دانشکده ریاضی و آمار

هوش مصنوعی

UnderGrad.Q5/7: Artificial Intelligence

فریا نصیری مفخّم

Faria Nassiri-Mofakham

https://eng.ui.ac.ir/**fnasiri**

4016282-01

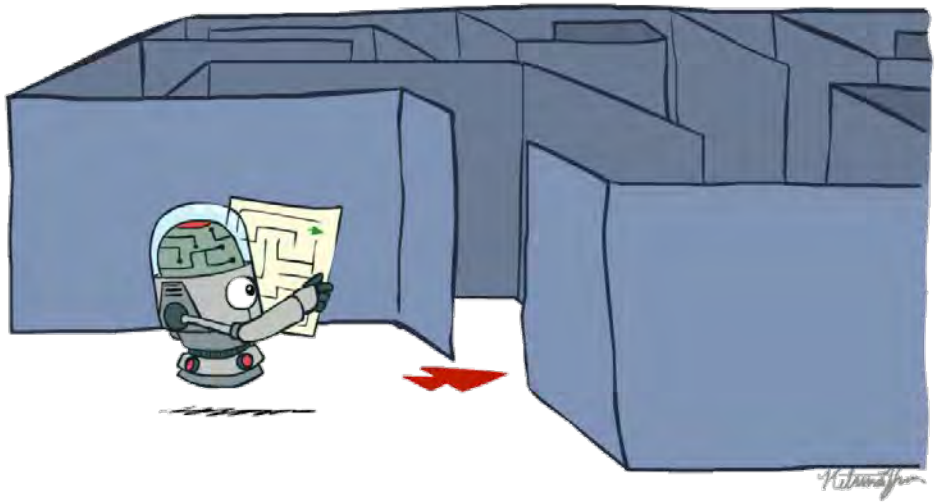
یکشنبه‌ها ۱۸–۱۷–۱۶، سه‌شنبه‌ها ۱۰–۸

lms.ui.ac.ir, ۳۶۰–۷

نیمسال ۴۰۴۲

CS 188: Artificial Intelligence

Search



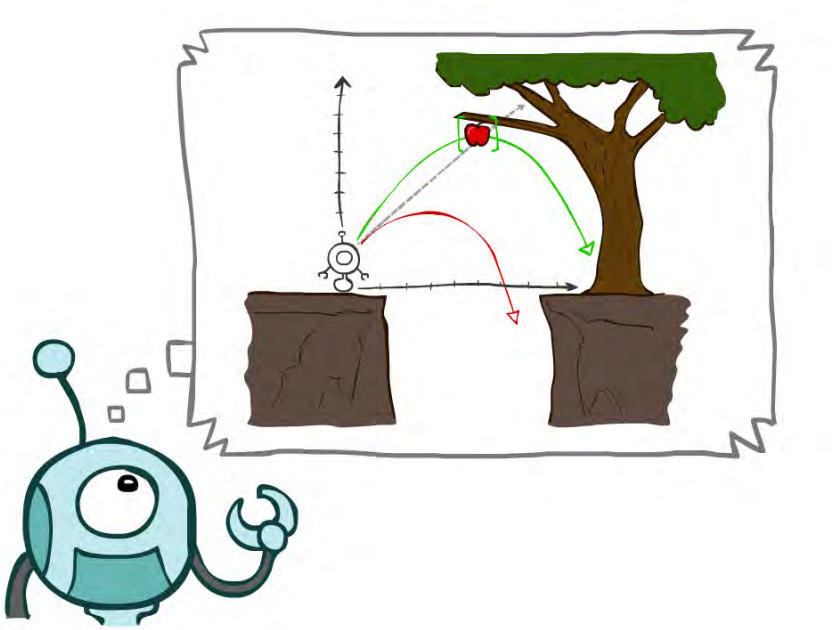
Instructors: Stuart Russell and Dawn Song

