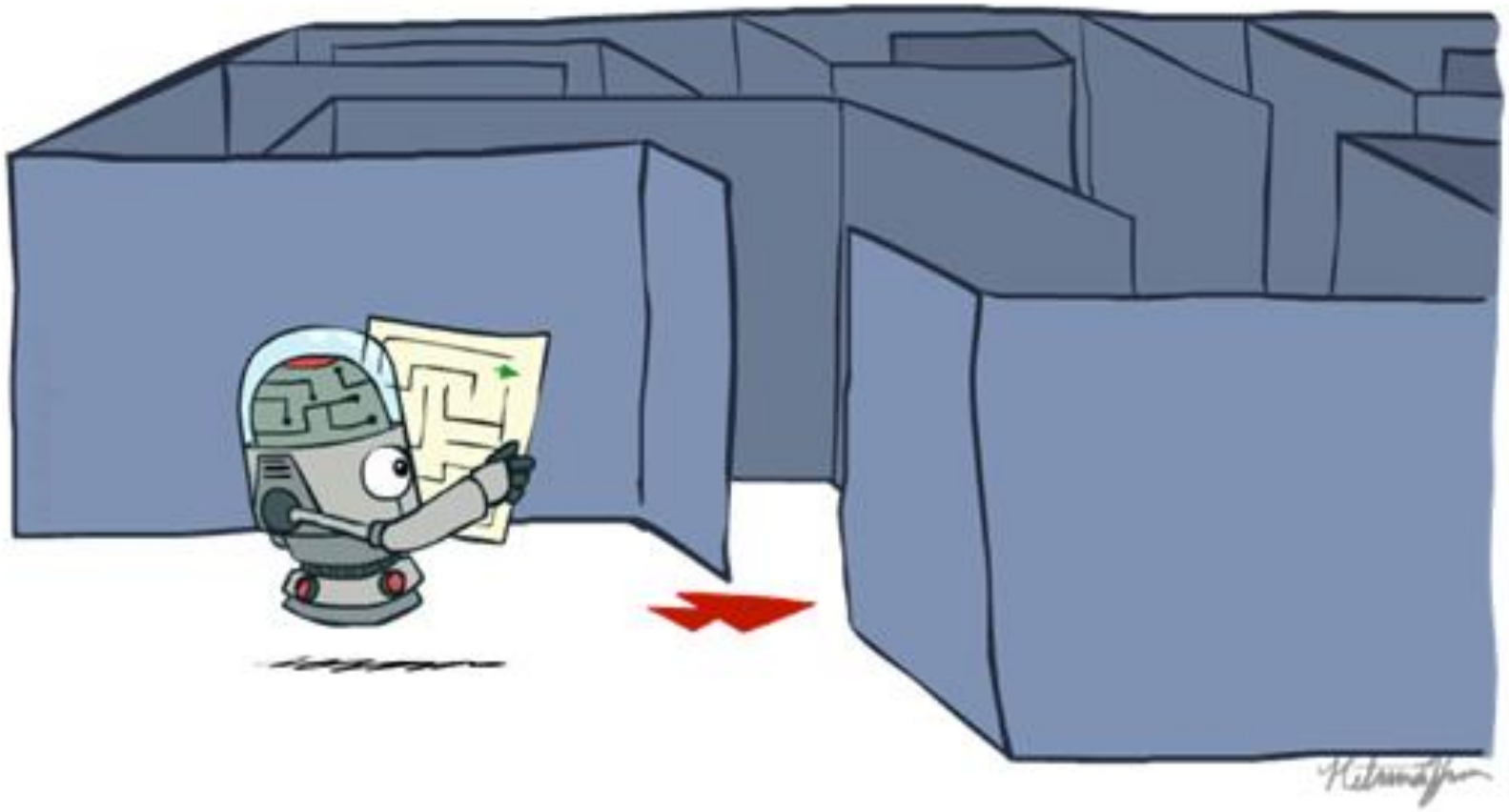


انواع جستجو

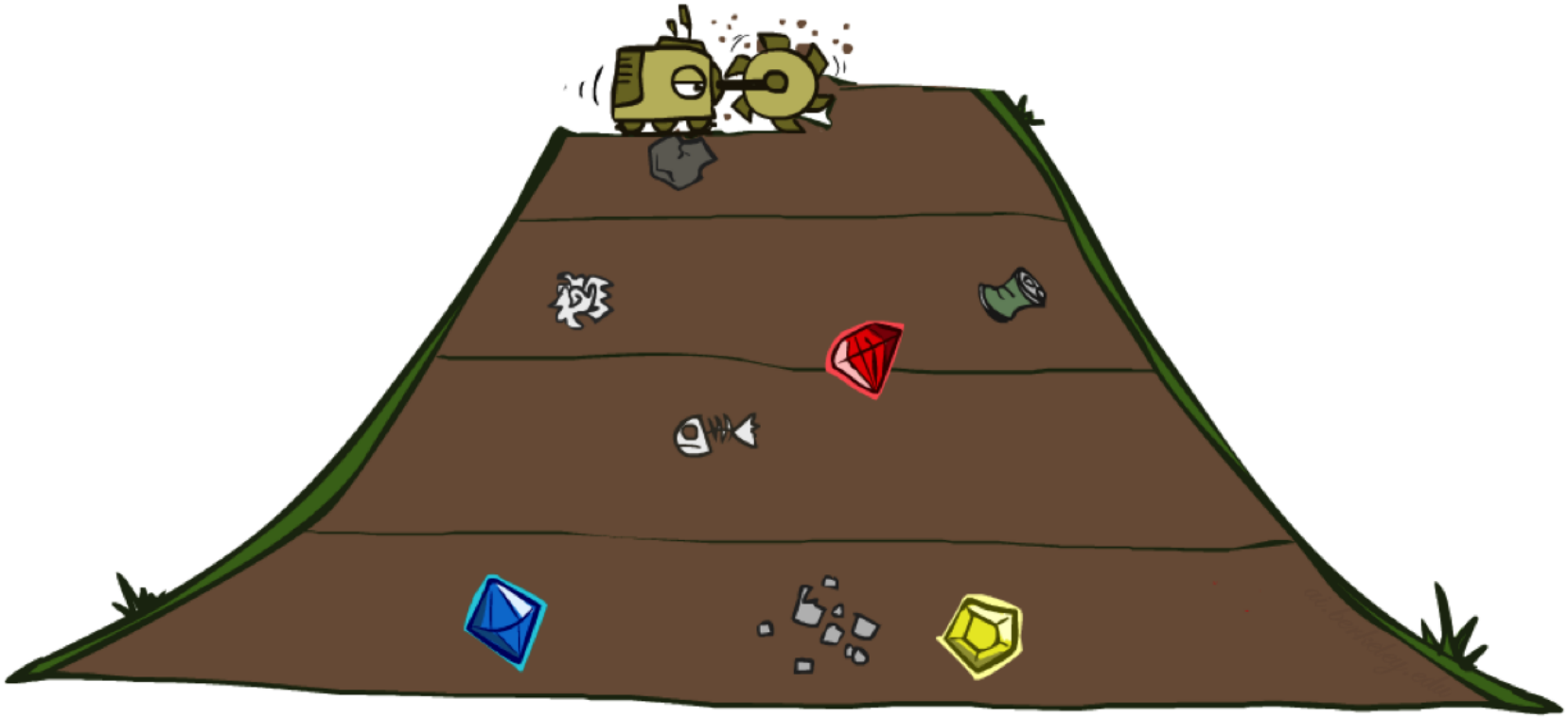
- روش های جستجو در حالت کلی به دو دسته تقسیم می شوند:

- ناآگاهانه یا کورکورانه
  - جستجوی اول سطح
  - جستجوی هزینه یکنواخت
  - جستجوی اول عمق
  - جستجوی عمق محدود
  - جستجوی عمیق شونده تکراری
  - جستجوی دو طرفه
- آگاهانه
  - جستجوی اول بهترین (حریصانه)
  - جستجوی A*
  - جستجو با حافظه محدود (IDA* و RBFS و SMA*)
  - الگوریتم های جستجوی محلی (تپه نوردی، سخت سازی شبیه سازی شده)

جستجوهای ناآگاهانه



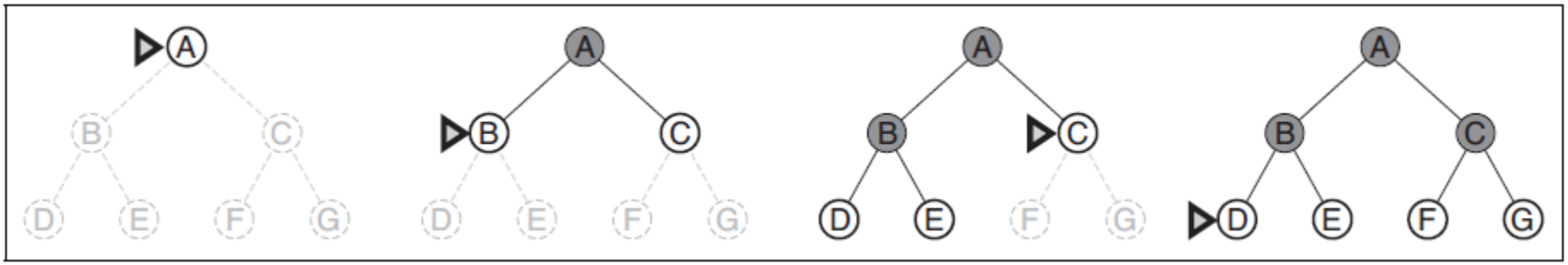
جستجوی اول سطح



جستجوی اول سطح

- در جستجوی اول سطح (Breadth-First Search) یا BFS ابتدا ریشه درخت جستجو گسترش می یابد، سپس تمام گره های فرزند ریشه گسترش می یابند و این کار تا آخر ادامه می یابد.

- بطور کلی این روش جستجو یک سطح درخت را کامل گسترش داده و جستجو می کند، سپس به سطح بعدی درخت می رود.

