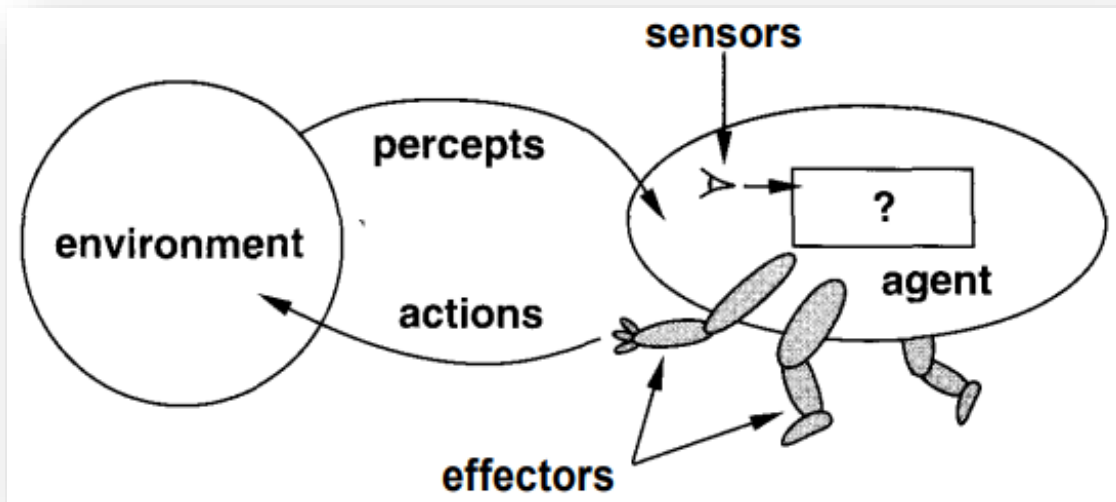


هوش مصنوعی (جستجو-بخش اول)

صادق اسکندری - دانشکده علوم ریاضی، گروه کامپیوتر

eskandari@guilan.ac.ir





هر چیزی که در یک محیط قرار گرفته و با استفاده از سنسورهای خود محیط را درک کرده و با استفاده از افکتورهای خود بر روی محیط عمل انجام می دهد.

مثال:

عامل	محیط	سنسورها	ادارات	افکتورها	اعمال
راننده تاکسی	ترافیک	چشم - گوش - سرعت سنج - آمپر بنزین	سرعت - مسافرین - وضعیت ماشین - ترافیک	دنده - پدال گاز - بوق - زبان - دست - ...	بوق زدن - سوار کردن مسافر - گاز دادن - ترمز کردن - ...
شطرنج باز	بازی شطرنج	چشم و گوش	حرکات حریف - موفقیت مهره ها - ساعت - ...	دست - زبان	حرکت دادن مهره ها - کیش دادن - ...

ساختار کلی یک عامل:

$Agent = Architecture + Program$

در این درس فرض بر این است که یک معماری در اختیار ما قرار داده شده است (سنسورها و افکتورها مشخص هستند) و ما باید برنامه آن را مشخص کنیم.

عوامل های برنامه ریز

مسائل جستجو

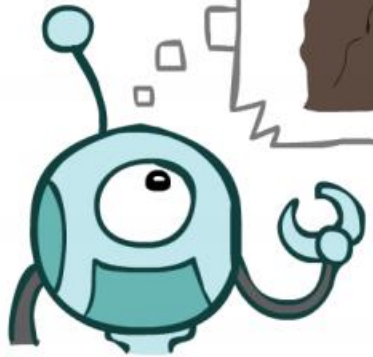
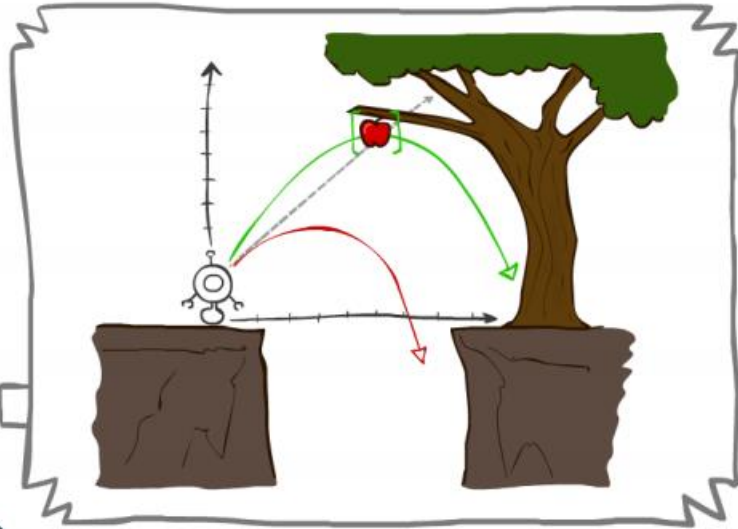
روش های جستجوی گورگورانه

جستجوی عمق اول (Depth First Search)

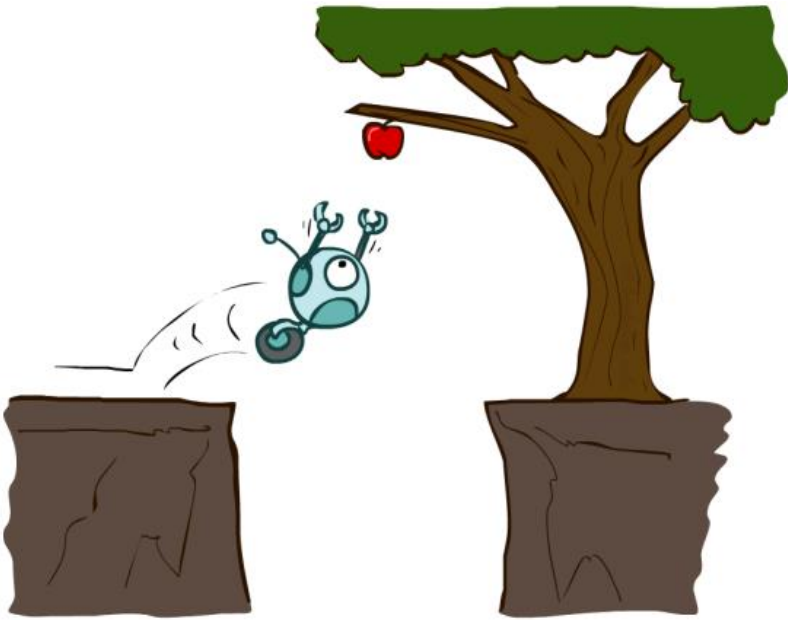
جستجوی سطح اول (Breadth First Search)

جستجوی هزینه یکنواخت (Uniform Cost Search)

جستجوی عمیق سازی تکراری (Iterative Deeping Search)

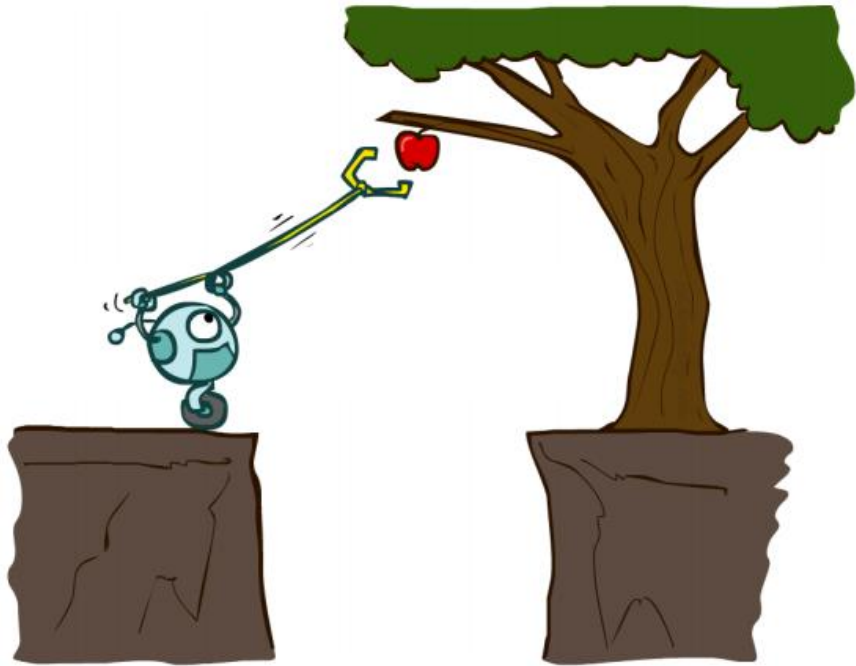


عامل های، فلکس



تصمیمات در هر لحظه را فقط بر اساس ادراکات همان لحظه می گیرند.
نتایج و پیامدهای اعمال خود را قبل از انجام عمل نمی سنجنند.