University of California, Berkeley

[slides adapted from Dan Klein, Pieter Abbeel]

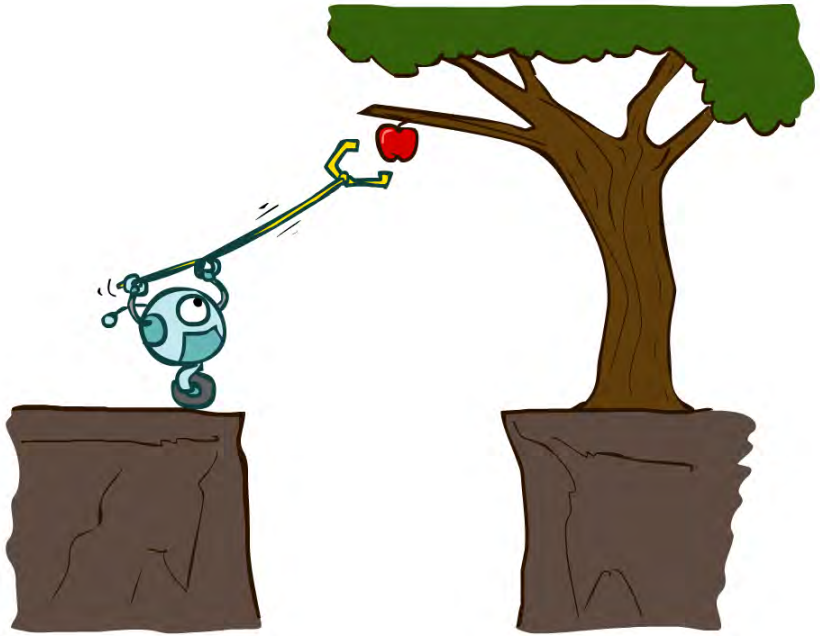
Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



Planning Agents

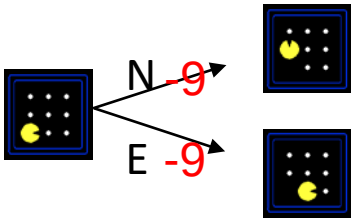
- Planning agents decide based on evaluating future action sequences
- Must have a model of how the world evolves in response to actions
- Usually have a definite goal
- Optimal: Achieve goal at least cost



Search Problems

Search Problems

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model *Result*(s,a)
  - A goal test $G(s)$
    - S has no dots left
  - Action cost $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost



- A solution is an action sequence that reaches a goal state
- An optimal solution has least cost among all solutions