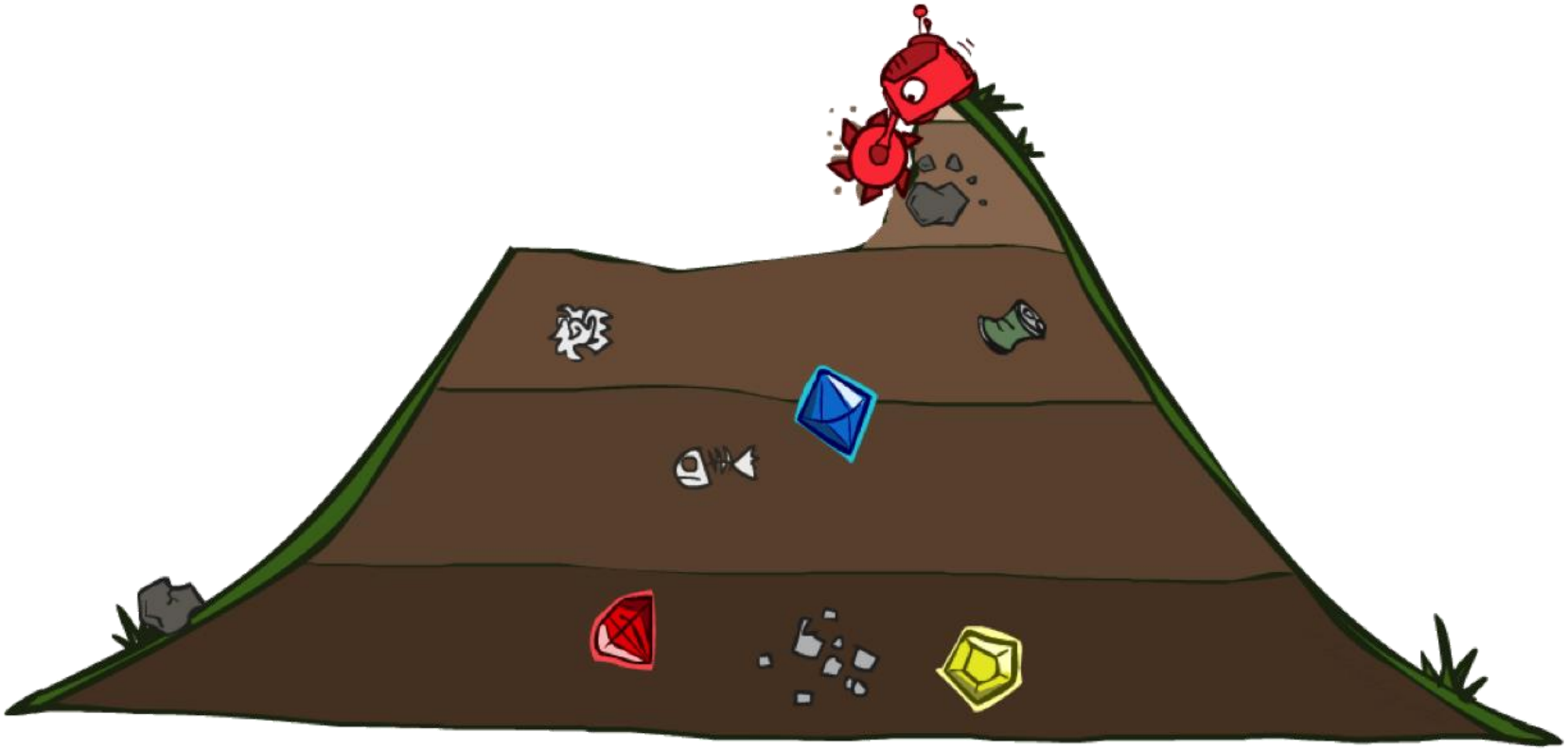


روش عملکرد جستجوی اول سطح

جستجوی اول سطح

- ویژگی های جستجوی اول سطح:
  - کامل است، یعنی حتماً هدف را پیدا کرده و آن، کم عمق ترین هدف موجود است.
  - ضرورتاً بهینه نیست، یعنی ممکن است کم عمق ترین هدف، بهینه ترین هدف نباشد.
  - اگر فاکتور انشعاب درخت جستجو b باشد و سطرها با d مشخص شوند، پیچیدگی مکانی و زمانی جستجو $O(b^d)$ خواهد بود.
  - اگر درخت جستجو پهنای کم و عمق زیاد داشته باشد، کارایی خوبی دارد.

جستجو با هزینه یکنواخت



جستجو با هزینه یکنواخت

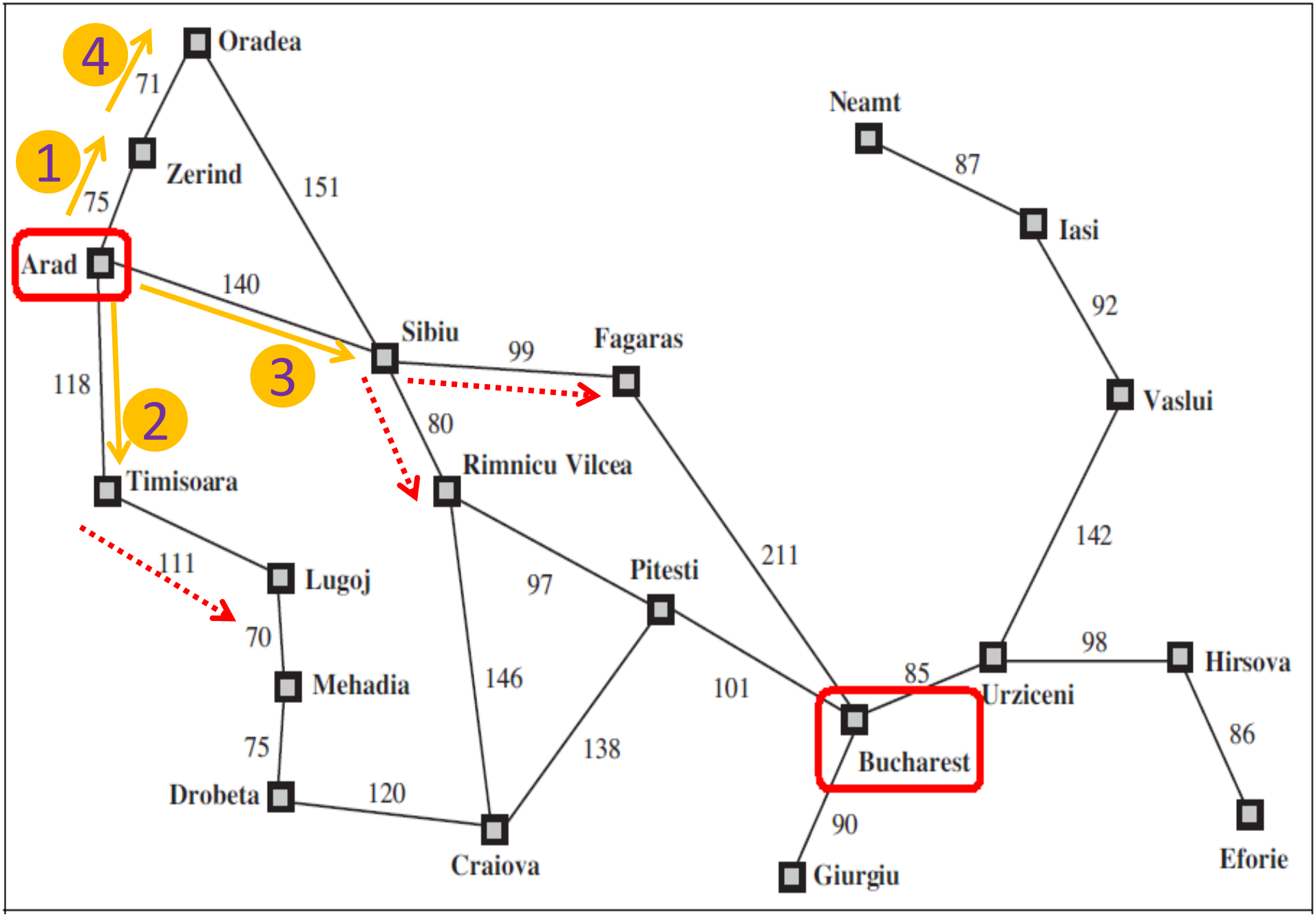
- جستجوی با هزینه یکنواخت (Uniform Cost Search) یا UCS گره n را که کم هزینه ترین گره بعدی در درخت جستجو است، گسترش می دهد.
- اگر تابع هزینه مسیر را g(n) فرض کنیم، آنگاه روش جستجوی با هزینه یکسان همان جستجوی اول سطح خواهد بود با این شرط که:

g(n) = n عمق گره

- ویژگی های جستجو با هزینه یکسان:
  - کامل و بهینه است به شرط اینکه هزینه مسیر کاهش نداشته باشد، یعنی

g(n فرزند گره) ≥ g(n)

  - پیچیگی زمانی و مکانی آن براحتی محاسبه نشده، ولی با هزینه یکسان $O(b^d)$ است.



روش عملکرد جستجوی با هزینه یکنواخت

جستجوی اول عمق

جستجوی اول عمق

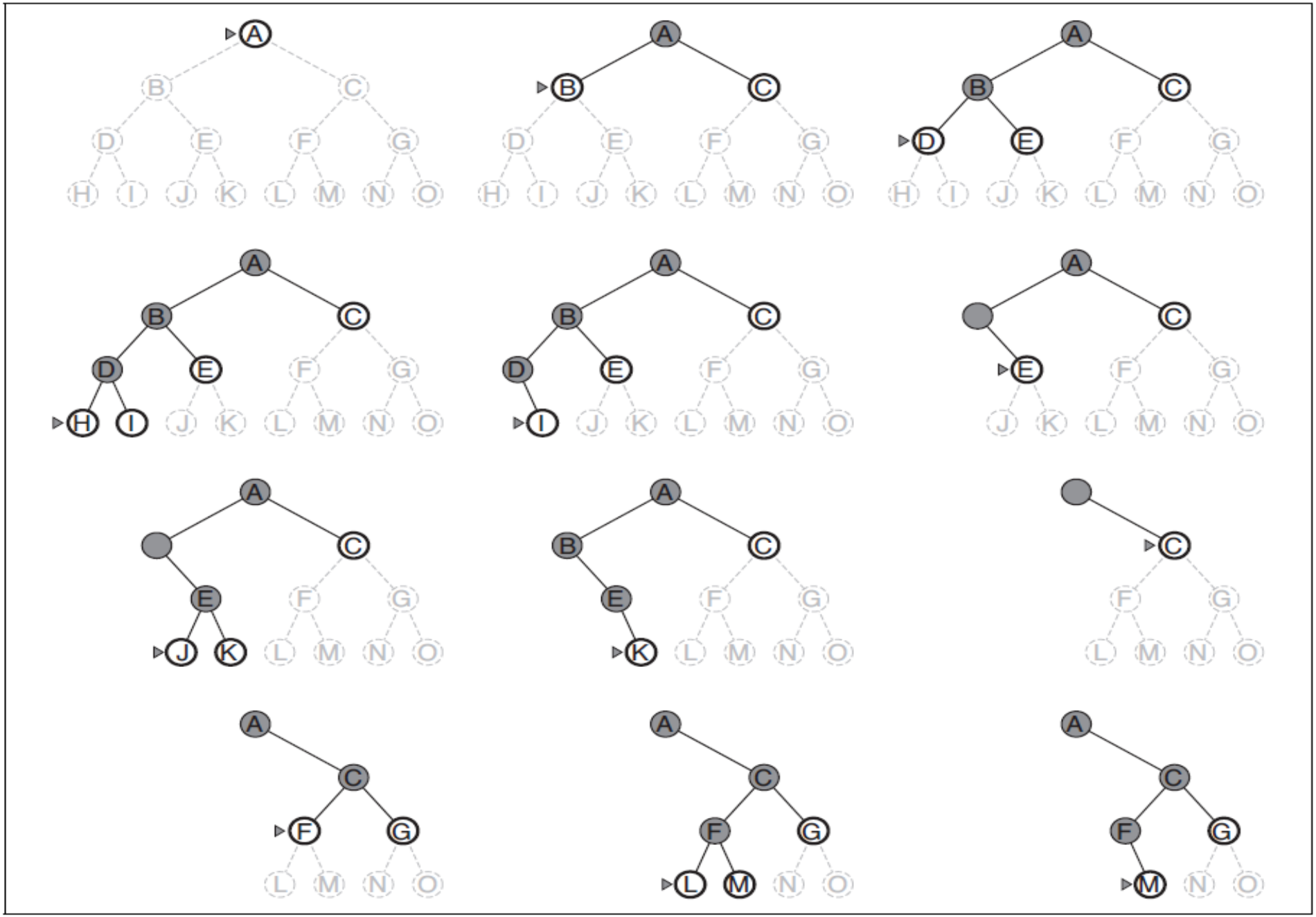
- جستجوی اول عمق (Depth-First Search) یا DFS همیشه عمیق ترین گره در لبهٔ فعلی از درخت جستجو را گسترش می دهد.