عامل های برنامه ریز

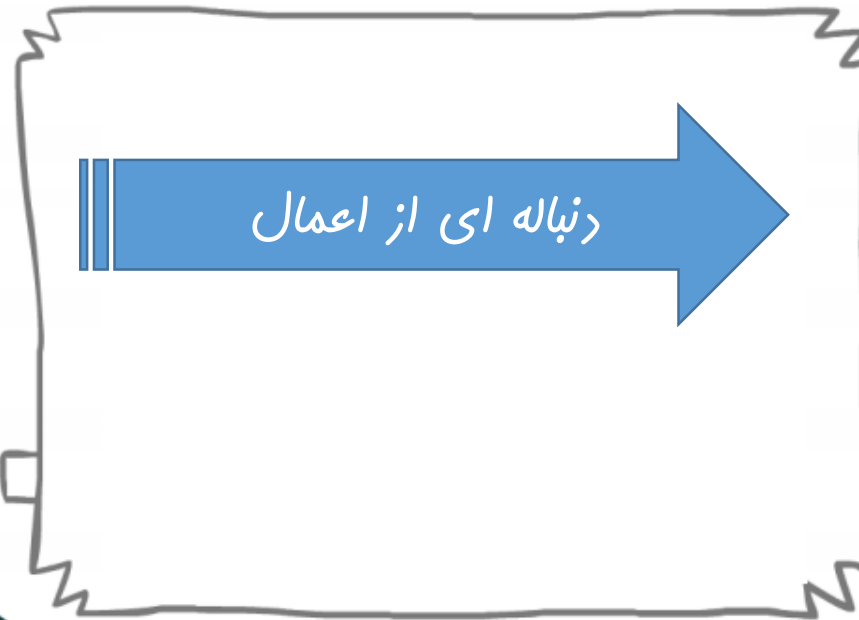


همواره قبل از انجام عمل، نتیجه آن را می سنجند.
همواره یک مدل از نحوه تغییر جهان در پاسخ با اعمال خود دارد.
هر عملی را برای رسیدن به یک هدف انجام می دهد.

عامل های برنامه ریز

در مسائل برنامه ریزی، هدف تغییر وضعیت جهان از حالت فعلی به یک حالت هدف است.

حالت فعلی جهان



حالت هدف

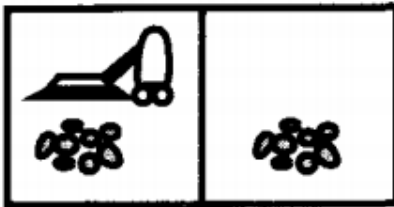


وظیفه یک عامل برنامه ریز، یافتن دنباله ای از اعمال است که با انجام آنها، وضعیت جهان از حالت فعلی به حالت هدف تبدیل می شود. به این عمل جستجو گفته می شود.

عوامل های برنامه ریز

در مسائل برنامه ریزی، هدف تغییر وضعیت جهان از حالت فعلی به یک حالت هدف است.

حالت فعلی جهان



دنباله ای از اعمال

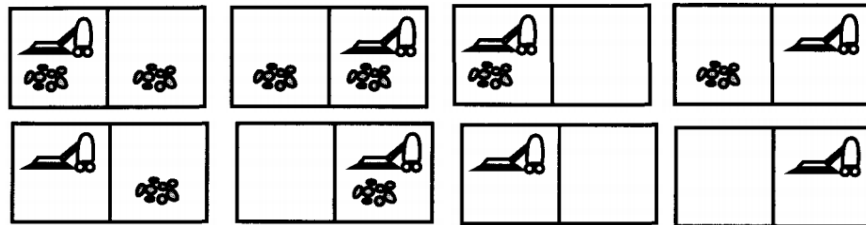
$S \rightarrow R \rightarrow S$

حالت هدف



مسائل جستجو

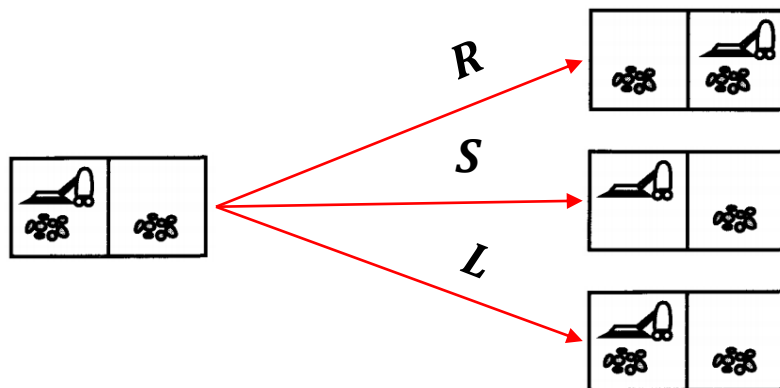
یک مسئله جستجو شامل موارد زیر است:



۱- یک فضای حالت (State Space)

$\{S, L, R\}$

۲- مجموعه اعمال (Actions)



۳- یک تابع بعدی (Successor Function): تابعی که با دریافت یک حالت و یک عمل، نتیجه انجام آن عمل در آن حالت را مشخص می‌کند.

مسائل جستجو

یک مسئله جستجو شامل موارد زیر است:

۴- یک تابع هزینه مسیر: تابعی که یک مسیر (دنباله ای از اعمال) را به عنوان ورودی دریافت کرده و هزینه آن را برمی گرداند. معمولا هزینه مسیر برابر با مجموع هزینه های گامهای مسیر می باشد.

۵- یک حالت اولیه (Initial State)

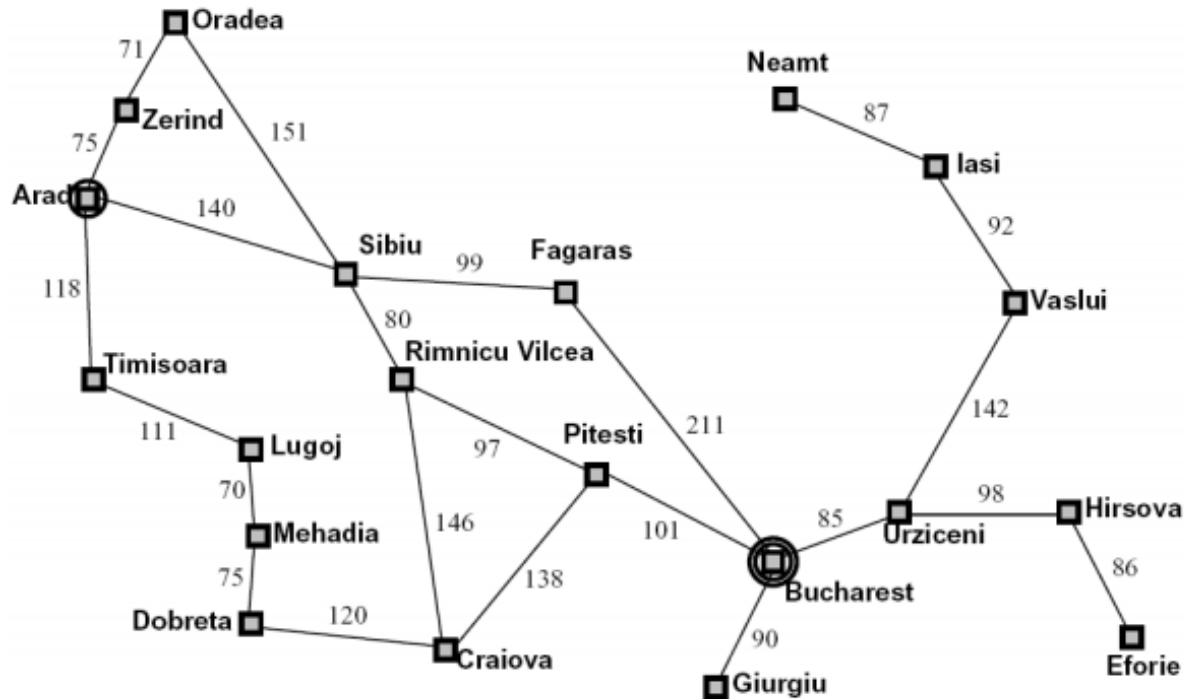
۶- یک تابع تست هدف (Goal Test Function)

$$G: \text{State Space} \rightarrow \{\text{True}, \text{False}\}$$

یک جواب (Solution) برای مسئله جستجو دنباله ای از اعمال (مسیر) است که حالت اولیه را به حالت هدف تبدیل می کند.