Search Problems Are Models

Example: Traveling in Romania

But then…

Bucharest to London

Lufthansa • Tue, Jan 26

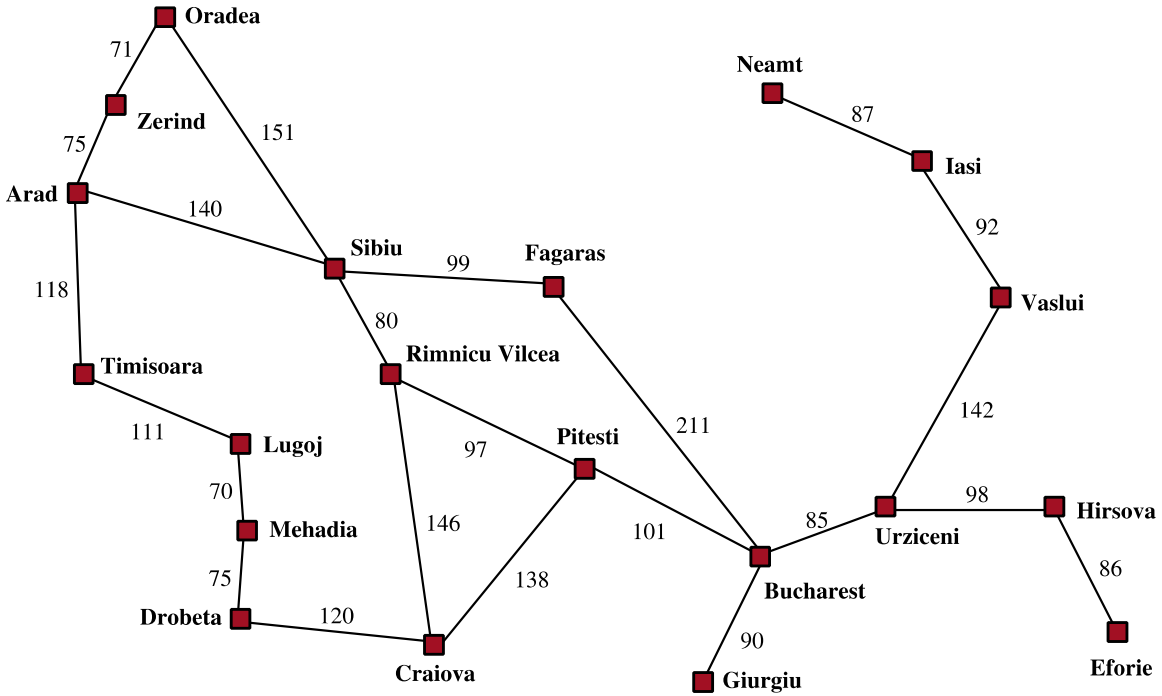
7:10am - 4:45pm

11h 35m (1 stop)

7h 15m in Frankfurt (FRA)

Show details

Example: Traveling in Romania



- State space:
  - Cities
- Initial state:
  - Arad
- Actions:
  - Go to adjacent city
- Transition model:
  - Reach adjacent city
- Goal test:
  - *s* = Bucharest?
- Action cost:
  - Road distance from *s* to *s'*
- Solution?

A *problem* is defined by four items:

initial state e.g., "at Arad"

successor function $S(x)$ = set of action–state pairs
e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

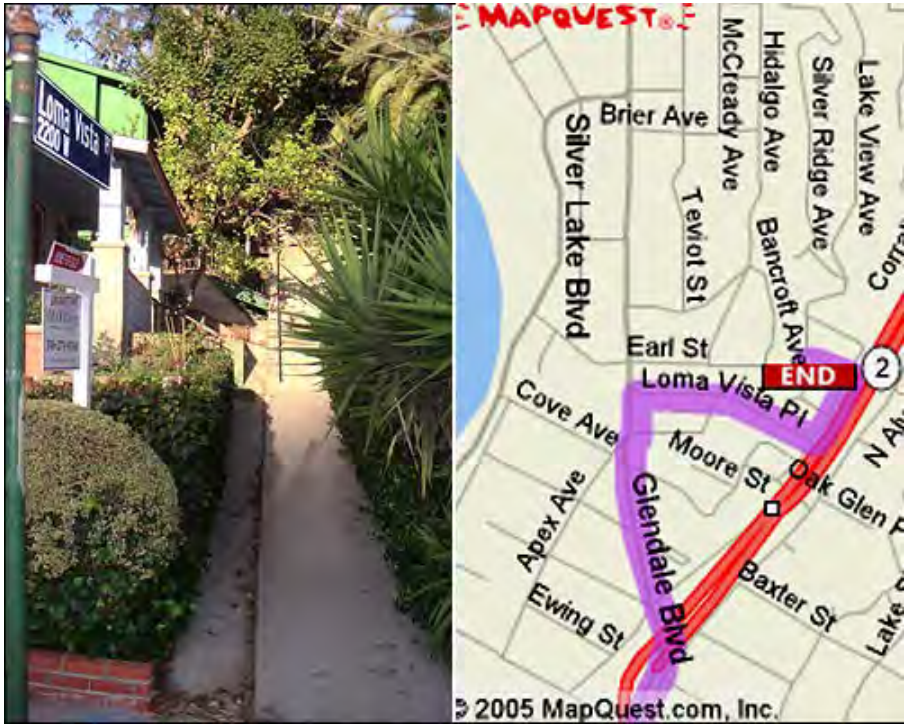
goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of actions executed, etc.
$c(x, a, y)$ is the step cost, assumed to be ≥ 0

