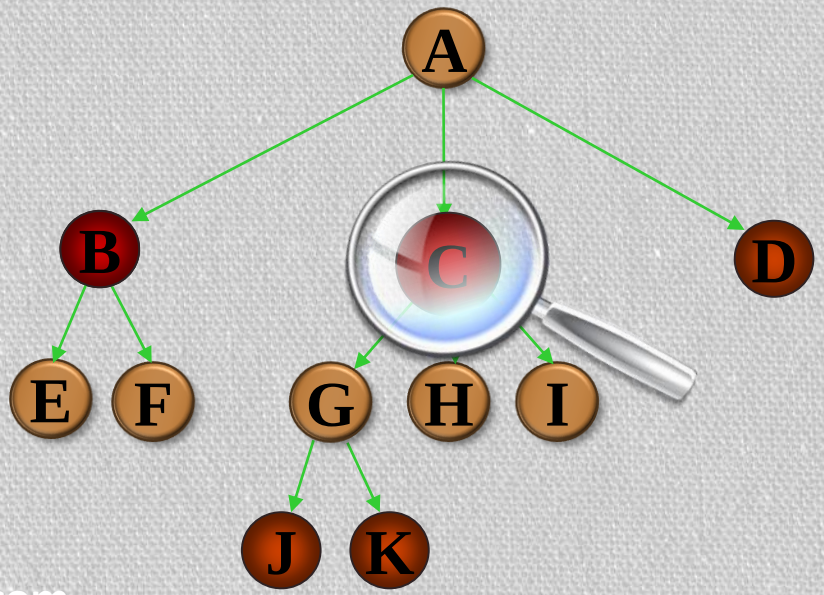
هوش مصنوعی

حل مسئله با جستجو



www.students.madadyar.com

عامل حل مساله

- یک نوع عامل **هدف گرا**، عامل حل مسئله نامیده می شود.

- عامل های حل مسئله توسط یافتن **ترتیب عملیات** تصمیم می گیرند که چه انجام دهند تا آنها را به حالت های مطلوب سوق دهد.

- الگوریتم جستجو مسئله ای را به عنوان ورودی دریافت نموده و **راه حلی را به صورت دنباله عملیات** بر می گرداند.

www.students.madadyar.com

عامل ساده حل مساله

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  inputs: percept, a percept
  static: 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 do
     goal ← FORMULATE-GOAL(state)
     problem ← FORMULATE-PROBLEM(state, goal)
     seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

گامهاي حل مساله

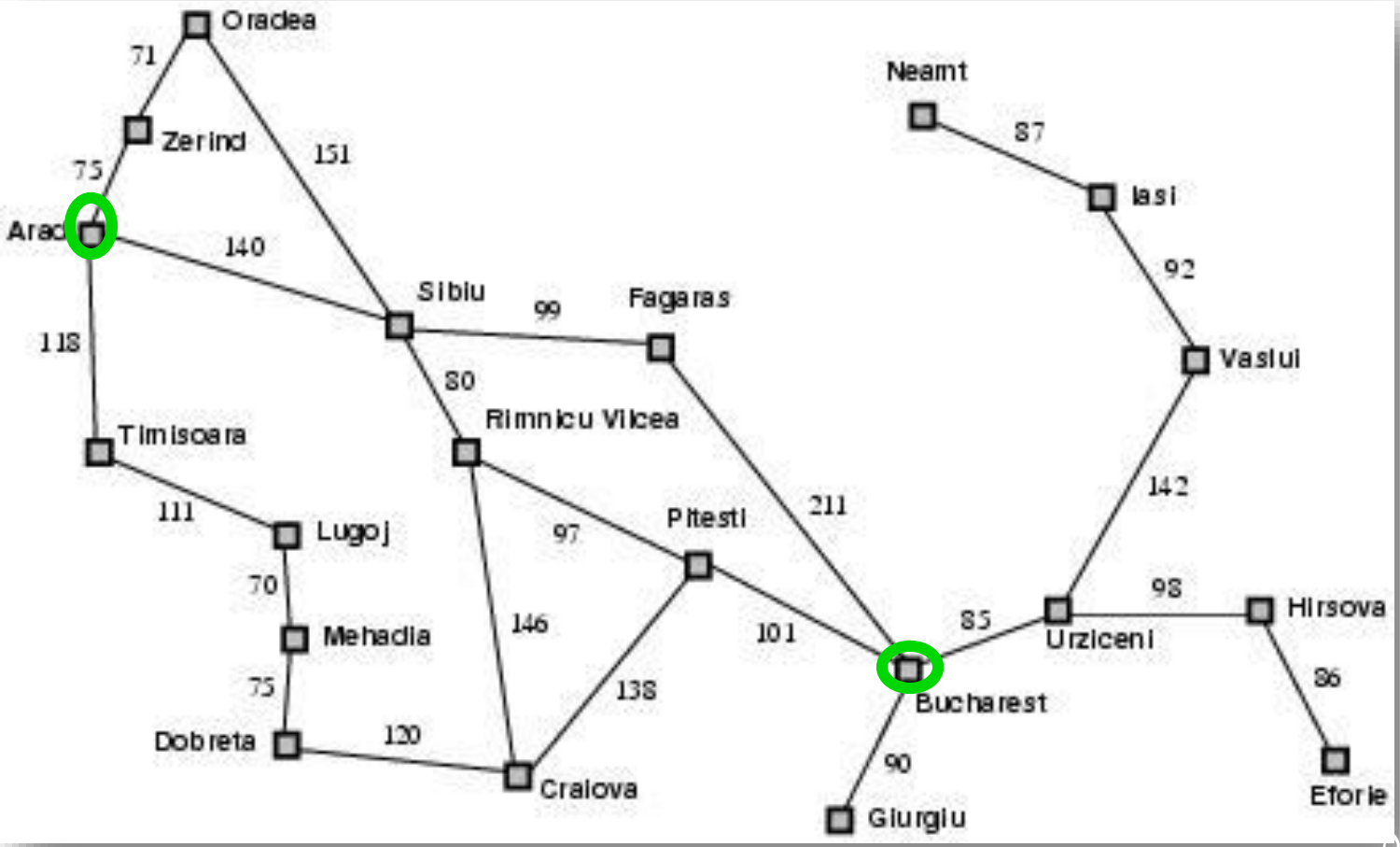
چهار گام اساسی برای حل مسائل

- **فرموله کردن هدف**
  - وضعیتهای مطلوب نهایی کدامند؟
- **فرموله کردن مسئله**
  - چه فعالیتها و وضعیتهایی برای رسیدن به هدف موجود است؟
- **جستجو**
  - انتخاب بهترین دنباله از فعالیتهایی که منجر به حالاتی با مقدار شناخته شده می‌شود.
- **اجرا**
  - وقتی دنباله فعالیت مطلوب پیدا شد، فعالیتهای پیشنهادی آن میتواند اجرا شود.

www.students.madadyar.com

حل مسئله با جستجو

مثال: نقشه رومانی



www.students.madadyar.com

مثال: نقشه رومانی

- صورت مسأله: رفتن از آراد به بخارست.
- فرموله کردن هدف: رسیدن به بخارست.
- فرموله کردن مسئله:
  - وضعیتها: شهرهای مختلف.
  - فعالیتها: حرکت بین شهرها.
- جستجو: دنباله ای از شهرها مثل:آراد، سیبیو، فاگارس، بخارست.
  - این جستجو با توجه به کم هزینه ترین مسیر انتخاب می‌شود.

www.students.madadyar.com

مسئله

- حالت اولیه: حالتی که عامل از آن شروع مي‌کند.
  - در مثال رومانی: شهر آراد In(Arad)
- تابع پسین: توصيفي از فعالیتهای ممکن که برای عامل مهیا است.
  - در مثال رومانی: S(Arad)={Zerind,Sibui,Timisoara}
- فضای حالت: مجموعه ای از حالتها که از حالت اولیه مي‌توان به آنها رسید.
  - در مثال رومانی: کلیه شهرها که با شروع از آراد میتوان به آنها رسید.

تابع جانشین + حالت اولیه = فضای حالت

www.students.madadyar.com

- آزمون هدف: تعيين مي‌كند كه آيا حالت خاصی ، حالت هدف است يا خير.
  - هدف صريح: در مثال رومانی ، رسيدن به بخارست.
  - هدف انتزاعی: در مثال شطرنج ، رسيدن به حالت كيش و مات.
- مسير: دنباله ای از حالتها كه دنباله ای از فعاليتها را به هم متصل مي‌كند.
  - در مثال رومانی: Arad, Sibiu, Fagaras يک مسير است.

www.students.madadyar.com

- **هزینه مسیر**: برای هر مسیر یک هزینه عددی در نظر مي‌گیرد.
  - در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر.

هزینه گام برای اجرای اقدام a که از حالت x به حالت y انتقال می یابد به صورت C(x,a,y) نمایش داده می شود.

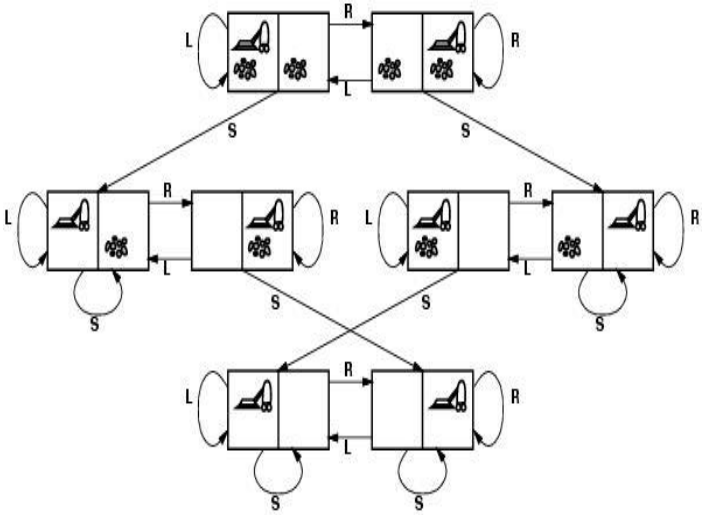
www.students.madadyar.com

راه حل مسئله مسیری از حالت اولیه به حالت هدف است.

راه حل بهینه کمترین هزینه مسیر را دارد.

www.students.madadyar.com

مثال: دنیای جارو برقي



حالتها:

حالت اولیه:

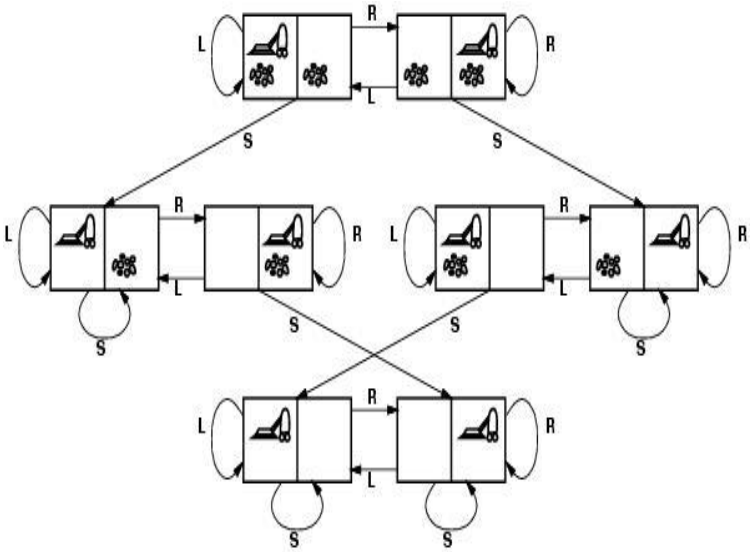
تابع جانشین:

آزمون هدف:

هزینه مسیر:

www.students.madadyar.com

مثال: دنیای جارو برقي



حالتها: دو مکان که هر یک ممکن است کثیف یا تمیز باشند.لذا ۸ = ۲^۲* ۲حالت در این جهان وجود دارد.

حالت اولیه: هر حالتی مي‌تواند به عنوان حالت اولیه طراحی شود.

تابع جانشین: حالتهای معتبر از سه عملیات: راست ، چپ ، مکش.

آزمون هدف: تمیزی تمام مربعها.

هزینه مسیر: تعداد مراحل در مسیر.

www.students.madadyar.com

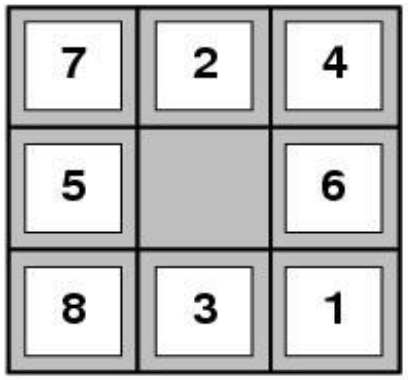
بحث کنید :

معمای هشت را فرموله سازی کنید ؟

مثال: معمای ۸

حالتها:

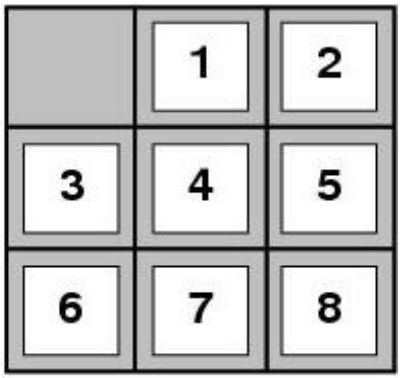
حالت اولیه:



Start State

تابع جانشین:

آزمون هدف:

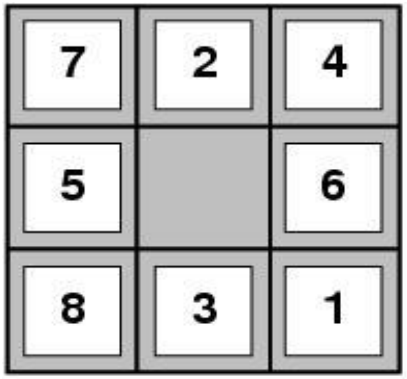


Goal State

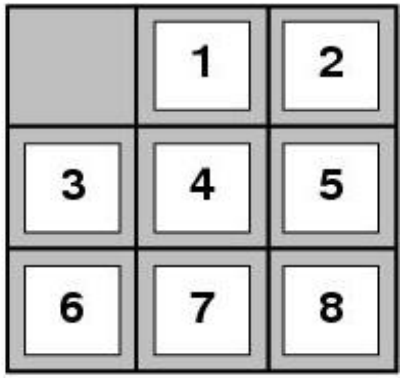
هزینه مسیر:

www.students.madadyar.com

مثال: معمای ۸



Start State



Goal State

حالتها: مکان هر هشت خانه شماره دار و خانه خالی در يكي از ۹ خانه

حالت اولیه: هر حالتي را مي‌توان به عنوان حالت اولیه در نظر گرفت.

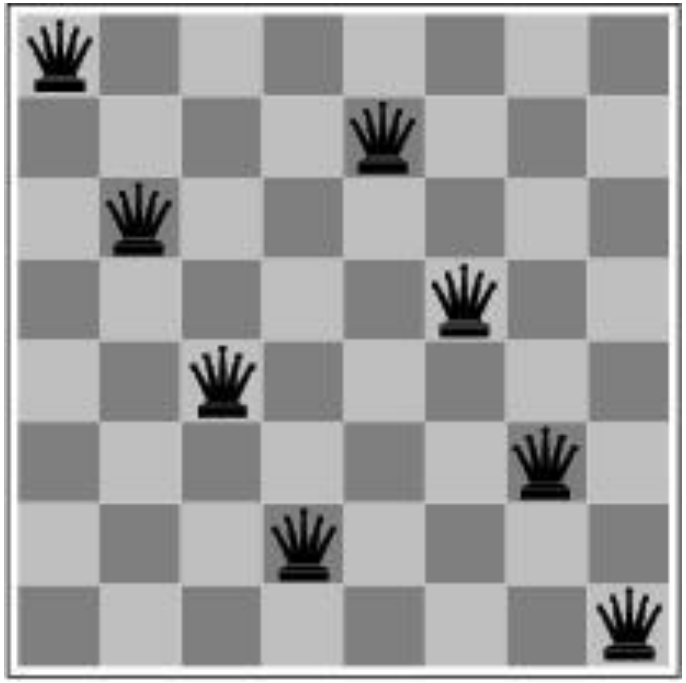
تابع جانشین: حالتهای معتبر از چهار عمل ، انتقال خانه خالی به چپ ، راست ، بالا یا پایین.

آزمون هدف: بررسی مي‌کند که حالتی که اعداد به ترتیب چیده شده اند(طبق شکل روبرو) رخ داده یا نه.

هزینه مسیر: برابر با تعداد مراحل در مسیر.

www.students.madadyar.com

مثال: مسئله ۸ وزیر



فرمول بندی افزایشي

حالتها:

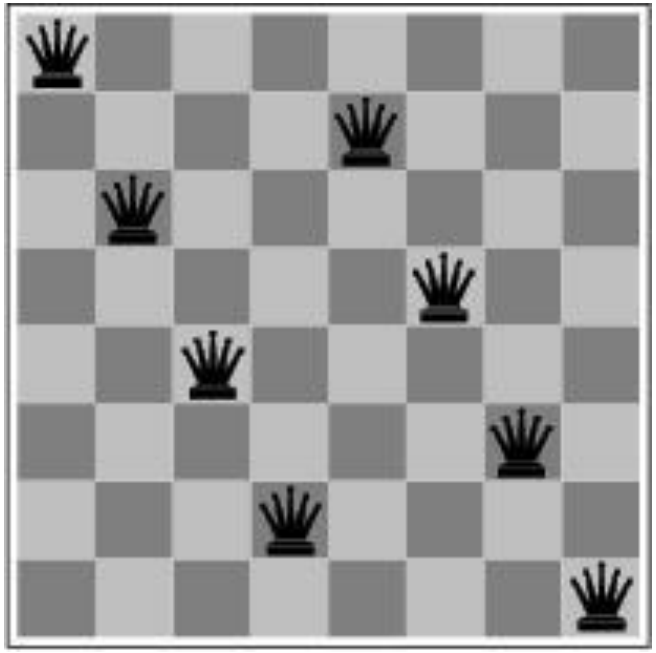
حالت اولیه:

تابع جانشین:

آزمون هدف:

www.students.madadyar.com

مثال: مسئله ۸ وزیر



فرمول بندی افزایشي

حالتها: هر ترتيبي از ۰ تا ۸ وزیر در صفحه ، یک حالت است.

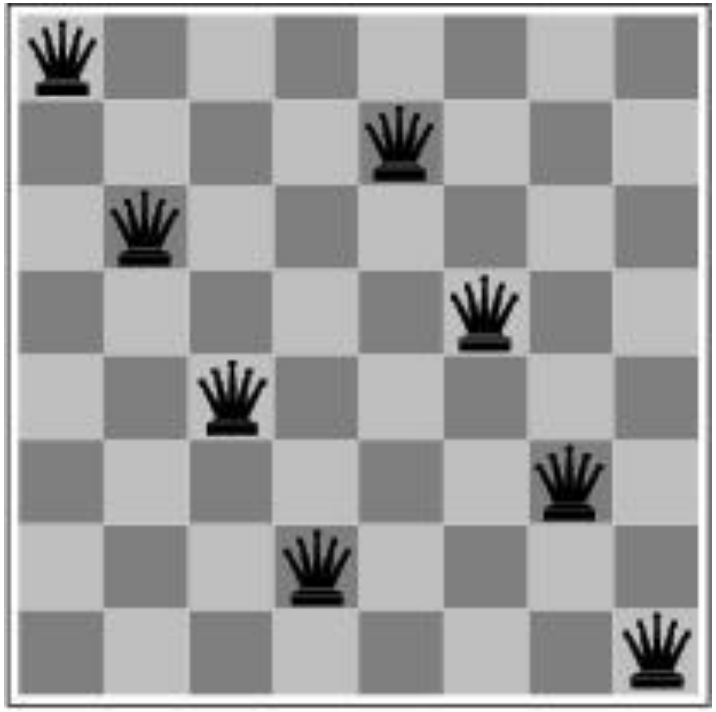
حالت اولیه: هیچ وزیری در صفحه نیست.

تابع جانشین: وزیری را به خانه خالی اضافه مي‌کند.

آزمون هدف: ۸وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمیگیرند.

در این فرمول بندی تقریبا باید ۳*۱۰^۱۴ دنباله ممکن بررسی مي‌شود.

مثال: مسئله ۸ وزیر



حالتها:

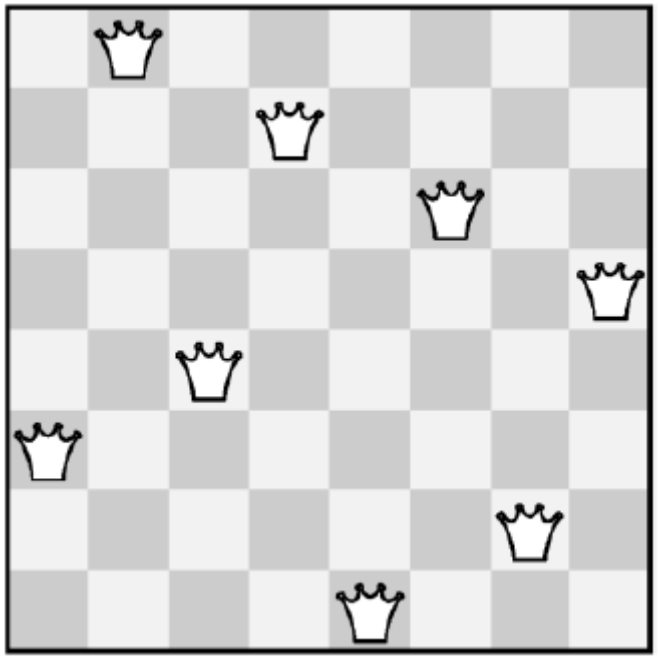
حالت اولیه:

تابع جانشین:

آزمون هدف:

www.students.madadyar.com

مثال: مسئله ۸ وزیر



حالتها: چیدمان n وزیر $(0\leq n\leq 8)$ ، بطوریکه در هر ستون از n ستون سمت چپ ، یک وزیر قرار گیرد و هیچ دو وزیری بهم گارد نگیرند.

حالت اولیه: با ۸ وزیر در صفحه شروع میشود.

تابع جانشین: وزیری را در سمت چپ ترین ستون خالي قرار میدهد ، بطوری که هیچ وزیری آن را گارد ندهد.

آزمون هدف: ۸وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمي‌دهند.

این فرمول بندی فضای حالت را از ۳*۱۰^۱۴ به ۲۰۵۷ کاهش میدهد.

جستجوی راه حل

تدوین مساله ،

مشخص شدن فضای حالت،

تشکیل درخت جستجو (حالت ابتدایی + تابع پسین) .

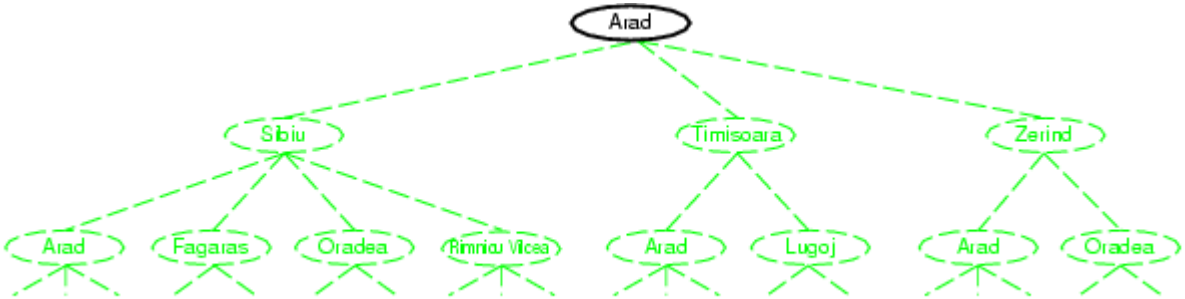
استفاده از روشهای جستجو برای یافتن راه حل.