- ویژگی های جستجوی اول عمق:

  - نه کامل است و نه بهینه. چون ممکن است در یک مسیر اشتباه هنگام حرکت عمقی گیر کند. لذا در جستجو با درخت های با عمق نامحدود و بزرگ استفاده نمی شود.

  - اگر فاکتور انشعاب درخت جستجو b باشد و حداکثر عمق با m مشخص شود، پیچیدگی مکانی جستجو $O(bm)$ و پیچیدگی زمانی جستجو $O(b^d)$ خواهد بود.

  - در حل مسائلی که راه حل های زیادی دارند، روش اول عمق سرعت خوبی دارد.



روش عملکرد جستجوی اول عمق

جستجوی عمقی محدود

- جستجوی عمقی محدود (Depth-Limited Search) یا DLS با ایجاد یک برش روی درخت جستجو، از به دام افتادن روش عمقی در درختان نامحدود پیشگیری می کند.

- ویژگی های جستجوی عمقی محدود:

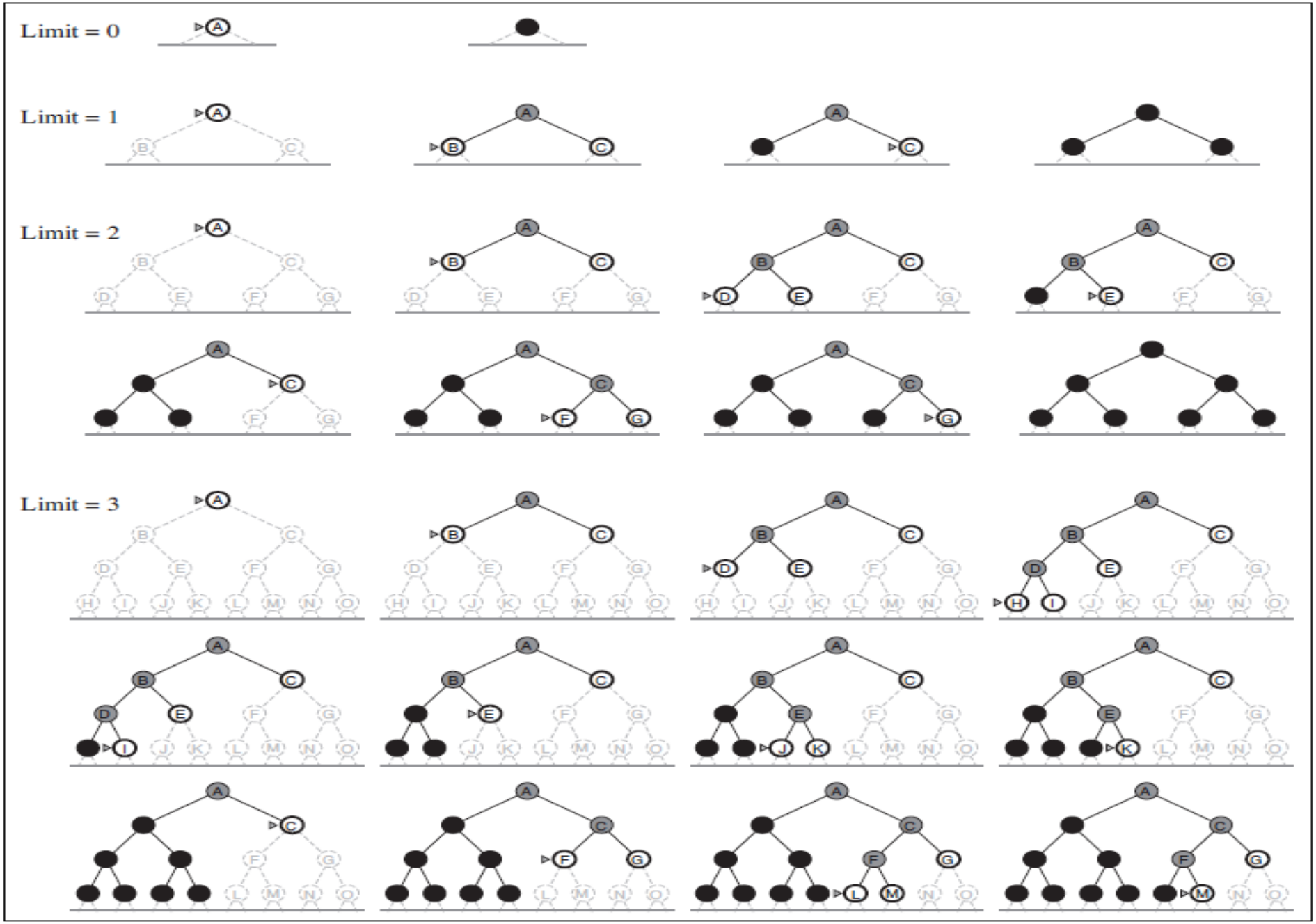
  - اگر هدف جستجو در سطوح قبل از سطح برش یا در همان سطح برش وجود داشته باشد، جستجو کامل خواهد بود ولی بهینه نمی باشد.

  - اگر فاکتور انشعاب درخت جستجو b باشد و عمق برش درخت با L مشخص شود، پیچیدگی مکانی جستجو $O(bL)$ و پیچیدگی زمانی جستجو $O(b^L)$ خواهد بود.

جستجوی عمیق شونده تکراری

- جستجوی عمیق شونده تکراری (Iterative Deepening Search) یا IDS برای جلوگیری از مشکل روش عمقی محدود، تمام عمق های ممکن برای جستجوی عمقی محدود را امتحان می کند.

- ویژگی های جستجوی عمیق شونده تکراری:
  - مزیت سرعت روش جستجوی اول عمق و کامل بودن جستجوی اول سطح را دارا است.
  - اگر هزینه مسیر برابر با عمق گره باشد، این روش بهینه خواهد بود.
  - پیچیدگی مکانی جستجو $O(bd)$ و پیچیدگی زمانی جستجو $O(b^d)$ خواهد بود.
  - وقتی فضای جستجو بزرگ و عمق راه حل مجهول می باشد، روش مناسبی خواهد بود.



روش عملکرد جستجوی عمیق شونده تکراری

جستجوی دو طرفه

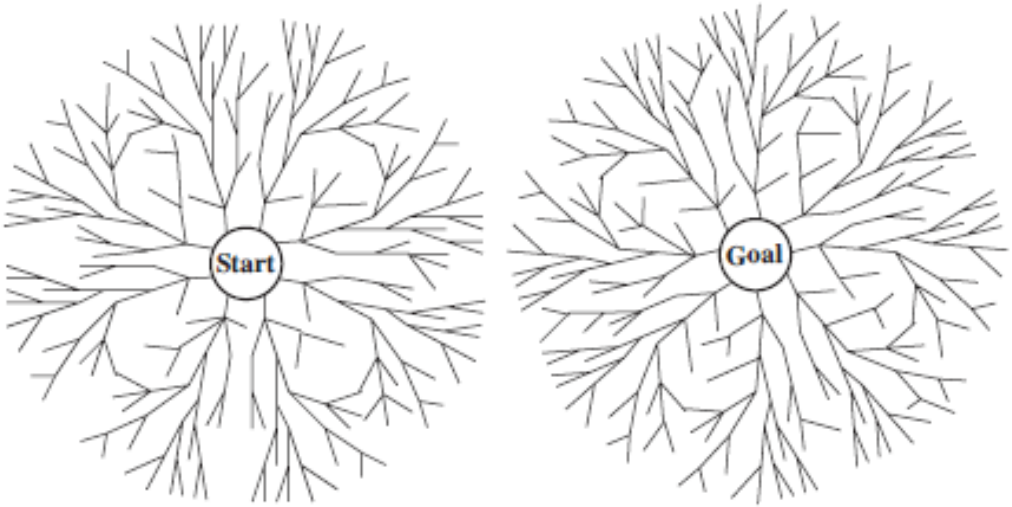
- جستجوی دو طرفه (Bidirectional Search) یا BS همزمان از حالت شروع بصورت روبه جلو و از سمت هدف رو به عقب عمل می کند و زمانیکه این دو جستجو به هم برسند، به اتمام میرسد.

- در این جستجو یکی از جستجوها یا هردوی آنها قبل از گسترش گره بعدی بررسی می کند که آیا این گره در جستجوی دیگر ملاقات شده است یا نه؟ اگر چنین بود یک راه حل پیدا شده است.

- در این جستجو باید حداقل یکی از جستجوها، گره های گسترش یافته خود را در حافظه ذخیره کند.

جستجوی دو طرفه

- ویژگی های جستجوی دوطرفه:
  - اگر یکی از دو جستجو از نوع اول سطح باشد این روش می تواند کامل باشد ولی در حالت های دیگر الزاماً بهینه یا کامل نیست.
  - پیچیدگی زمانی و مکانی جستجو $O(b^{d/2})$ خواهد بود.