مسائل جستجو

مثال: مسافرت در رومانی



فضای حالت: بودن در هر یک از شهرها

تابع بعدی و اعمال: رفتن به شهرهای همسایه

تابع هزینه: مجموع مسافت میان شهرها در مسیر

حالت اولیه: Arad

تابع تست هدف: آیا شهر (حالت) فعلی

Bucharest است؟

مسائل جستجو

مثال: مسأله ۸-پازل (8-Puzzle)

فضای حالت: هر ترتیبی از خانه ها

تابع بعدی و اعمال: حرکت خانه خالی به سمت

{بالا، پایین، چپ، راست}

5	4	
6	5 1	8
7	3	2

حالت شروع

1	2	3
8		4
7	6	5

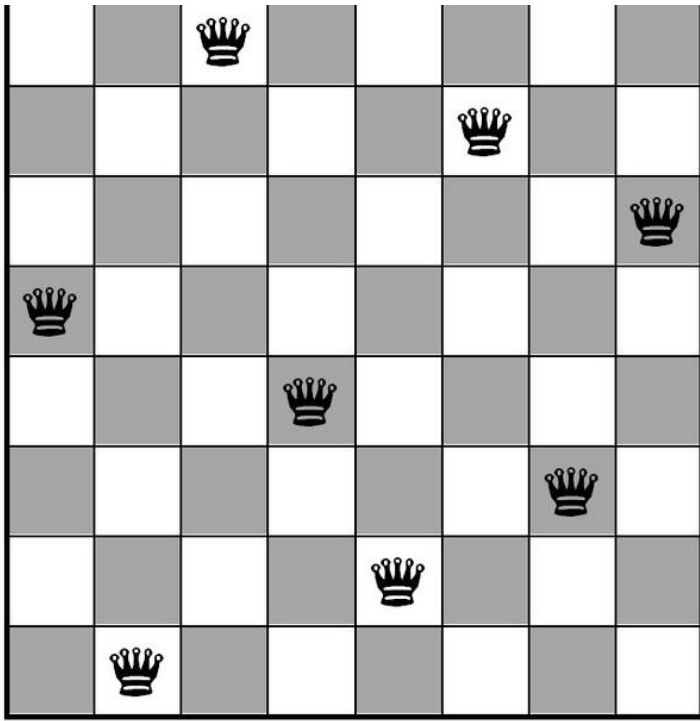
حالت هدف

تابع هزینه: هزینه هر گام برابر یک واحد و هزینه مسیر برابر با تعداد گام ها

حالت اولیه: هر حالتی از فضای حالت

تابع تست هدف: آیا خانه ها مرتب هستند؟

مسائل جستجو



مثال: مسأله ۸ وزیر

فضای حالت: تمامی چینش های ۸ وزیر در صفحه شطرنج

تابع بعدی و اعمال: جایجا کردن هر یک از وزیرها در خانه های شطرنج

تابع هزینه: هزینه هر گام (جایجا کردن) برابر یک واحد و

هزینه مسیر برابر با تعداد گام ها

حالت اولیه: یک چینش تصادفی

تابع تست هدف: آیا تمامی وزیرها در موقعیت امن هستند؟

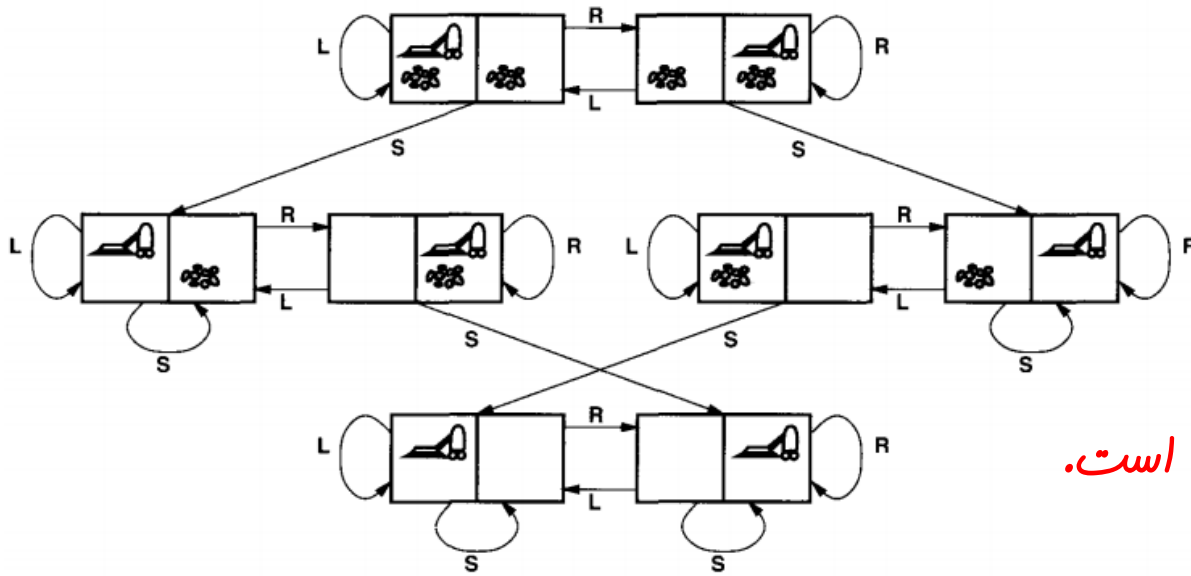
گراف فضای حالت

گراف فضای حالت (State Space Graph) یا گراف جستجو یک نمایش ریاضی برای مسئله جستجو است.

گره ها نمایش دهنده حالت های جهان هستند.

یالها نشان دهنده تابع بعدی (نتیجه اعمال) هستند.

در این گراف هر حالت فقط یکبار ظاهر می شود.



برای اغلب مسائل، تشکیل این گراف امکان پذیر نیست زیرا اندازه آن بسیار بزرگ خواهد بود. ولی برای تحلیل، ایده مناسبی است.

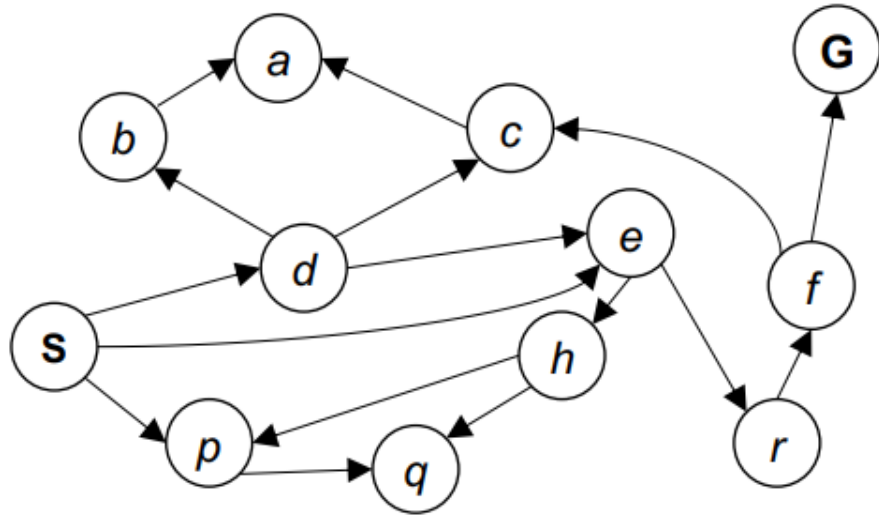
گراف فضای حالت

گراف فضای حالت (State Space Graph) یا گراف جستجو یک نمایش ریاضی برای مسئله جستجو است.

گره ها نمایش دهنده حالت های جهان هستند.

یالها نشان دهنده تابع بعدی (نتیجه اعمال) هستند.

در این گراف هر حالت فقط یکبار ظاهر می شود.



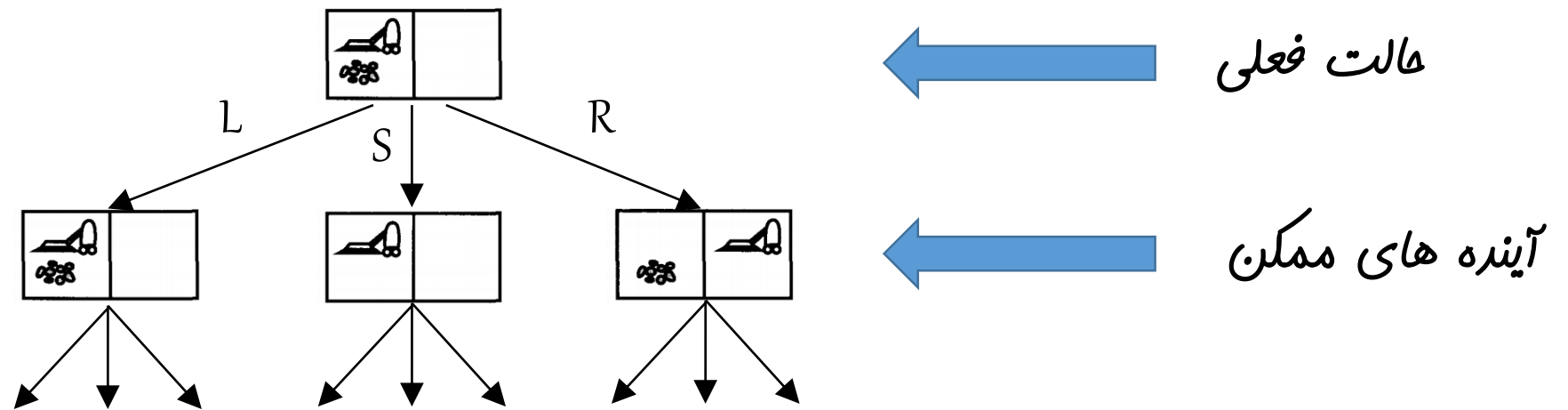
برای اغلب مسائل، تشکیل این گراف امکان پذیر نیست زیرا اندازه آن بسیار بزرگ خواهد بود. ولی برای تحلیل، ایده مناسبی است.