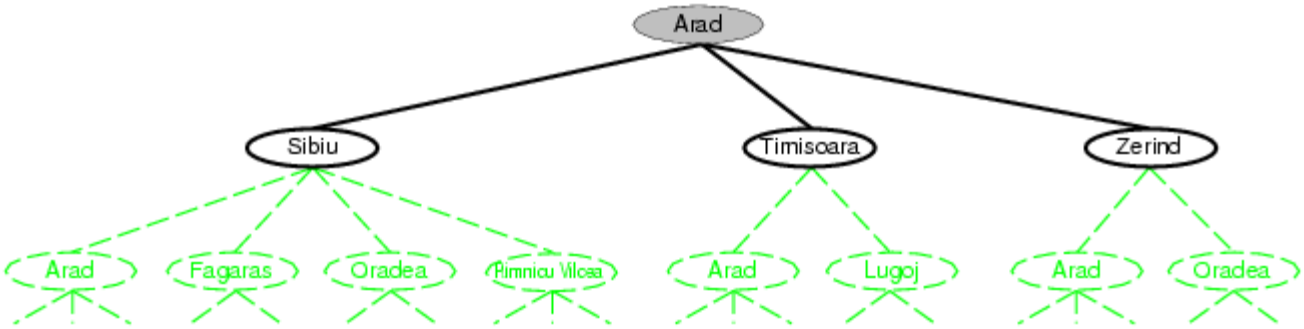
www.students.madadyar.com

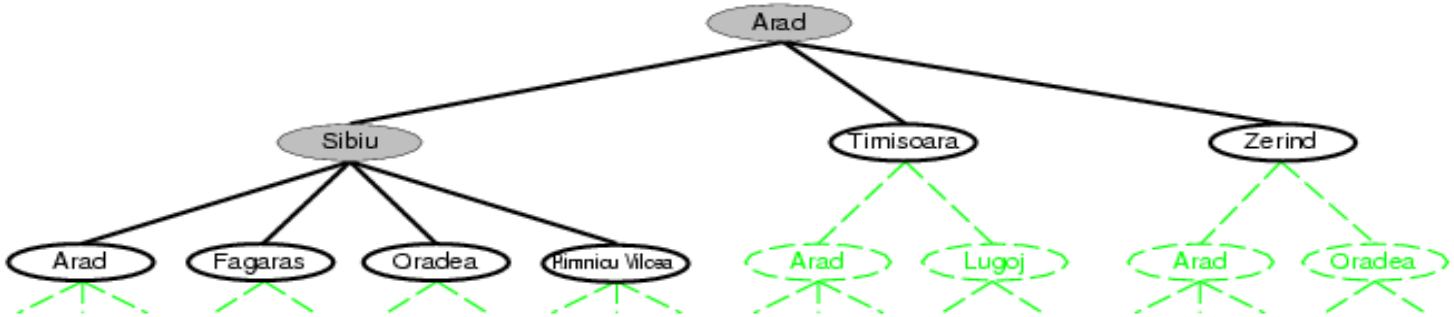
الگوریتم کلی جستجوی درختی

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
     if there are no candidates for expansion then return failure
     choose a leaf node for expansion according to strategy
     if the node contains a goal state then return the corresponding solution
     else expand the node and add the resulting nodes to the search tree
```

www.students.madadyar.com

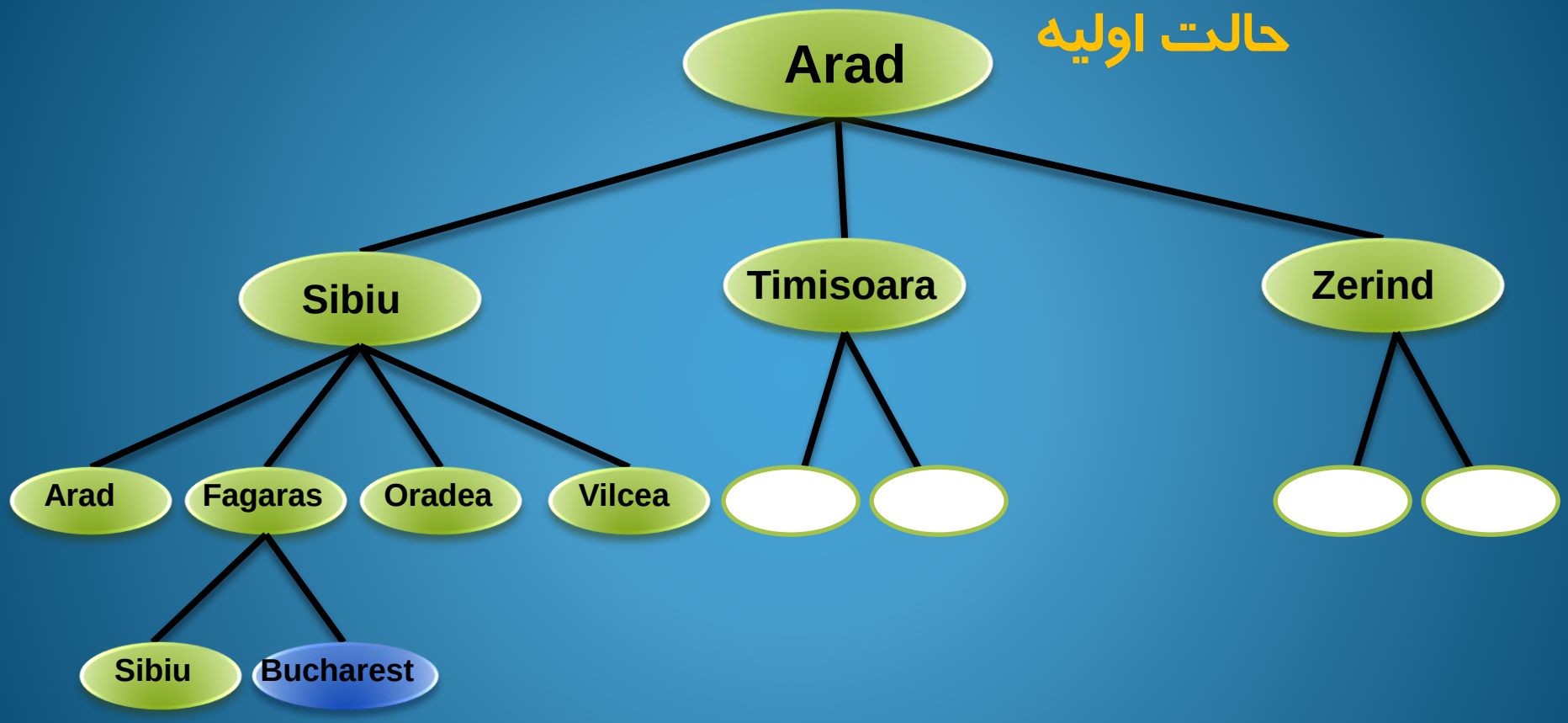






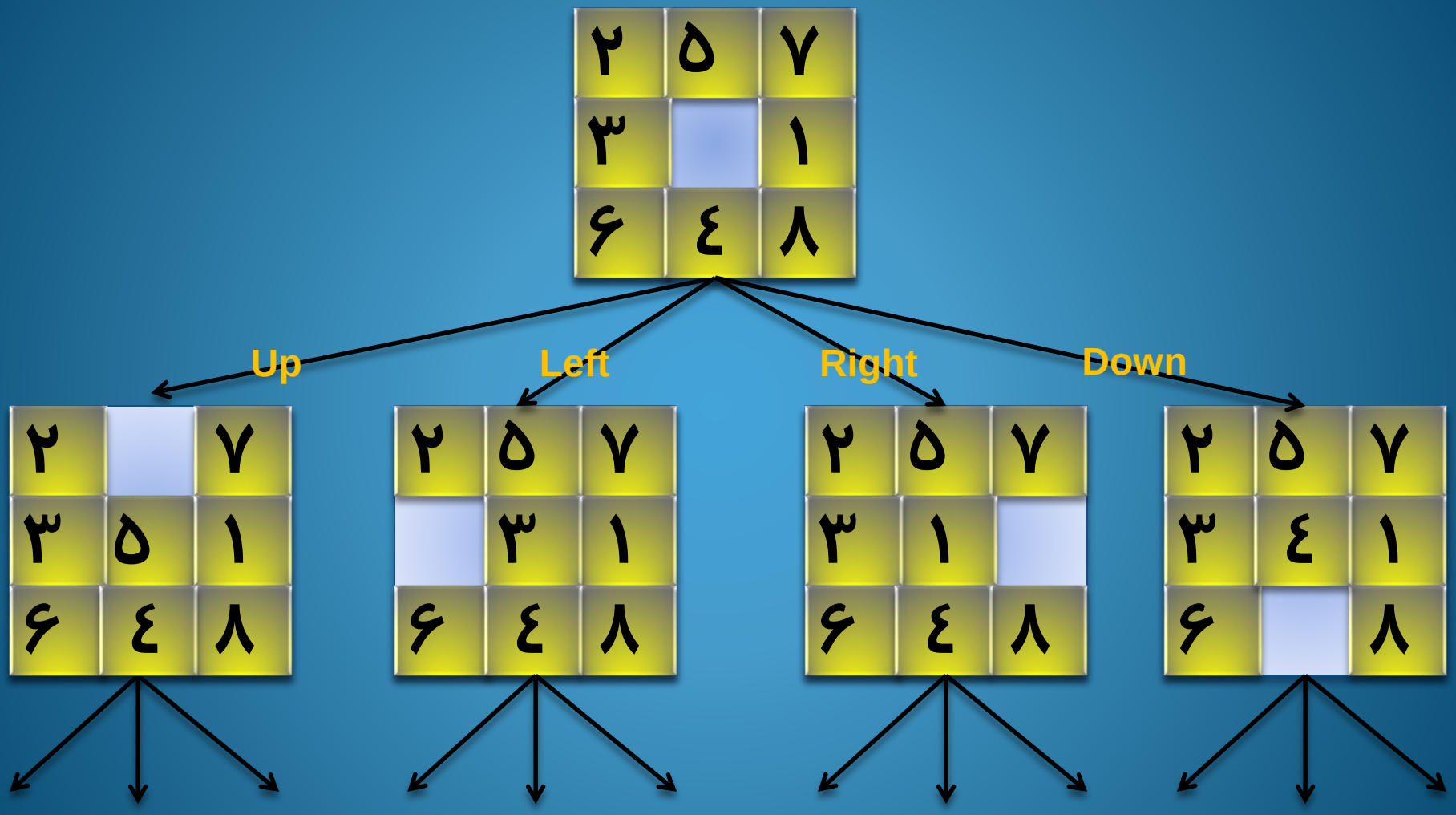
www.students.madadyar.com

درخت جستجو

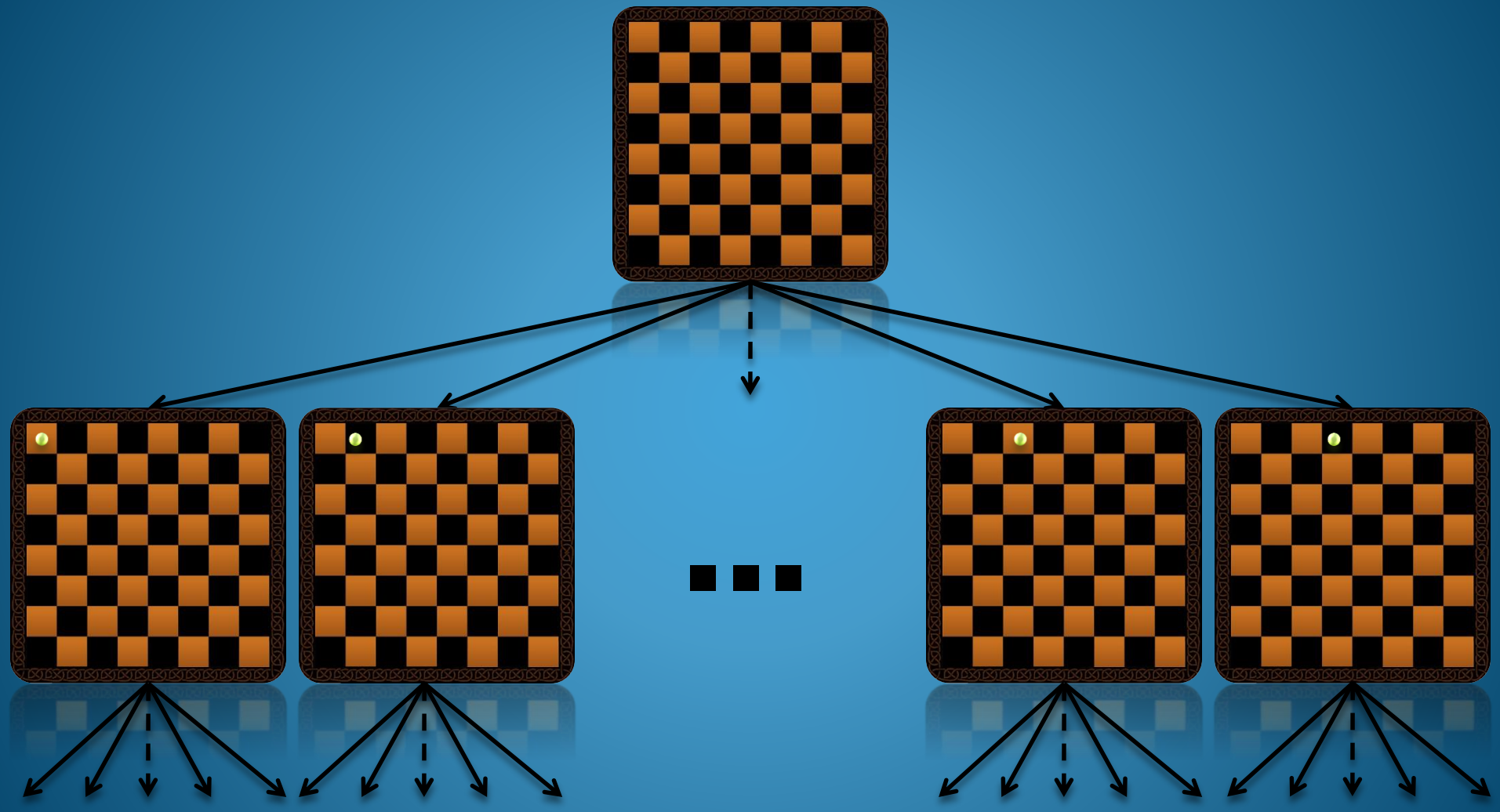


حالت هدف

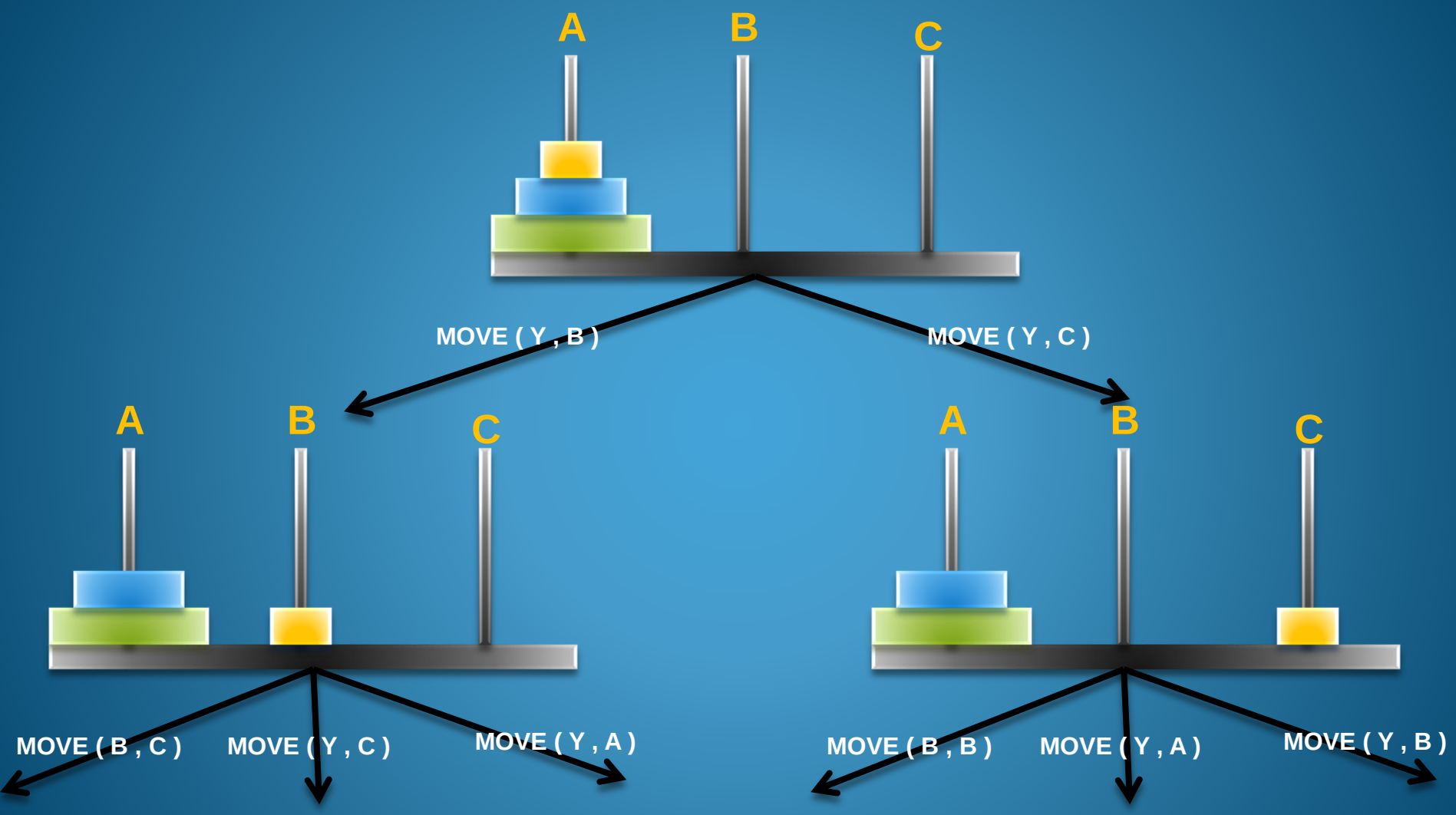
گراف حالت معمای هشت



گراف حالت هشت وزیر



گراف حالت برج هانوی



گرگ و کلم و گوسفند

کشاورزی پس از خرید یک گرگ, یک کلم و یک گوسفند از بازار بازمیگردد. در راه بازگشت باید از رودخانه ای عبور کند.اما قایق او تنها برای یکی از اینها جا دارد.او نمی تواند گوسفند و کلم را یک جا نگاه دارد زیرا ممکن است گوسفند کلم را ببلعد.از سوی دیگر ماندن گرگ و گوسفند هم در کنار هم موجب از بین رفتن گوسفند می شود.پس چه طور می تواند آنها را بدون اینکه آسیب ببینند جابه جا کند؟

فضای حالت ؟ درخت جستجو

- A **state** is a (representation of) a physical configuration.

- A **node** is a data structure constituting part of a search tree includes state, parent node, action, path cost $g(n)$, depth.

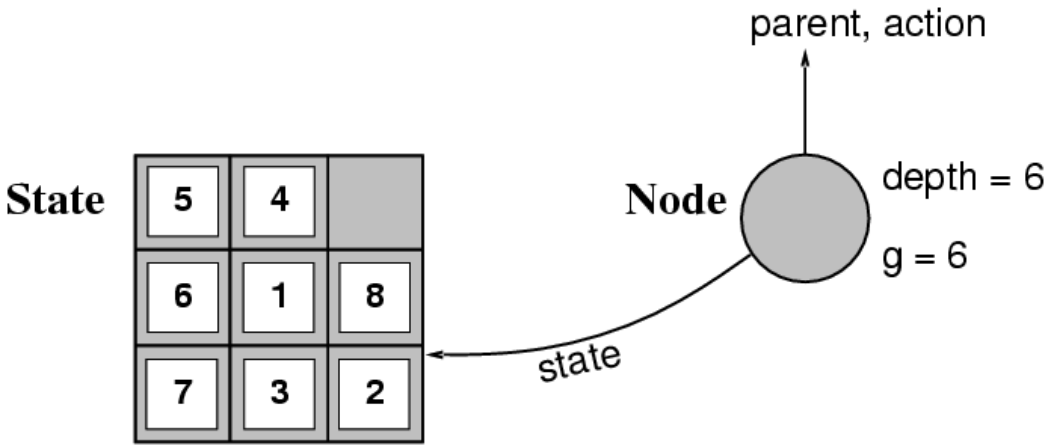
www.students.madadyar.com

بازنمایی گره (Node)

- حالت (state)
  - حالتي از فضاي حالت گه گره با آن متناظر است.
- گره والد (parent)
  - گره‌اي از درخت جستجو که این گره را تولید کرده است.
- اقدام (action)
  - عملي که بر گره والد اعمال شده تا این گره تولید شود.
- هزینه مسیر (path cost)
  - هزینه مسیر از حالت اولیه تا این گره و با $g(n)$ نشان داده مي‌شود.
- عمق (depth)
  - تعداد گامهاي مسیر از حالت اولیه.

www.students.madadyar.com

- تعداد حالت ها در فضاي حالت ممكن است محدود ولي تعداد گره ها نامحدود باشد.
  - مثلا در روماني تعداد حالت ها ۲۰ عدد و لي تعداد گره‌ها نامتناهي است!!
- لبه (fringe)
  - گره هايي از درخت جستجو كه تاكنون گسترش نيافته اند.



www.students.madadyar.com

الگوریتم عمومی جستجوی درختی

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure

  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
      if EMPTY?(fringe) then return failure
      node ← REMOVE-FIRST(fringe)
      if GOAL-TEST[problem] applied to STATE[node] succeeds
          then return SOLUTION(node)
      fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes

  successors ← the empty set
  for each ⟨action, result⟩ in SUCCESSOR-FN[problem](STATE[node]) do
      s ← a new NODE
      STATE[s] ← result
      PARENT-NODE[s] ← node
      ACTION[s] ← action
      PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
      DEPTH[s] ← DEPTH[node] + 1
      add s to successors
  return successors
```

www.students.madadyar.com

راهبرد جستجو

تابعی برای انتخاب
گره بعدی از **لبه**
برای گسترش است.

www.students.madadyar.com

اندازه گیری کارایی حل مسئله

کامل بودن: آیا الگوریتم تضمین میکند که در صورت وجود راه حل، آن را بیابد؟

بهینگی: آیا این راهبرد، راه حل بهینه ای را ارائه می‌کند.

پیچیدگی زمانی: چقدر طول می‌کشد تا راه حل را پیدا کند؟

- تعداد گره های تولید شده در اثنای جستجو.

پیچیدگی فضا: برای جستجو چقدر حافظه نیاز دارد؟

- حداکثر تعداد گره های ذخیره شده در حافظه.

www.students.madadyar.com

انواع جستجو

جستجو

جستجوی نا آگاهانه یا جستجوی کور

۱) هیچ اطلاعات اضافی در مورد حالتها ندارد.

۲) تنها می تواند حالات را تولید و تشخیص دهد کدام حالت هدف است یا هدف نیست .

جستجوی آگاهانه یا جستجوی هیوریستیک

جستجوی ناآگاهانه

ناآگاهی این است که الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد.

این الگوریتمها فقط می‌توانند جانشینهایی را تولید و هدف را از غیر هدف تشخیص دهند.

راهبردهایی که تشخیص می‌دهند یک حالت غیر هدف نسبت به گره غیر هدف دیگر، امید بخش تر است، جست و جوی آگاهانه یا جست و جوی اکتشافی نامیده میشود.

www.students.madadyar.com

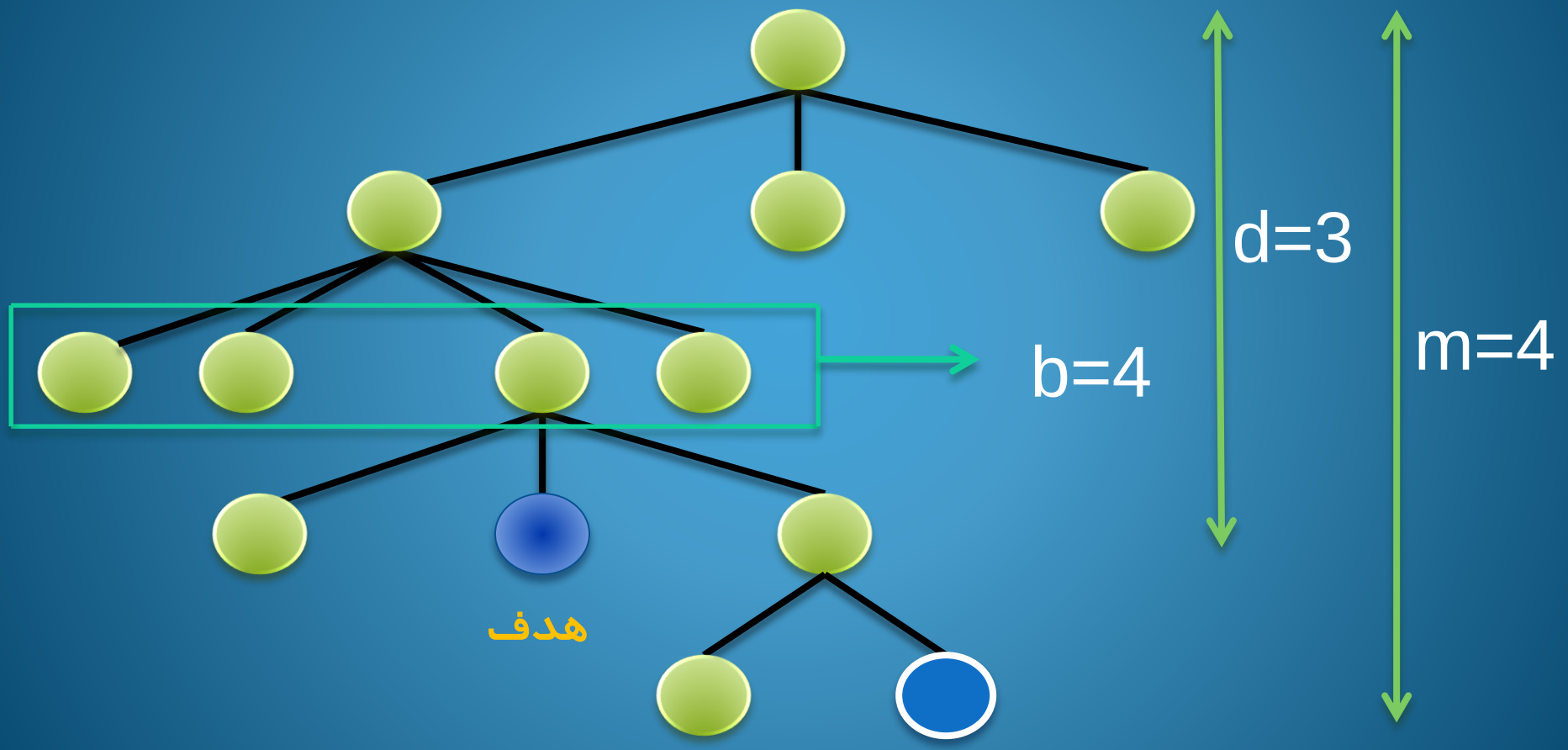
تعاريف

- فاکتور انشعاب (b)
  - حداکثر تعداد پسين‌هاي هر گره.
- d
  - عمق کم عمق ترين گره هدف.
- m
  - طولاني ترين مسير در فضاي حالت.
  - يا حداکثر عمق فضاي حالت (ممکن است بي نهايت باشد !!)

- زمان بر اساس تعداد گرههاي توليد شده در زمان جستجو.
- فضا بر اساس حداکثر گره هاي ذخيره شده در حافظه. سنجيده مي شود.

www.students.madadyar.com

یک مثال



انواع روش های جستجوی ناآگاهانه

جست و جوی اول سطح (سطحی)

جست و جوی هزینه یکنواخت

جست و جوی عمقی

جست و جوی عمیق کننده تکراری

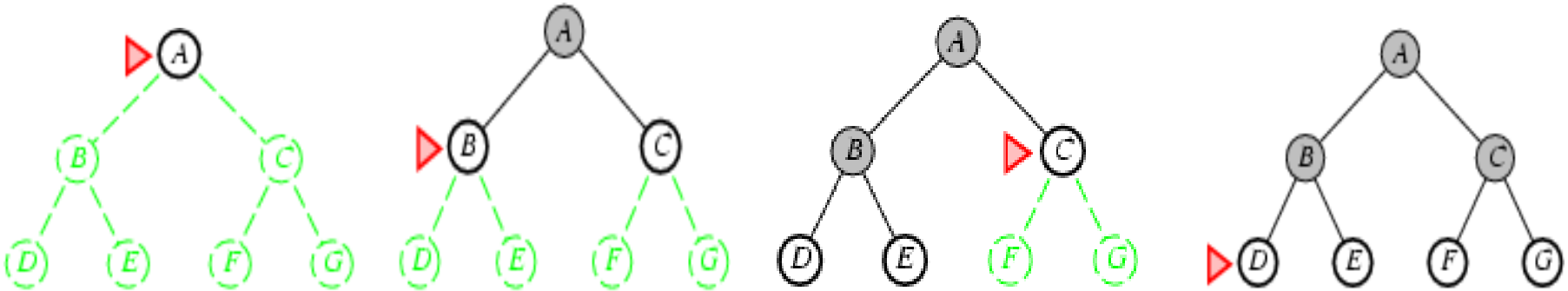
جست و جوی عمقی محدود

جست و جوی دو طرفه

www.students.madadyar.com

جستجوی اول سطح (BFS)

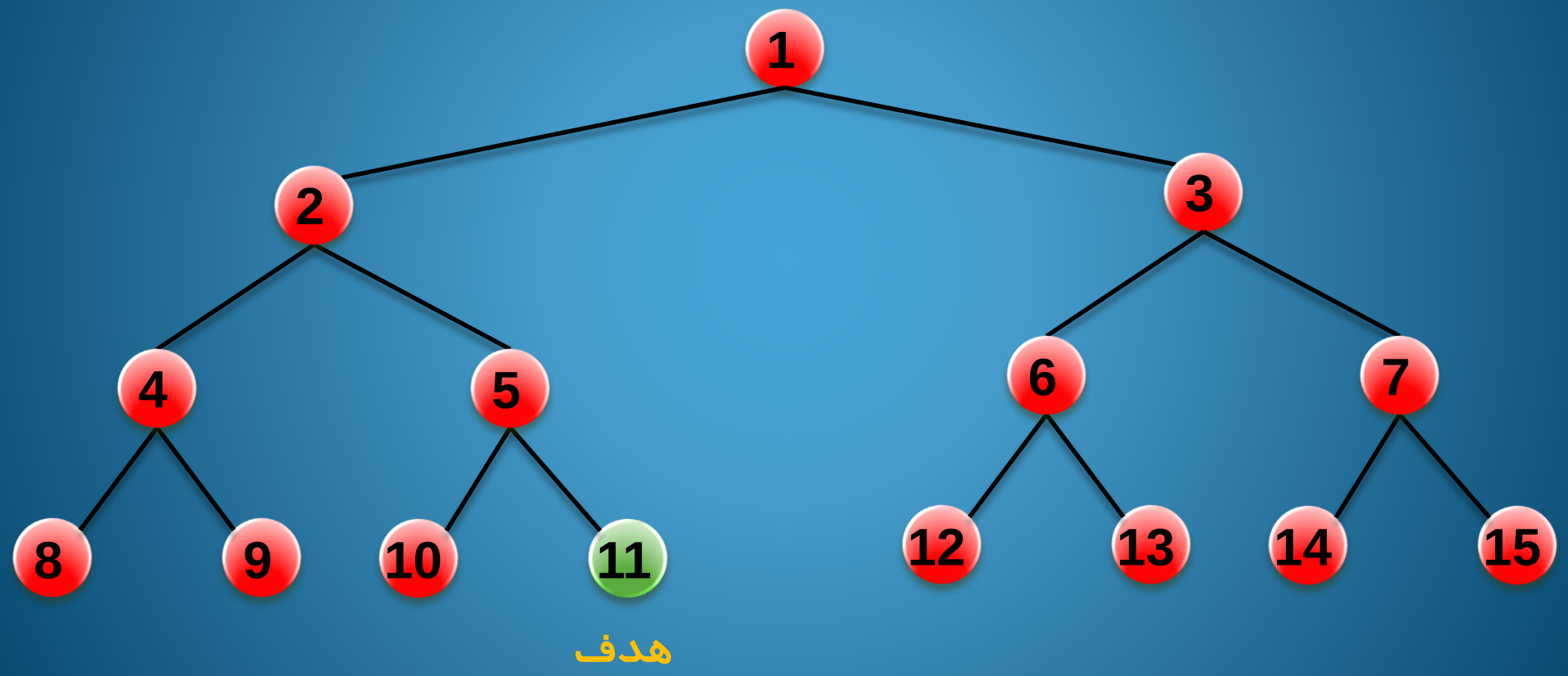
- ابتدا ریشه و سپس تمامی گره های یک سطح گسترش می‌یابند.
- به عبارت کلی تر، تمام گره های عمق d ، قبل از عمق d+1 گسترش داده می شوند.
- گرههای جدید برای هر سطح در انتهای یک صف قرار داده می شوند و به ترتیب گسترش داده خواهند شد.
- اگر چندین راه حل وجود داشته باشد BFS کم عمق ترین مسیر را پیدا می کند.



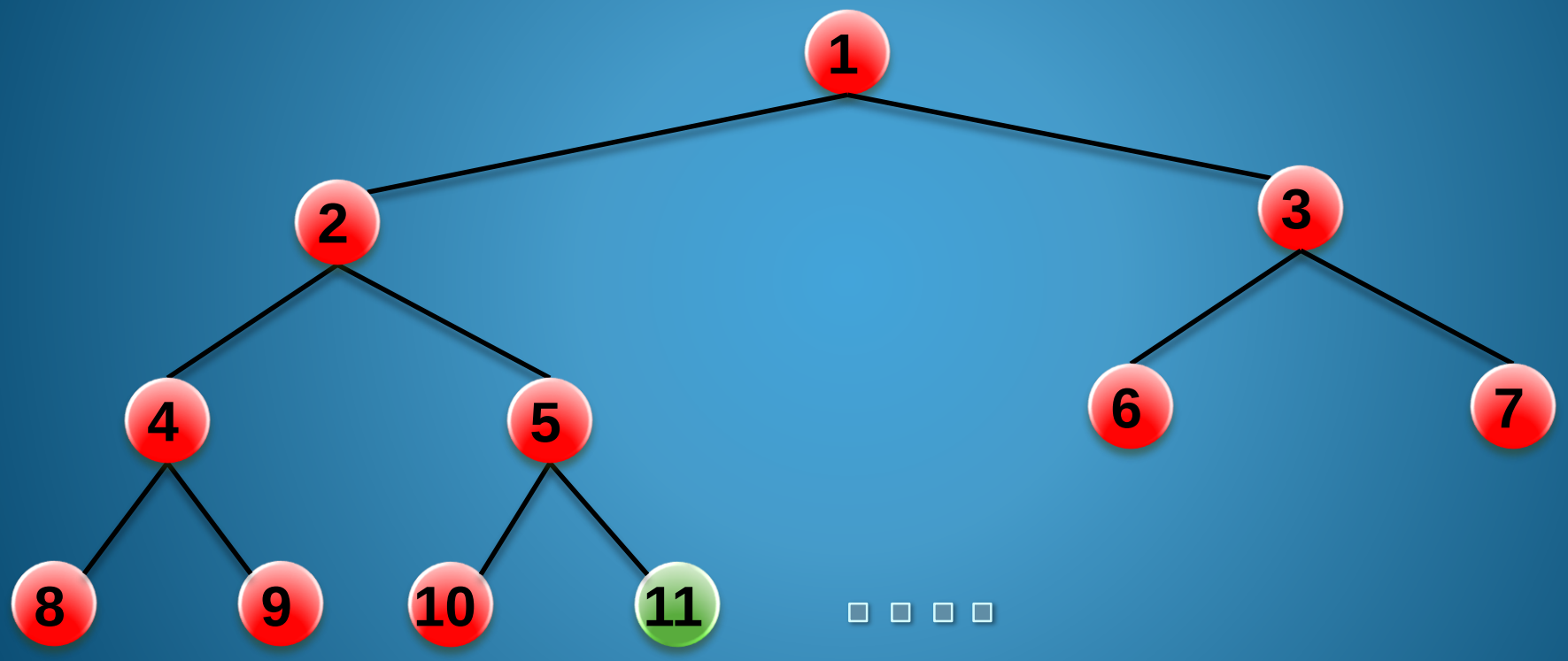
www.students.madadyar.com

مثال :

روی درخت زیر اگر جستجوی سطحی را انجام دهیم بعد از چند تست به هدف می رسیم ؟



مثال : پیمایش سطحی درخت زیر را بدست آورید ؟



توجه !

الگوریتم جستجوی سطحی

توجه : برای پیاده سازی الگوریتم جستجوی سطحی از ساختمان داده صف استفاده می شود .