A *solution* is a sequence of actions
leading from the initial state to a goal state

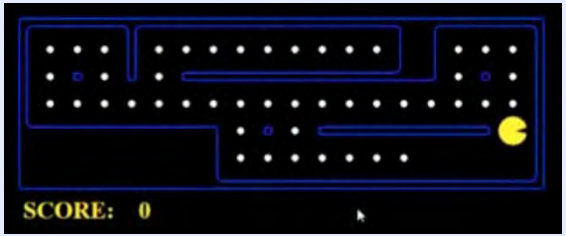
Models are almost always wrong

Models are almost always wrong



What's in a State Space?

The world state includes every last detail of the environment



A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Transition: update x,y value
  - Goal test: is (x,y)=destination

- Problem: Eat-All-Dots
  - States: {(x,y), dot Booleans}
  - Actions: NSEW
  - Transition: update x,y and possibly a dot Boolean
  - Goal test: dots all false

State Space Sizes

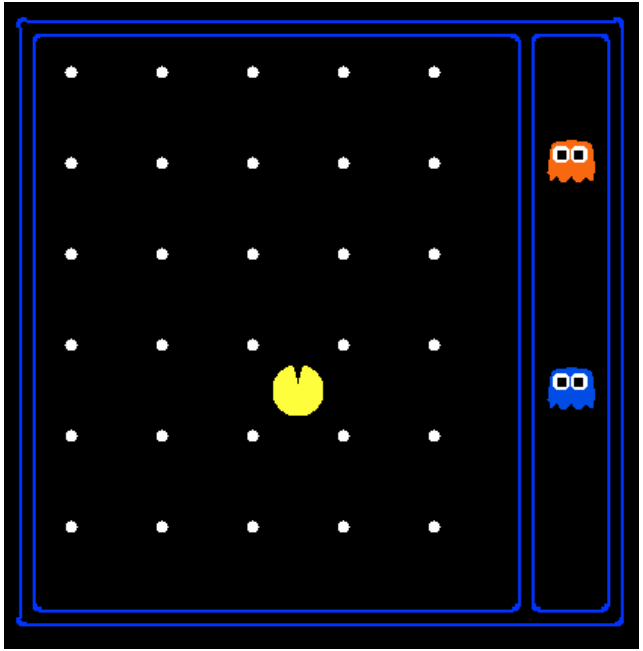
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

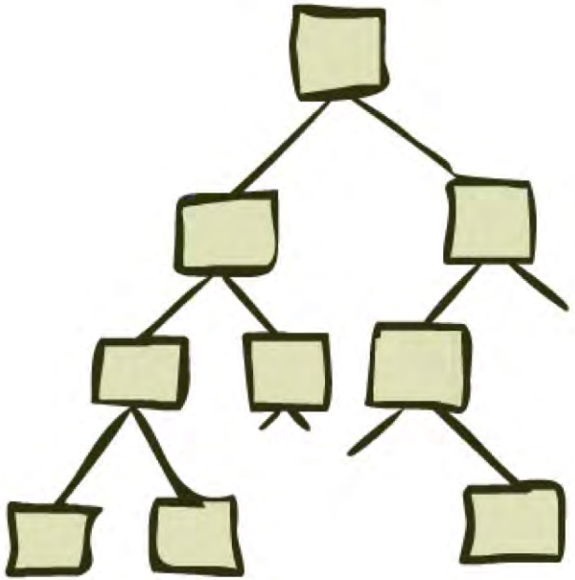
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120 \times (2^{30})$