درخت جستجو

یک درخت جستجو یک درخت اگر-آنگاه برای برنامه ها و نتایج آنها است.

حالت اولیه ریشه درخت است

فرزندان معادل با حالت های بعدی هستند



پیاده سازی درخت جستجو

علاوه بر این، نیاز به مملی برای نگهداری گره‌هایی داریم که منتظر بسط داده شدن هستند. بدین منظور از یک صف (Queue) استفاده خواهیم کرد.

اعمال زیر بر روی این صف قابل انجام هستند:

MAKE-QUEUE(Elements): یک صف با عناصر داده شده ایجاد می‌کند.

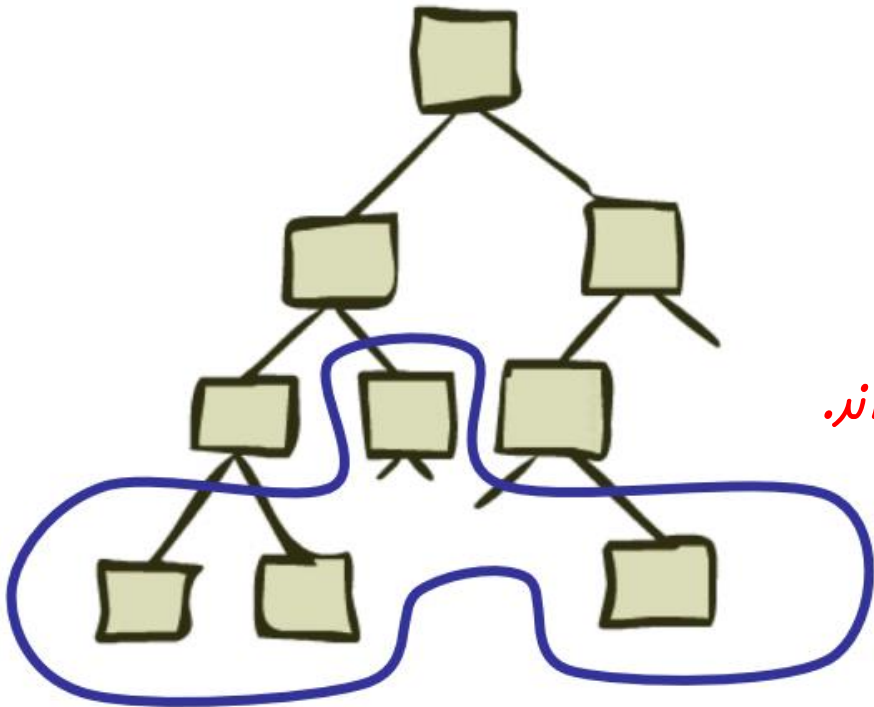
EMPTY?(Queue): آیا صف خالی است؟

REMOVE-FRONT(Queue): عنصر ابتدای صف را حذف کرده و برمی‌گرداند.

QUEUING-FN(Elements, Queue): مجموعه ای از عناصر را در صف

قرار می‌دهد. اینکه عناصر جدید به کجای صف اضافه شوند، استراتژی‌های

مختلف جستجو را تعریف می‌کند.



function GENERAL-SEARCH(*problem*, QUEUING-FN) **returns** a solution, or failure

nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[*problem*]))

loop do

if *nodes* is empty **then return** failure

node ← REMOVE-FRONT(*nodes*)

if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **then return** *node*

nodes ← QUEUING-FN(*nodes*, EXPAND(*node*, OPERATORS[*problem*]))

end

در زمان ایجاد درخت جستجو، تعدادی گره جدید تولید شده و منتظر بسط داده شدن هستند. اینکه کدامیک باید زودتر بررسی و بسط داده شوند، استراتژی جستجو نامیده می شود.

استراتژی های جستجو: عمق اول

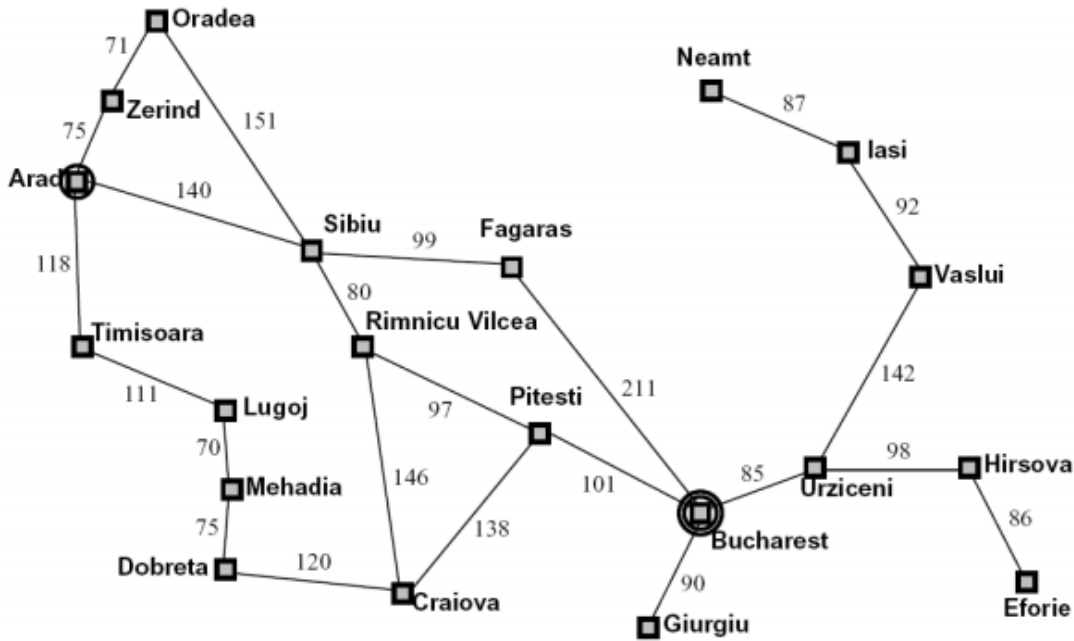
در این استراتژی، همواره عمیق ترین گره زودتر از بقیه بسط داده می شود.

در این استراتژی، فرزندان تولید شده از بسط گره ها، به ابتدای صف اضافه می شوند. به عبارت دیگر، صف همانند یک پشته عمل می کند.



استراتژی های جستجو: عمق اول

مثال: مسافرت در رومانی

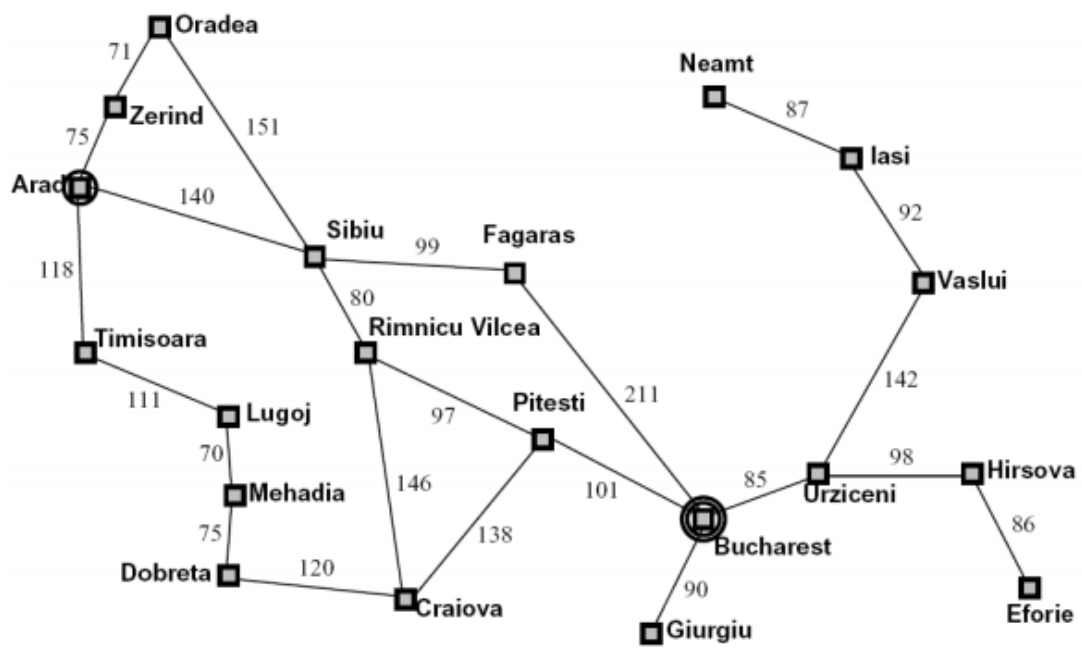


1
STATE: Arad
PARENT: null
OPERATOR: null
DEPTH: 0
PATH-COST: 0

nodes



$nodes \leftarrow \text{MAKE-QUEUE}(\text{MAKE-NODE}(\text{INITIAL-STATE}[\text{problem}])))$