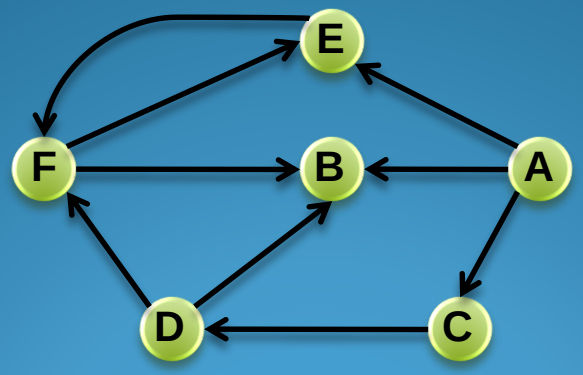
۱) یک صف خالی ایجاد کن و حالت اولیه را در آن قرار بده

۲) اگر لیست خالی است جواب نداریم در غیر این صورت عنصر سر صف را بخوان

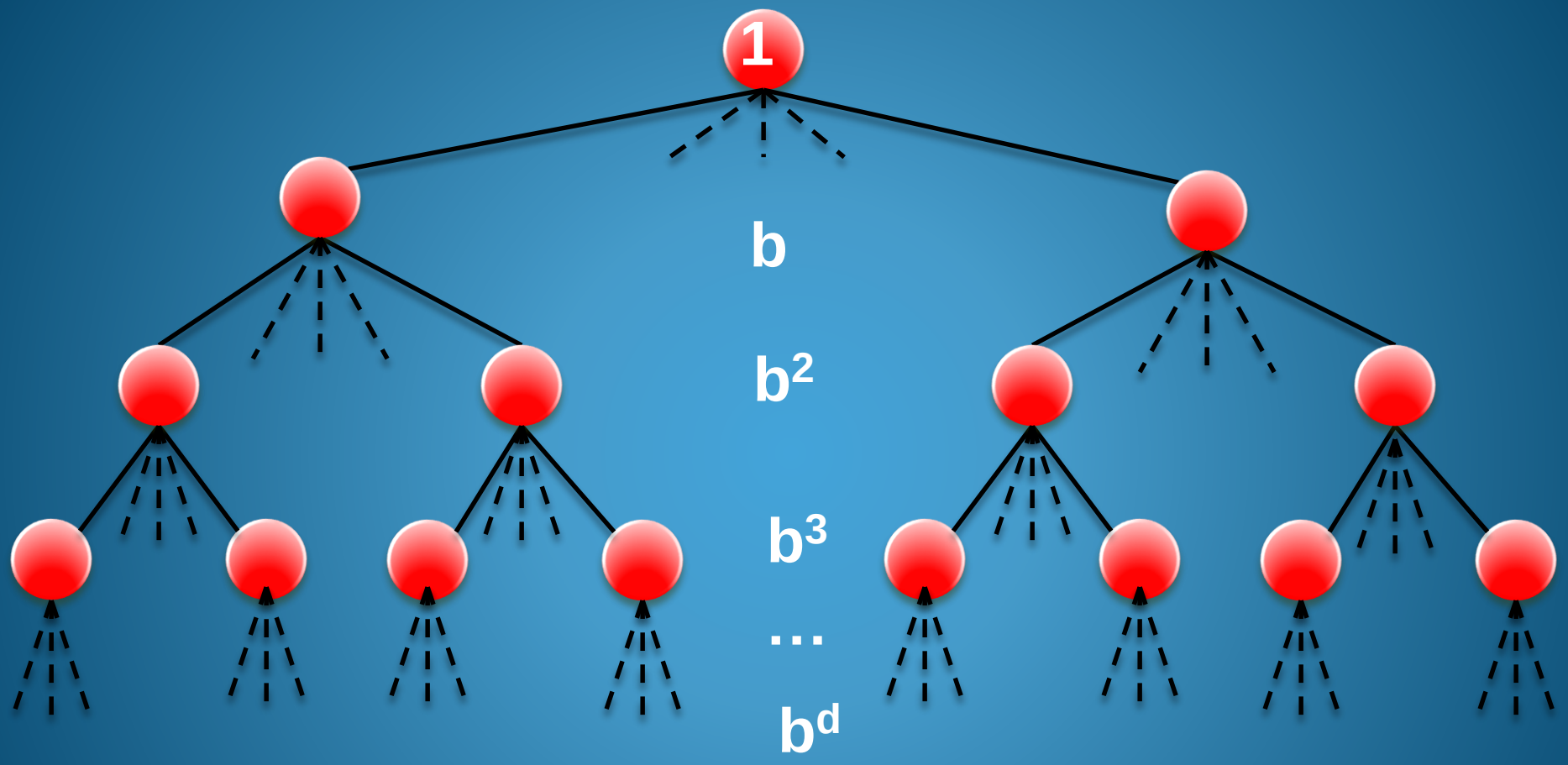
۳) اگر عنصر خوانده شده جواب است مسیر را به عنوان جواب برگردان

۴) در غیر این صورت عنصر خوانده شده را از صف خارج کن و فرزندان آن را در صورت وجود و **ملاقات نشدن** در انتهای صف قرار بده و به مرحله ۲ برو.

یک مثال از پیاده سازی الگوریتم جستجوی سطحی



محاسبه پیچیدگی الگوریتم جستجوی سطحی در بدترین حالت



$1 + b + b^2 + \ldots + b^d$ $\quad$ b^{d+1} $\quad$ d ارتفاع درخت است

جستجوی اول سطح

Complete? Yes (if *b* is finite)

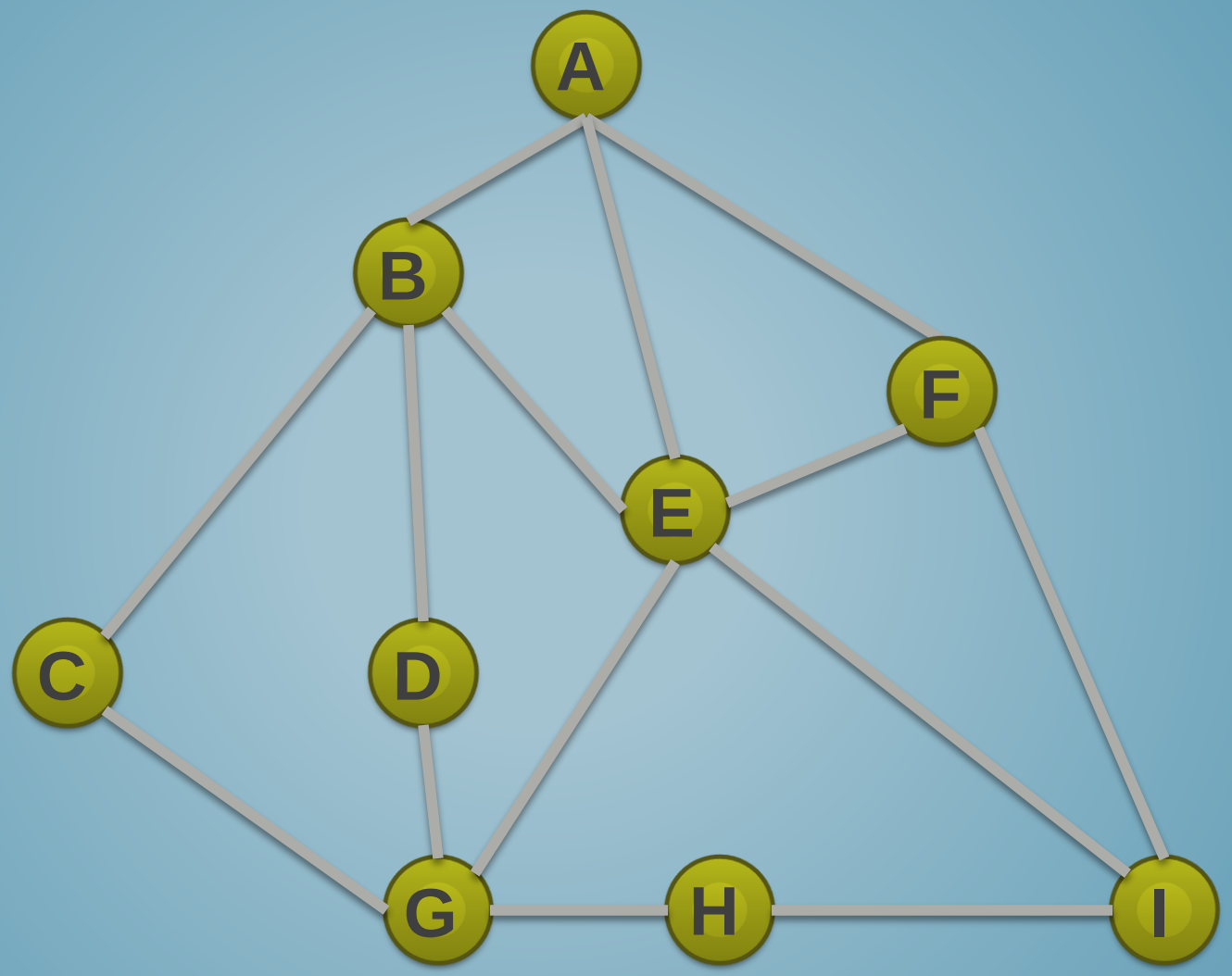
Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$

Space? $O(b^{d+1})$ (keeps every node in memory)

Optimal? Yes (if cost = 1 per step)

www.students.madadyar.com

سوال : الگوریتم جستجوی سطحی را روی گراف زیر پیاده کنید ؟



جستجوی اول سطح

این استراتژی اولین جواب در نزدیک ترین سطح را خواهد یافت.

جستجوی سطحی، کامل و در مسائل مرحله ای بهینه می باشد زیرا هزینه مسیر، یک تابع غیر نزولی از عمق گره است.

زمان اجرای بسیار زیادی هم دارد. (هر دو نمایی هستند!)

مشکل اصلی حافظه است.

www.students.madadyar.com

میزان مصرف حافظه و زمان در BFS

Depth	Nodes	Time		Memory	
0	1	1	Millisecond	100	bytes
2	111	.1	seconds	11	kilobytes
4	11,111	11	Seconds	1	megabyte
6	10^6	18	Minutes	111	megabytes
8	10^8	31	Hours	11	gigabytes
10	10^{10}	128	Days	1	terabytes
12	10^{12}	35	Years	111	terabytes
14	10^{14}	3500	years	11,111	terabytes

b=10

۱۰۰۰ گره در ثانیه

هر گره ۱۰۰ بایت

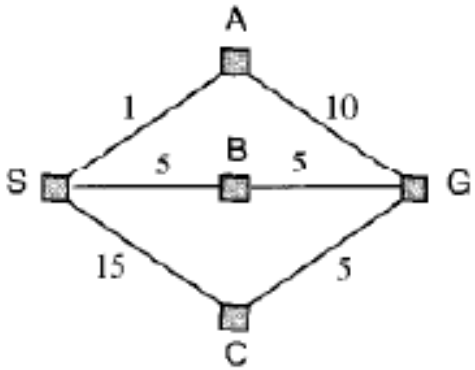
www.students.madadyar.com

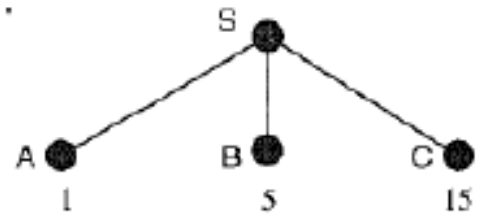
جستجوی هزینه یکنواخت (UCS)

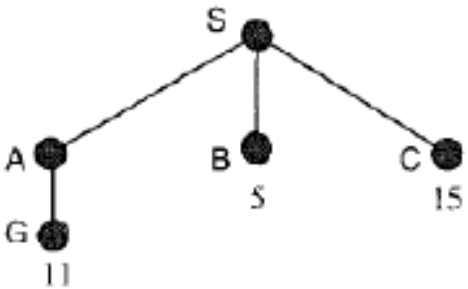
- جستجوي سطحي كم عمق ترين هدف را مي يابد كه لزوما كم هزينه ترين هدف نيست.
- جستجوي با هزينه يكسان روش قبل را با يافتن كم هزينه ترين هدف اصلاح مي كند.
- در اين روش همواره گرهي بسط داده مي شود كه كم هزينه ترين مسير را داشته است.
- در اين استراتژي ، در شرايط عمومي ، اولين راه حل ، ارزان ترين راه نيز هست.
- معادل جستجوي اول سطح اگر هزينه گامها **مساوي** باشد و يا
- $g(n)=DEPTH(n)$
- پياده سازي توسط (fringe) صفي كه بر اساس هزينه مرتب شده است.

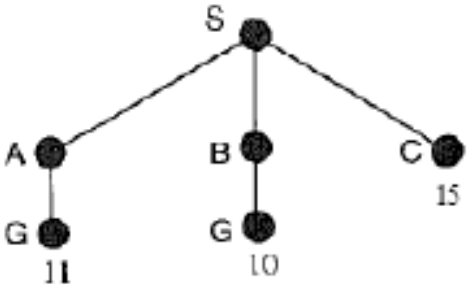
www.students.madadyar.com

جستجوی هزینه یکنواخت (UCS)









www.students.madadyar.com

جستجوی هزینه یکنواخت (UCS)

Complete? Yes, if step cost ≥ ε.

Time? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$.

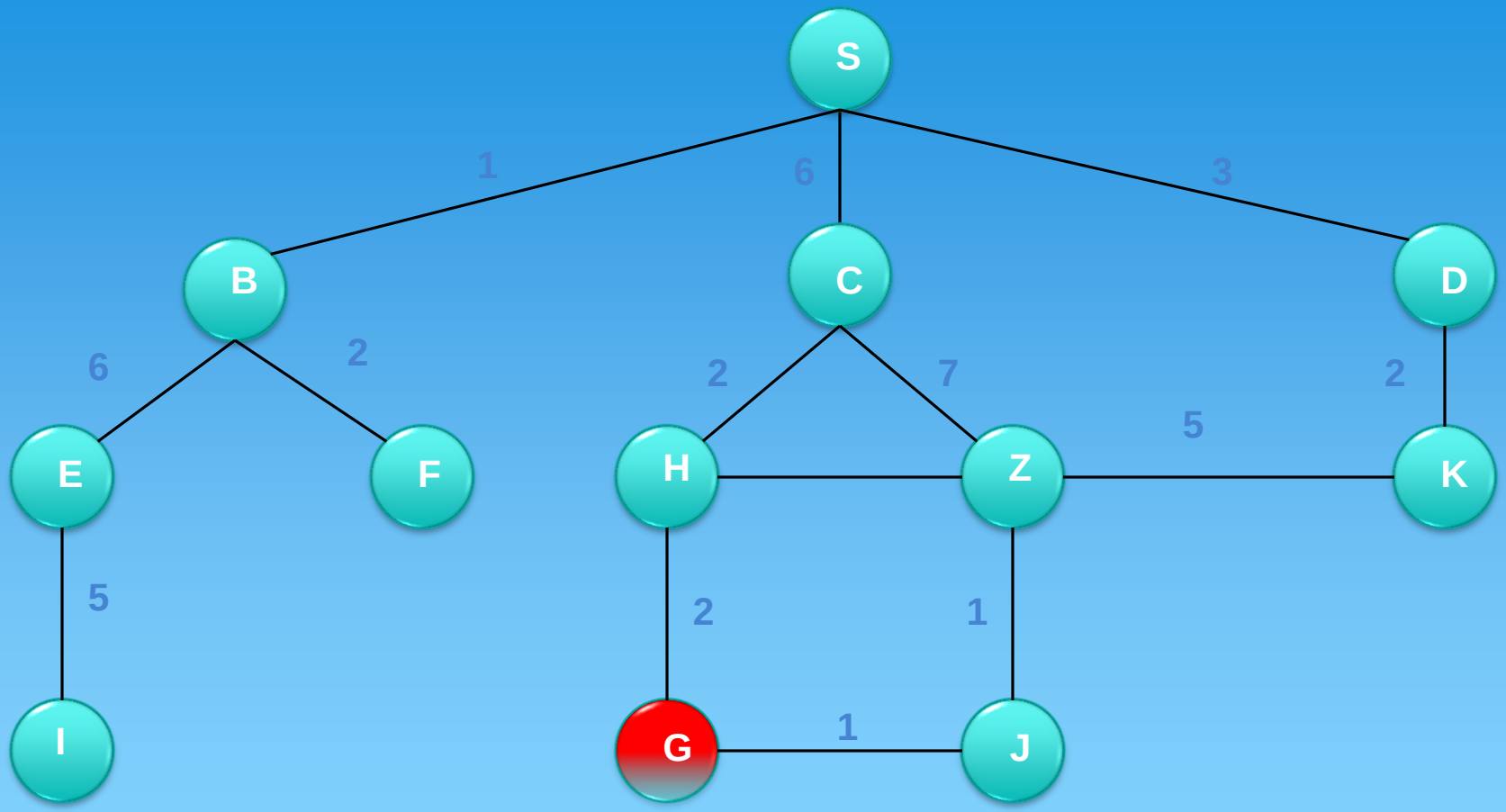
where C^* is the cost of the optimal solution.

Space? # of nodes with $g \leq$ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$.

Optimal? Yes – nodes expanded in increasing order of $g(n)$.

www.students.madadyar.com

سوال : الگوریتم جستجو با هزینه یکسان را روی گراف زیر پیاده کنید ؟

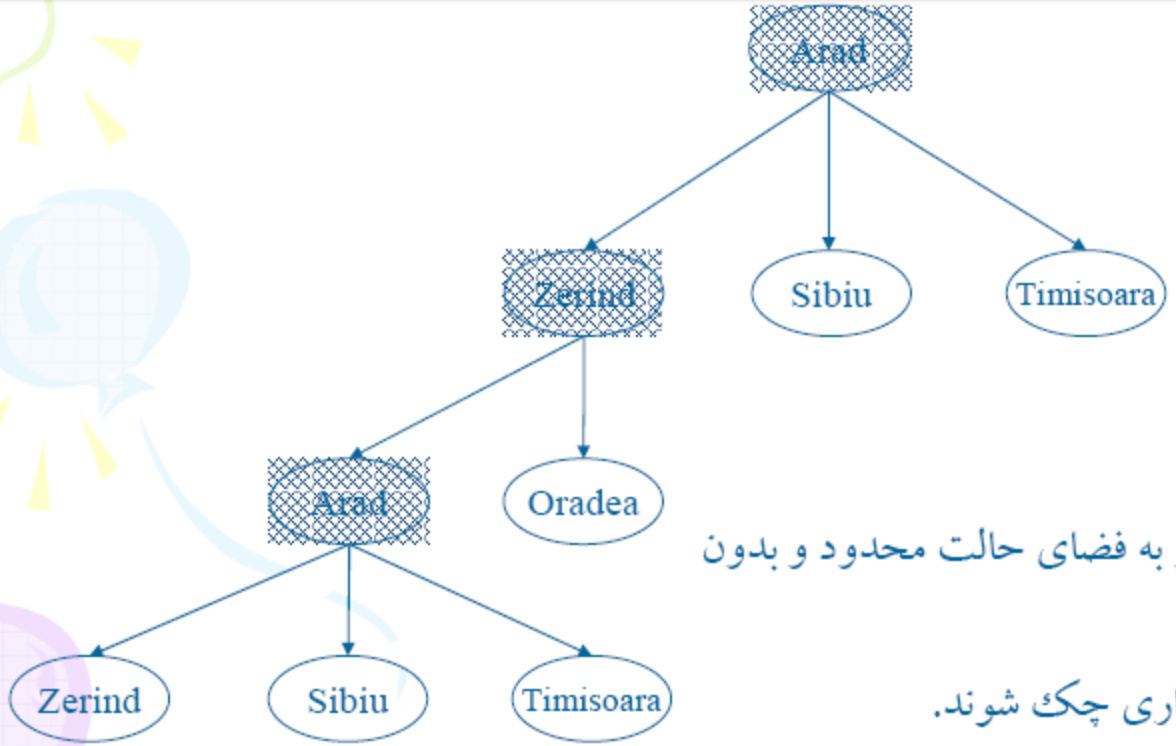


جستجوی اول عمق (DFS)

- همیشه عمیقترین گره در لبه فعلي از درخت جستجو را گسترش مي دهد.
  - اما اگر به نتیجه نرسید ، به سراغ گره هایي در سطوح کم عمق تر مي رود.
- پیاده سازي توسط پشته (LIFO or Stack) انجام مي‌شود.
  - این جستجو ، نیاز به حافظه نسبتاً کمي فقط براي ذخیره مسیر واحدي از ریشه به یك گره برگي داشته ، و گره هاي باقي مانده بسط داده نشده نیاز به حافظه ندارد.
- مشکل : حلقه بي پایان!

www.students.madadyar.com

جستجوی اول عمق (DFS)



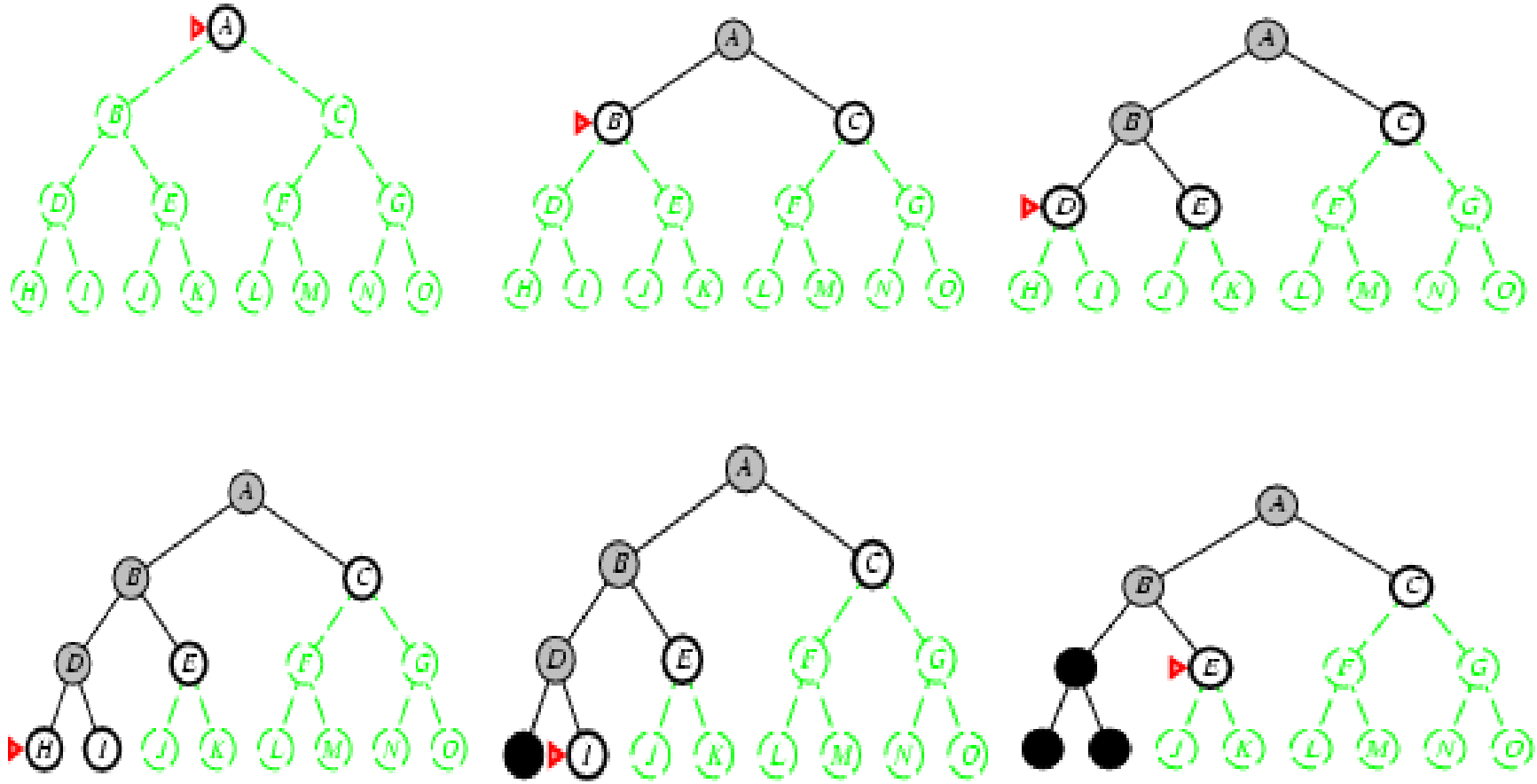
حلقه بی پایان !

در این جستجو نیاز به فضای حالت محدود و بدون چرخه داریم،

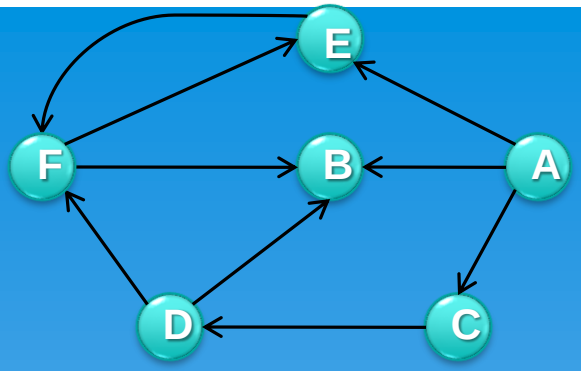
یا باید حالات تکراری چک شوند.

www.students.madadyar.com

مثال :جستجوی عمقی



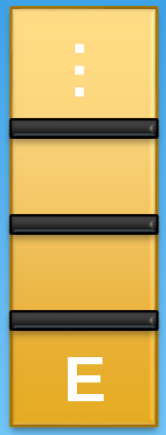
جستجوی عمقی به کمک پشته



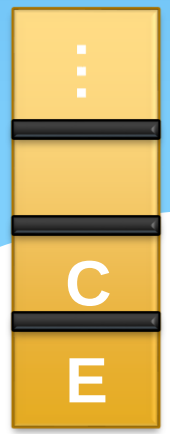
1 A

3 ABC

5 ABCDF



7 ABCDFE



2 AB

4 ABCD

6 ABCDF

8 ABCDFE

جستجوی اول عمق (DFS)

- **Complete?** No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path.
  - Complete in finite spaces

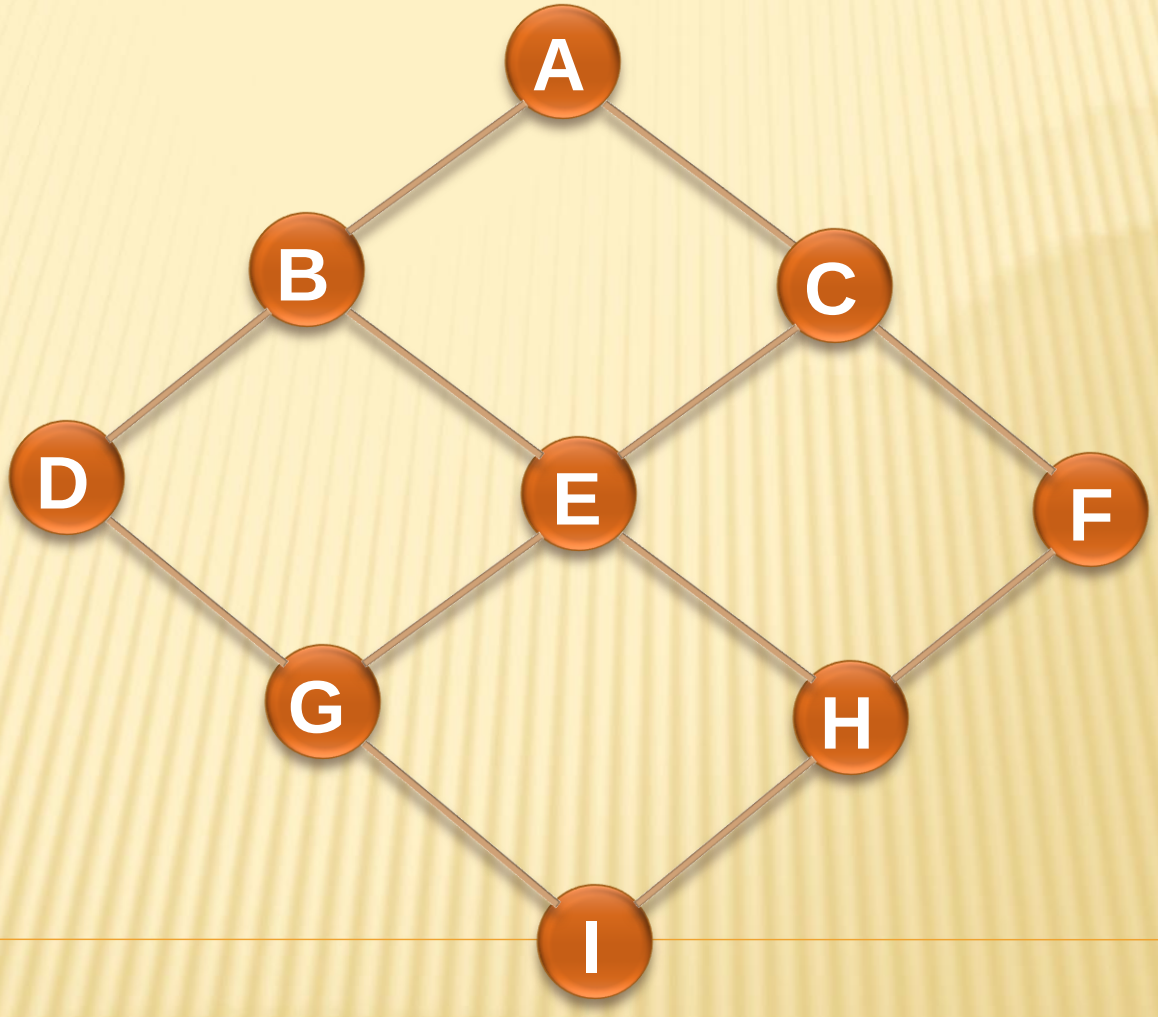
- **Time?** $O(b^m)$: terrible if m is much larger than d.

- **Space?** $O(bm)$, i.e., linear space!

- **Optimal?** No.

www.students.madadyar.com

سوال : الگوریتم جستجوی عمقی را روی گراف زیر پیاده کنید ؟



جستجوی عمقی محدود (DLS)

- مسئله درختهای نامحدود مي تواند به وسيله جست و جوی عمقي با عمق محدود L بهبود يابد.
- يعني گرههاي با عمق L گسترش نمي يابند.
- مثلا در مساله روماني حداكثر عمق راه حل ۱۹ مي تواند باشد !