اجتناب از حالت های تکراری

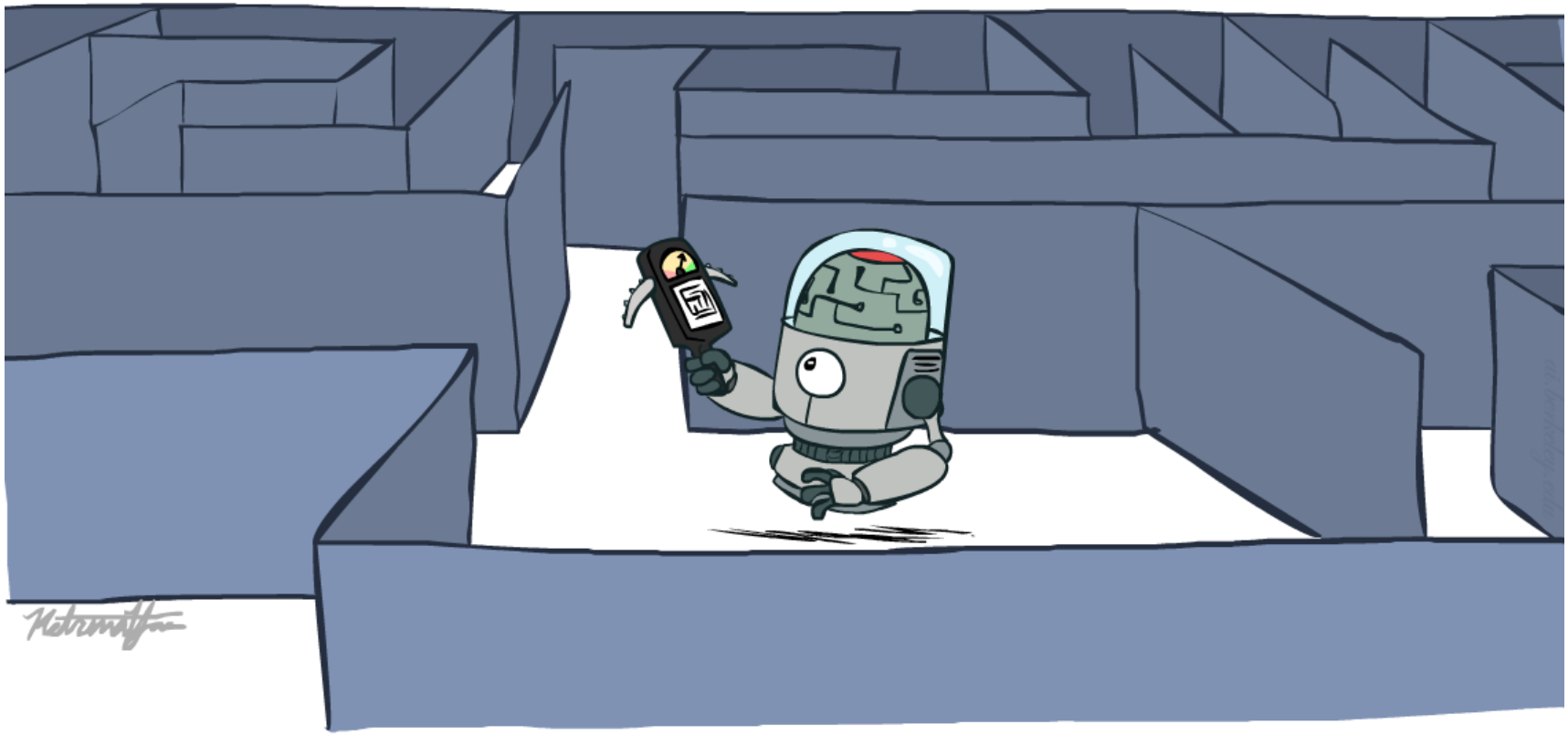
- در بسیاری از مسائل از جمله مسائل دارای اقدامات معکوس پذیر، حالت های تکراری غیر قابل اجتناب هستند و درخت جستجوی این مسائل نامتناهی خواهد بود.

- به روش های زیر می توان اندازه درخت جستجو را محدود کرد:
  - به گره ای که هم اکنون از آن آمده اید بر نگردید.
  - از ایجاد سیکل (دور) پرهیز کنید.
  - حالتی که قبلاً تولید شده است، مجدداً تولید نکنید.

جستجو با اطلاعات ناقص

- انواع مختلف نقص (کامل نبودن) به سه نوع مسأله منجر می شود:
  - مسائل بدون حسگر
    - دنیا کاملاً رؤیت پذیر نیست و عامل بجای یک حالت، مجموعه حالت هایی که ممکن است در آنها قرار گیرد (حالت باور) را استدلال می کند.
  - مسائل اقتضایی
    - محیط نیمه رؤیت پذیر یا اقدامات غیر قطعی است.
    - عامل پس از هر اقدام، اِدراکات و اطلاعات جدید فراهم می کند و هر اِدراک، اقتضایی را تعریف می کند که باید برای آن برنامه ریزی کرد.
  - مسائل اکتشافی
    - حالت ها و اقدامات محیط ناشناخته بوده و عامل باید آنها را کشف کند.
    - این مسائل حد نهایی مسائل اقتضایی هستند.

جستجوهای آگاهانه



جستجوی اول بهترین

- در جستجوی اول بهترین (Best-First Search) یا BFS یک تابع ارزیابی مورد استفاده قرار می گیرد. بر این اساس گره ای که بهترین ارزیابی را داشته باشد، گسترش خواهد یافت.

- نام این روش اغراق آمیز است. چون اگر واقعاً بهترین گره را گسترش دهیم، دیگر یک جستجو نخواهد بود.

- یک خانواده کامل از جستجوی های اول بهترین دارای تابع ارزیابی هیوریستیک (Heuristic) یا $h(n)$ هستند:

$h(n)$: هزینه تخمینی ارزان ترین مسیر از گره n تا گره هدف

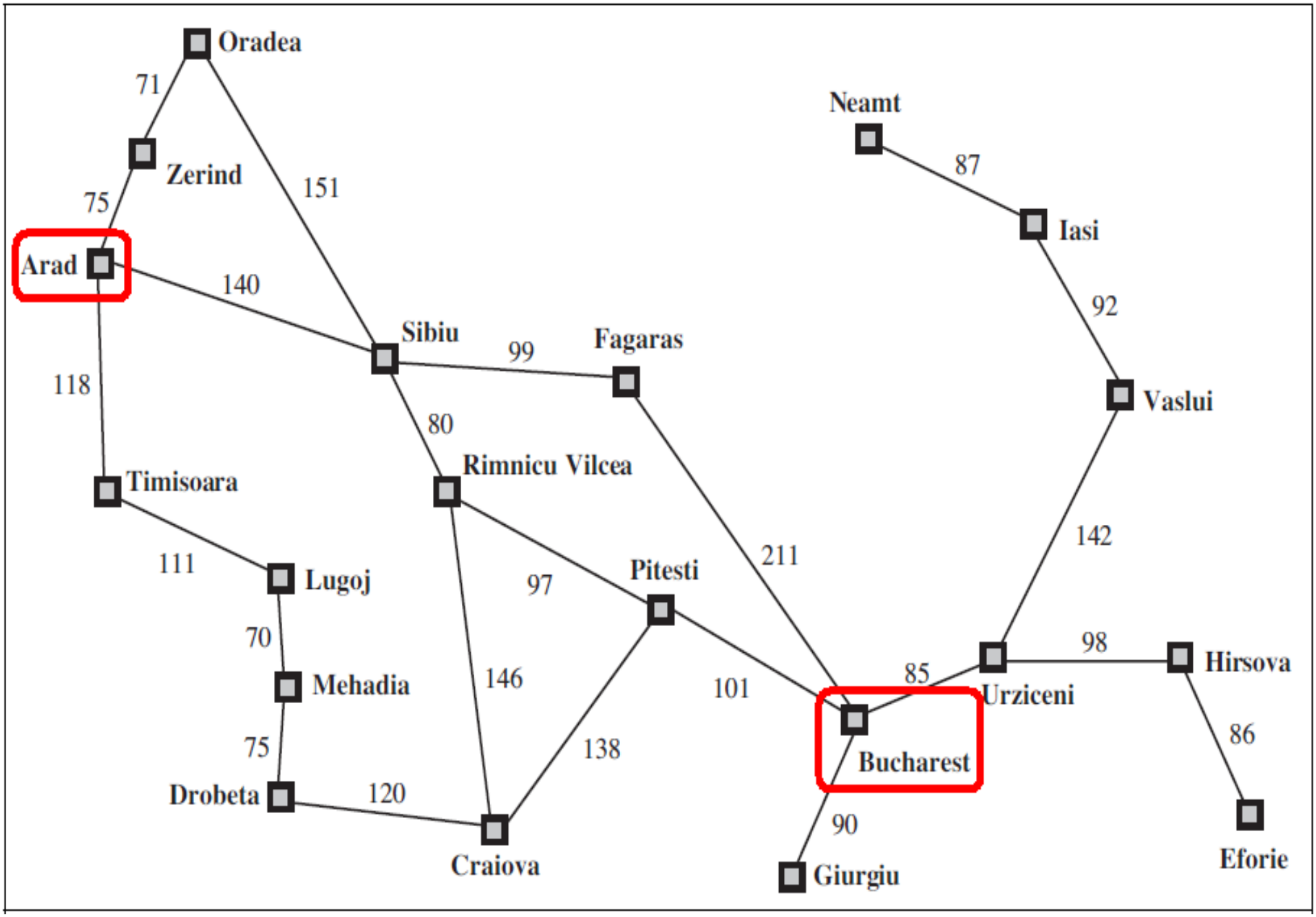
جستجوی اول بهترین حریصانه

جستجوی اول بهترین حریصانه

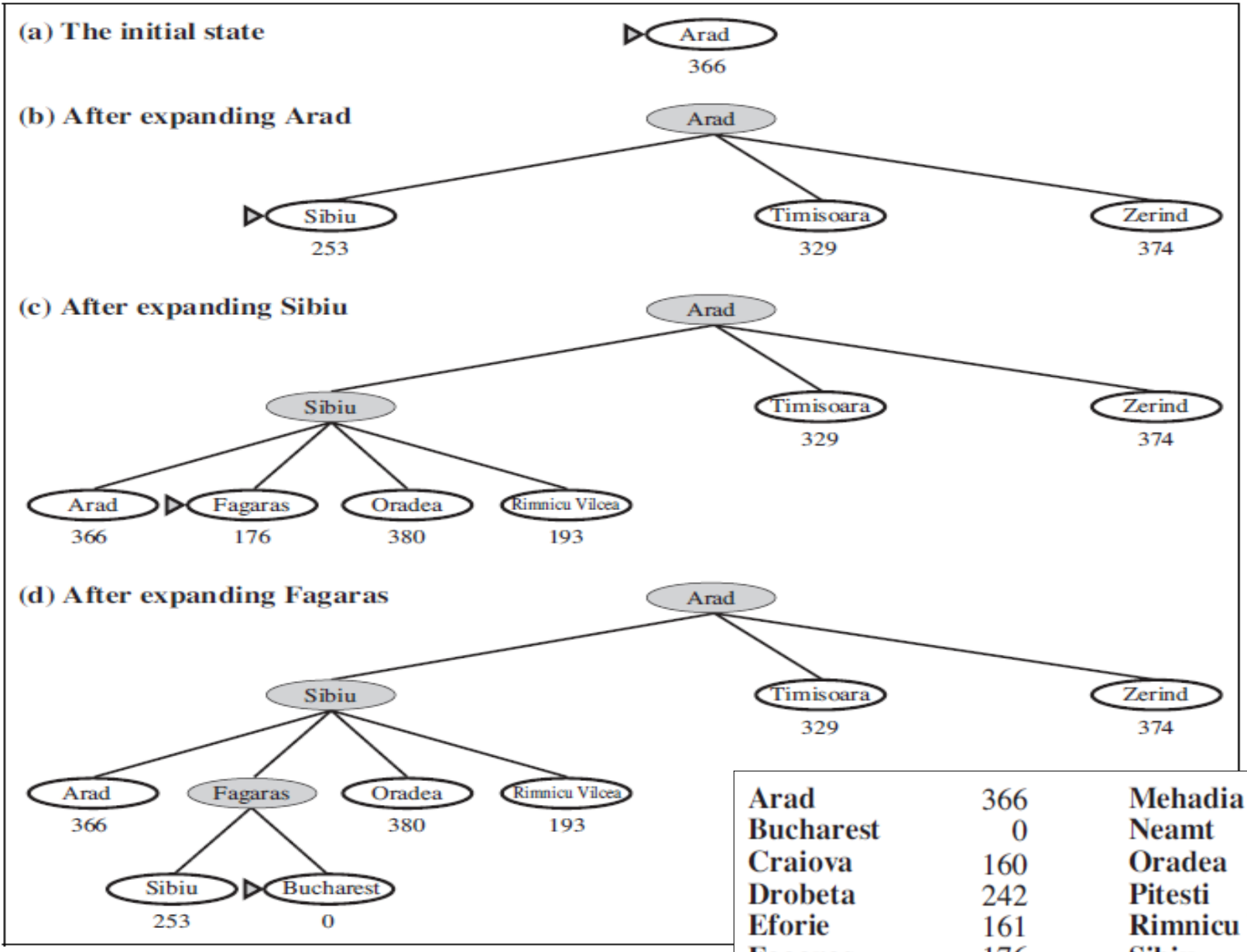
- جستجوی حریصانه (Greedy Search) سعی می کند نزدیکترین گره به هدف را گسترش دهد، بدلیل اینکه احتمال زیادی دارد که سریعترین راه حل باشد.

- ویژگی های جستجوی حریصانه:
  - از نظر عملکرد به جستجوی عمقی شباهت دارد.
  - اغلب کارآیی خوبی دارد ولی مشکوک به شروع های نادرست است.
  - همیشه راه حل های بهینه را پیدا نمی کند چون توجهی به هزینه مسیر g(n) ندارد.
  - این جستجو کامل نیست.
  - پیچیدگی زمانی و مکانی آن $O(b^m)$ می باشد که m حداکثر عمق فضای جستجو است.

نکته! پیچیدگی زمانی و مکانی جستجو به نوع مسأله و کیفیت تابع هیوریستیک بستگی دارد.



گراف مسئله آراد به بخارست در رومانی



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

روش عملکرد جستجوی حریصانه

جستجوی A^*

جستجوی A^*

- شناخته شده ترین شکل جستجوی اول بهترین است که ترکیبی از دو روش جستجوی حریصانه و جستجو با هزینه یکسان می باشد.

- این جستجو بر اساس تابع $f(n)$ گره ها را برای رسیدن به هدف ارزیابی خواهد کرد:

$$f(n) = g(n) + h(n)$$

  - $g(n)$ هزینه رسیدن به گره n و $h(n)$ هزینه رسیدن از گره n به هدف

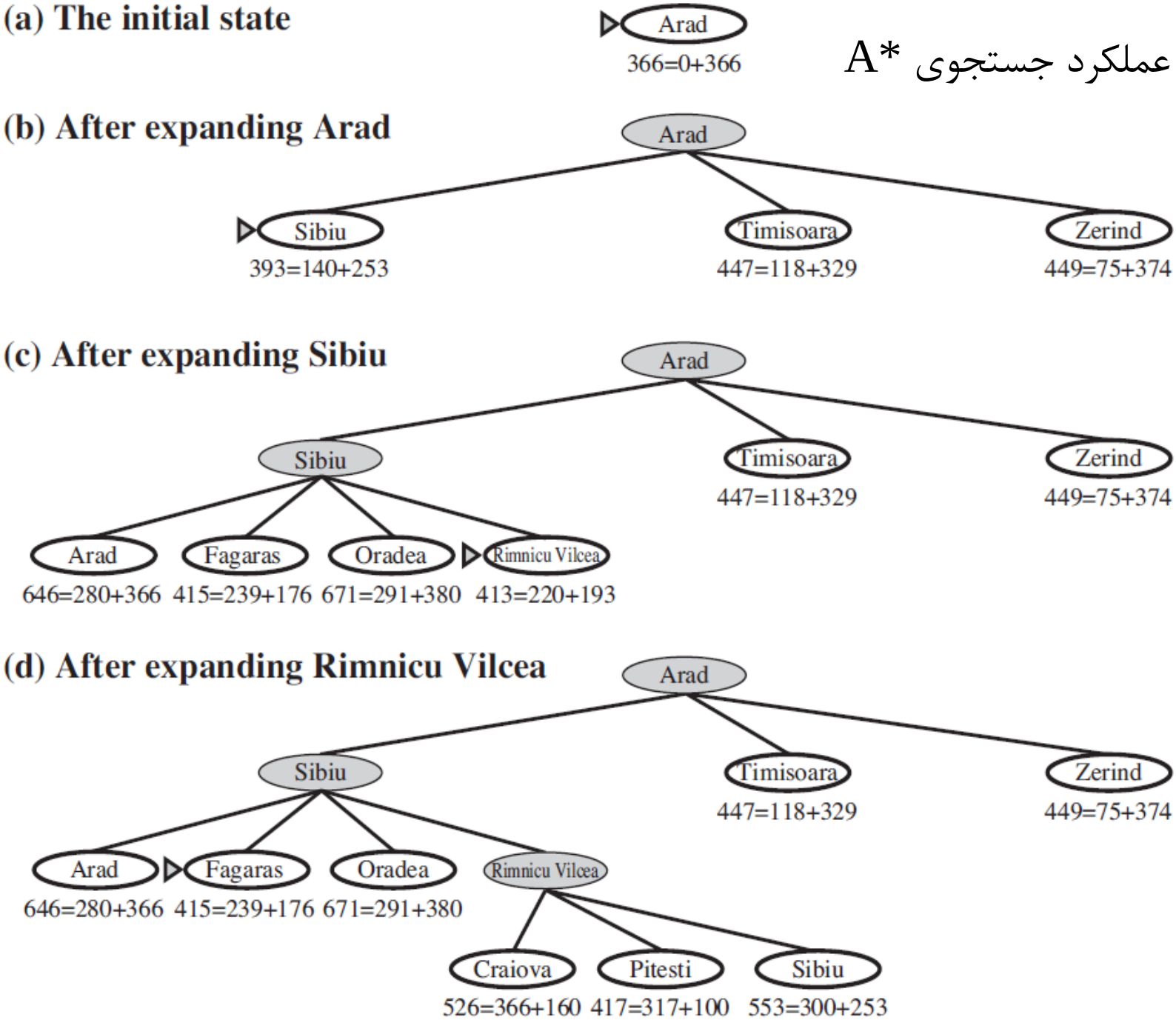
- با توجه به رابطه بالا می توان تعریف $f(n)$ را بصورت زیر بیان کرد:

$f(n)$ = هزینه تخمین زده شده برای ارزان ترین راه حل از طریق گره n

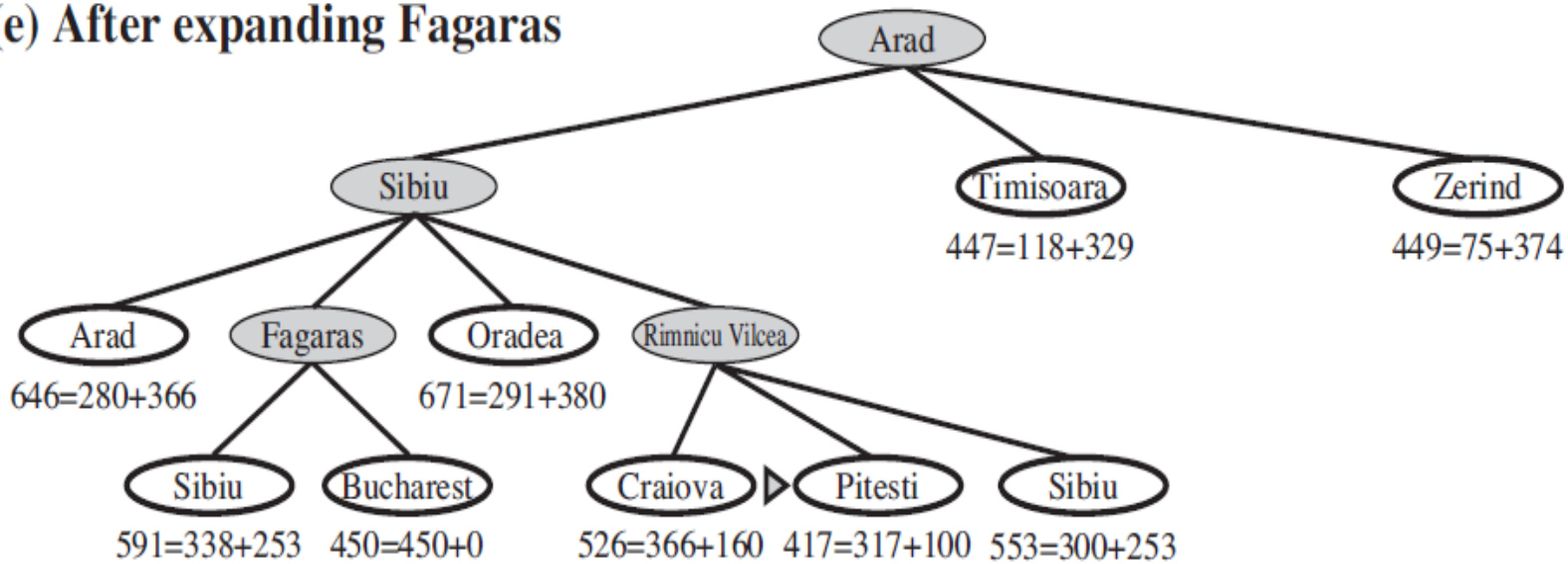
جستجوی A*

- این جستجو زمانی بهینه و کامل است که $h(n)$ یک تابع هیوریستیک **قابل قبول و سازگار** باشد.
  - تابع هیوریستیک قابل قبول است به شرط اینکه هرگز هزینهٔ رسیدن به هدف را بیش از اندازه واقعی برآورد نکند.
  - تابع هیوریستیک سازگار (یکنوا) است که در طول هر مسیر از ریشه، هزینه تابع $f(n)$ هرگز کاهش پیدا نکند.
- جستجوی **A*** با تابع هیوریستیک قابل قبول از هر الگوریتم بهینه ای تعداد گره کمتری ایجاد کند.
- این جستجو معمولاً با زمان مشکلی ندارد، بلکه بطور معمول دچار کمبود حافظه می شود.