1

STATE: Arad
 PARENT: null
 OPERATOR: null
 DEPTH: 0
 PATH-COST: 0

nodes



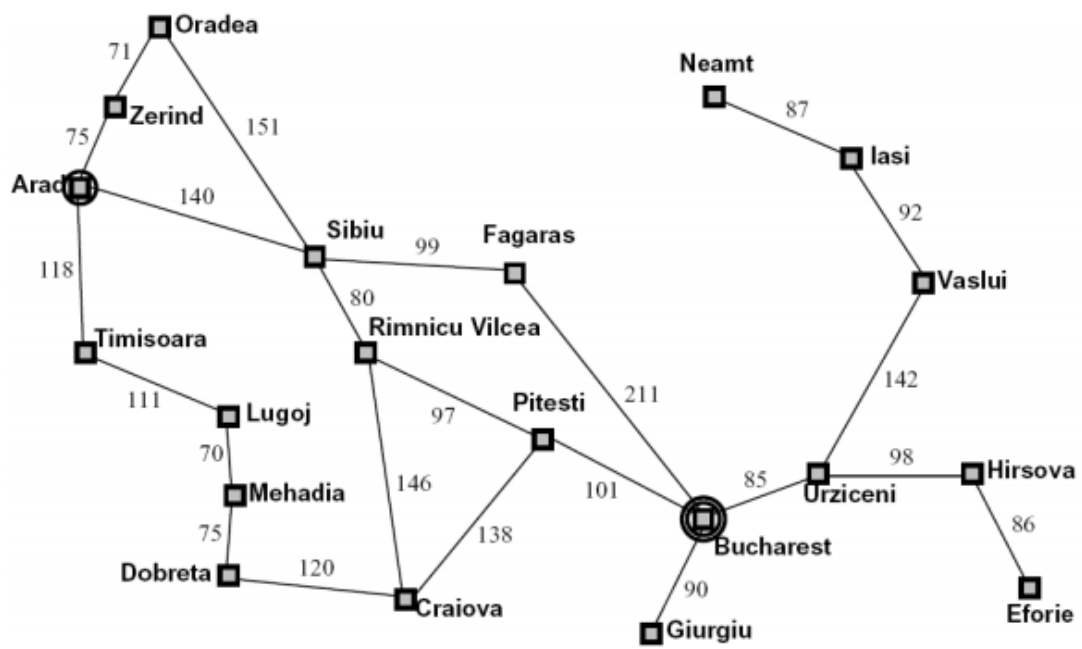
loop do

if nodes is empty **then return** failure

node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds **then return** node

nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))



1

STATE: Arad
 PARENT: null
 OPERATOR: null
 DEPTH: 0
 PATH-COST: 0

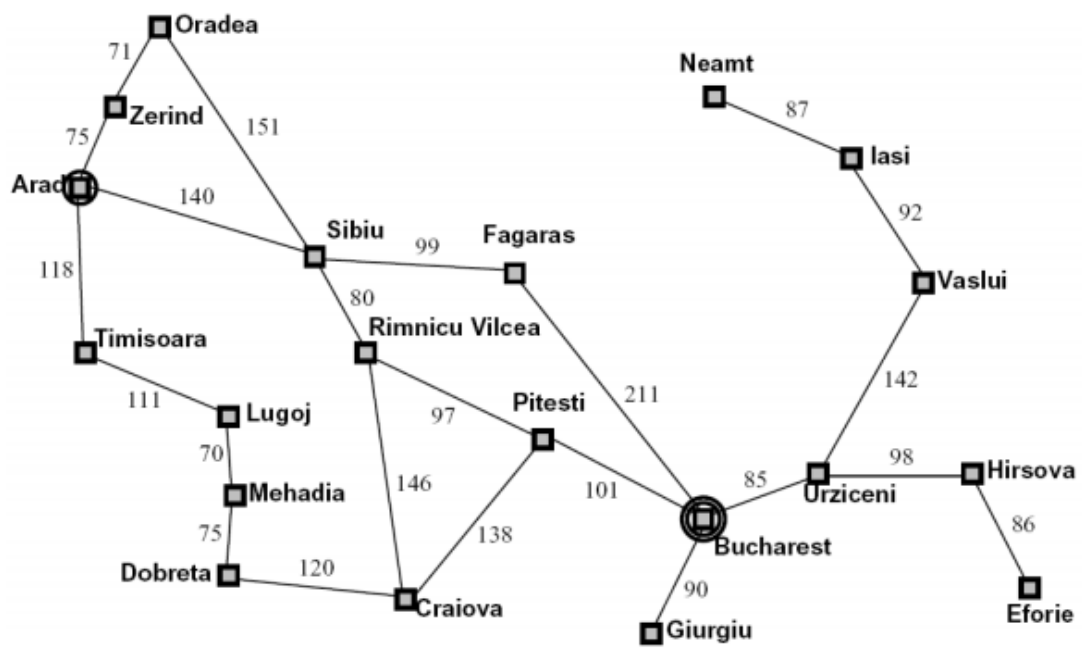
nodes



```

loop do
  if nodes is empty then return failure
  node ← REMOVE-FRONT(nodes)
  if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
  nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

```

1

STATE: Arad
 PARENT: null
 OPERATOR: null
 DEPTH: 0
 PATH-COST: 0

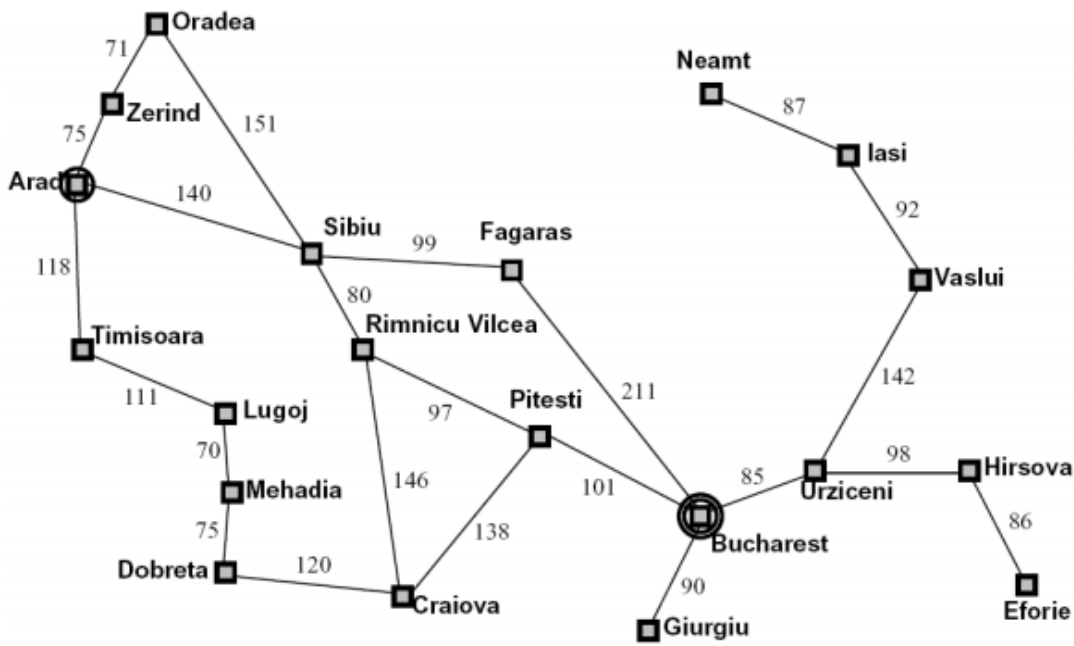
nodes

```

loop do
  if nodes is empty then return failure
  node ← REMOVE-FRONT(nodes)
  if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
  nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

```

nodes



1

STATE: Arad
 PARENT: null
 OPERATOR: null
 DEPTH: 0
 PATH-COST: 0

2

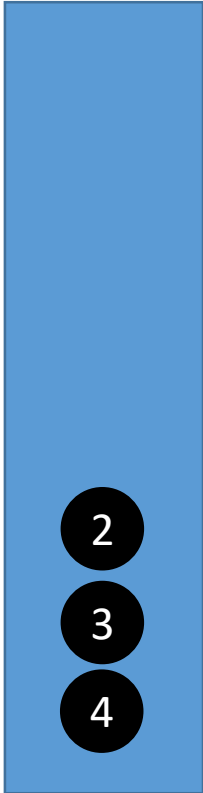
STATE: Zerind
 PARENT: 1
 OPERATOR: go Zerind
 DEPTH: 1
 PATH-COST: 75