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

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  cutoff_occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff_occurred? ← true
      else if result ≠ failure then return result
  if cutoff_occurred? then return cutoff else return failure
```

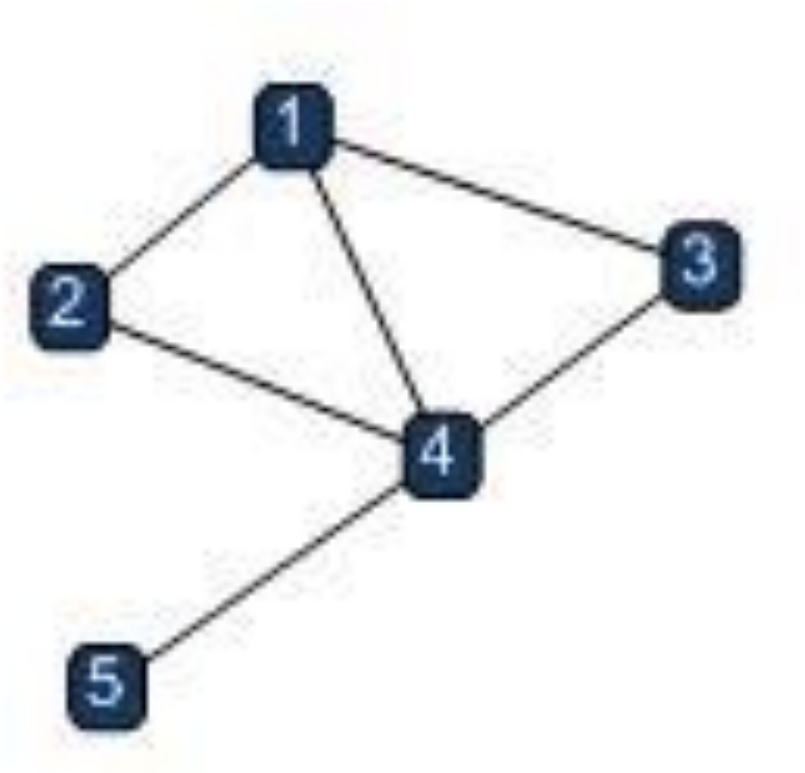
www.students.madadyar.com

جستجوی عمقی محدود(DLS)

- **Complete?** **Yes, if L≥d.**
- **Time?** $O(b^L)$
- **Space?** $O(bL)$, **i.e., linear space!**
- **Optimal?** **No.**
  - **Yes, if l=d and step cost has constant value.**

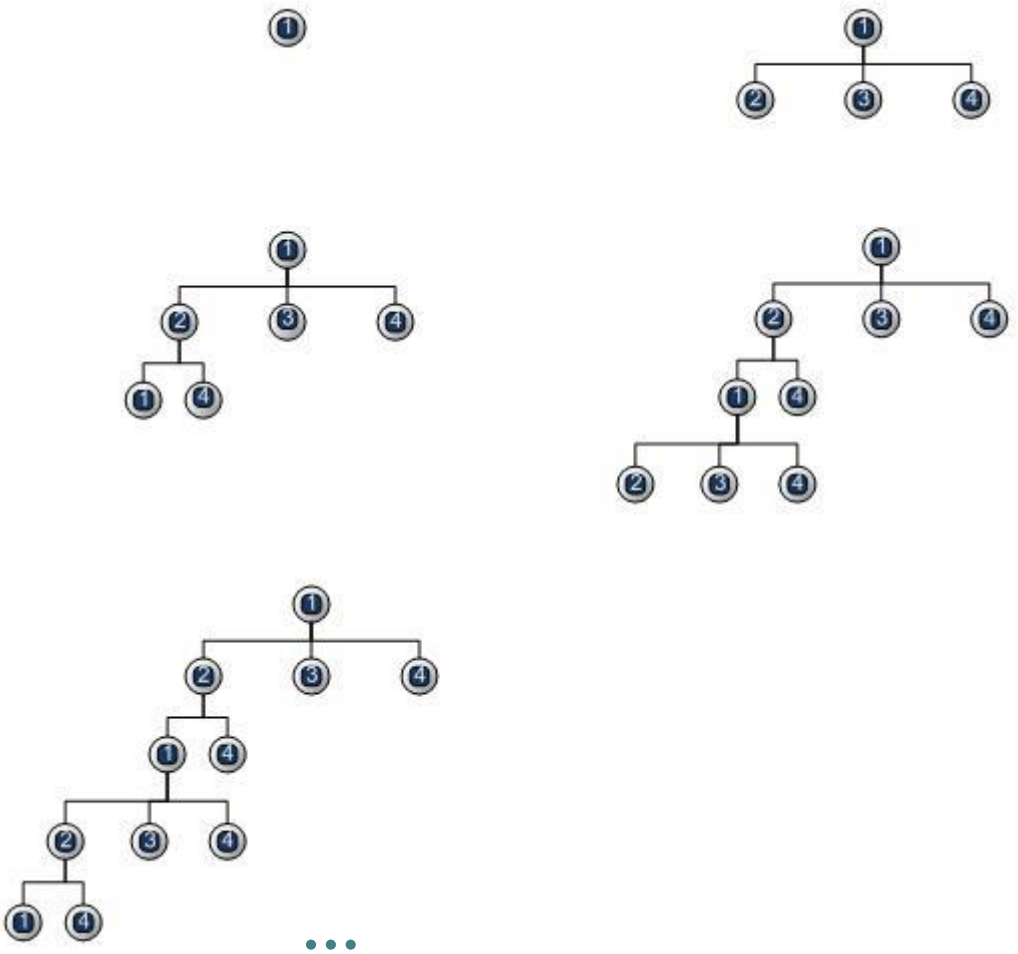
www.students.madadyar.com

- تمرين: گراف زير را به روش جستجوي عمقي و عمقي محدود شده براي يافتن مسيري از ۱ به ۵ بيابيد؟



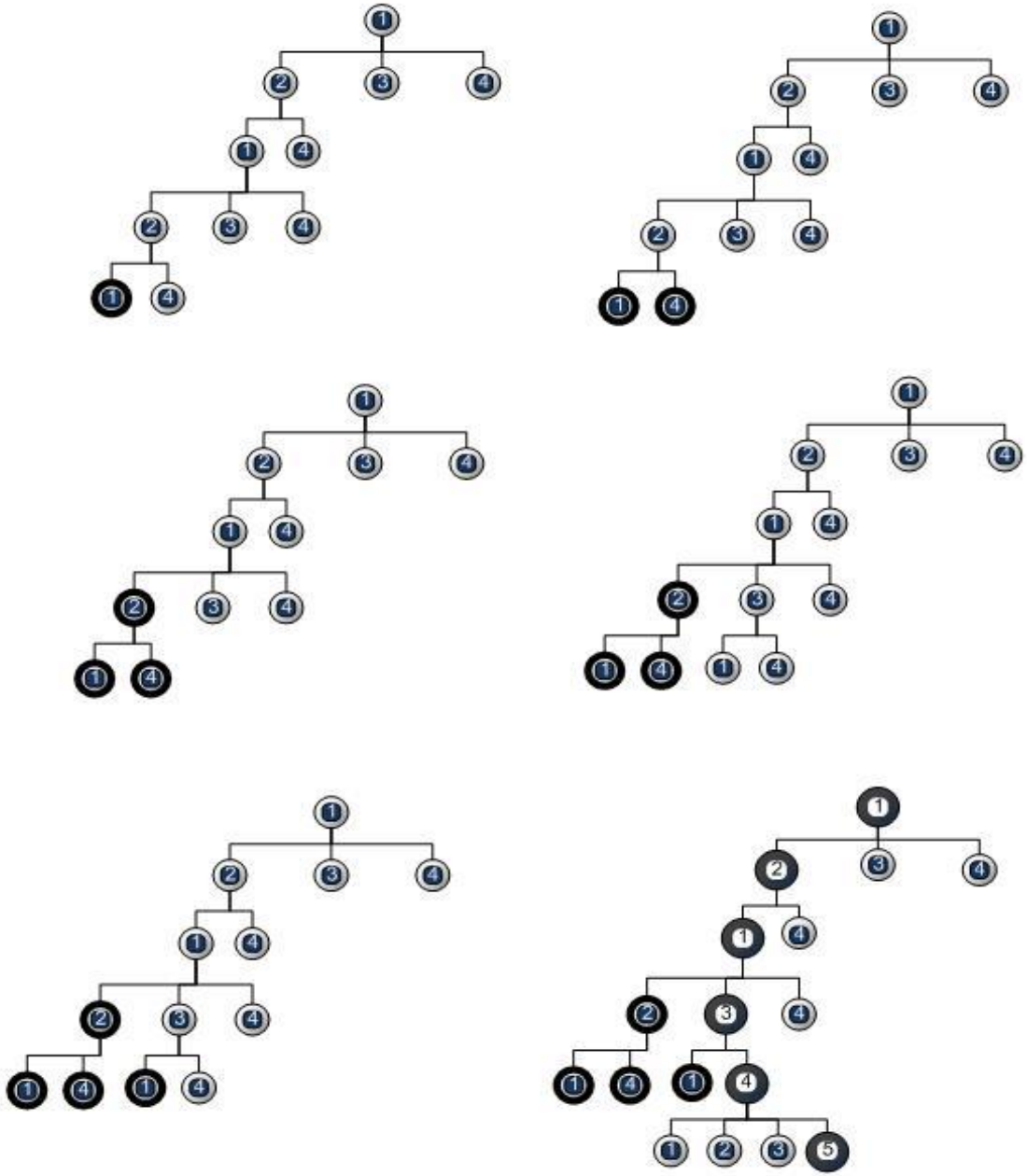
www.students.madadyar.com

جستجوی عمقی (DFS)



www.students.madadyar.com

جستجوی عمقی محدود شده (DLS)
(L=5)



www.students.madadyar.com

جستجوی عمیق کننده تکراری (IDS)

- برای بیشتر مسائل، محدودهٔ عمقی مناسب را تا زمانی که مسئله حل نشده است، نمی شناسیم.
- جستجوی عمیق کنندهٔ تکراری استراتژی است که نظریه انتخاب بهترین محدودهٔ عمقی، توسط امتحان تمام محدودهٔ مسیرهای ممکن را یادآوری می کند.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

www.students.madadyar.com

Iterative deepening search *l* =0

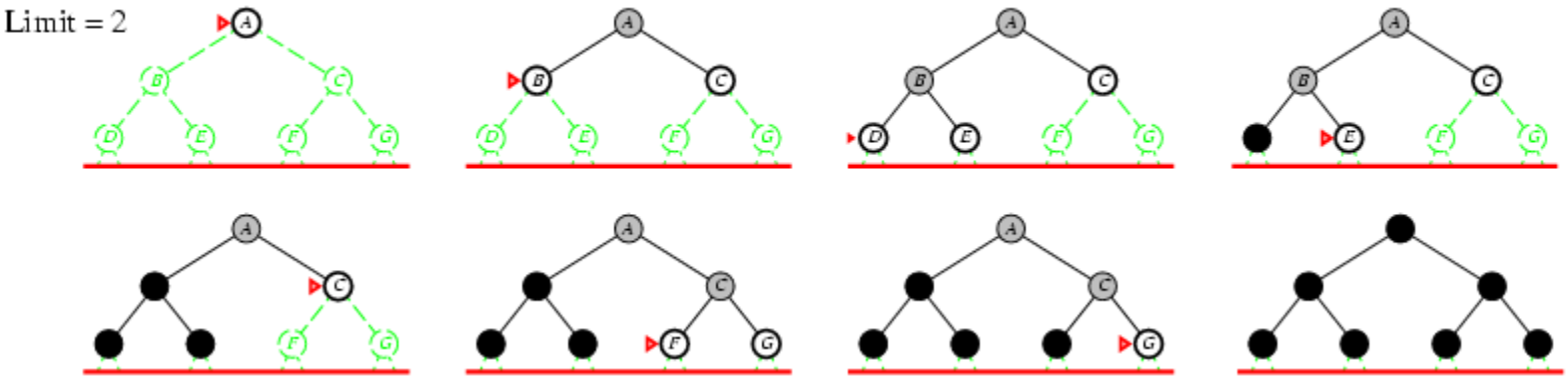
Limit = 0

www.students.madadyar.com

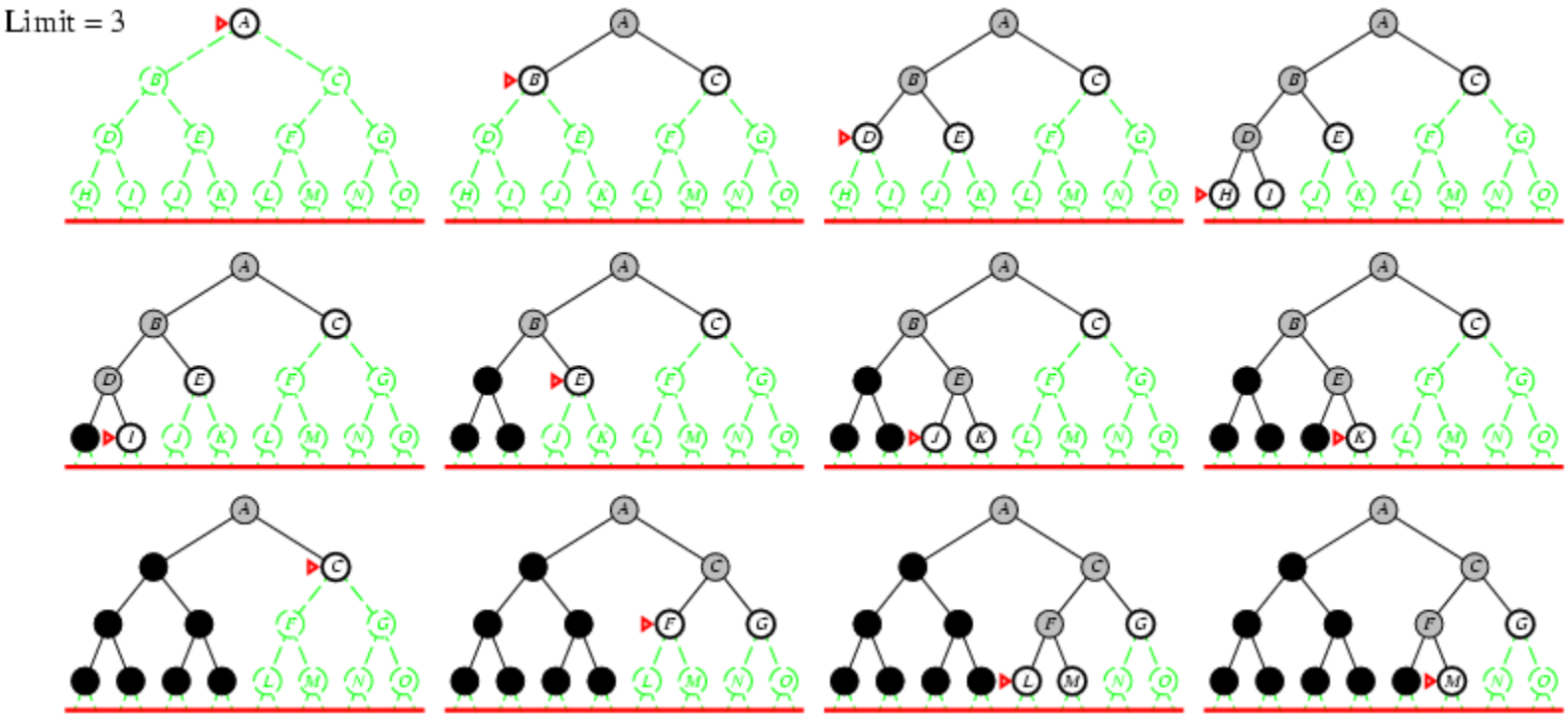
Iterative deepening search *l* =1

www.students.madadyar.com

Iterative deepening search *l* =2



Iterative deepening search *l* =3



- Number of nodes generated in a depth-limited search to
- depth *d* with branching factor *b*:
-

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening
- search to depth *d* with branching factor *b*:

$$N_{IDS} = (d+1)b^0 + d\, b^{\wedge 1} + (d-1)b^{\wedge 2} + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- For $b = 10$, $d = 5$,
-
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  -
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$

- Overhead = (123,456 - 111,111)/111,111 = 11%

www.students.madadyar.com

جستجوی عمیق کننده تکراری (IDS)

- Complete? Yes , (if b is finite).
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$.
- Space? $O(bd)$.
- Optimal? Yes, if step cost = 1.

www.students.madadyar.com

جستجوی عمیق کننده تکراری (IDS)

در حالت کلی، عمیق کننده تکراری، روش جستجوی برتری است؛ زمانی که فضای جستجوی بزرگی وجود دارد و عمق راه حل نیز مجهول است.

مرتبه بسط حالات مشابه جستجوی سطحی است، به جز اینکه بعضی حالات چند بار بسط داده می شوند.

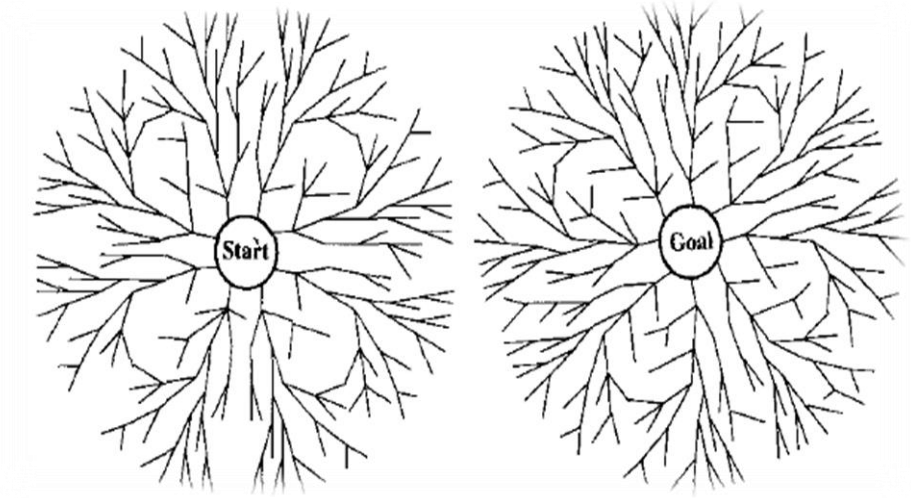
این جستجو مانند جستجوی سطحی کامل و بهینه است، اما فقط مزیت درخواست حافظه اندک را از جستجوی عمقی دارد.

ترکیبی از مزایای جستجوی سطحی و عمقی را دارد.

www.students.madadyar.com

جستجوی دو طرفه (BS)

- انجام دو جست و جوی همزمان، یکی از حالت اولیه به هدف و دیگری از هدف به حالت اولیه تا زمانی که دو جست و جو به هم برسند. (زمانی انجام پذیر است عملیات معکوس پذیر باشند)
- نیاز به فضا نقطه ضعف این روش است.



- forward
- backward

- **جستجو زمانی خاتمه می یابد که هر دو جستجو به یک گره مشترک برسند.**

www.students.madadyar.com

جستجوی دو طرفه

- Complete? Yes , (if both of searches are BFS and b is finite)
- Time? $O(b^{d/2})$
- Space? $O(b^{d/2})$
- Optimal? Yes, (if both of searches are BFS and cost is fixed)

www.students.madadyar.com

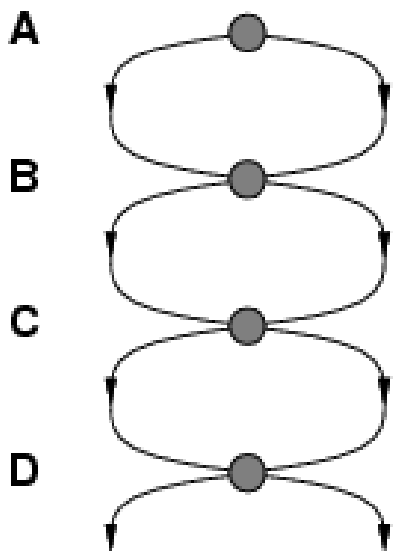
مقایسه

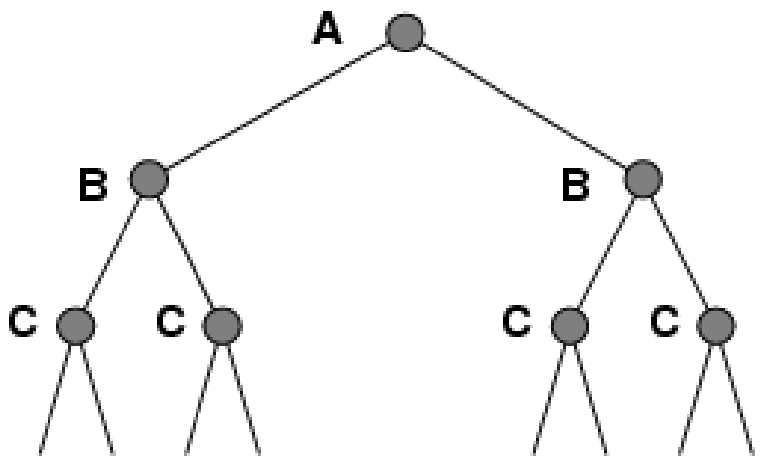
Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	YES*	YES*	NO	YES*	YES*	YES*
Time	b^{d+1}	$b^{C^*/e}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{C^*/e}$	bm	bl	bd	$b^{d/2}$
Optimal?	YES*	YES	NO	NO	YES*	YES*

www.students.madadyar.com

اجتناب از حالتهای تکراری

- وجود حالتهای تکراری در یک مسئله قابل حل ، مي تواند آن را به مسئله غیر قابل حل تبدیل کند.
- و یا یک مساله خطي را به یک مساله نمايي تبدیل کند !!





www.students.madadyar.com

- مقايسه گره ها قبل از گسترش با گرههايي كه قبلا گسترش يافته اند.

- زمان ، حافظه ؟!؟
  - الگوريتم هايي كه گذشته را فراموش مي كنند مجبور به تكرار آن هستند.

- فهرست Closed
  - تمام گرههاي گسترش يافته.

- فهرست open
  - گرههاي لبه كه تاكنون گسترش نيافته اند.

www.students.madadyar.com

جستجوی گراف

- افزودن قابلیت close list به Tree search باعث پیدایش الگوریتم جستجوي گراف (graph search) مي شود.
- اين الگوريتم تمامي گرهها را در حافظه نگه داري مي كند.
  - محدوديت در به كارگيري در برخي مسائل.

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure

  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
      if EMPTY?(fringe) then return failure
      node ← REMOVE-FIRST(fringe)
      if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
      if STATE[node] is not in closed then
          add STATE[node] to closed
          fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

www.students.madadyar.com

جستجوی آگاهانه

HEURISTIC

www.students.madadyar.com

جستجوی آگاهانه

- جستجوي ناآگاهانه فقط با توليد حالتهاي جديد و بررسي حالات براي دستيابي به هدف سعي در يافتن راه حل مساله دارد.

- **جستجوي آگاهانه از دانش مساله براي حل آن استفاده مي كند.**

- در جستجوي آگاهانه يک تابع ارزياب به نام $f(n)$ براي بررسي ميزان **اميد بخش** بودن يک گره براي گسترش وجود دارد.

هيوريستيک

- در جستجوي آگاهانه اطلاعاتي در رابطه با هزينه رسيدن به هدف ، در اختيار عامل قرار داده مي شود.

- هيوريستيک ها (Heuristic)
  - استراتژيهايي براي جستجو که اغلب اوقات ما را به جواب نزديکتر مي کنند.