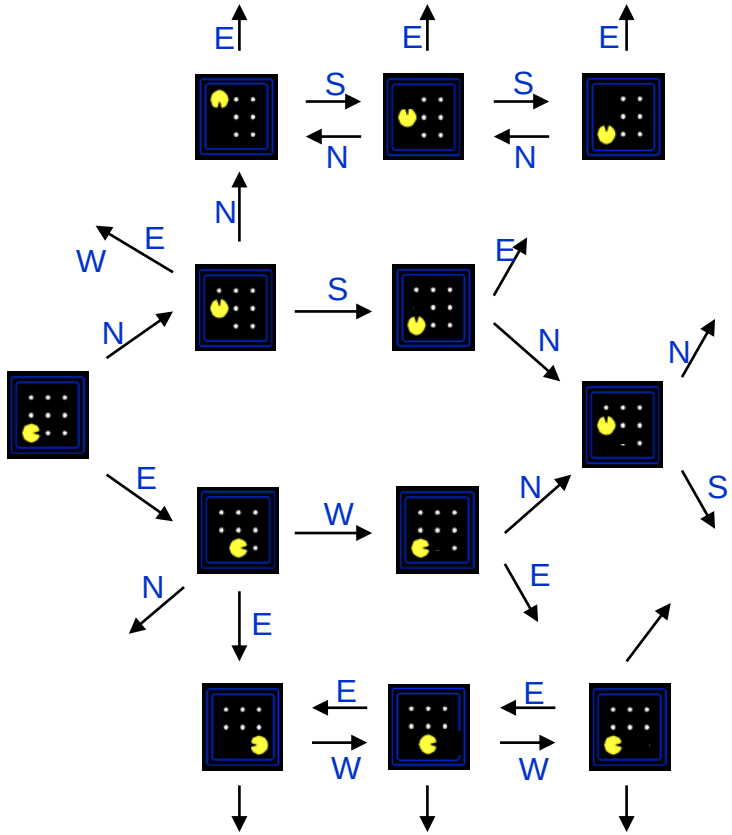


State Space Graphs and Search Trees



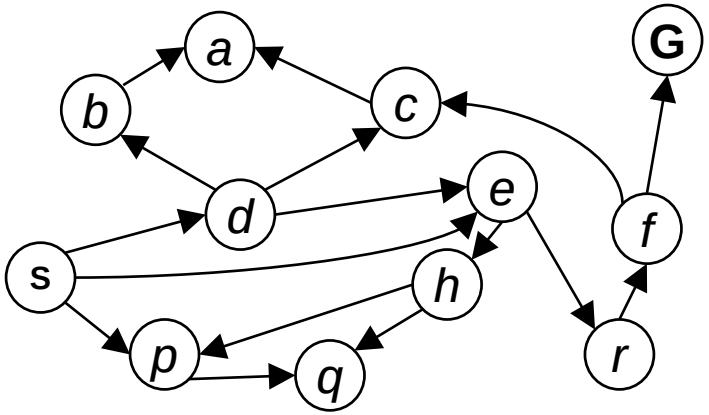
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions (labeled with actions)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



State Space Graphs

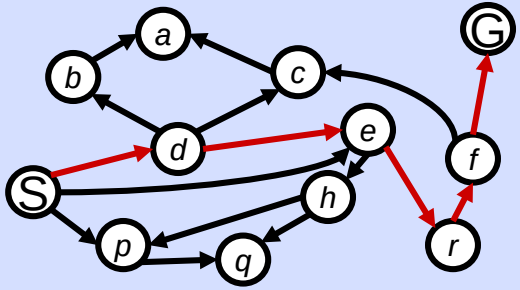
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