3

STATE: Sibiu
 PARENT: 1
 OPERATOR: go Sibiu
 DEPTH: 1
 PATH-COST: 140

4

STATE: Timisoara
 PARENT: 1
 OPERATOR: go Tim...
 DEPTH: 1
 PATH-COST: 118



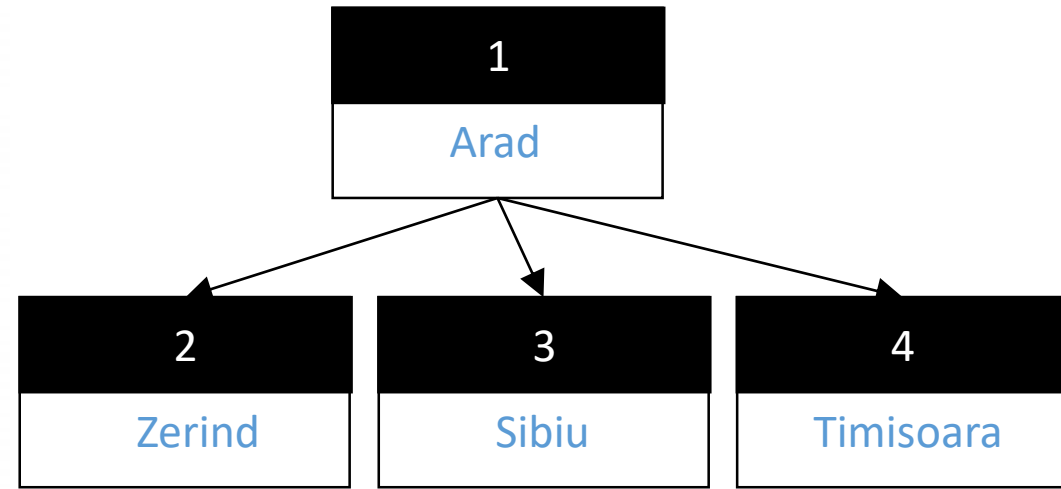
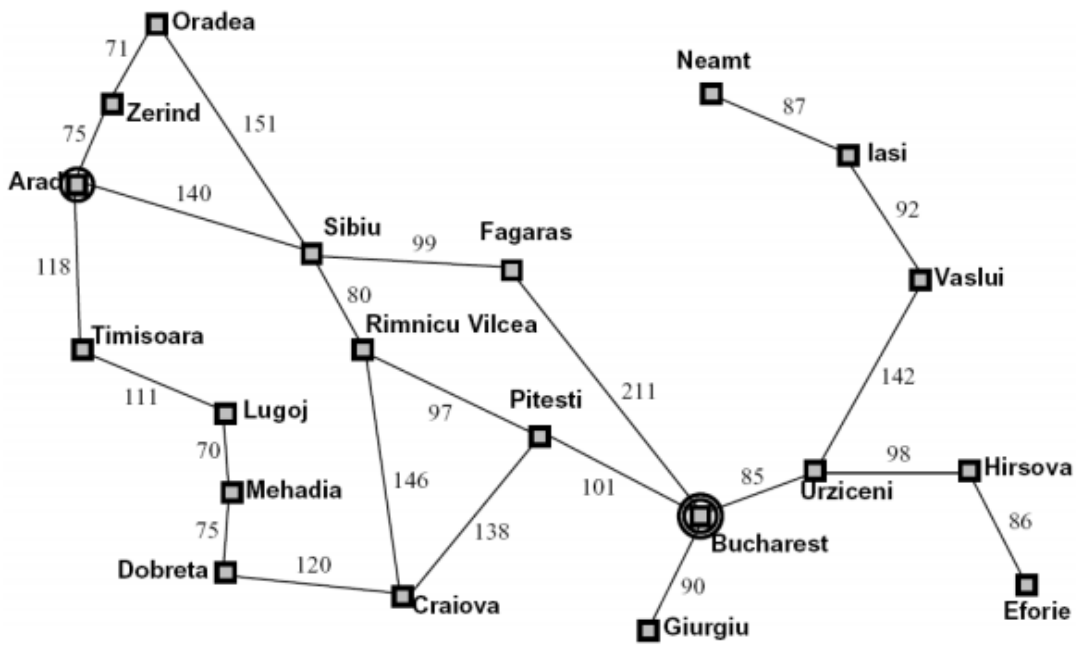
loop do

if nodes is empty then return failure

node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds then return node

nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

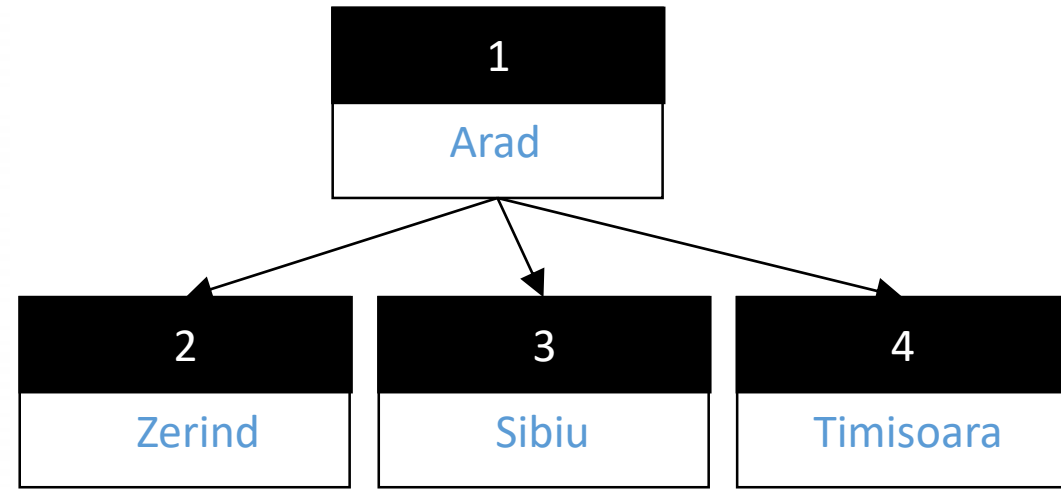
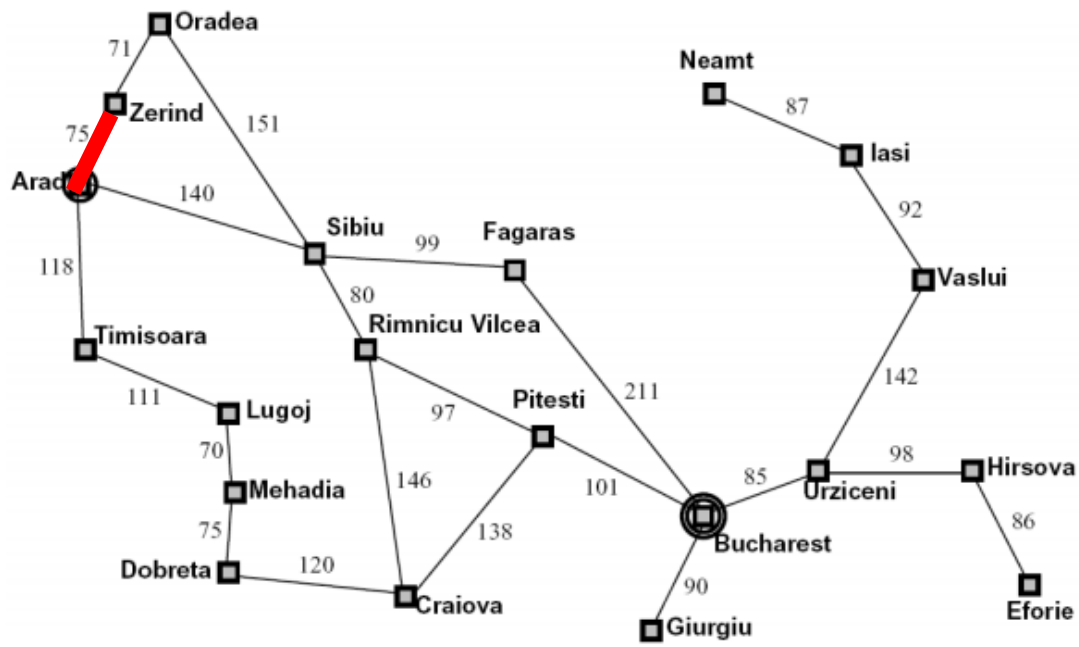