  - از ساختار مساله نشات مي گيرند و هدفشان راهنمايي و هدايت جستجو مي باشد.

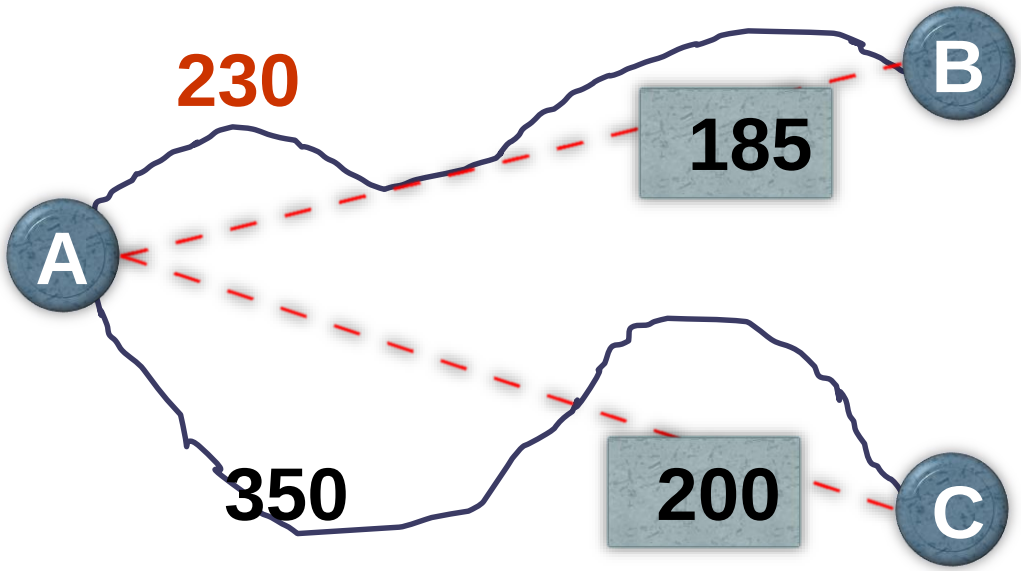
www.students.madadyar.com

هیوریستیک (Heuristic)

- نکته کلیدي در الگوریتم هاي جستجوي آگاهانه تابع هیوریستیک h(n) است.
- **h(n) : هزینه تخمیني ارزانترین مسیر از گره n تا هدف.**
  - مانند مسافت خط مستقیم مابین شهرها در مساله نقشه روماني
- اگر n گره هدف باشد در اینصورت h(n)=0.
- **در جستجوي آگاهانه طراحي مناسب تابع h(n) مساله حیاتي است!!!**

www.students.madadyar.com

یک مثال از تابع هیورستیک



State	h(n)
B	185
C	200

متدهای جستجوی آگاهانه

جستجوي اول بهترين

حریصانه (Greedy)

A*

IDA*

RBFS

MA* و SMA*

جستجوی محلی و بهینه سازی

تپه نوردی (HC)

شبیه سازی حرارت (SA)

پرتو محلی (LBS)

الگوریتمهای ژنتیک (GA)

www.students.madadyar.com

تعاریف

- تابع هزینه مسیر، $g(n)$: هزینه مسیر از گره اولیه تا گره n
- تابع اکتشافی، $h(n)$: هزینه تخمینی ارزان ترین مسیر از گره n به گره هدف.
- تابع بهترین مسیر، $h^*(n)$: ارزان ترین مسیر از گره n تا گره هدف.
- تابع ارزیابی، $f(n)$: هزینه تخمینی ارزان ترین مسیر از طریق n.

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

- $f^*(n)$: هزینه ارزان ترین مسیر از طریق n

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

www.students.madadyar.com

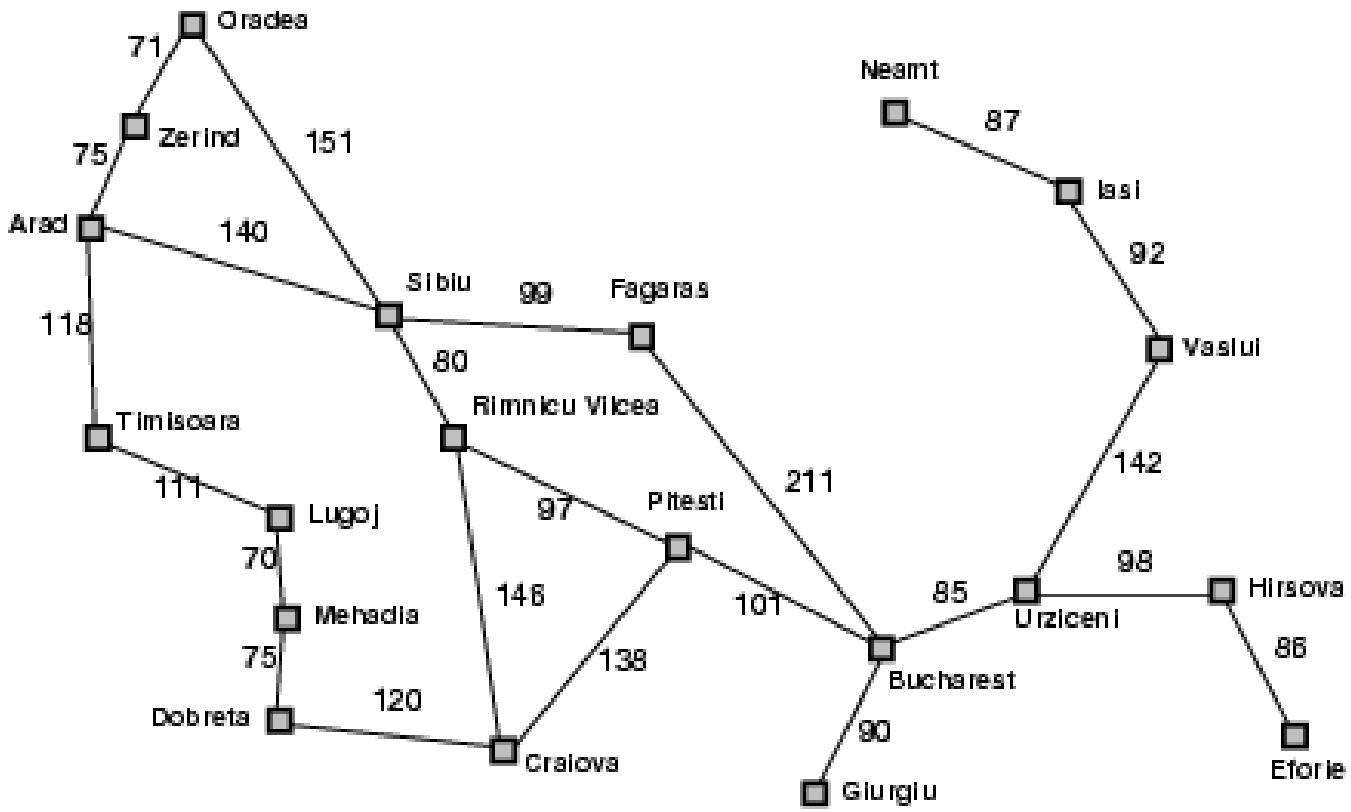
یکنوایی (monotony)

- يعني تابع f در طول مسير غير نزولي باشد.
  - يعني با افزايش گام راه حل ميزان هزينه طي شده افزايش يابد.

www.students.madadyar.com

جستجوی حریصانه (greedy search)

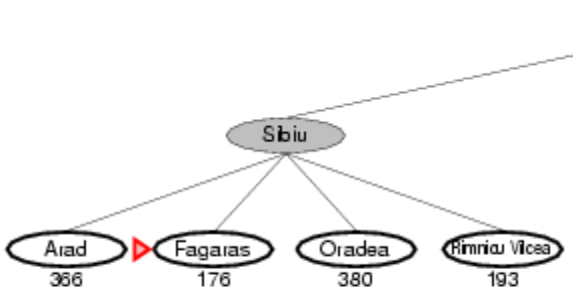
- سعي در گسترش نزديكترين گره به هدف را دارد.
- F(n)=h(n)
- اگر گره هاي تكراري شناسايي نشوند شايد الگوريتم به <u>جواب نرسد.</u>
  - فرض كنيد در مساله روماني h نشان دهنده فاصله مستقيم تا شهر هدف باشد. در اين صورت جستجو در حال نوسان بين Neamt و Iasi خواهد بود (حالت اوليه:Iasi ، هدف: Fagares).
- پياده سازي
  - **صف بر اساس ميزان مطلوبيت گره ها مرتب شده باشد.**

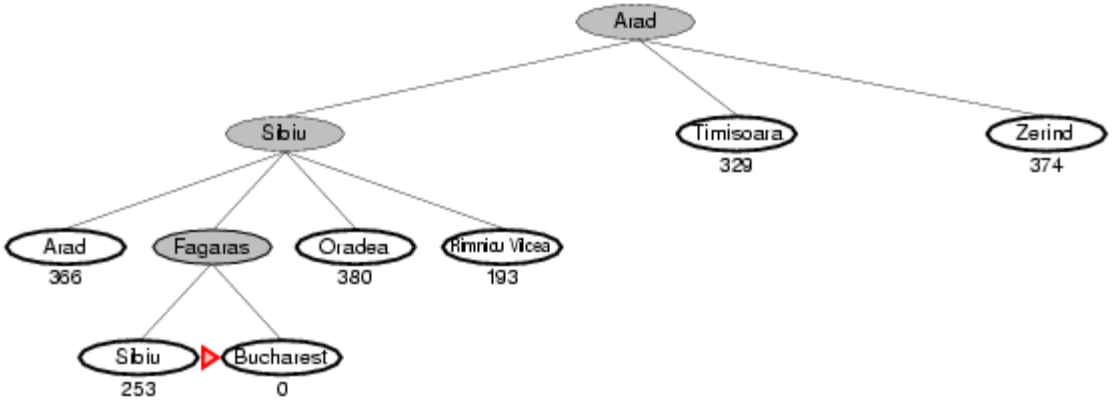


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

www.students.madadyar.com

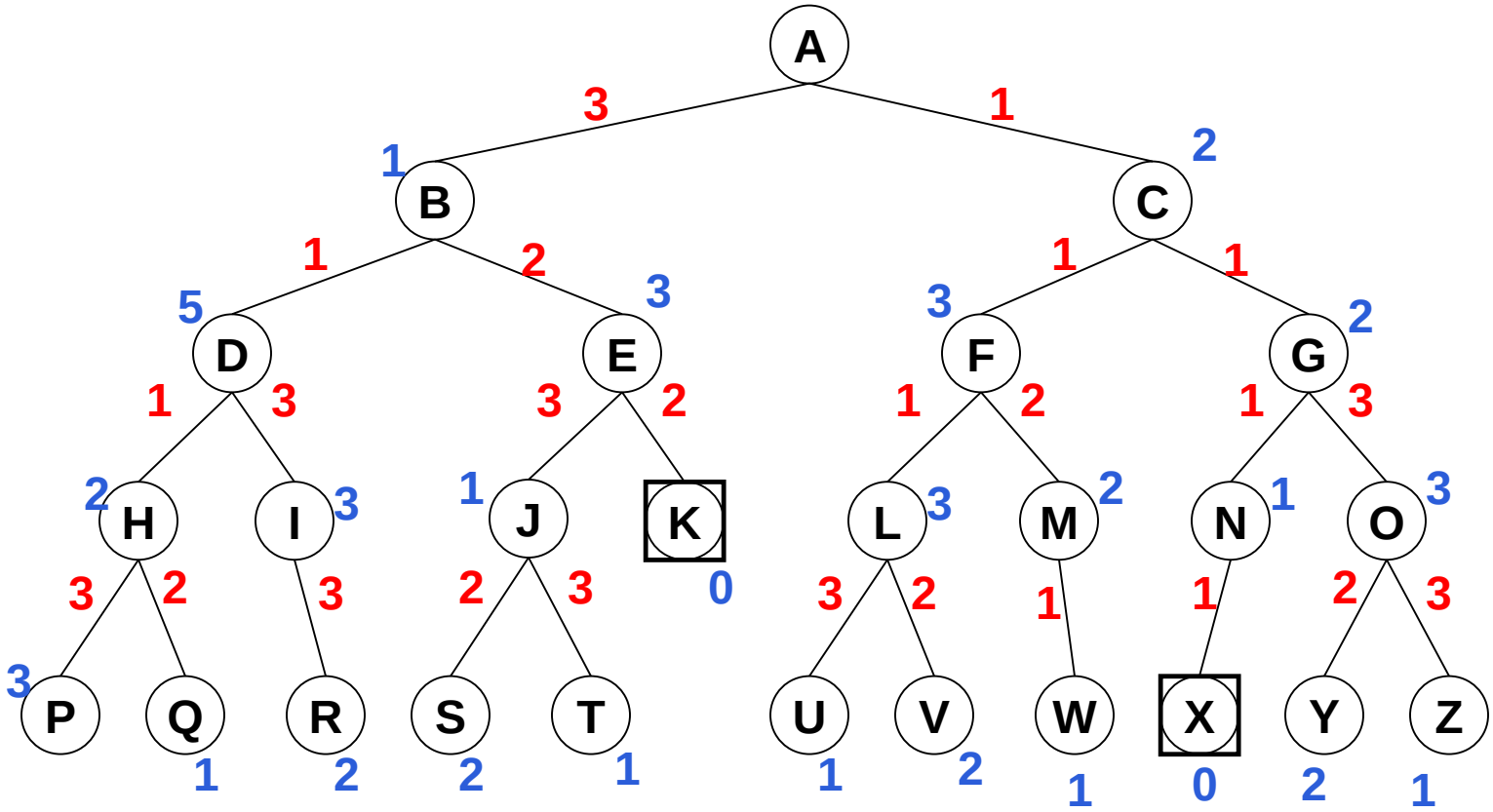
جستجوی حریصانه در مساله رومانی





www.students.madadyar.com

مثال ۱ : جستجوی حریصانه

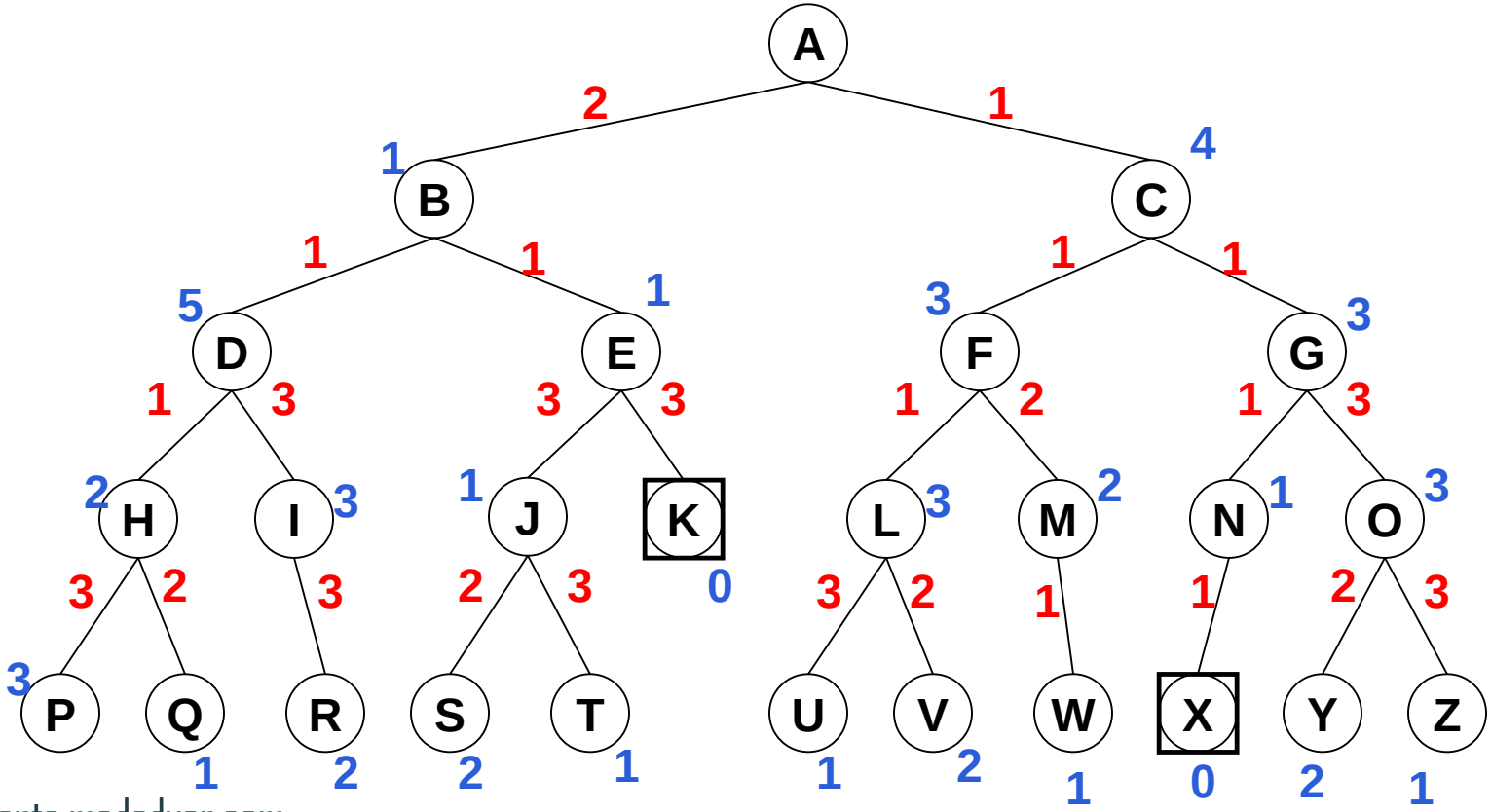


مثال ۱ : جستجوی حریصانه

A

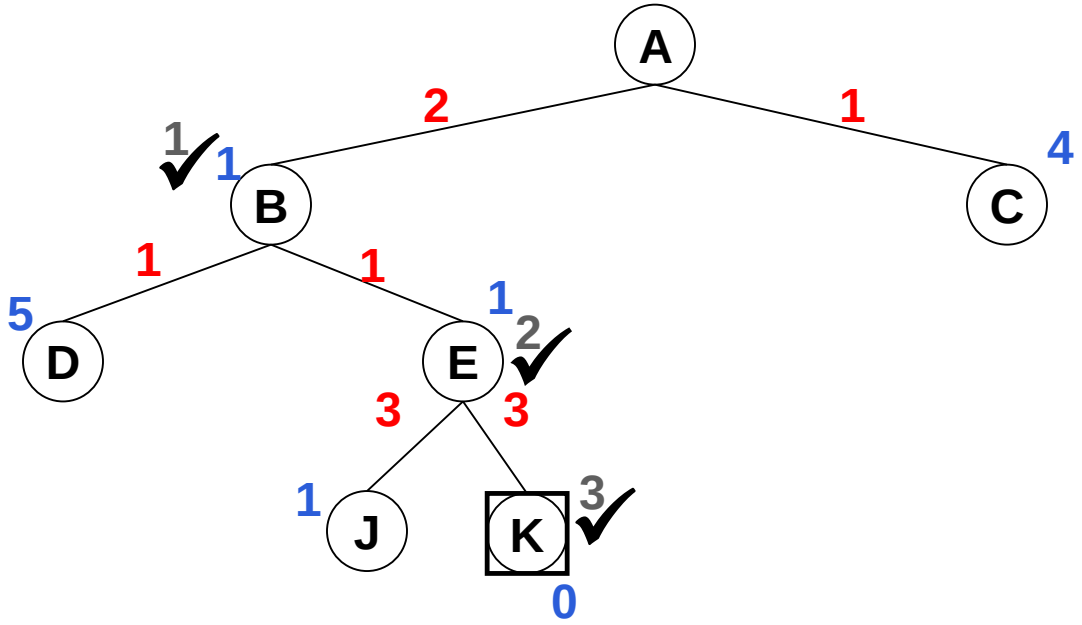
www.students.madadyar.com

مثال ۲ : جستجوی حریصانه



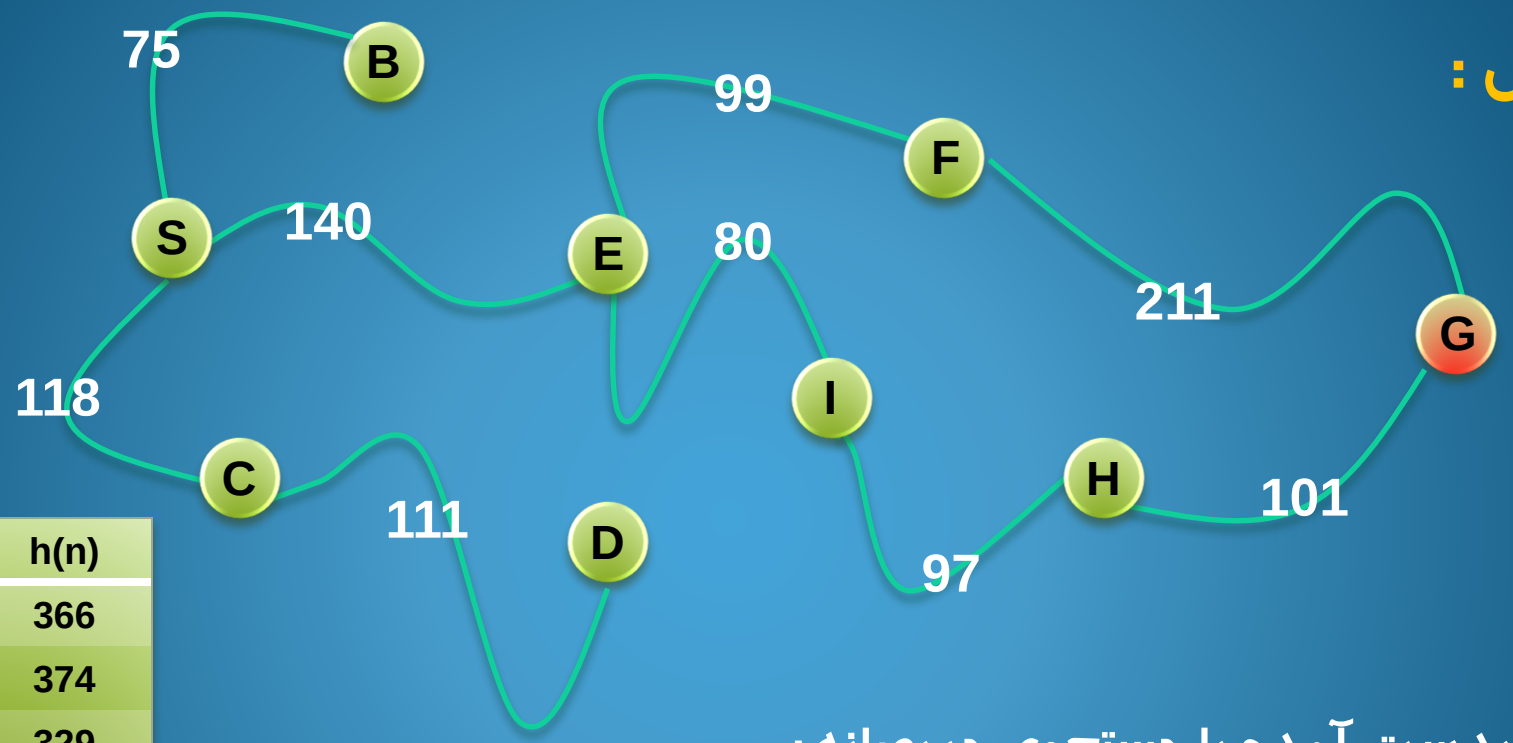
www.students.madadyar.com

مثال ۲ : جستجوی حریصانه



مسیر یابی با استراتژی حریصانه بهینه نیست

مثال :



State	h(n)
S	366
B	374
C	329
D	244
E	253
F	178
I	193
H	98
G	0

مسیر بدست آمده با جستجوی حریصانه :

S-E-F-G = 140 + 99 + 211= 450

مسیر بدست آمده بهینه :

S-E-I-H-G = 140 + 80 + 97+101= 418

جستجوی حریصانه

Complete? No – can get stuck in loops,
e.g., Iasi → Neamt → Iasi → Neamt →

Time? $O(b^m)$,
but a good heuristic can give dramatic improvement.

Space? $O(b^m)$
keeps all nodes in memory

Optimal? No

www.students.madadyar.com

حالات خاص در الگوریتم حریصانه

اگر $h = h^*$ آنگاه جستجو کامل می شود.

اگر $h = h^*$ آنگاه جستجو بهینه می شود.

جستجوی A*

- در اين جستجو از بسط گرههايي كه تاكنون پر هزينه بودن آنها اثبات شده پرهيز مي شود.
- در حقيقت جستجوي A* تركيب مزاياي جستجو با هزينه يكسان و جستجوي حريصانه مي باشد.
- تابع ارزيابي در A*
- $f(n) = g(n) + h(n)$
- در واقع f(n) هزينه تخميني ارزانترين راه حل از طريق n

www.students.madadyar.com

جستجوی A*

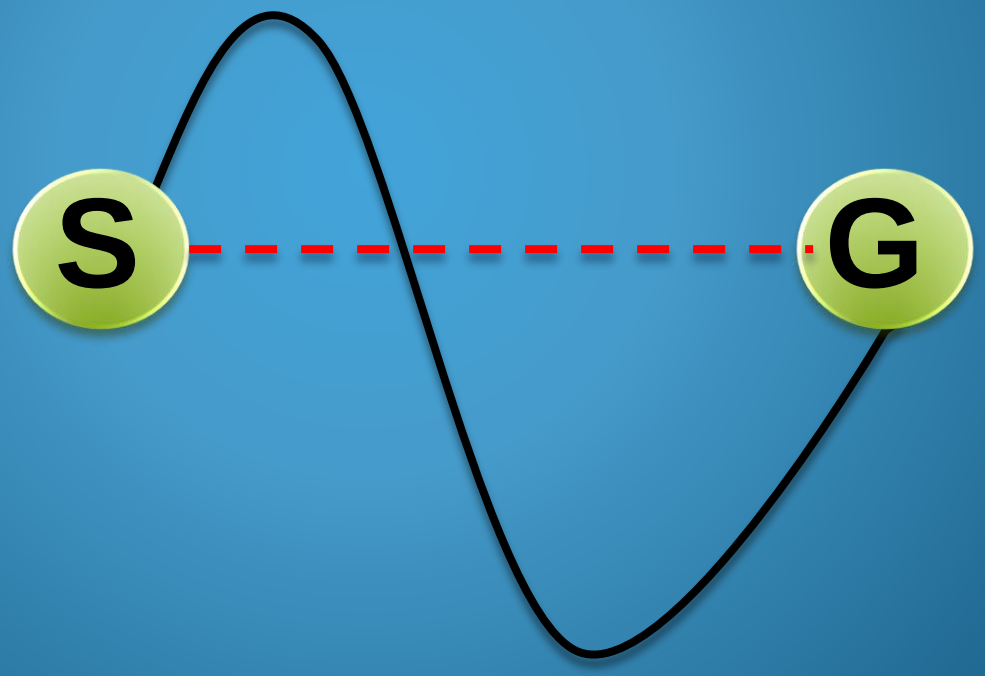
- بهینگي در A*
  - بهینه است اگر h(n) یک **هیوریستیک قابل قبول** باشد.

- هیوریستیک قابل قبول (Admissible)
  - یعني هزینه رسیدن به هدف را **بیش از اندازه واقعي** تخمین نزند.

  - $h(n) \le h^*(n)$
  - $h(n) \ge 0$

$$0 \le h(n) \le h^*(n)$$

سوال : چرا در مسله مسیر یابی هیورستیک فاصله مستقیم قابل قبول است ؟

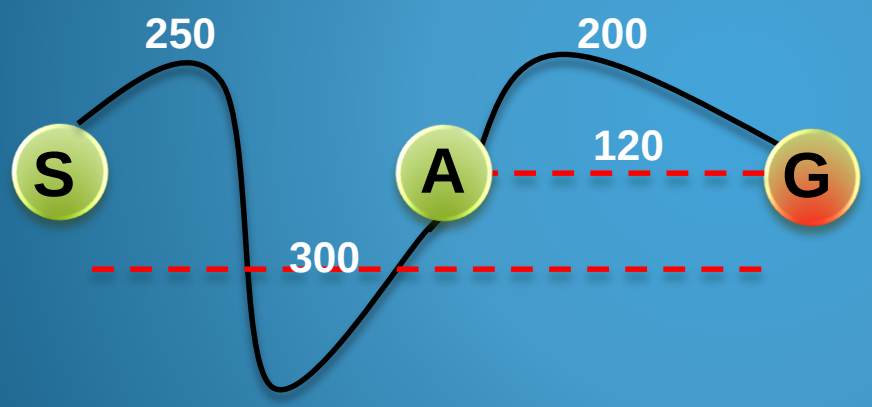


جستجوی اول بهترین با استراتژی الگوریتم A*

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

هزینه واقعی رسیدن از گره شروع به گره n

هزینه تقریبی رفتن از گره n به گره هدف



محاسبه تابع f(n) در گره S

$f(n) = 0 + 300 = 300$

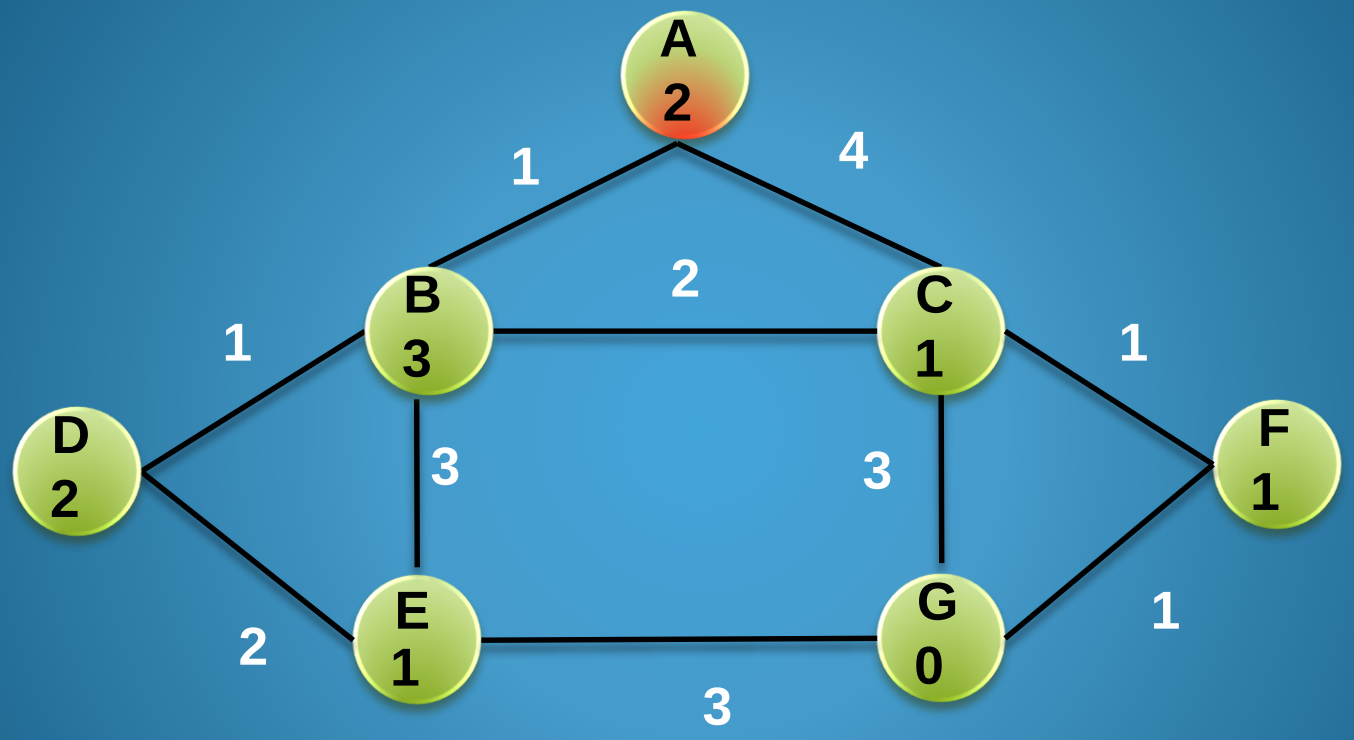
محاسبه تابع f(n) در گره A

$f(n) = 250 + 120 = 370$

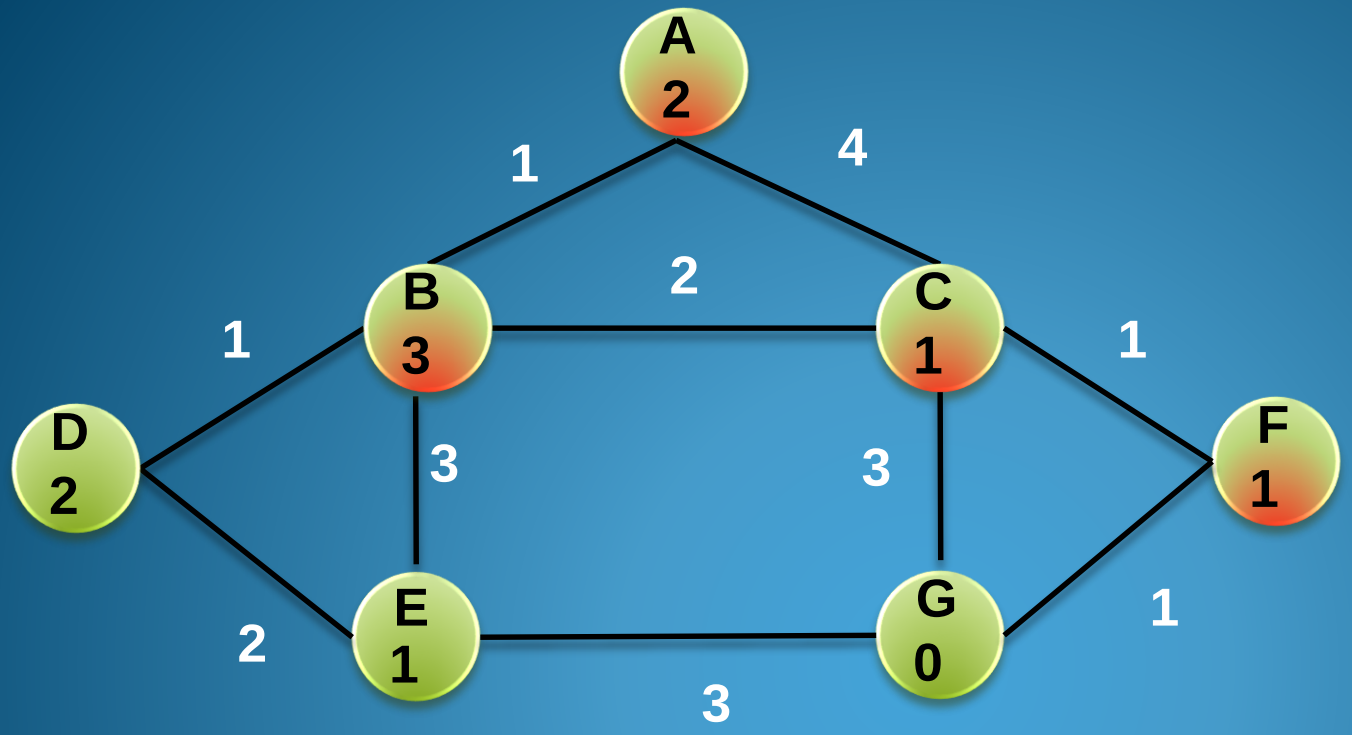
محاسبه تابع f(n) در گره G

$f(n) = (250+200) + 0 = 450$

مثال : استراتژی * A را روی گراف زیر پیاده سازی کنید و مسیری که توسط این الگوریتم تولید می شود را محاسبه کنید ؟



جواب :



$f(G) = g(G) + h(G) =(1 + 2 + 3) + 0 = 6$

انتخابی دیگر نداریم و به هدف رسیدیم با مسیر زیر :

A B C F G هزینه آن ۵ می شود

البته چند مسیر دیگر داریم :

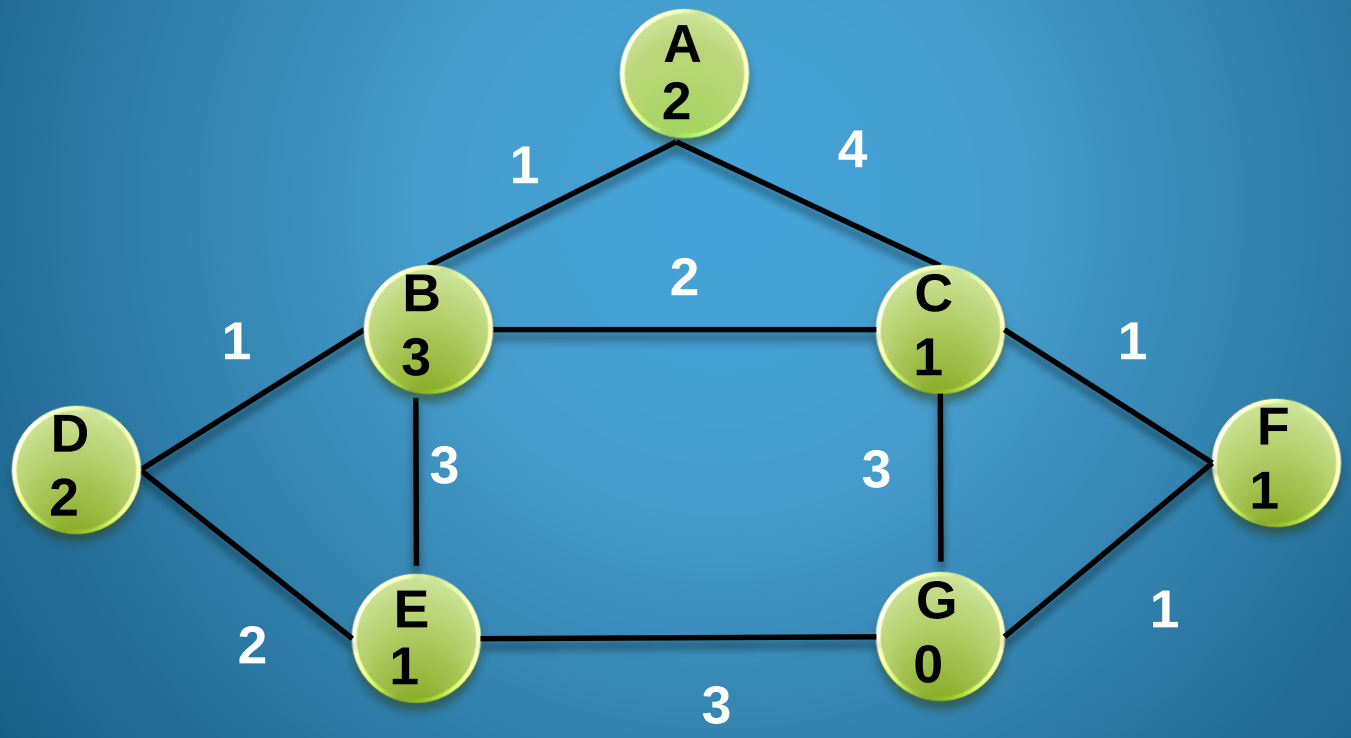
A C G هزینه آن ۷ می شود

A B C G هزینه آن ۶ می شود

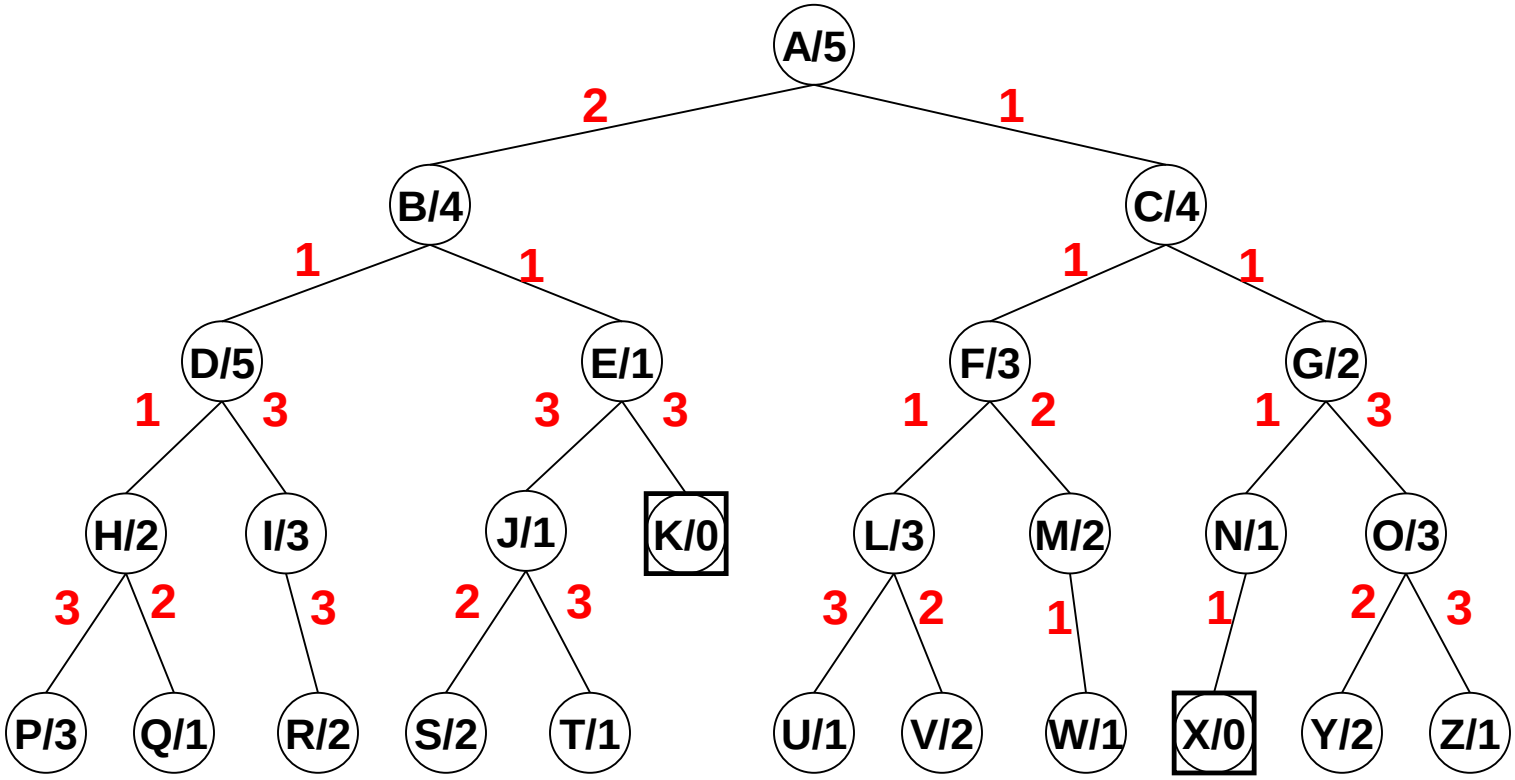
A B E G هزینه آن ۷ می شود

A* بهینه است به شرط اینکه تابع h(n) قابل قبول باشد.

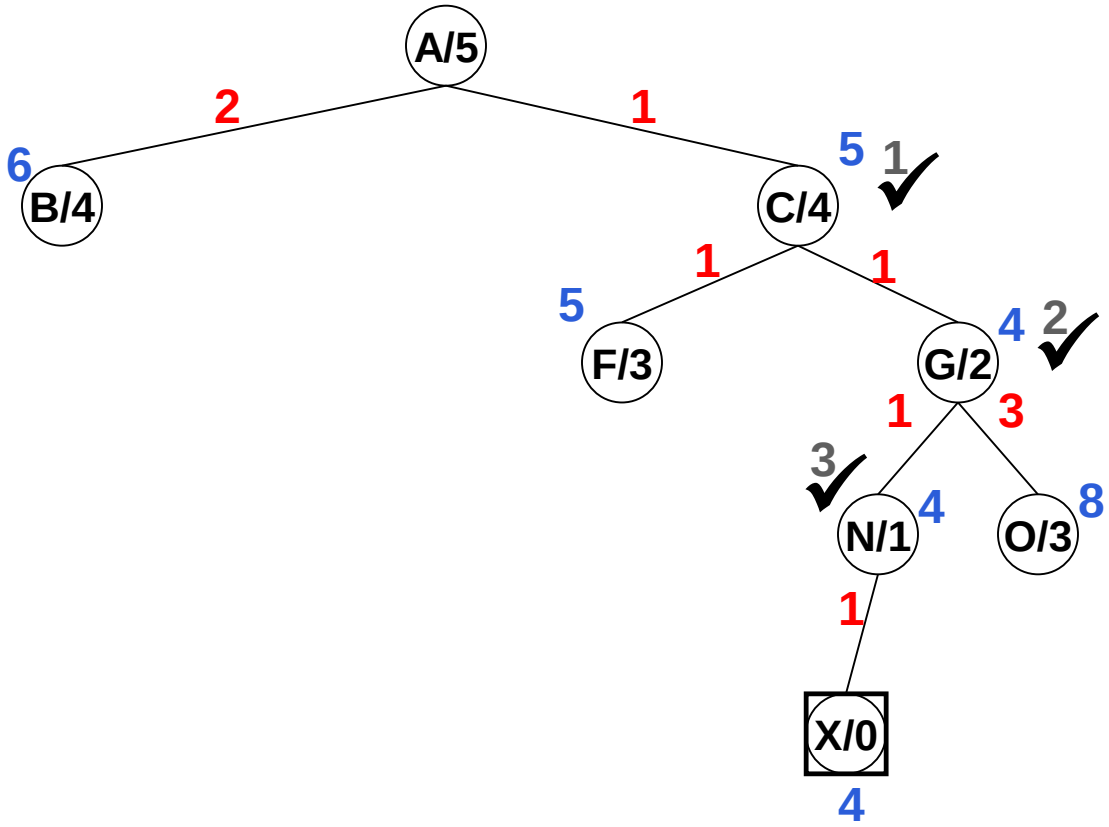
سوال : چرا تابع h(n) زیر قابل قبول است ؟



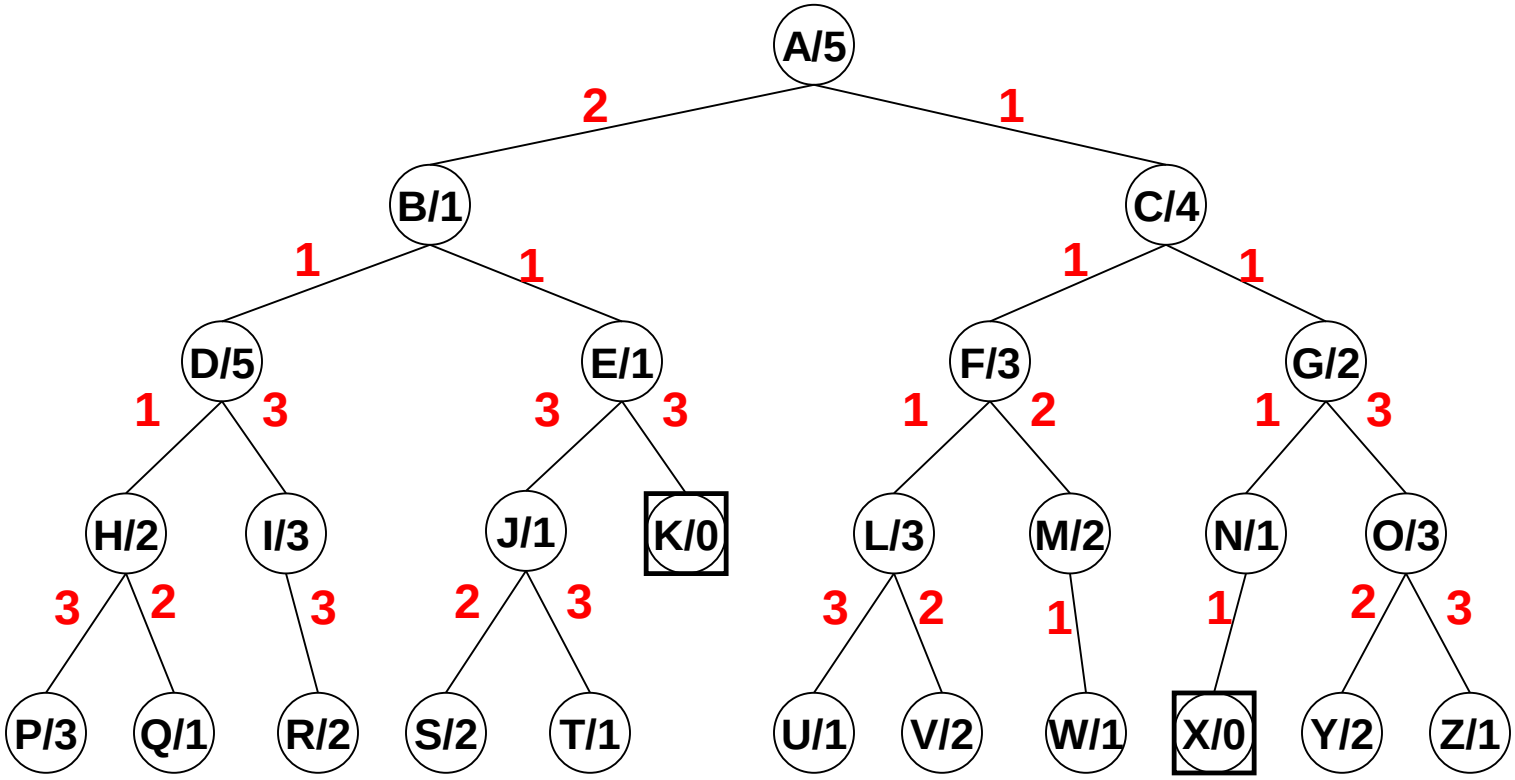
مثال : جستجوی A*



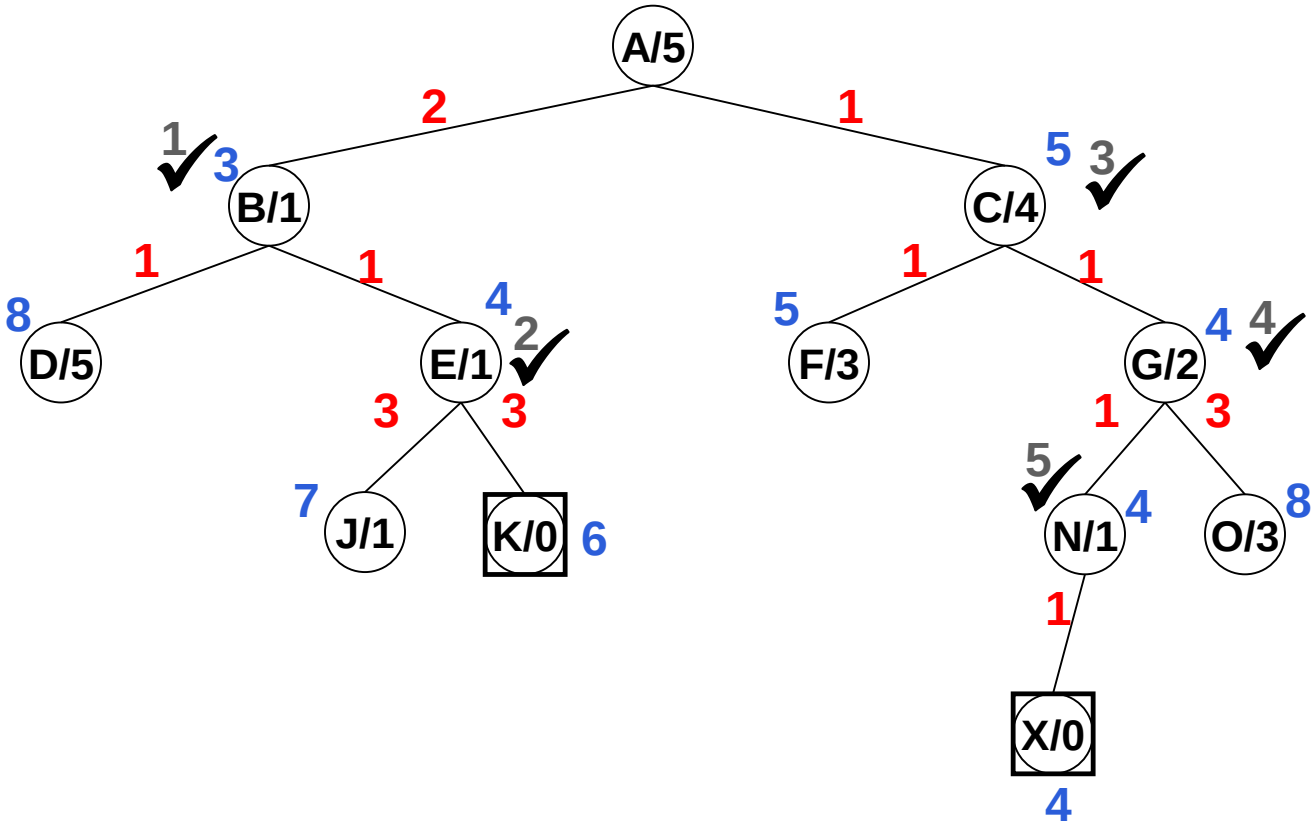
مثال : جستجوی A*



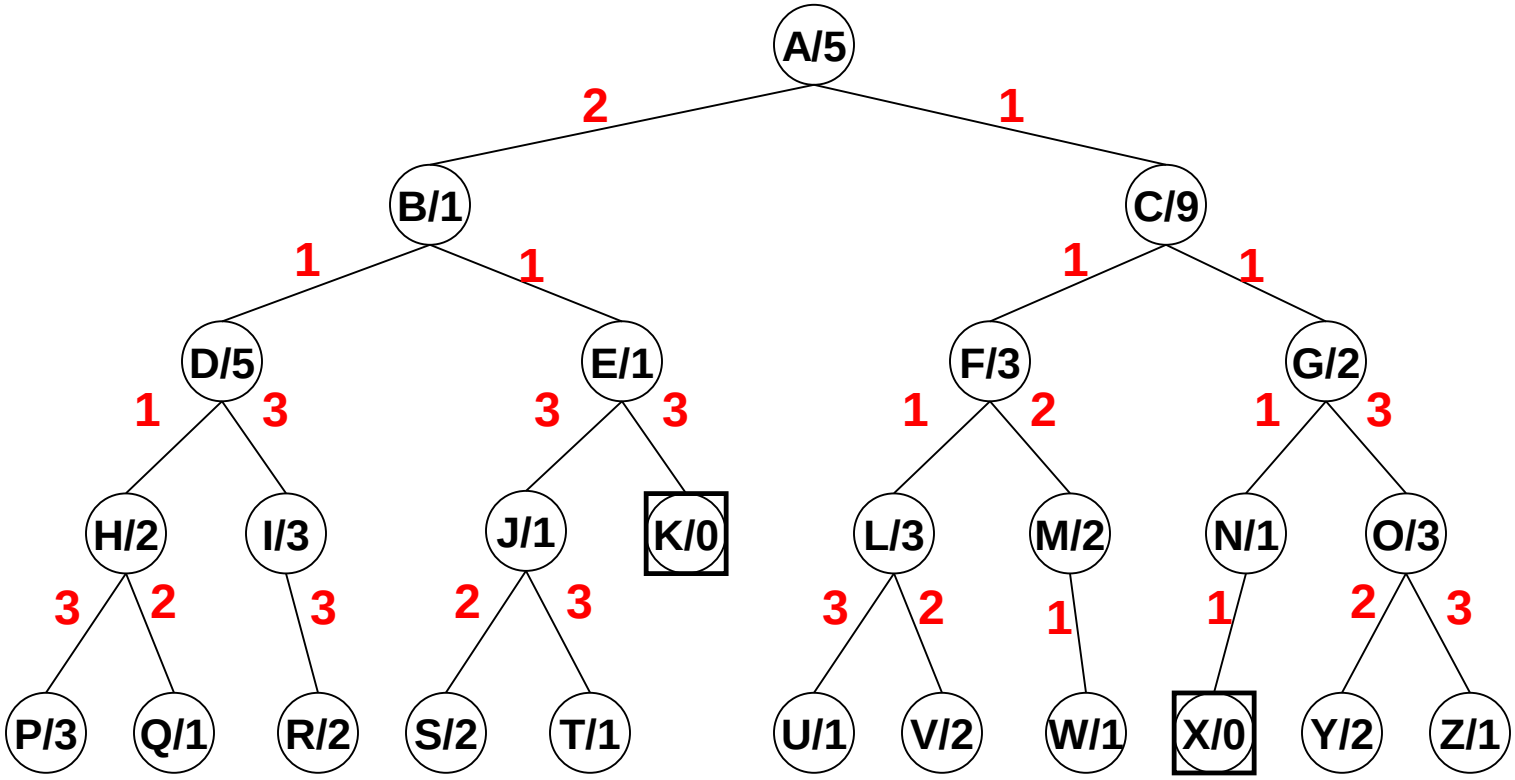
مثال ۲:جستجوی A*



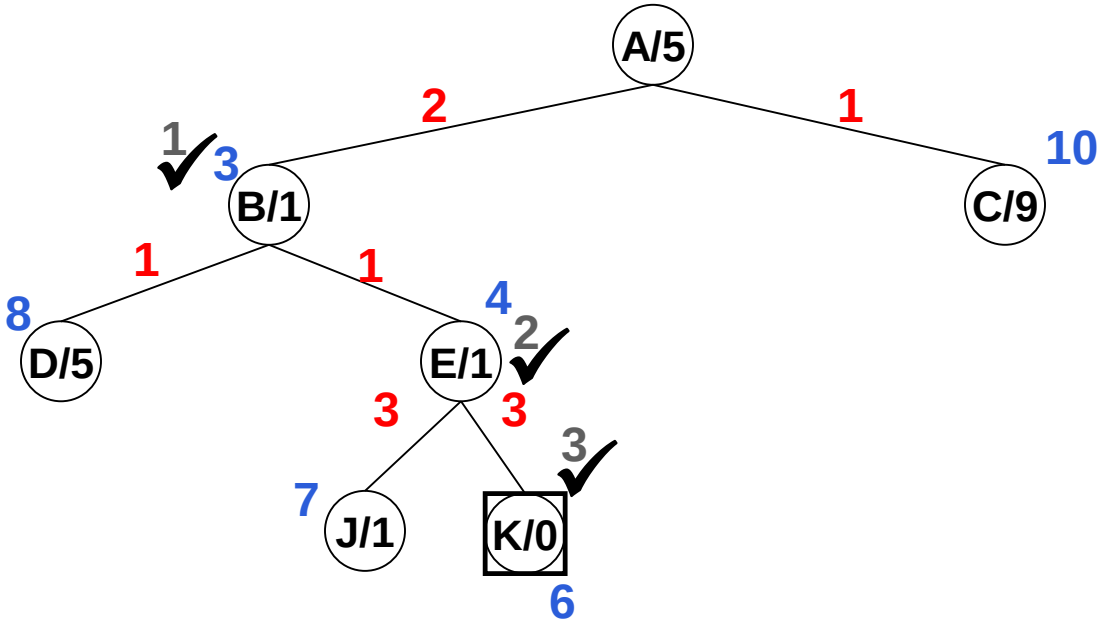
مثال ۲ :جستجوی A*



جستجوی A*

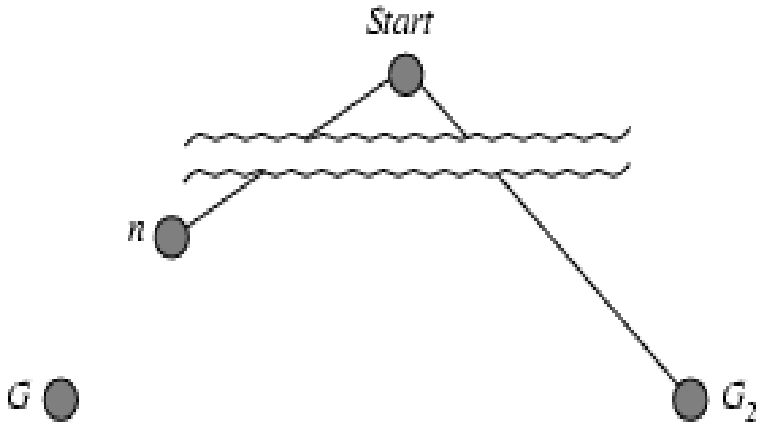

A/5
2
1
B/1
C/9
1
1
1
1
D/5
E/1
F/3
G/2
1
3
3
3
1
2
1
3
H/2
I/3
J/1
K/0
L/3
M/2
N/1
O/3
3
2
3
2
3
3
2
1
1
2
3
P/3
Q/1
R/2
S/2
T/1
U/1
V/2
W/1
X/0
Y/2
Z/1

جستجوی A*



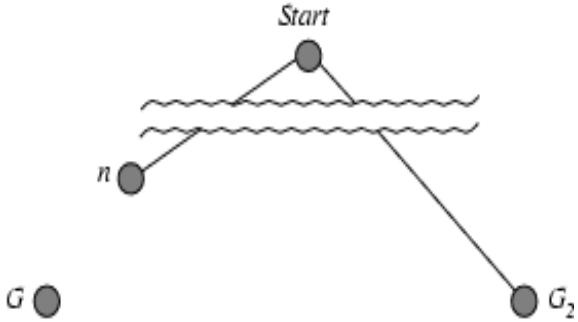
اثبات بهینگی در A*

- اگر G حالت هدف بهینه با هزینه h^* مسیر باشد.
- فرض کنید G2 یک هدف زیر بهینه (suboptimal) مي باشد. که هزینه یك هدف زیر بهینه $g(G_2)>h^*$.
- نشان مي دهیم A^* به هیچ وجه با این هدف نیمه بهینه خاتمه نمي یابد.