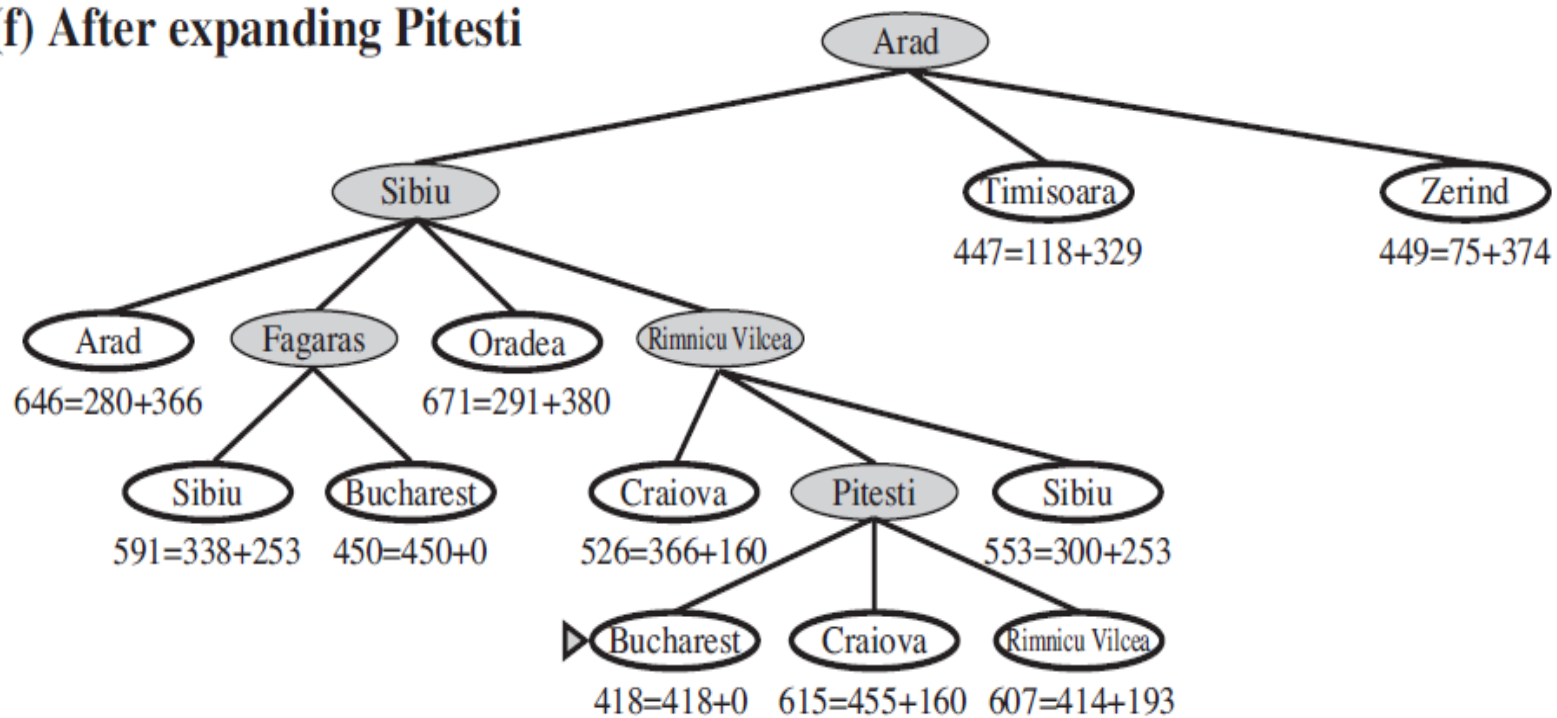
روش عملکرد جستجوی A*



(e) After expanding Fagaras

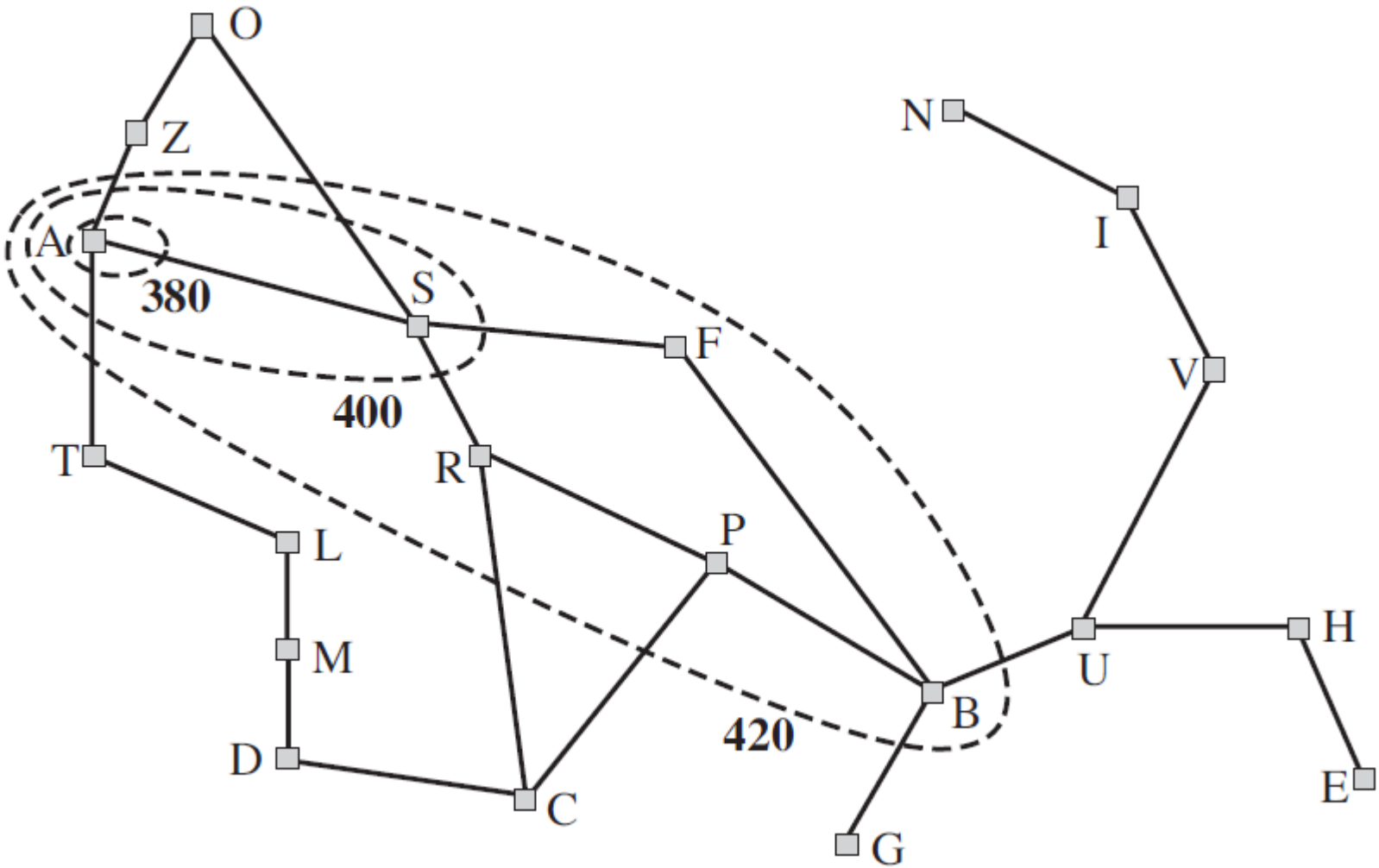


(f) After expanding Pitesti



جستجوی IDA*

- جستجوی A* عمیق شونده تکراری (Iterative Deepening A*) یا IDA* در واقع جستجوی A* با حافظه محدود می باشد.
- تفاوت این جستجو با روش عمیق شونده تکراری استاندارد اینست که برش محدوده جستجو بوسیله تابع $f(n)$ انجام می شود و نه عمق محدوده!
- در جستجوی IDA* در هر مرحله، مقدار برش، کوچکترین مقدار $f(n)$ میان تمام گره هایی است که از $f(n)$ گره های برش تکرار قبلی بیشتر باشند. f_{limit}
- ویژگی های جستجو IDA*:
  - با شرایط جستجوی A* این جستجو نیز کامل و بهینه است.
  - پیچیدگی حافظه آن $O(bd)$ است و پیچیدگی زمانی آن به تعداد مقادیر تابع هیوریستیک وابسته است.



نقشه رومانی با محدوده های برش f=380 و f=400 و f=420 که شهر آراد حالت اولیه جستجو در این نقشه می باشد. گره هایی که داخل هر محدوده هستند هزینه تابع f کوچکتر یا مساوی مقدار f محدوده دارند.

جستجوی اولین بهترین بازگشتی

- جستجوی اولین بهترین بازگشتی (Recursive Best-First Search) یا RBFS مشابه جستجوی اول عمق بازگشتی است با این تفاوت که برای ادامه دادن مسیر همیشه گره دارای کمترین *f(n)* را انتخاب می کند.
  - این جستجو برای انتخاب گره بعدی همیشه خود را از مقدار *f(n)* بهترین مسیر جایگزین قابل دسترس از تمام اجداد گره فعلی مطلع نگه می دارد.
  - اگر گره فعلی از این محدوده فراتر رود، تابع بازگشتی آنرا به عقب برگردانده تا روی یک مسیر جایگزین دیگر قرار گیرد.
  - در حال بازگشت به عقب، مقدار *f(n)* هر گره در طول مسیر را به بهترین مقدار *f(n)* فرزندانش تغییر می دهد.