2

3

4

loop do**if** nodes is empty **then return** failure $node \leftarrow \text{REMOVE-FRONT}(\text{nodes})$ **if** GOAL-TEST[problem] applied to STATE(node) succeeds **then return** nodenodes $\leftarrow \text{QUEUING-FN}(\text{nodes}, \text{EXPAND}(\text{node}, \text{OPERATORS}[\text{problem}])))$



loop do

if nodes is empty then return failure

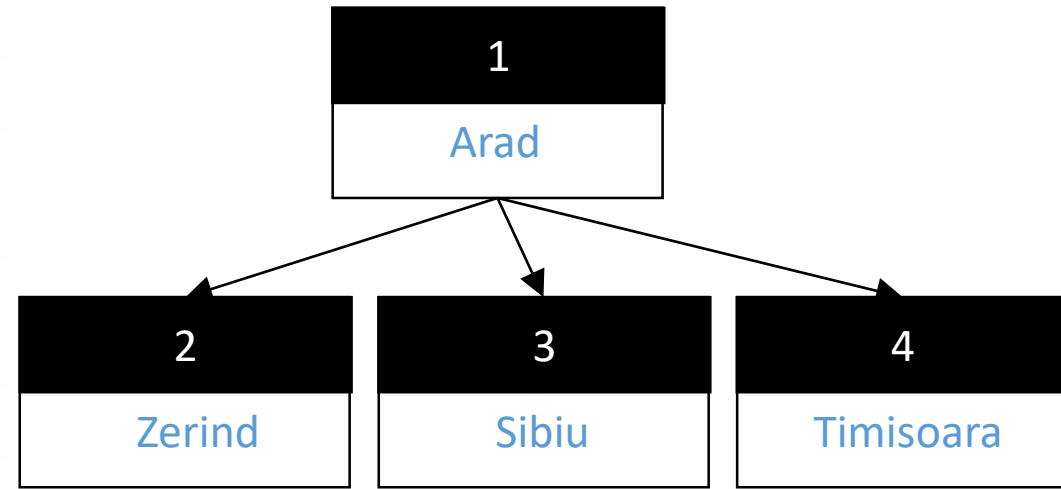
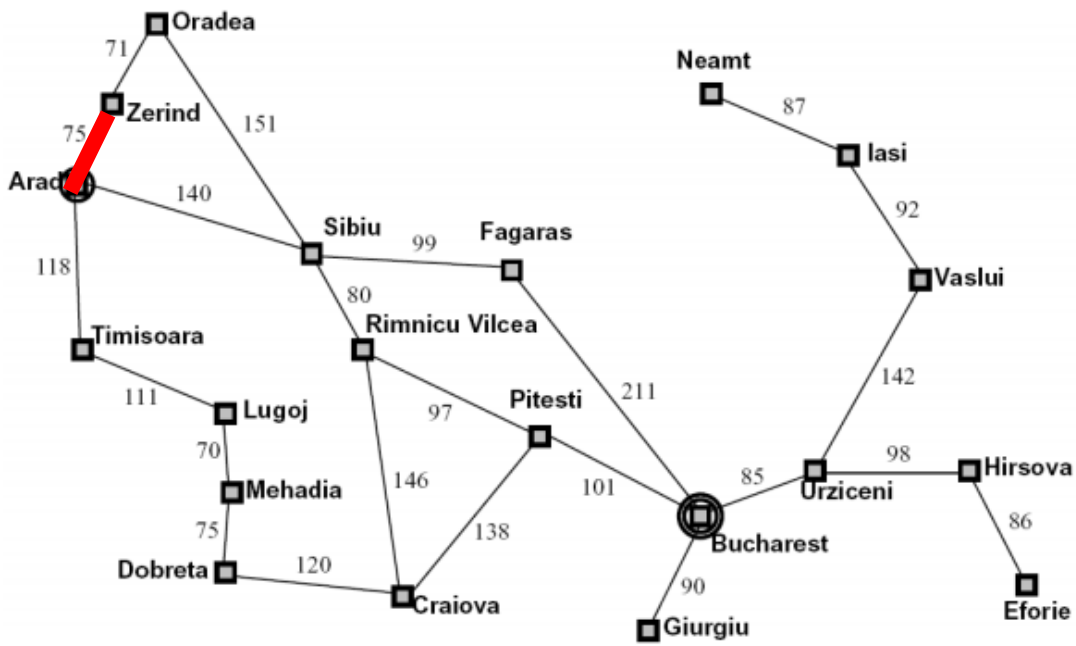
node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds then return node

nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

3

4



3

4

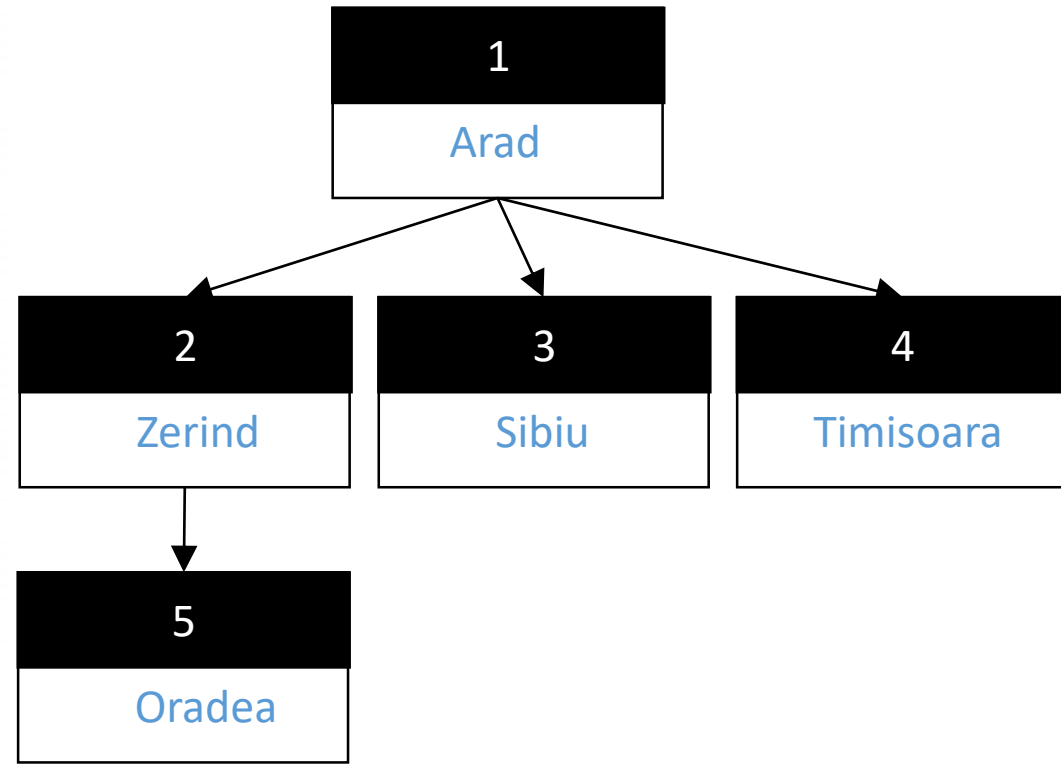
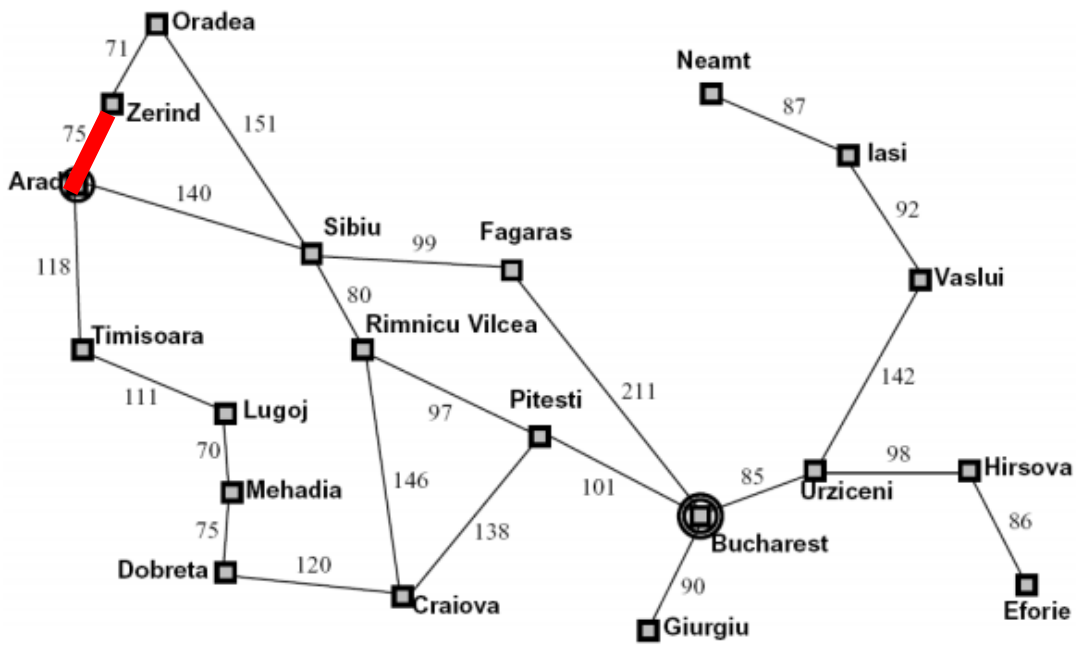
loop do

if nodes is empty **then return** failure

node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds **then return** node

nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))