www.students.madadyar.com

اثبات بهینگی در A*



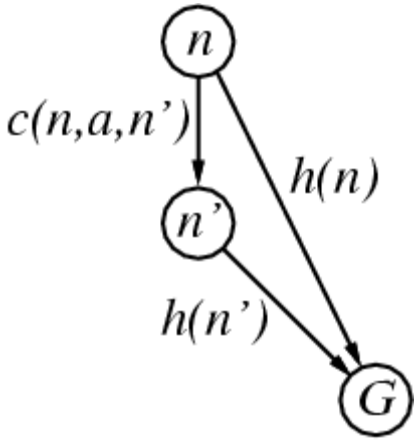
- $f(G_2) = g(G_2)$,since $h(G_2) = 0$
- $g(G_2) > g(G)$,since G_2 is suboptimal
- $f(G) = g(G)$,since $h(G) = 0$
- $f(G_2) > f(G)$,from above
- $h(n) \leq h^*(n)$,since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$,f(G)<f(G2)

- Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

اثبات شد ؟!؟؟

سازگاری هیوریستیک

- هيوريستيكي سازگار است كه اگر براي هر گره n و هر پسين آن گره n` كه با اقدام a توليد شده است هزينه برآورد شده رسيدن به هدف از n بزرگتر از هزينه گام رسيدن به n` + هزينه برآورد شده رسيدن به هدف از n` نباشد.
- $h(n) \leq c(n,a,n') + h(n')$



- هر هيوريستيک سازگار قابل قبول نيز هست.
- A*اي كه از جستجوي گراف بهره مي برد در
- صورتي بهينه است كه h(n) سازگار باشد.

ویژگی های A^*

اگر f^* هزینه واقعی مسیر بهینه از شروع تا هدف تعریف شود، می توان گفت:

A^* برای هر تابع هیوریستیک قابل قبول، دارای **کارایی بهینه موثر** می باشد. • یعنی نسبت به الگوریتم های بهینه دیگر کمترین تعداد گره را گسترش می‌دهد.	A^* هرگز گرهی را با $f(n)>f^*$ گسترش نمی دهد.	ممکن است تعدادی از گره هایی که دقیقا روی **کانتور** هدف $f(n)=f^*$ قرار دارد را گسترش دهد قبل از اینکه هدف انتخاب شود.	A^* تمام گره ها با $f(n)<f^*$ را گسترش خواهد داد.

www.students.madadyar.com

جستجوی A*

Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)*).

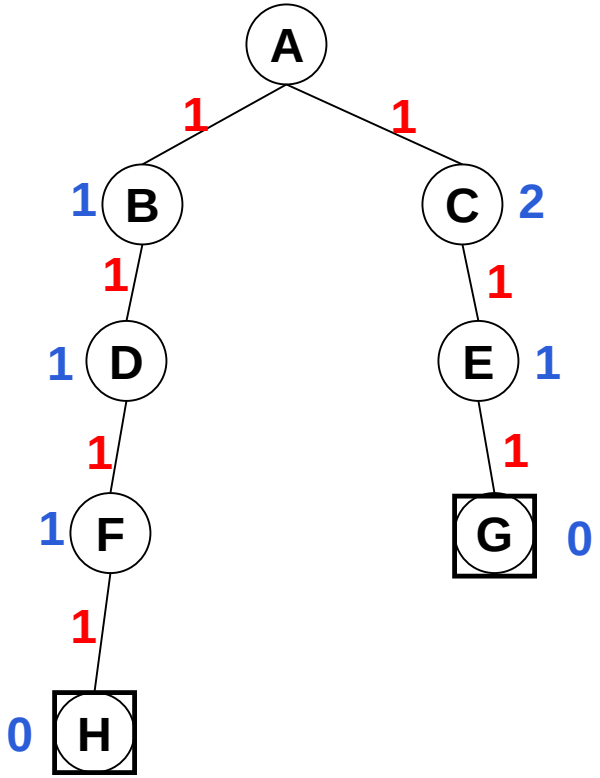
Time? Exponential. $O(b^m)$

Space? Keeps all nodes in memory. $O(b^m)$

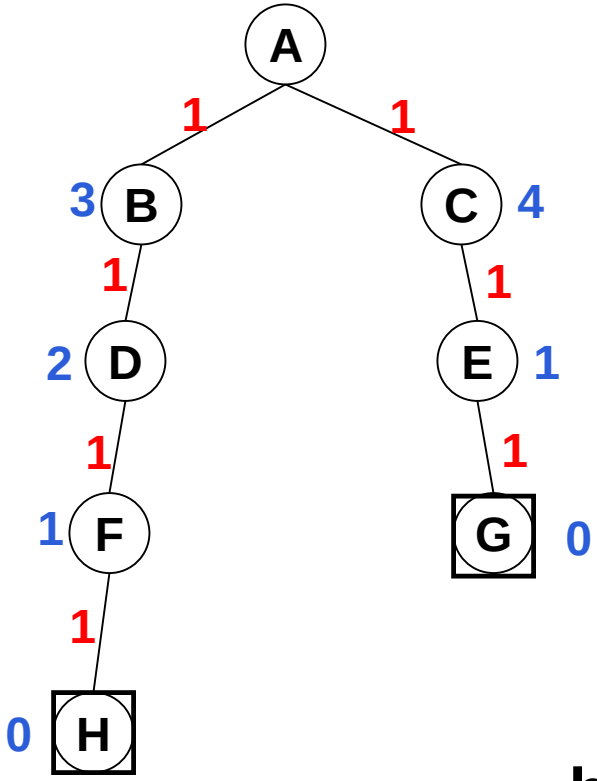
Optimal? Yes. (efficient optimal)

www.students.madadyar.com

تاثیر تابع هیوریستیک در جستجوی A*

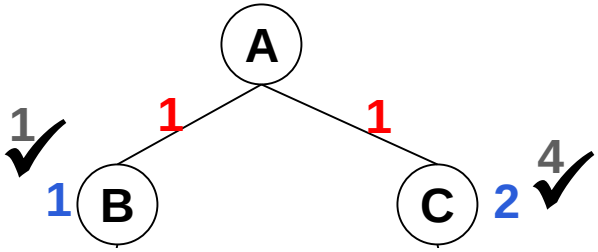


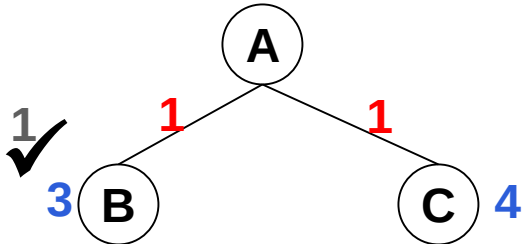
$h \leq h^*$



$h \nleq h^*$

تاثیر تابع هیوریستیک در جستجوی A*

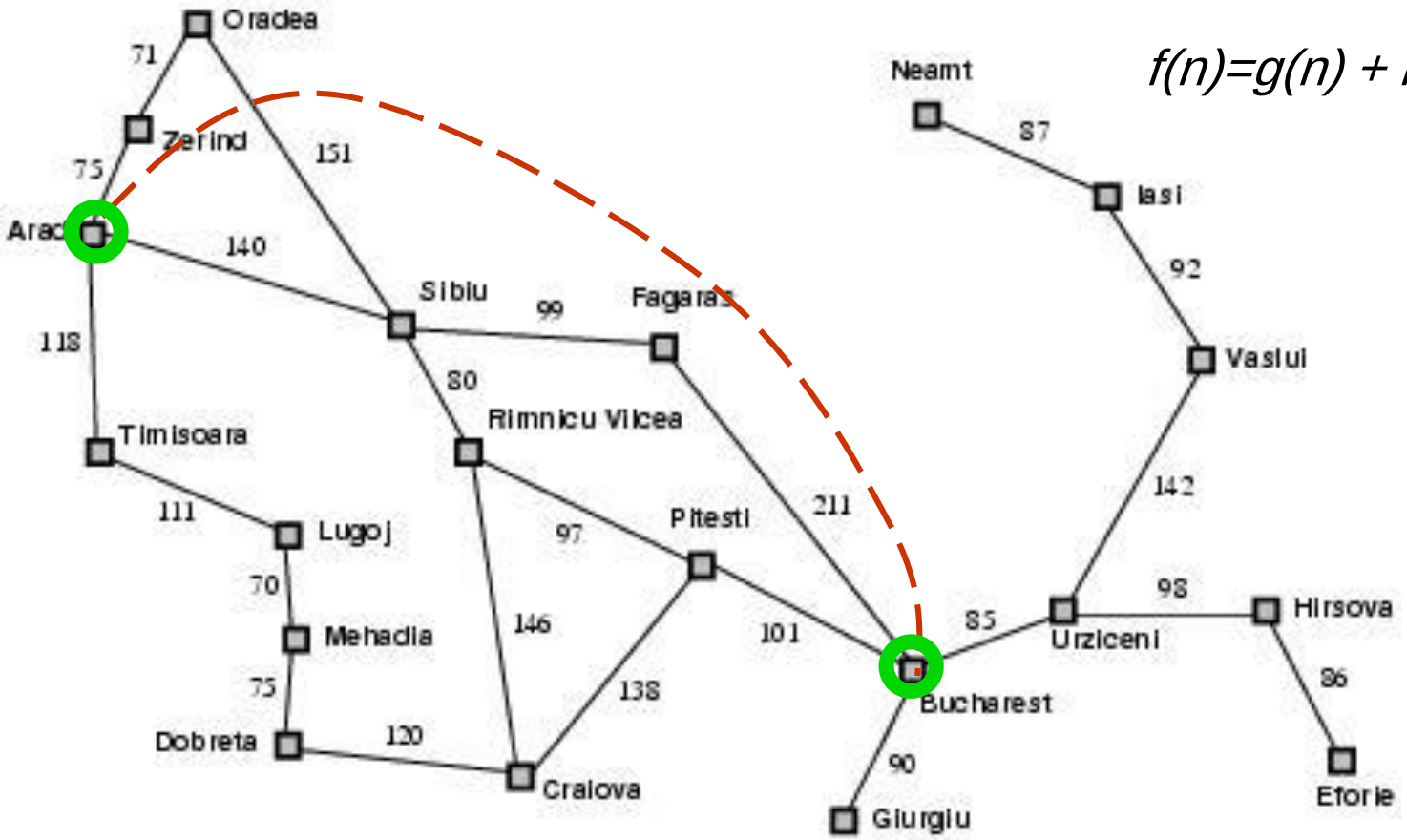




$h \leq h^*$

$h \nleq h^*$

مثال دیگر از جستجوی A*



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

www.students.magadyar.com

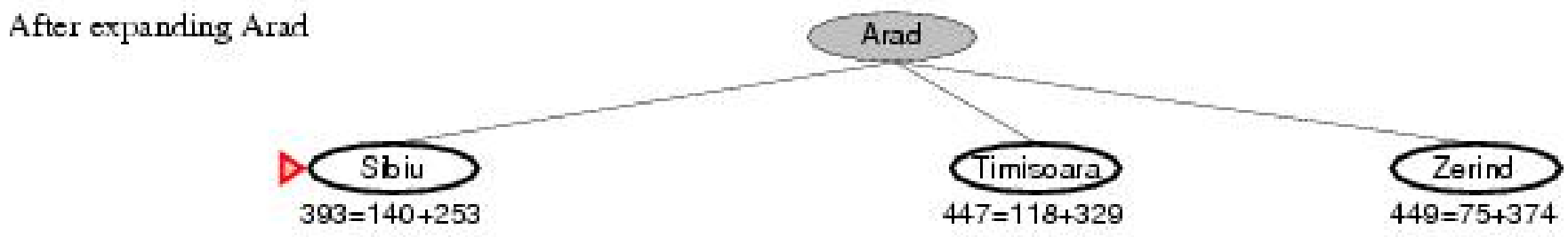
جستجوی A* در نقشه رومانی

(a) The initial state

جستجوی Bucharest با شروع از Arad

f(Arad) = g(Arad)+h(Arad)=0+366=366

www.students.madadyar.com



Arad را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

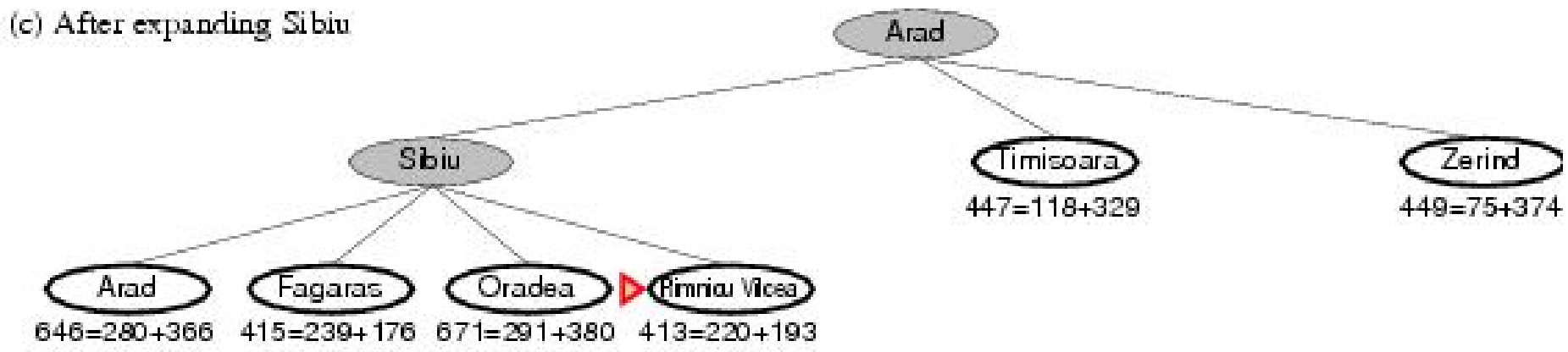
f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393

f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447

f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449

بهترین انتخاب شهر Sibiu است.

www.students.madadyar.com

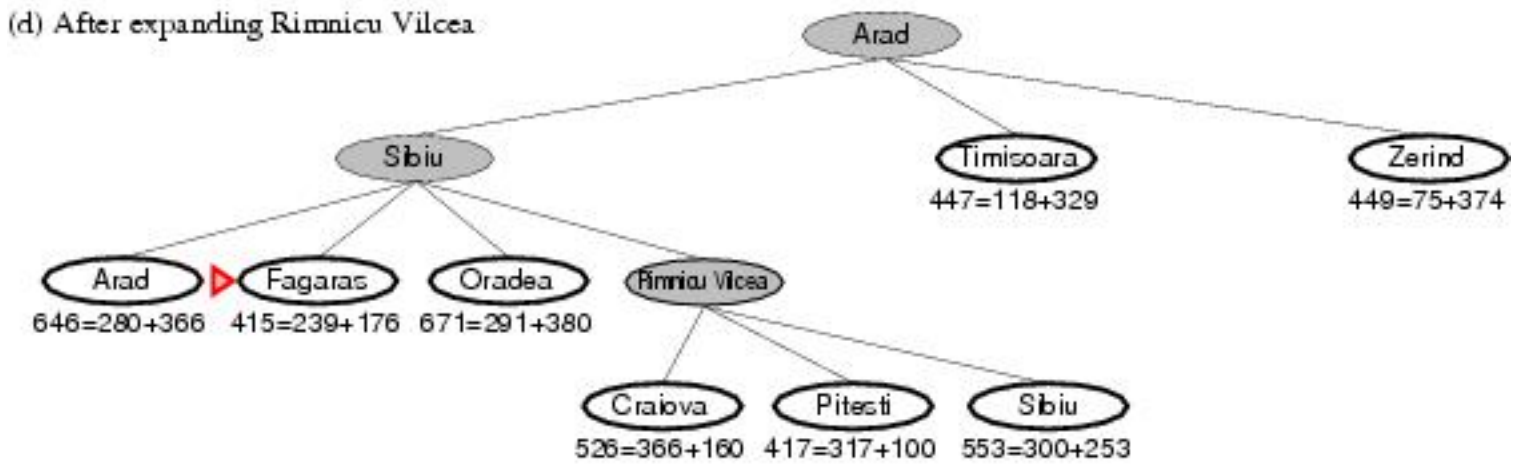


Sibiu را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415
f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413

بهترین انتخاب شهر **Rimnicu Vilcea** است.

www.students.madadyar.com

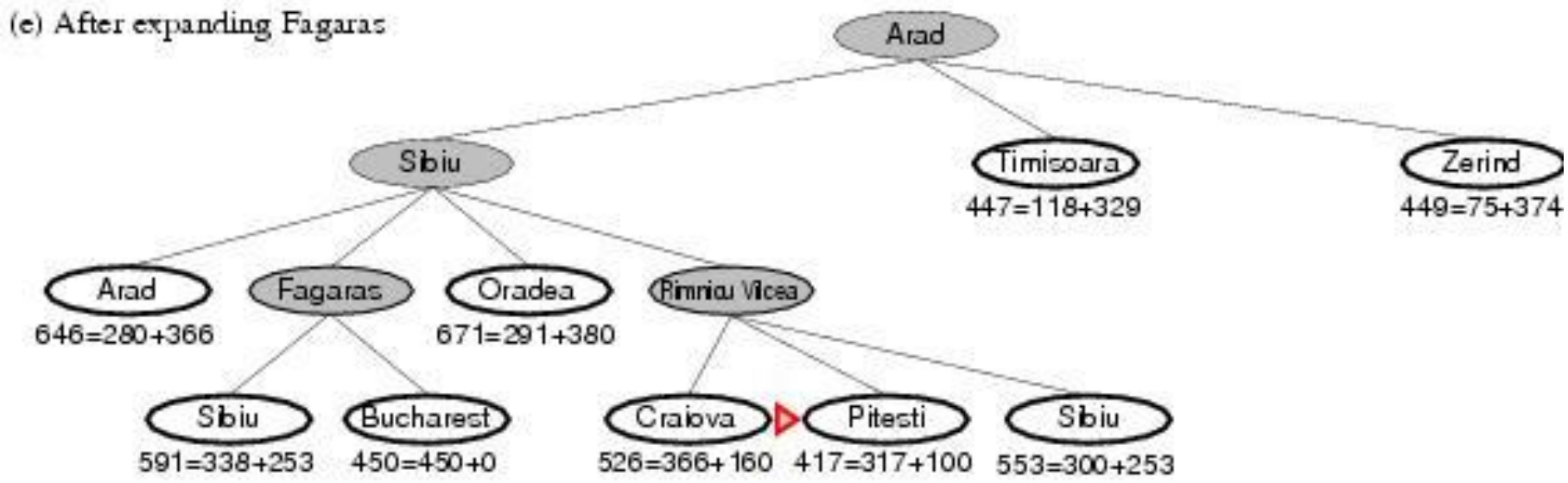


Rimnicu Vilcea را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553

بهترین انتخاب شهر **Fagaras** است.

www.students.madadyar.com



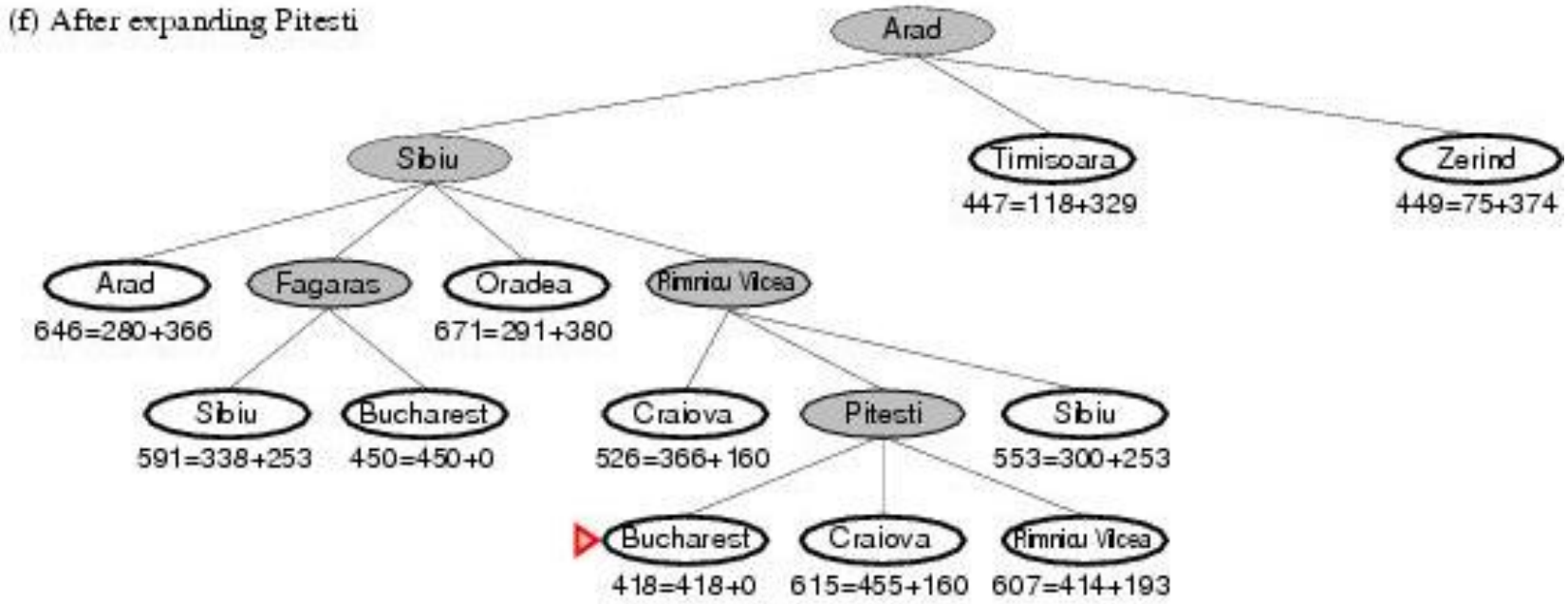
Fagaras را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591

f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450

بهترین انتخاب شهر Pitestiاست.

www.students.madadyar.com



- Pitesti را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418

بهترین انتخاب شهر **Bucharest** است.

www.students.madadyar.com

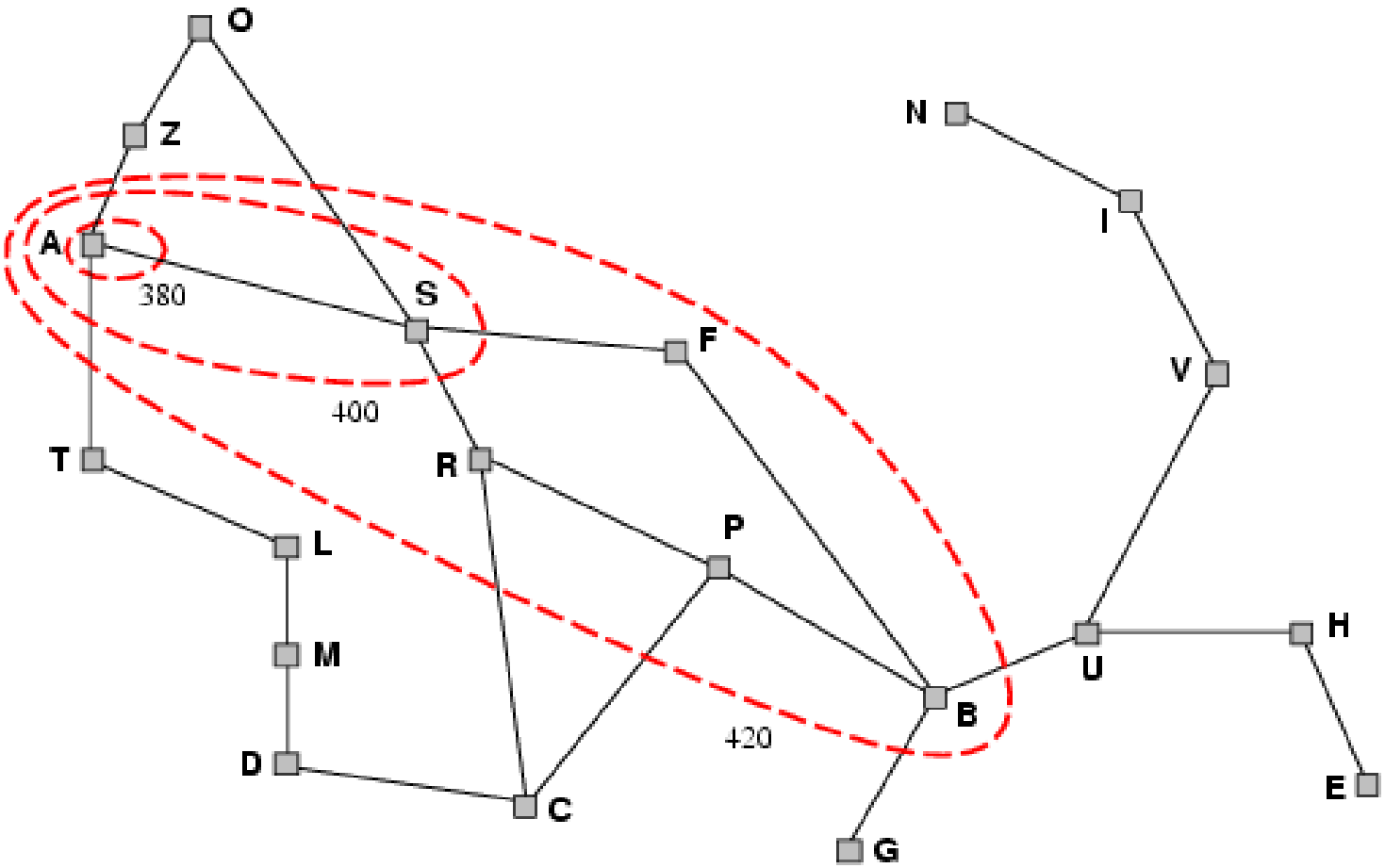
مشکلات جستجوی A*

- نیاز به **حافظه و زمان زیادي** دارد چرا که لازم است تمامي گره ها را در حافظه نگه داري کند.

- از رویکردي که در جستجوي عمقي براي کنترل حافظه استفاده کردیم بهره مي بریم.
  - به جاي محدودیت در عمق ، محدودیت در میزان تابع f با نام تراز (contour) خواهیم داشت.

www.students.madadyar.com

کانتور (محدوده تابع ارزیابی برای گسترش)



جستجوی هیوریستیک با حافظه محدود IDA*

- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از عمیق کننده تکرار در زمینه جست و جوی اکتشافي است.
- الگوریتم عمیق کننده تکرار A* => IDA*
- در جستجوی IDA* مقدار برش مورد استفاده ، عمق نیست بلکه هزینه g+h به نام f-cost است. در هر مرحله مقدار برش کوچکترین مقدار f میان تمامي گره هايي است که از برش قبلي بزرگترند.
- IDA* ترکیب جستجوي عمیق کننده تکراري و A*مي باشد تا از مزاياي هر دو بهره مند شود.
- IDA* برای اغلب مسئله های با هزینه های مرحله ای ، مناسب است .

www.students.madadyar.com

ویژگی های الگوریتم IDA *

- هدف در تکراری پیدا می شود که f-limit = C*
- ثابت می شو د این الگوریتم کامل و بهینه است . (با نقطه ضعف دوباره کاری)
- در این الگوریتم در هر مرحله فقط گره های که f<f-limit است گسترش می یابد
- اگر تعداد تکرار زیاد نباشد این الگوریتم از نظر کارآیی مانند A * است .
- محدودیت هزینه در هر مرحله به گونه ای انتخاب می شود که در مراحل قبلی ثابت شده است که جوابی با هزینه کمتر از این مقدار وجود ندارد.

بهترین جستجوی بازگشتی RBFS

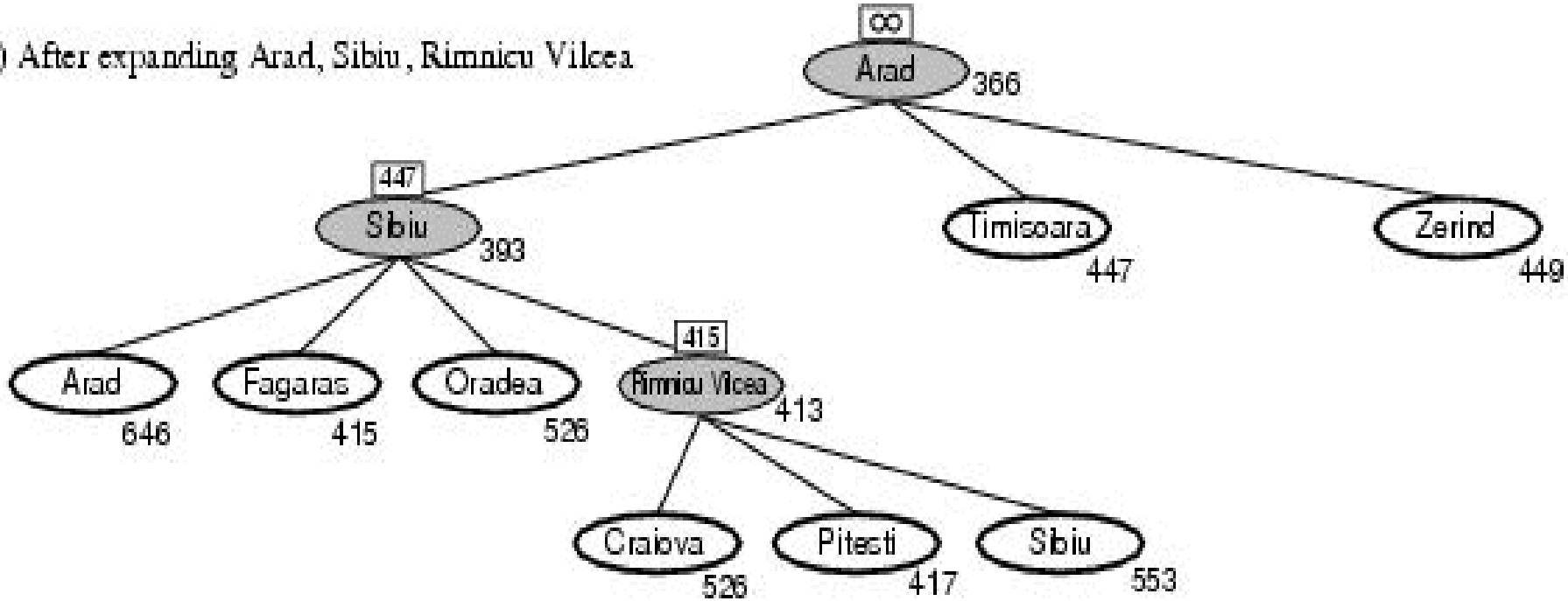
- ساختار آن شبیه جست و جوی عمقي ، بازگشتي است ، اما به جای اینکه دائما به طرف پایین مسیر حرکت کند ، مقدار f مربوط به بهترین مسیر از هر جد گره فعلی را نگهداری مي‌کند ، اگر گره فعلی از این حد تجاوز کند ، بازگشتی به عقب برمي‌گردد تا مسیر دیگري را انتخاب کند.
- این جستجو اگر تابع اکتشافی قابل قبولی داشته باشد ، **بهینه** است.
- تعیین پیچیدگی زمانی آن به دقت تابع اکتشافی و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگی دارد.

www.students.madadyar.com

RBFS

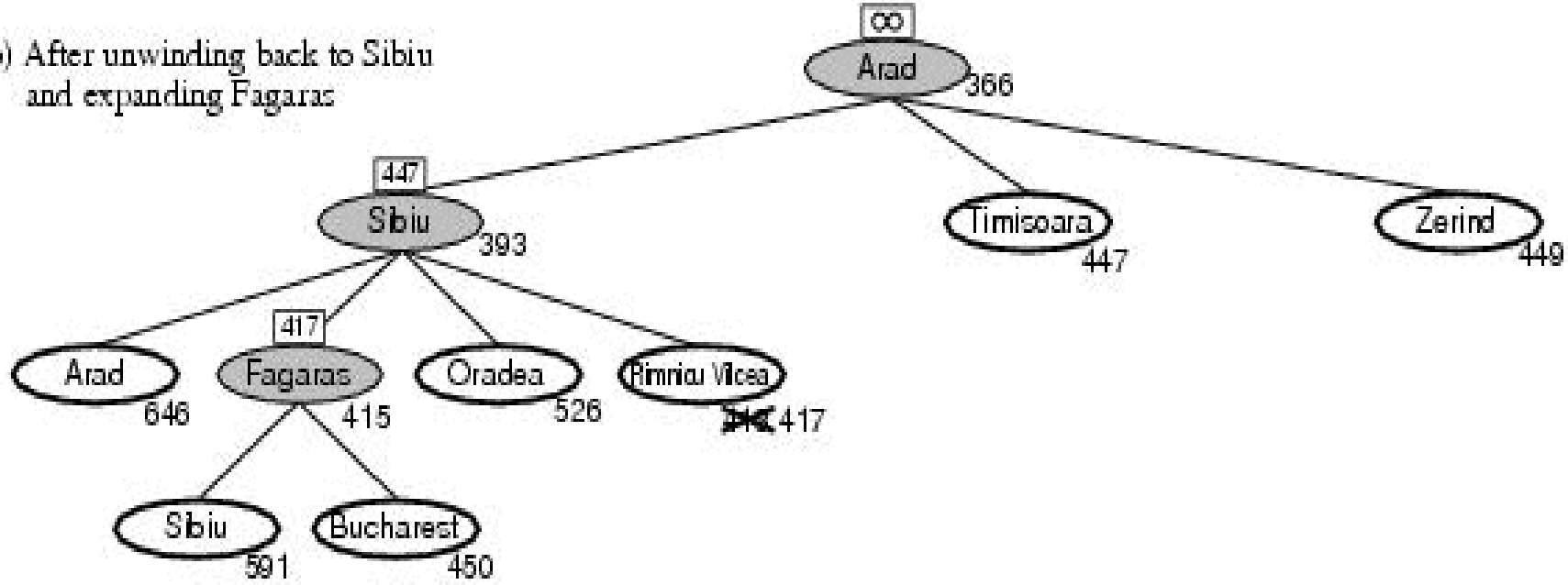
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    RBFS(problem, MAKE-NODE(INITIAL-STATE[problem]), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
    if GOAL-TEST[problem](state) 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
```