Tiny state space graph for a tiny search problem

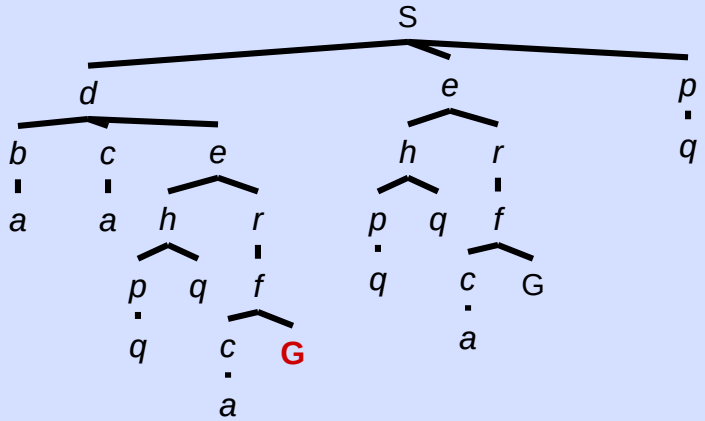
State Space Graphs vs. Search Trees

State Space Graph

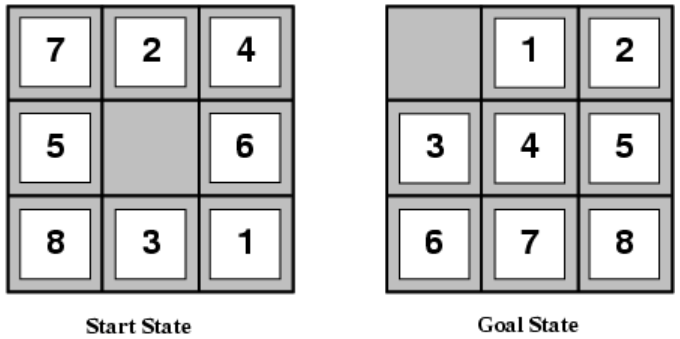


Each NODE in in the search tree is an entire PATH in the state space graph. We construct the tree on demand – and we construct as little as possible.