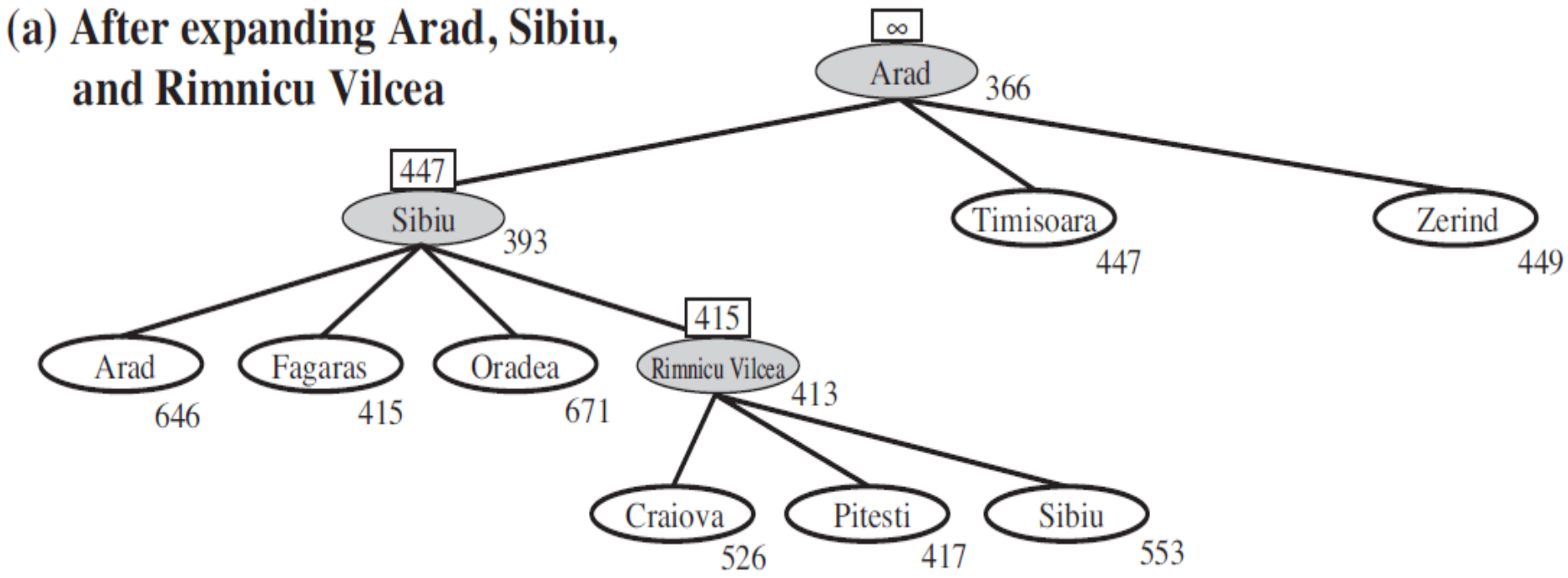
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RFBS( problem, node, f_limit) return a solution or failure and a new f-cost limit
    if GOAL-TEST[problem](STATE[node]) then return node
    successors ← EXPAND(node, problem)
    if successors is empty then return failure, ∞
    for each s in successors do
        f[s] ← max(g(s) + h(s), f[node])
    repeat
        best ← the lowest f-value node in successors
        if f[best] > f_limit then return failure, f[best]
        alternative ← the second lowest f-value among successors
        result, f[best] ← RBFS(problem, best, min(f_limit, alternative))
        if result ≠ failure then return result
```

الگوریتم جستجوی RBFS

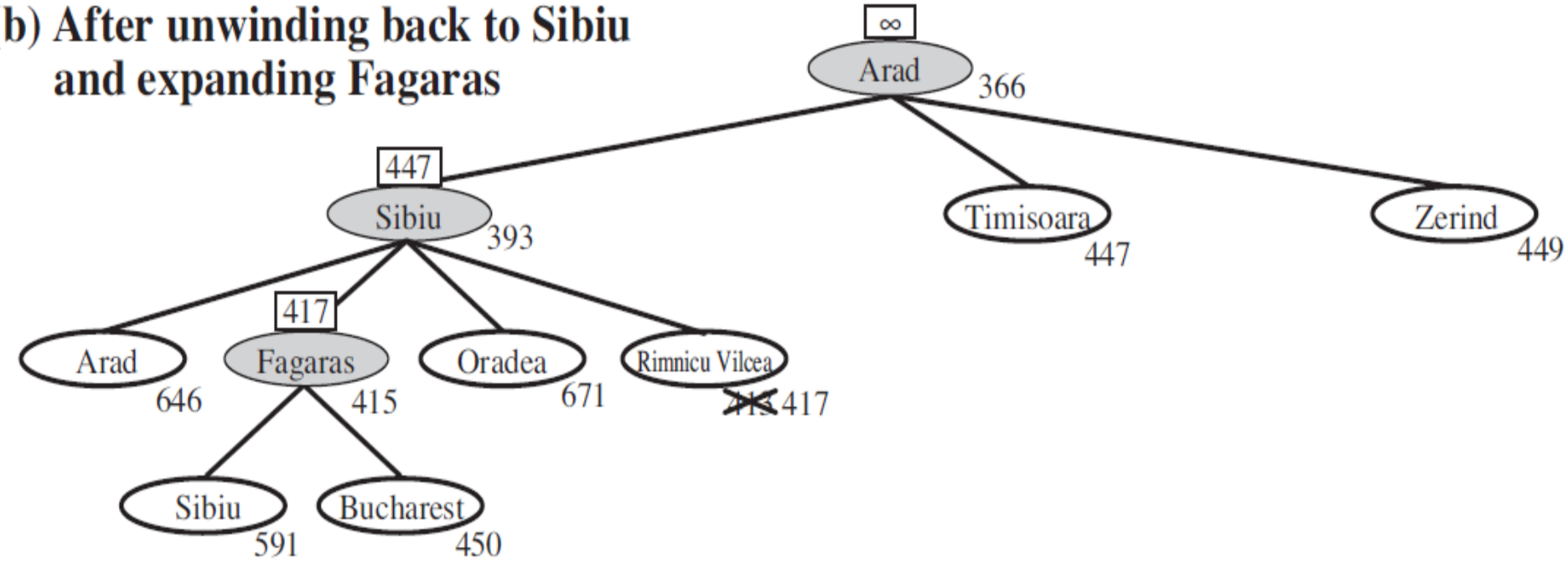
جستجوی اولین بهترین بازگشتی

- ویژگی های جستجوی RBFS :
  - بهینه است اگر تابع هیوریستیک قابل قبول و یکنوا باشد.
  - پیچیدگی حافظه آن $O(bd)$ است ولی بیان پیچیدگی زمانی آن مشکل می باشد.
  - نمی تواند از حافظه بخوبی استفاده کند.

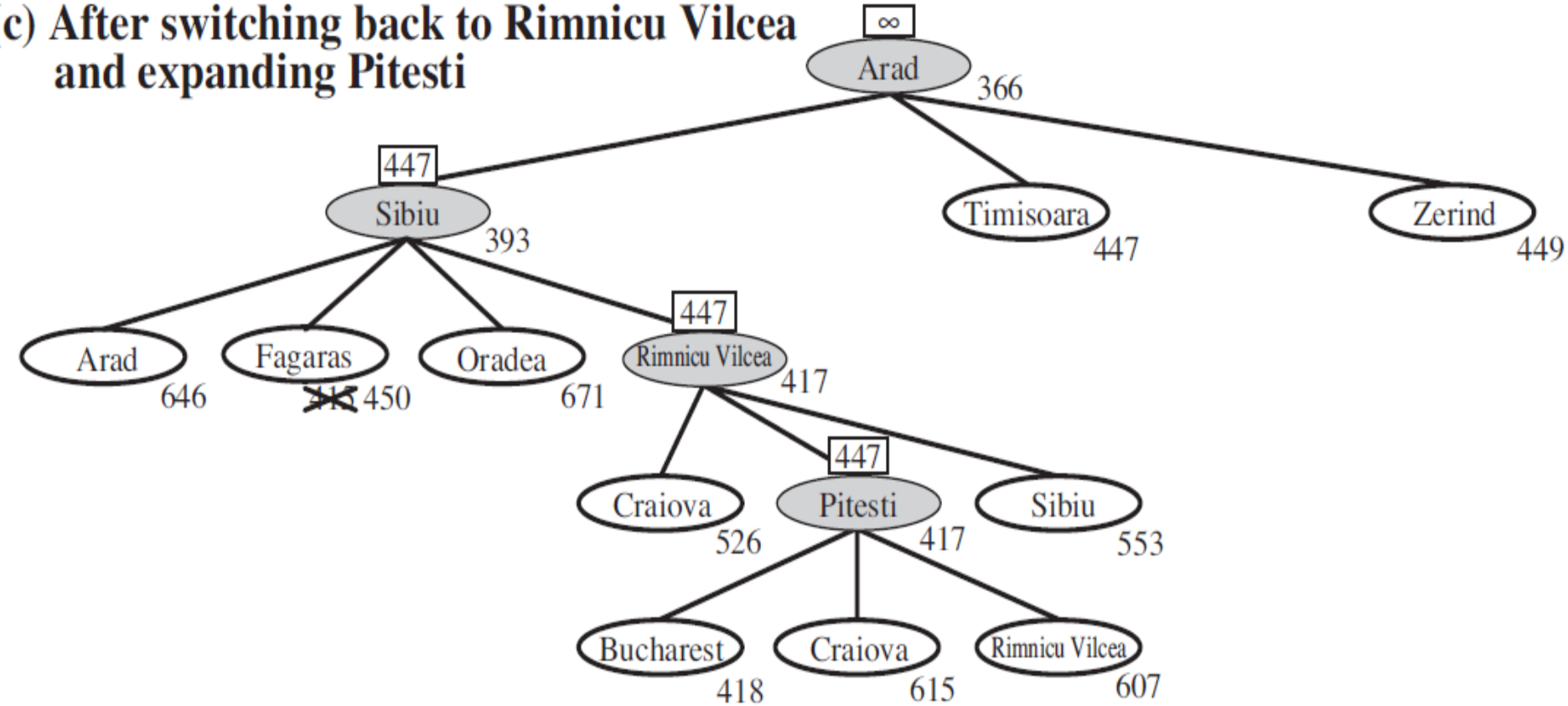


روش عملکرد جستجوی RBFS

(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



جستجوی A* مقید به حافظهٔ ساده شده

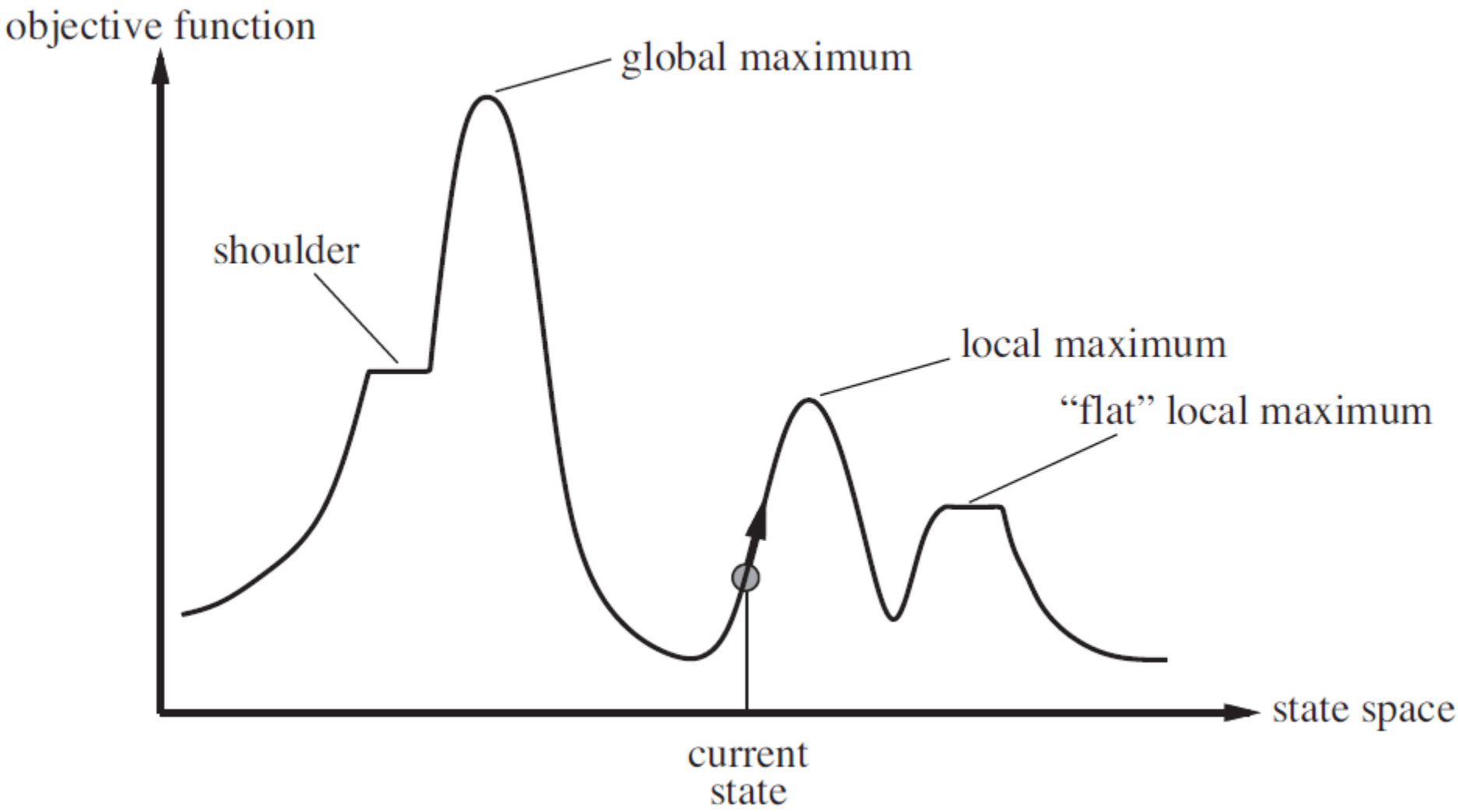
- A* مقید به حافظهٔ ساده شده (Simplified Memory bounded A*) یا SMA*، الگوریتم جستجویی است که مشابه با A* عمل می کند.
  - در حین جستجو وقتی حافظه پُر شود، گره دارای بیشترین $f(n)$ را حذف می کند و مقدار گره فراموش شده را به والدش بر می گرداند.
  - زمانی که به محدودیت حافظه می رسیم علامت ∞ را ثبت می کنیم.

کنفرانسی

- ویژگی های SMA*:
  - کامل است اگر کم عمق ترین هدف، با حافظه محدود جستجو قابل دسترس باشد.
  - در مورد پیچیدگی حافظه و زمان آن براحتی نمی توان بحث کرد.
  - با دادن حافظه لازم مسائل مشکل تری را نسبت به A* حل می کند و بالعکس.

الگوریتم های جستجوی محلی

- در مسائل مورد بحث قبلی، هنگامیکه یک هدف پیدا می شود، مسیر هدف هم جزئی از اهمیت راه حل مسأله است. اما در بسیاری از مسائل، مسیر به هدف ارتباطی ندارد مثل مسأله هشت وزیر و مسائل بهینه سازی.
- الگوریتم های جستجوی محلی اصلاً به مسیر اهمیت نمی دهند و با استفاده از حالت فعلی عمل می کنند.
- مزایای الگوریتم های جستجوی محلی:
  - بسیار کم از حافظه استفاده می کنند.
  - اغلب در فضای حالت نامتناهی راه حل های معقول می یابند.



یک فضای حالت یک بعدی که ارتفاع در آن متناظر با تابع هدف می باشد.

الگوریتم های جستجوی محلی

- الگوریتم های جستجوی محلی با نام **الگوریتم های اصلاح تکراری** نیز شناخته می شوند.

- الگوریتم های مشهور این دسته از الگوریتم ها عبارتند از:
  - جستجوی تپه نوردی
  - جستجوی سخت سازی شبیه سازی شده
  - الگوریتم ژنتیک (در فصل های آینده بحث می شود)

جستجوی تپه نوردی