(a) After expanding Arad, Sibiu, Rimnicu Vilcea

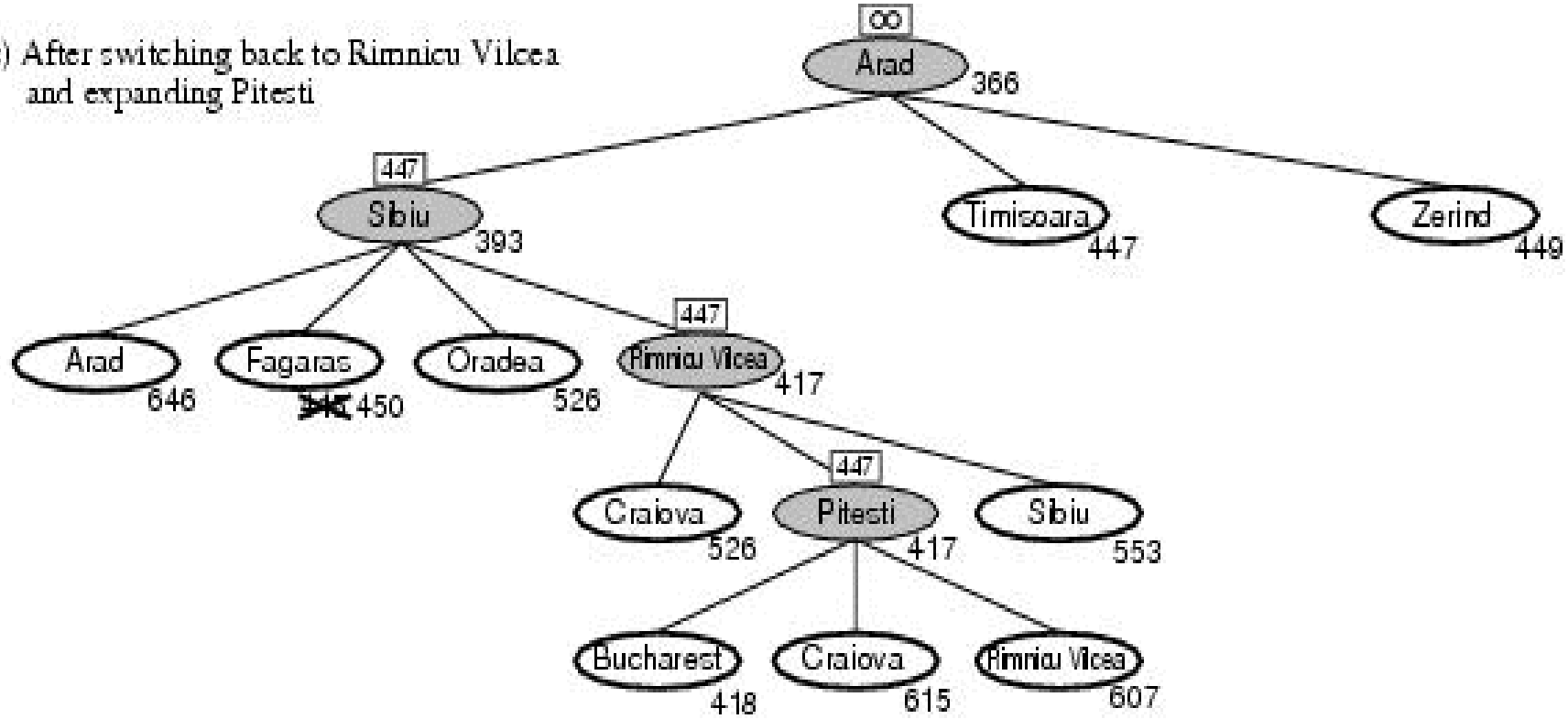


www.students.madadyar.com

(b) After unwinding back to Sibiu and expanding Fagaras



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



بهترین جستجوی بازگشتی RBFS

RBFS تا حدی از IDA* کارآمدتر است، اما گره های زیادی تولید می‌کند.

IDA* و RBFS در معرض افزایش توانی پیچیدگی قرار دارند که در جست و جوی گرافها مرسوم است، زیرا نمیتوانند حالتهای تکراری را در غیر از مسیر فعلی بررسی کنند. لذا، ممکن است یک حالت را چندین بار بررسی کنند.

IDA* و RBFS از فضای اندکی استفاده می‌کنند که به آنها آسیب می‌رساند.

IDA* بین هر تکرار فقط یک عدد را نگهداری می‌کند که هزینه فعلی f است. RBFS اطلاعات بیشتری در حافظه نگهداری می‌کند.

www.students.madadyar.com

جستجوی حافظه محدود ساده SMA*

- SMA* بهترین برگ را بسط (مانند A*) می‌دهد تا حافظه پر شود. در این نقطه بدون از بین بردن گره های قبلی نمی‌تواند گره جدیدی اضافه کند. SMA* همیشه بدترین گره برگ را حذف می‌کند و سپس از طریق گره فراموش شده به والد آن بر می‌گردد. پس جد زیر درخت فراموش شده، کیفیت بهترین مسیر را در آن زیر درخت می‌داند.
- اگر عمق سطحی ترین گره هدف کمتر از حافظه باشد, کامل است.
- اگر راه حل بهینه دست یافتنی باشد ، بهینه است.
- SMA* بهترین الگوریتم همه منظوره برای یافتن راه‌حلهای بهینه می‌باشد.

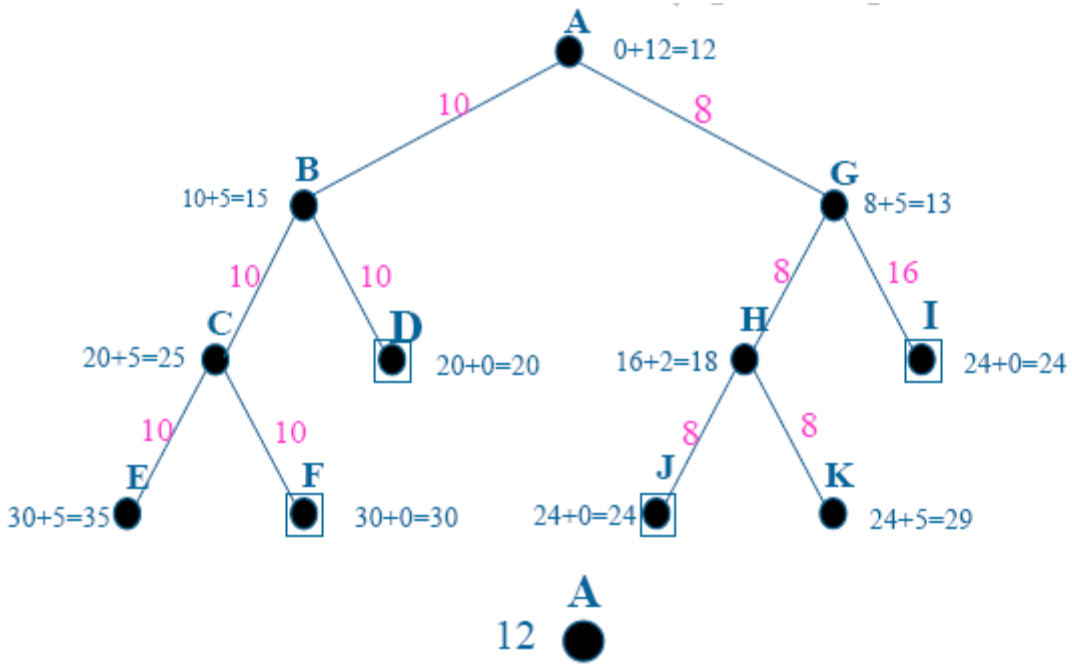
www.students.madadyar.com

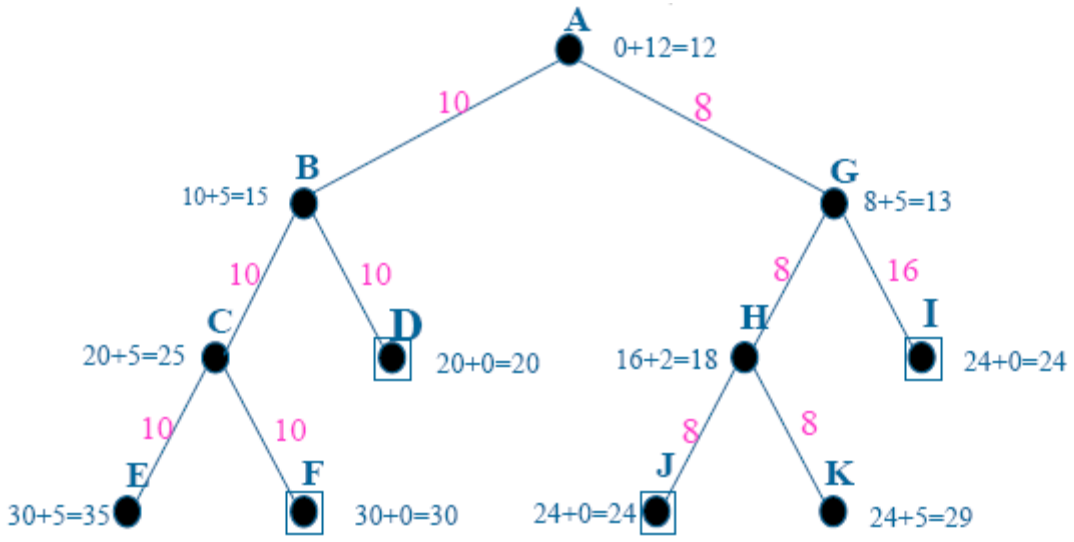
جستجوی حافظه محدود ساده SMA*

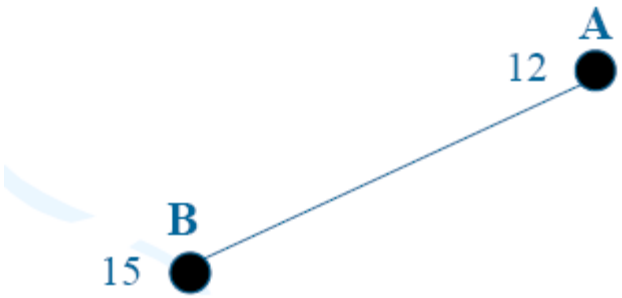
- اگر مقدار f تمام برگها یکسان باشد و الگوریتم باید یک گره را هم برای بسط و هم برای حذف انتخاب کند، SMA* این مسئله را با بسط بهترین برگ جدید و حذف بهترین برگ قدیمی حل می‌کند.