5

3

4

loop do

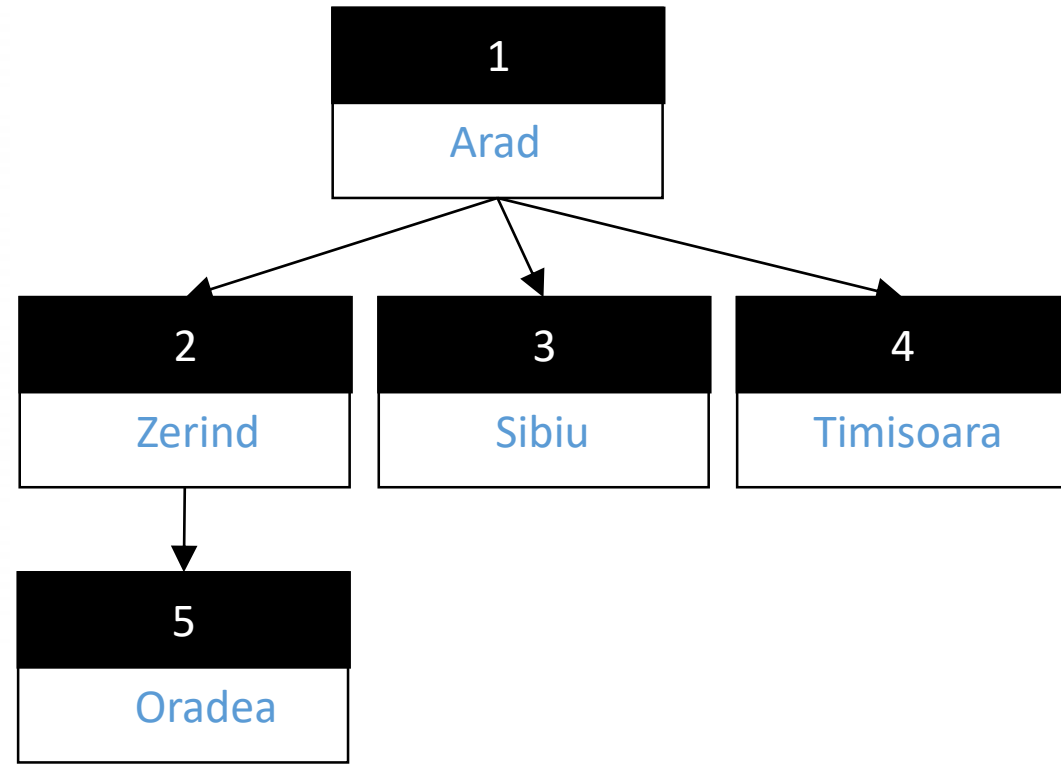
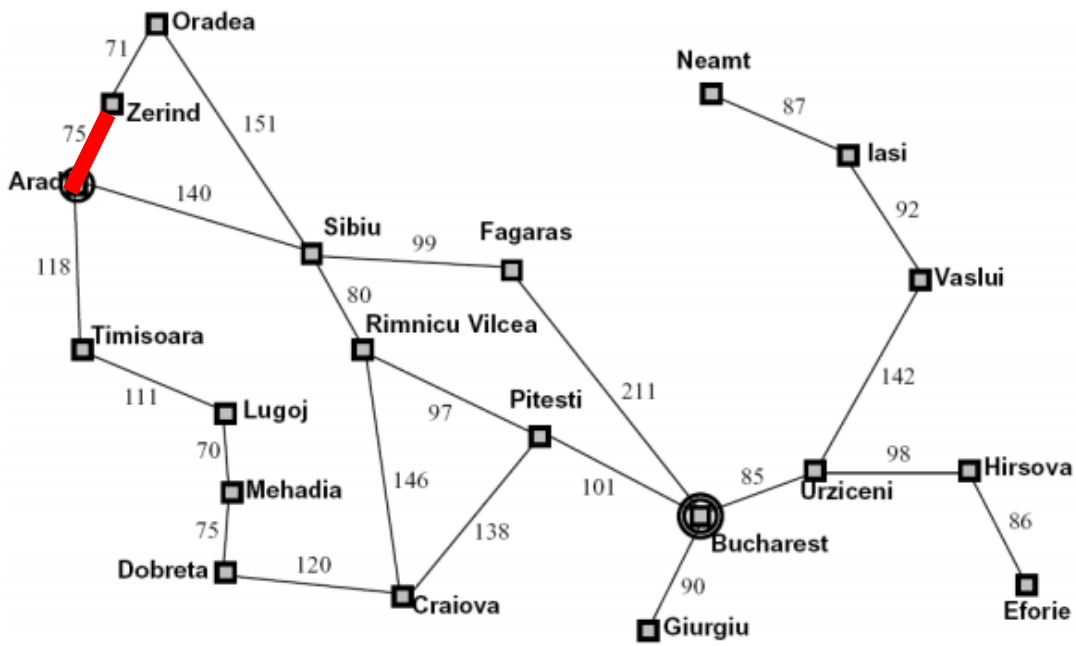
if nodes is empty then return failure

node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds then return node

nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

با فرض اینکه مسیرها یک طرفه باشد



5

3

4

ادامہ بہ عنوان تمرین

loop do

if nodes is empty then return failure

node ← REMOVE-FRONT(nodes)

if GOAL-TEST[problem] applied to STATE(node) succeeds then return node

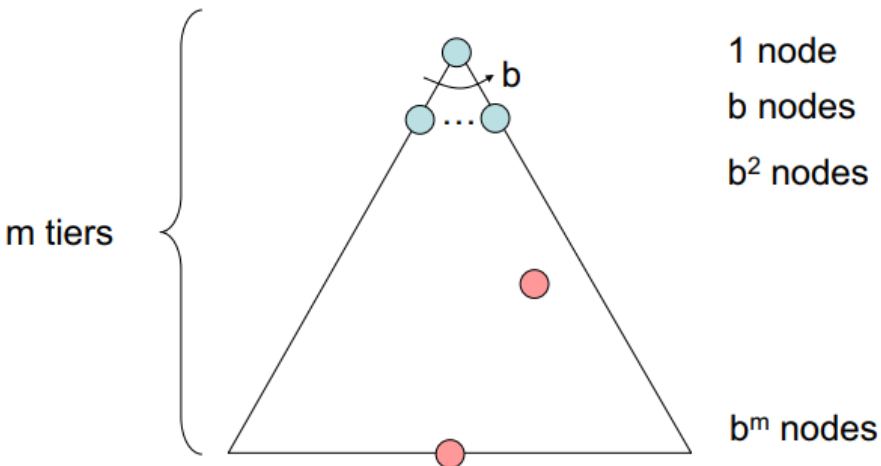
nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))

ارزیابی استراتژی های جستجو

معیارهای ارزیابی استراتژی های جستجو:

- کامل بودن (Completeness): در صورت وجود پاسخ، آیا استراتژی جستجو تضمین می کند که آن را خواهد یافت؟
- پیچیدگی زمانی (Time Complexity): چه مدت زمان نیاز است تا الگوریتم پاسخ را بیابد.
- پیچیدگی فضایی (Space Complexity): چه مقدار حافظه نیاز است تا الگوریتم کار خود را انجام دهد.
- بهینگی (Optimality): چه مقدار حافظه نیاز است تا الگوریتم کار خود را انجام دهد.

شکل کامل درخت جستجو



○ **B**: فاکتور انشعاب

○ **m**: حداکثر عمق

○ جواب ها می توانند در عمق های مختلف باشند.

$$1 + b + b^2 + \dots + b^m = O(b^m) \quad \text{تعداد کل گره های درخت}$$

ارزیابی استراتژی عمق اول

DFS چه گره‌هایی را بسط می‌دهد؟

این الگوریتم همواره یک پیشوند چپ از کل درخت جستجو را بسط می‌دهد در صورتی که حداکثر عمق محدود باشد، می‌تواند کل درخت جستجو را پردازش کند.

پیمیدگی زمانی

در صورتی که حداکثر عمق محدود باشد، دارای پیچیدگی زمانی $O(b^m)$ است.

پیمیدگی فضایی

تنها هموالدهای گره‌های تا ریشه را نگهداری می‌کند. بنابراین پیچیدگی فضایی $O(bm)$ است.

آیا DFS کامل است؟

فیر، m می‌تواند بینهایت باشد. تنها در صورتی که m محدود باشد یا از گره‌های تکراری جلوگیری کنیم، این استراتژی کامل خواهد بود.

آیا DFS بهینه است؟

فیر، این الگوریتم همواره سمت چپ‌ترین گره را بدون در نظر گرفتن عمق یا هزینه می‌یابد.