- این جستجو از یک حلقه ساده تکرار تشکیل می شود که بصورت پیوسته در جهت افزایش مقدار حرکت می کند.
- جستجوی تپ نوردی زمانی متوقف می شود که به مکانی می رسد که در آنجا هیچ همسایه ای مقدار بیشتری نداشته باشد.
- این الگوریتم درخت جستجو را نگهداری نکرده و فقط حالت و مقدار تابع هدفِ گره فعلی را نگهداری کرده و فراتر از همسایه های مجاور حالت فعلی را در نظَر نمی گیرد.

جستجوی تپه نوردی مثل تلاش برای یافتن قلهٔ کوه اِورست در مِه غلیظ است در حالیکه از بیماری فراموشی رنج می بریم.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

الگوریتم جستجوی تپه نوردی

جستجوی تپه نوردی

- تپه نوردی را گاهی **جستجوی حریصانهٔ محلی** می نامند، زیرا یک حالت همسایه خوب را بدون اینکه از قبل فکر کند که از کجا به آنجا خواهد رفت، انتخاب می کند.
- متأسفانه تپه نوردی اغلب به دلایل زیر گیر می کند:
  - ماکزیمم های محلی (Local Maxima)
    - قله ای است که از تمام همسایه هایش بلندتر بوده ولی از ماکزیمم اصلی کوتاهتر است.
  - فلات (Flat, Plateaus)
    - محوطه ای از فضای حالت است که تابع ارزیاب در آن مقدار یکنواختی دارد.
  - لبه یا تیغه (Ridge)
    - یک رشته ماکمزیمم محلی، که عبور از آن با حرکت تکی در هیچ جهتی ممکن نیست.

تپه نوردی با شروع تصادفی

- این جستجو برای حل مشکلات تپه نوردی مطرح شده است.
- این روش یک سری از جستجوهای تپه نوردی را از حالات اولیه تصادفی تا زمانیکه متوقف شوند یا پیشرفتی نداشته باشند، اجرا کرده و بهترین نتایج حاصل از هر جستجو را ذخیره می کند.
- اگر تعداد تکرارها کافی باشد، تپه نوردی با شروع تصادفی بالاخره راه حل بهینه را پیدا می کند.
- موفقیت تپه نوردی به ظاهر فضای حالت وابسته است. اگر ماکزیمم های محلی کم باشند، تپه نوردی با شروع تصادفی بسرعت راه حل خوبی پیدا می کند.

جستجوی سخت سازی شبیه سازی شده

- جستجوی سخت سازی شبیه سازی شده (Simulated Annealing) به نام سرد کردن تدریجی یا آبکاری فلزات نیز شناخته می شود.

- این جستجو از ترکیب تپه نوردی با یک راهپیمایی تصادفی کارآمد و کامل حاصل شده است.

- علت این ترکیب این است که تپه نوردی که هرگز حرکت رو به پایین انجام نمی دهد، ممکن است در یک ماکزیمم محلی گیر کند.

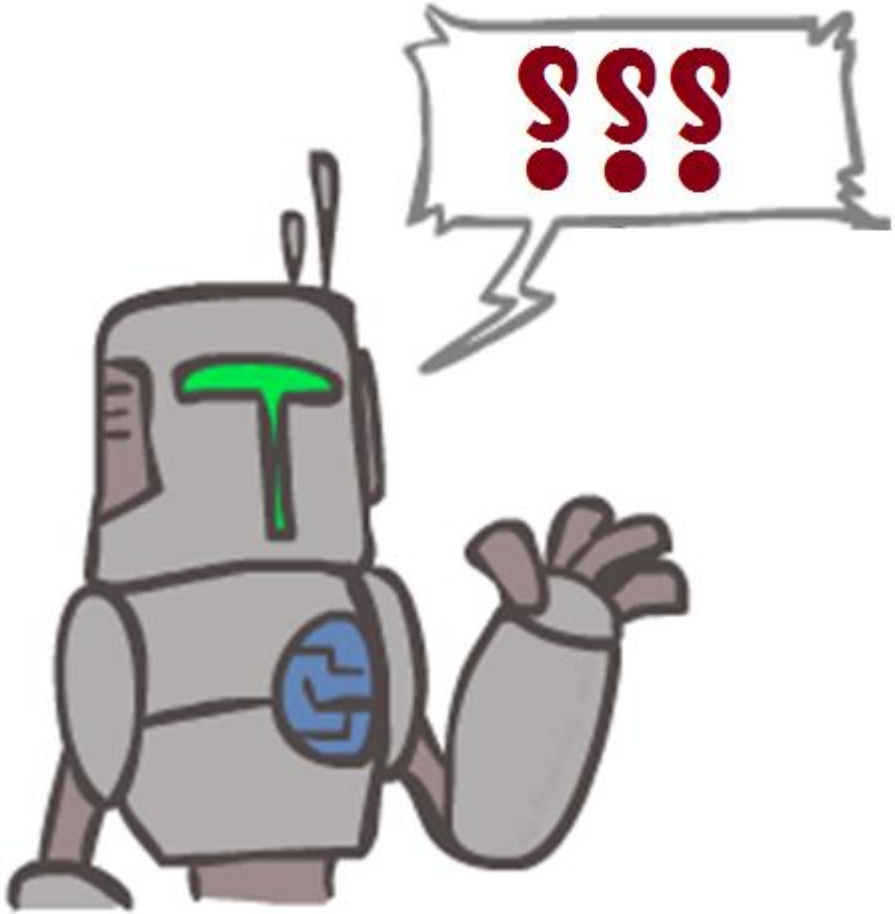
جستجوی سخت سازی شبیه سازی شده

- روش کار جستجوی سخت سازی شبیه سازی شده بصورت زیر است:
  - با توجه به تابع ارزیابی، یک وضعیت حرکت، بصورت تصادفی انتخاب می شود.
  - اگر حرکت جدید شرایط را بهبود دهد همیشه پذیرفته می شود.
  - اگر شرایط را بهبود ندهد، الگوریتم این حرکت را با احتمال کمتر از ۱ می پذیرد.
  - احتمال ΔE حاصل تفاضل ارزش حالت فعلی و ارزش حرکت تصادفی می باشد.
  - احتمال ΔE بصورت نمایی با کاهش دما T کاهش می یابد.
  - احتمال ΔE بصورت نمایی با افزایش ارزش حرکت تصادفی جدید افزایش می یابد.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

الگوریتم جستجوی سخت سازی شبیه سازی شده



سوال ؟