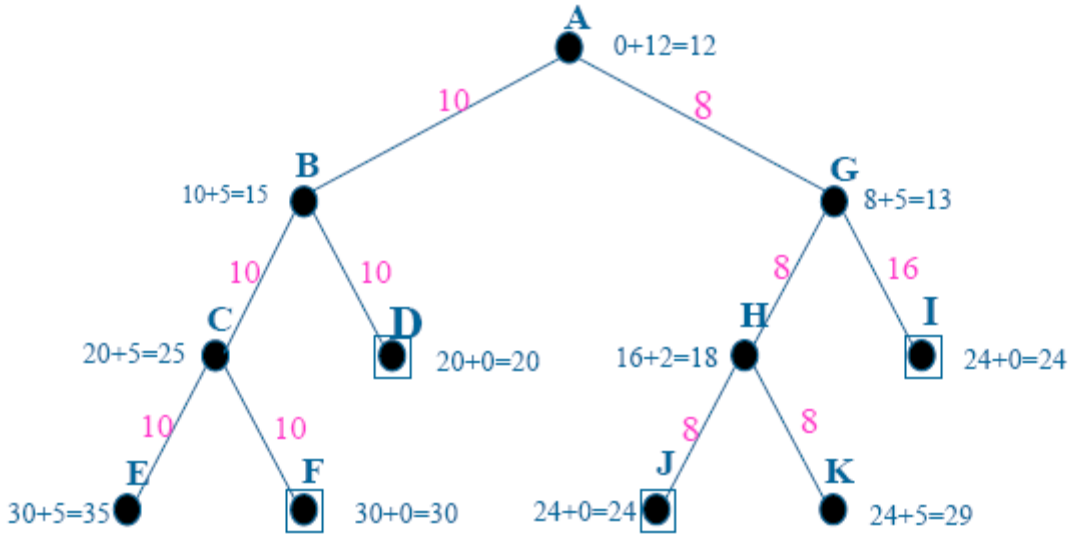
- محدودیتهای حافظه ممکن است مسئله ها را از نظر زمان محاسباتی، غیر قابل حل کند.

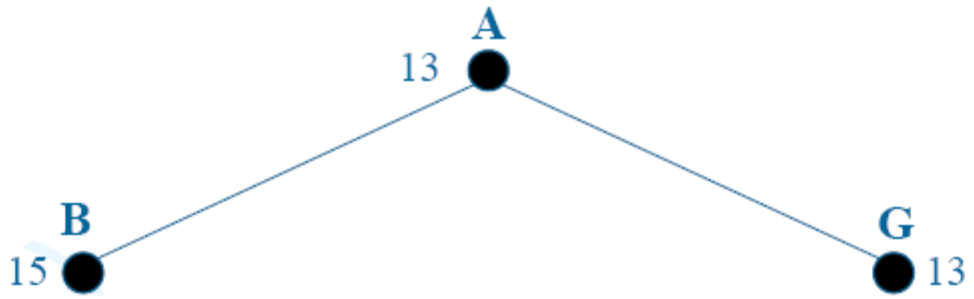
www.students.madadyar.com

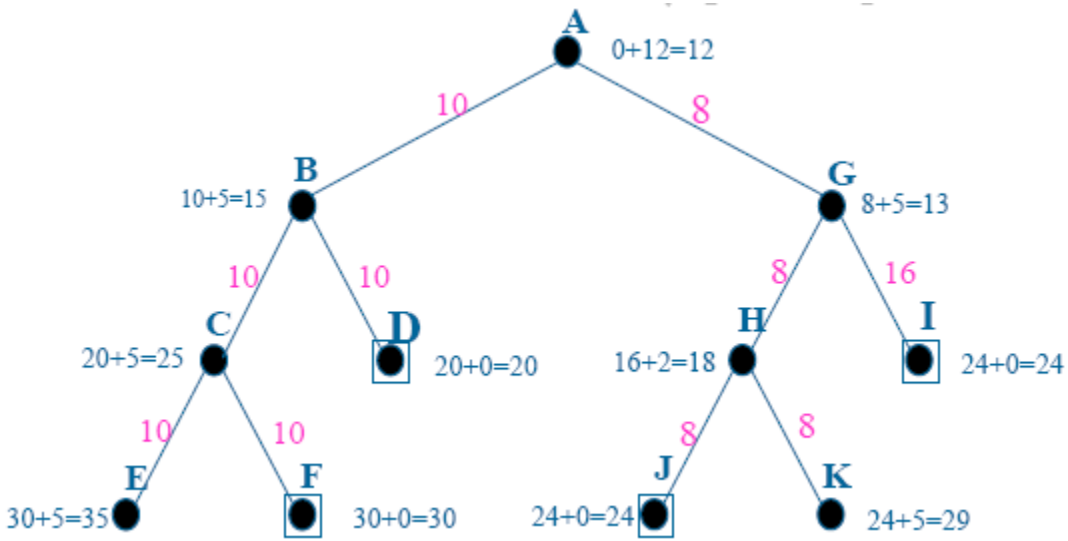

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29
A
12

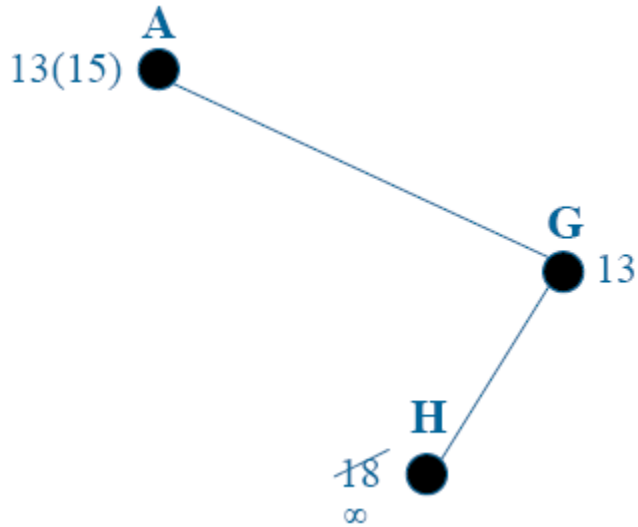

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29


A
12
B
15

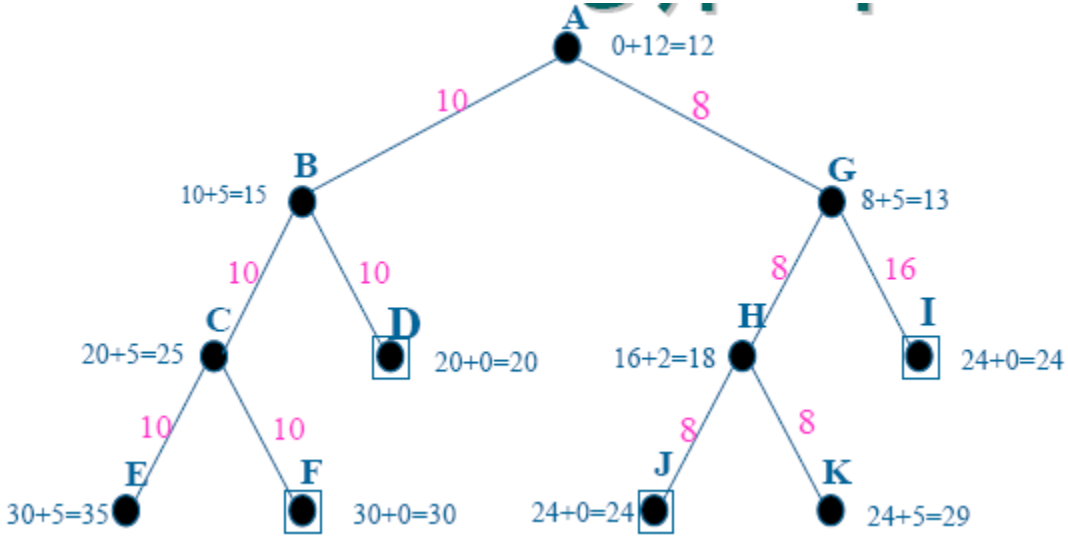

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29

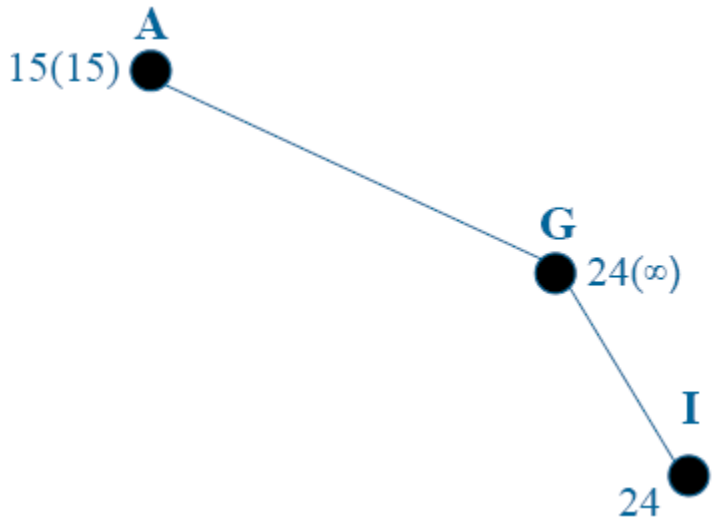

A
13
B
15
G
13

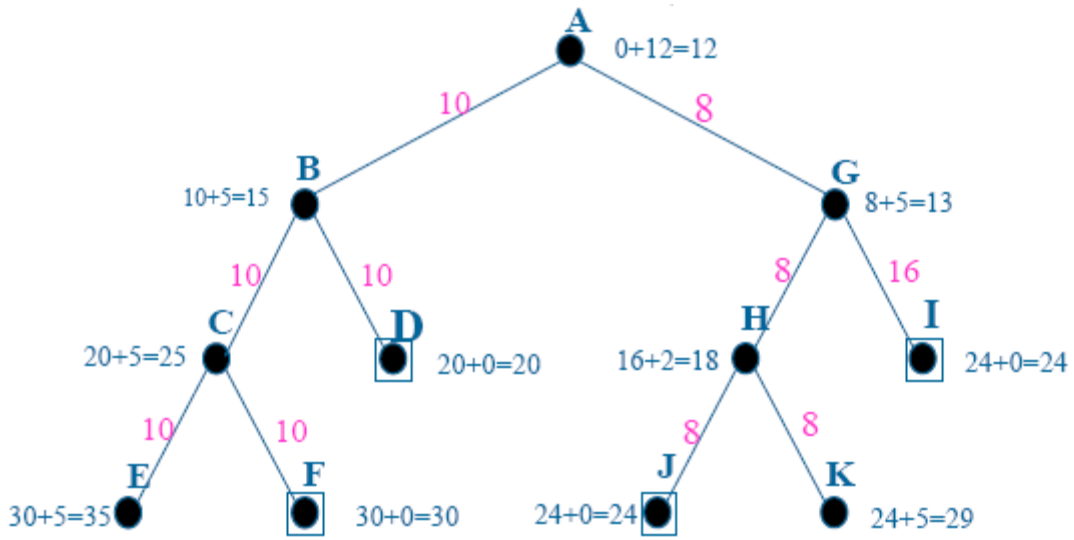


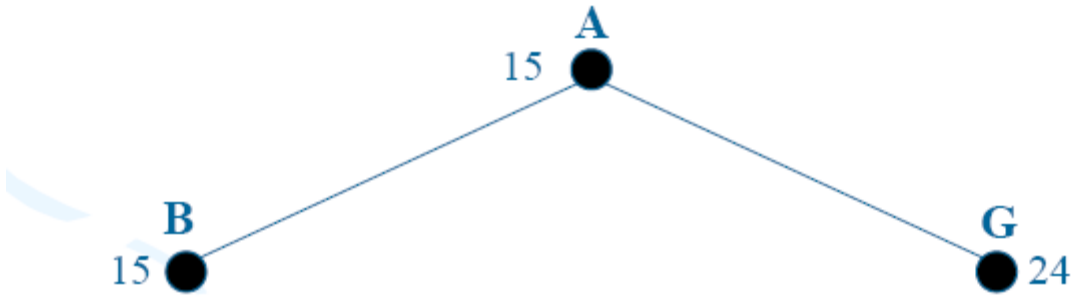


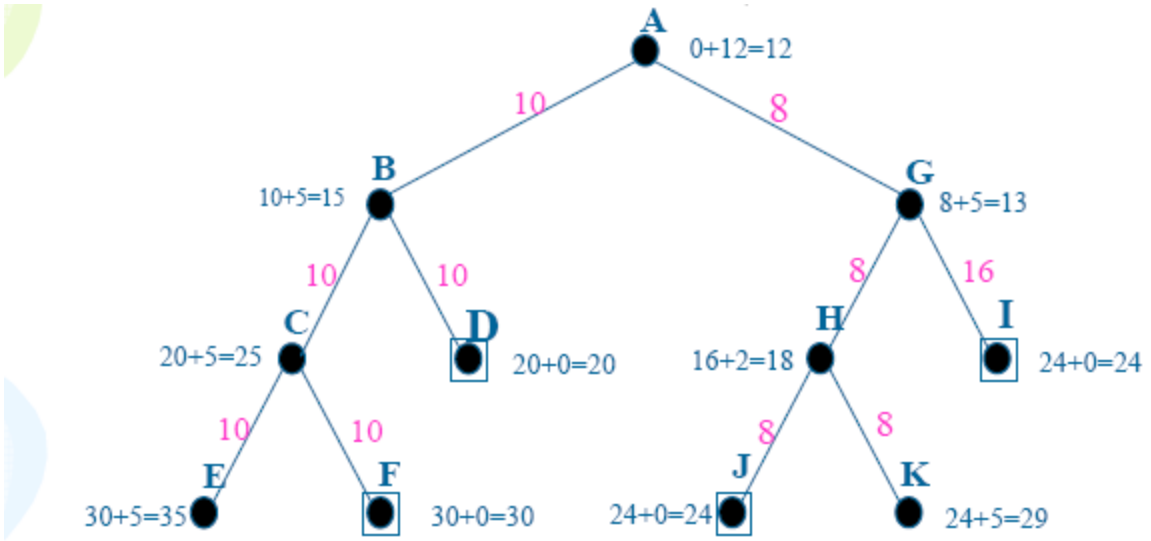
www.students.madadyar.com

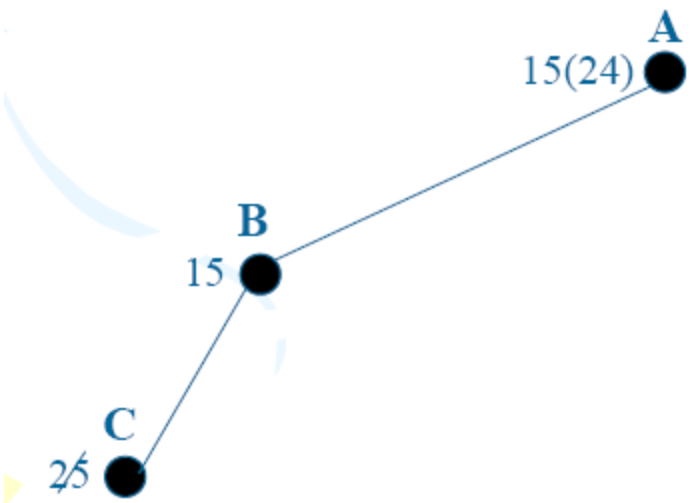

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29

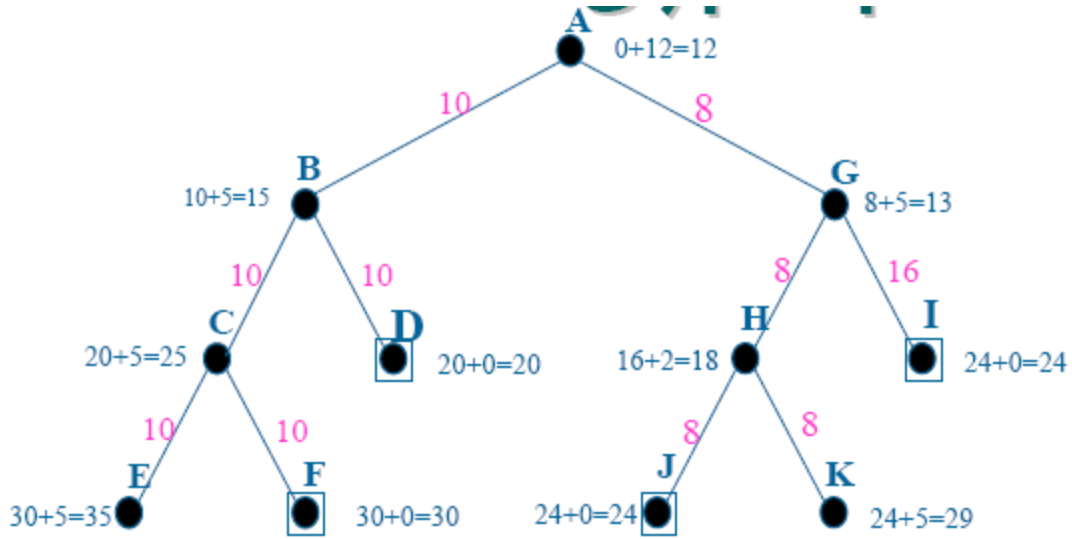

A
15(15)
G
24(∞)
I
24

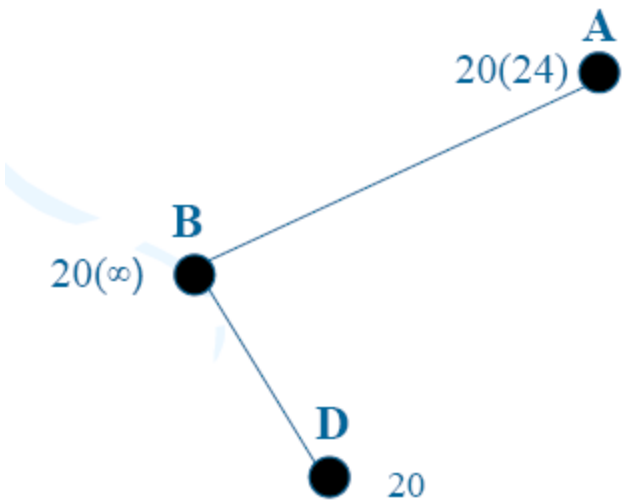

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29


A
15
B
15
G
24


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29


A
15(24)
B
15
C
25

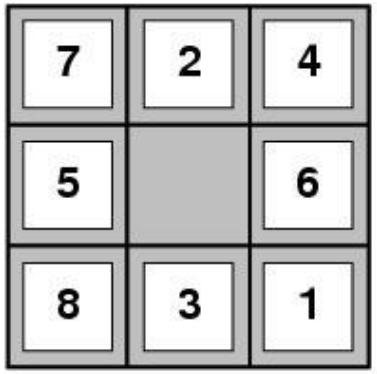

A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29


A
20(24)
B
20(∞)
D
20

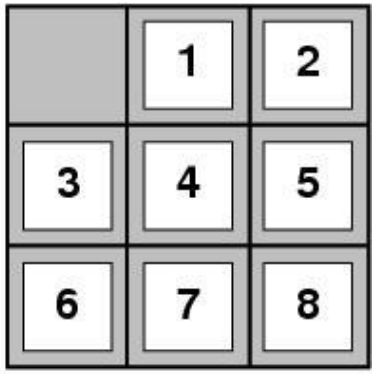
طراحی توابع هیوریستیک

Heuristic

توابع هیوریستیک قابل قبول



Start State



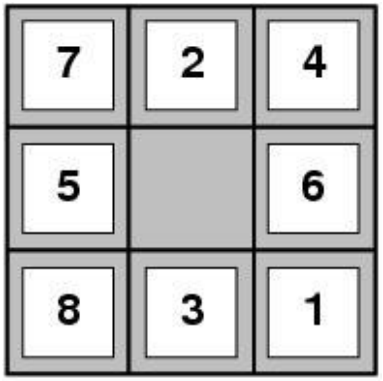
Goal State

مثال برای معمای ۸

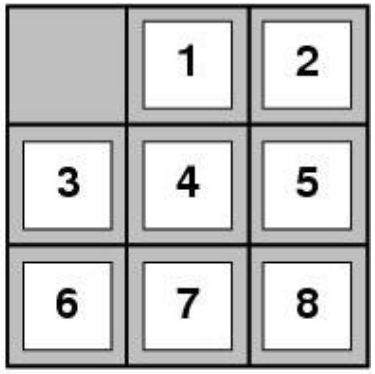
- میانگِین هزینه حل تقریبا ۲۲ مرحله و فاکتور انشعاب در حدود ۳ است.
- جستِ و جوی جامع تا عمق ۲۲ : $3^{22} \approx 3.1\times10^{10}$
- با انتخاب یک تابع اکتشافی مناسب میتوان مراحل جستجو را کاهش داد.

www.students.madadyar.com

دو تابع هیوریستیک برای معمای۸



Start State



Goal State

تعداد کاشیها در مکانهای نادرست $= h_1$

در حالت شروع $h_1 = 8$

h_1 اکتشاف قابل قبولی است، زیرا هر کاشی که در جای نامناسبی قرار دارد، حداقل یکبار باید جابجا شود.

www.students.madadyar.com

دو تابع هیوریستیک برای معمای۸

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$h_2 =$ مجموعه فواصل کاشیها از موقعیتهای هدف آنها در حالت شروع

$$h_2 = 3+1+2+2+2+3+3+2 = 18$$

چون کاشیها نمیتوانند در امتداد قطر جا به جا شوند, فاصله ای که محاسبه میکنیم مجموع فواصل افقی و عمودی است. این فاصله را فاصله **بلوک شهر** یا فاصله **منهتن** مینامند.

www.students.madadyar.com

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

مجموعه فواصل کاشیها از موقعیتهای هدف آنها $= h_2$

h_2 قابل قبول است، زیرا هر جابجایی که میتواند انجام گیرد، به اندازه یک مرحله به هدف نزدیک میشود.

هیچ کدام از این برآوردها، هزینه واقعی راه حل نیست

هزینه واقعی ۳۶ است.

www.students.madadyar.com

تسلط (dominance)

اگر برای هر گره n داشته باشیم: h2(n) >= h1(n)

- ☑ h2 بر h1غالب است.
- ☑ غالب بودن مستقیما به کارایی ترجمه می‌شود.
- ☑ تعداد گره هایی که با بکارگیری h2 بسط داده می‌شود، هرگز بیش از بکارگیری h1 نیست.

همیشه بهتر است از تابع اکتشافی با مقادیر بزرگ استفاده کرد، به شرطی که زمان محاسبه اکتشاف، خیلی بزرگ نباشد و قابل قبول باشد

www.students.madadyar.com

مقایسه

- مقایسه تعداد گره هاي بسط داده شده براي رسيدن به هدف در دو حالت:

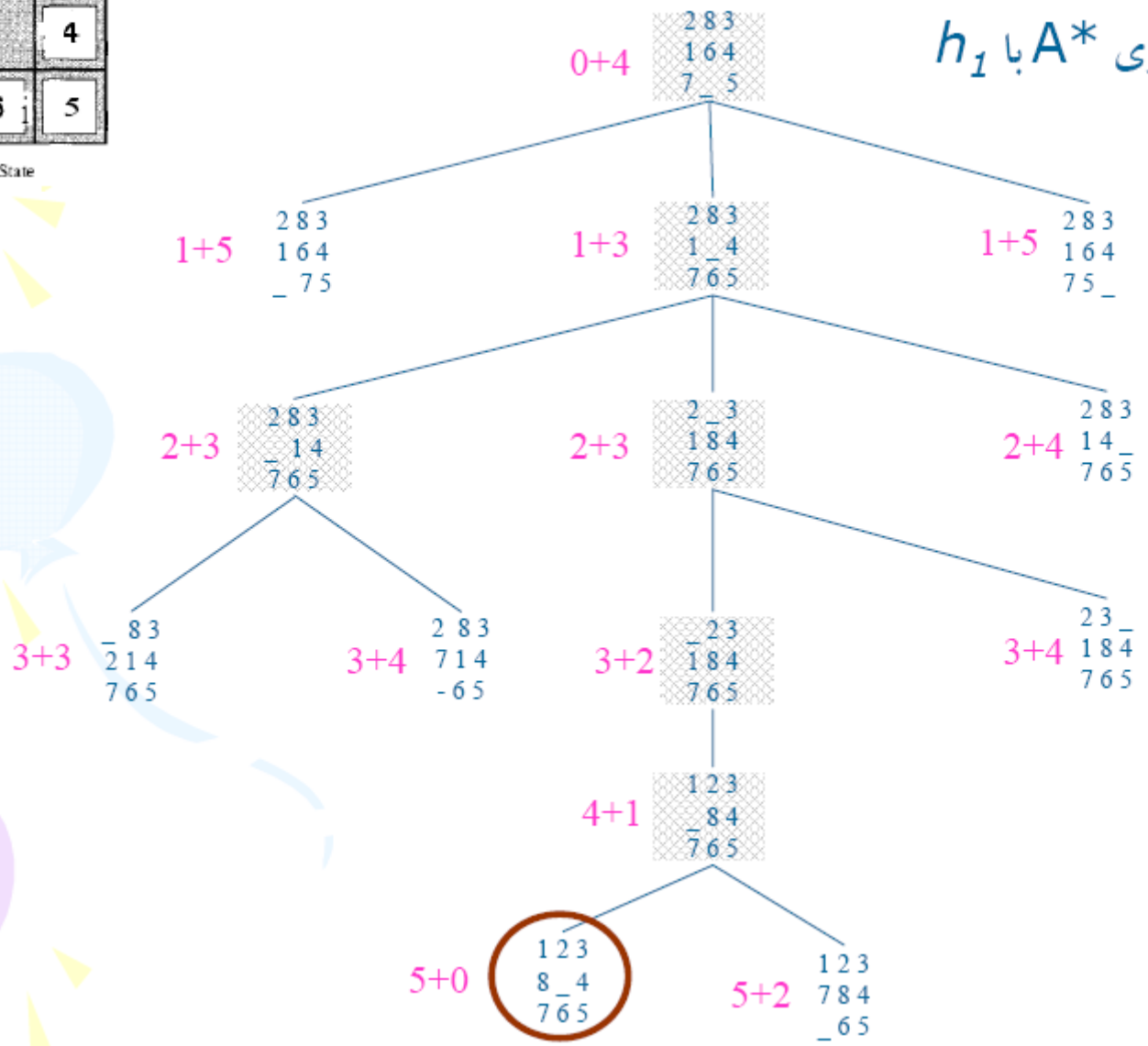
- *d=12*
  - *IDS = 3,644,035 nodes*
  - A*(h1) = 227 nodes
  - A*(h2) = **73** nodes

- *d=24*
  - *IDS = 54,000,000,000*
  - A*(h1) = 39,135 nodes
  - A*(h2) = **1,641** nodes

www.students.madadyar.com

مثال: جستجوی A* با h_1

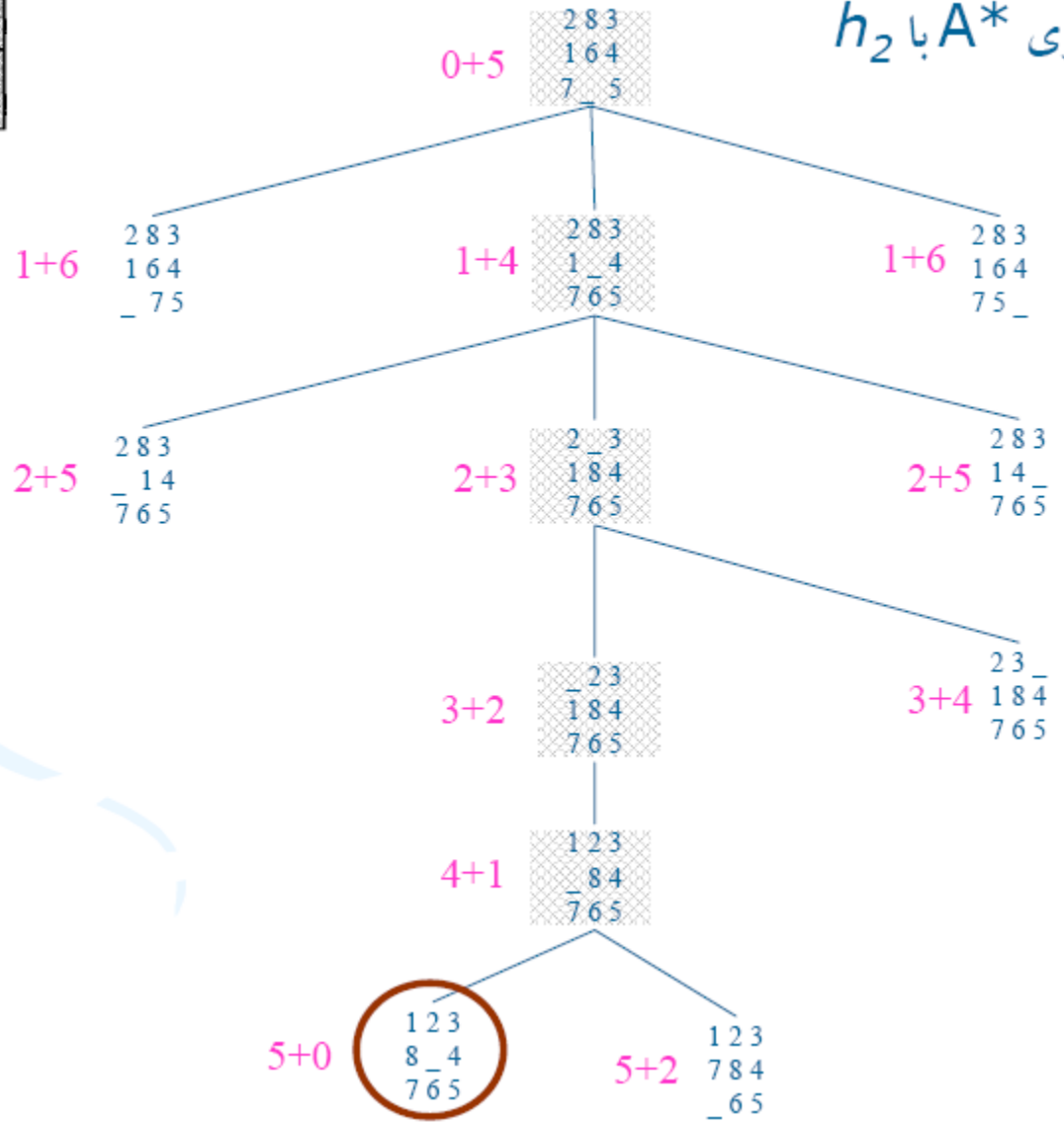
Goal State



www.students.mazbayar.com

Goal State

مثال: جستجوی A* با h_2



www.students.modaayar.com

اثر دقت اکتشاف بر کارایی

فاکتور انشعاب مؤثر b^*

- اگر تعداد گره هایی که برای یک مسئله خاص توسط A^* تولید می‌شود برابر با N و عمق راه حل برابر با d باشد، آن گاه b^* **فاکتور انشعابی** است که درخت یکنواختی به عمق d باید داشته باشد تا حاوی N+1 گره باشد.

$$N+1=1+b^*+(b^*)^2+\ldots+(b^*)^d$$

- فاکتور انشعاب مؤثر معمولاً برای **مسئله های سخت ثابت** است.
- اندازه گیریهای تجربی b^* بر روی مجموعه کوچکی از مسئله ها می‌تواند **راهنمای خوبی برای مفید بودن** اکتشاف باشد.
- مقدار b^* در اکتشافی با طراحی خوب، نزدیک ۱ است.

www.students.madadyar.com

اثر دقت اکتشاف بر کارایی

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6,384	39	25	2.80	1.33	1.24
10	47,127	93	39	2.79	1.38	1.22
12	364,404	227	73	2.78	1.42	1.24
14	3,473,941	539	113	2.83	1.44	1.23
16	-	1,301	211	-	1.45	1.25
18	-	3,056	363	-	1.46	1.26
20	-	7,276	679	-	1.47	1.27
22	-	18,094	1,219	-	1.48	1.28
24	-	39,135	1,641	-	1.48	1.26

میانگین گره های بسط یافته در جستجوی IDS و A* و فاکتور انشعاب مؤثر با استفاده از h1 و h2

www.students.madadyar.com

نتيجه

- براي مسائل مختلف سعي شود از تابع هيوريستيک با مقادير بالاتر استفاده كنيم:
  - بيش از اندازه واقعي تخمين نزند.
  - زمان محاسبات براي آن زياد نباشد.
- هيوريستيک هاي خوب از گسترش بيش از اندازه گره ها جلوگيري مي‌كنند.

www.students.madadyar.com