Search Tree



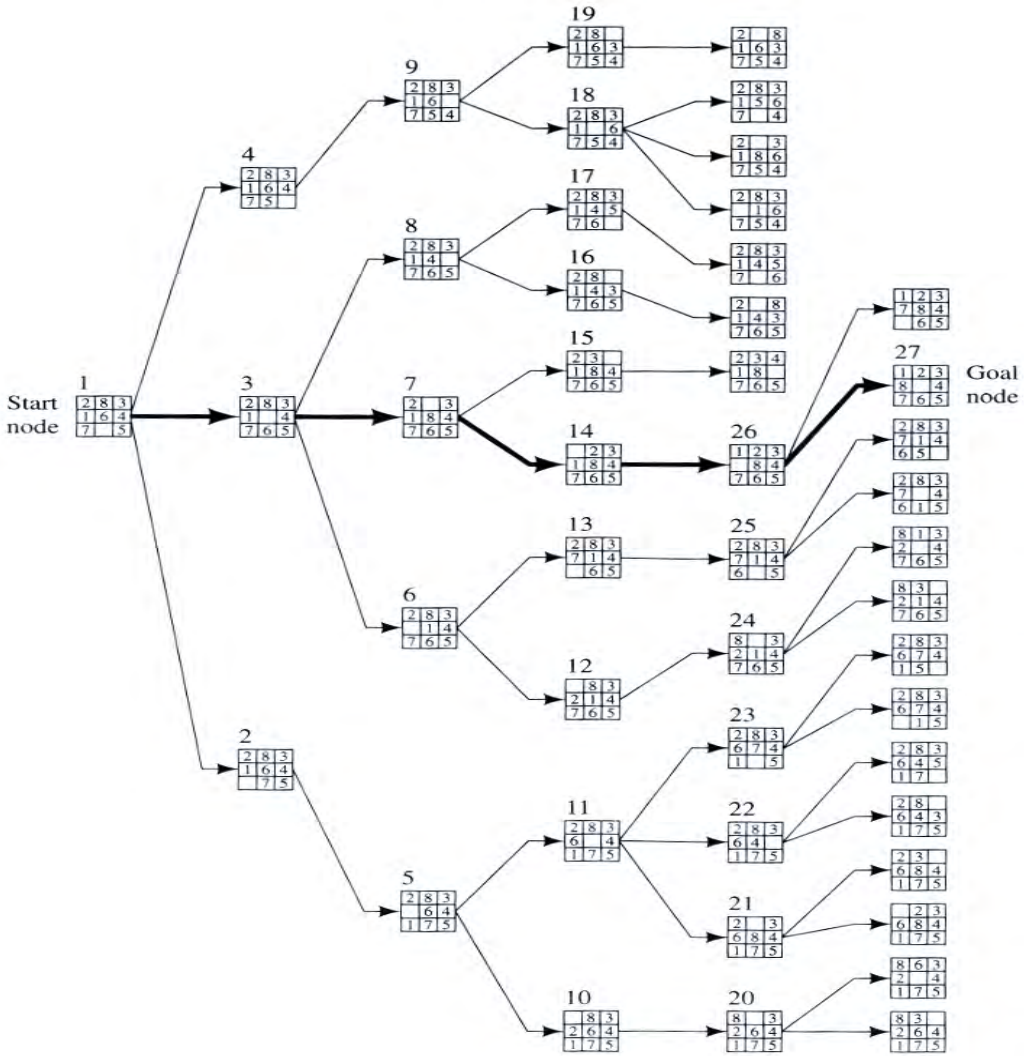
Example: The 8-puzzle



- states? locations of tiles
- initial state? given
- actions? move blank left, right, up, down
- goal test? goal state (given)
- path cost? 1 per move

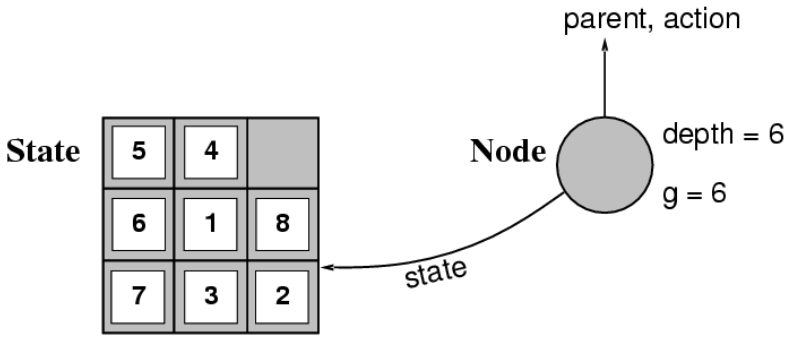
[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: The 8-puzzle



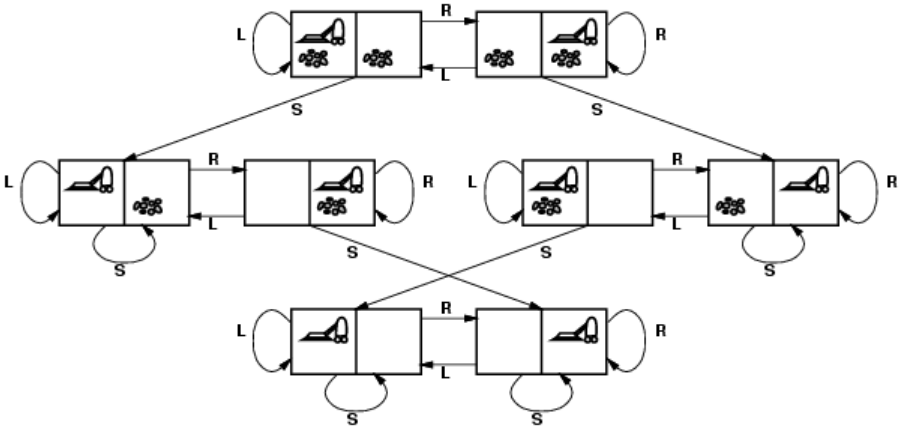
Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth

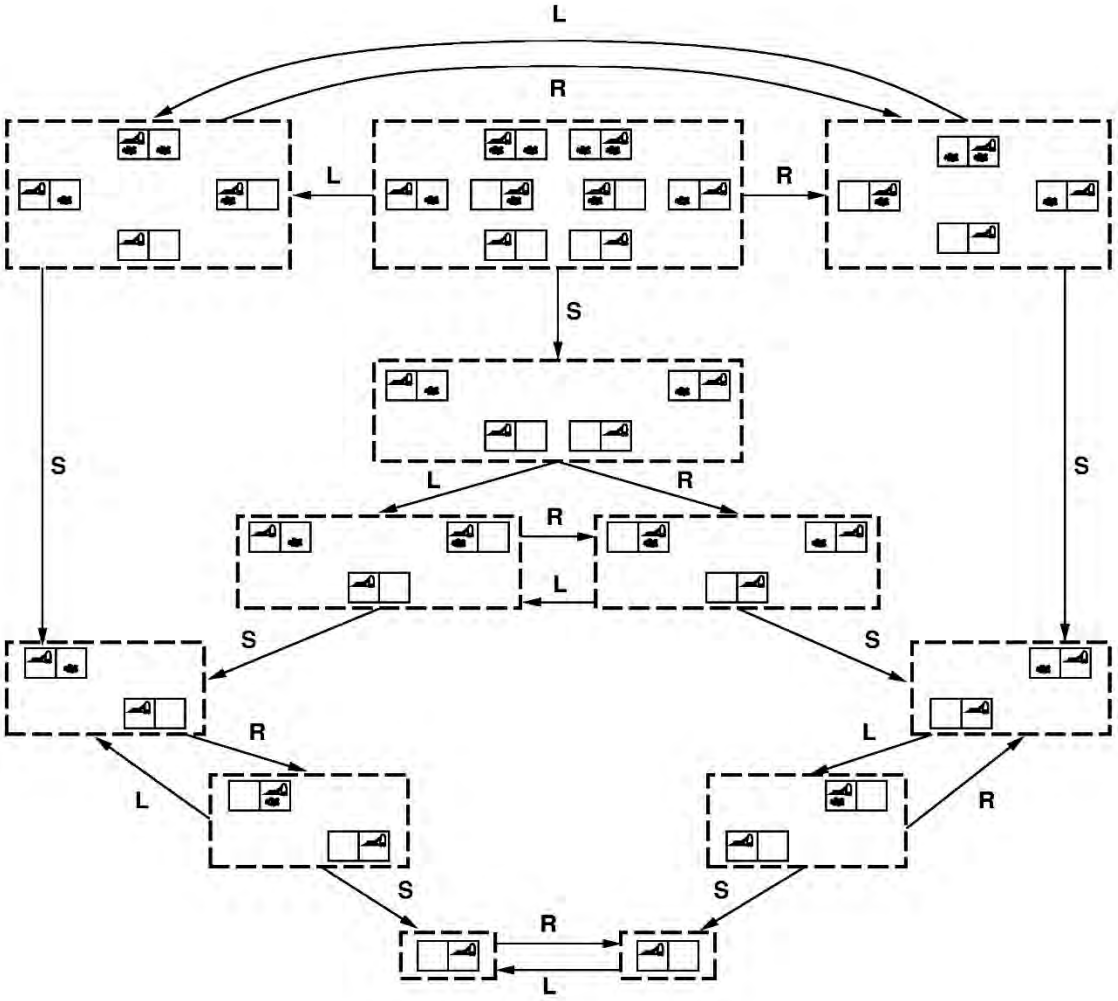


- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

Vacuum world state space graph

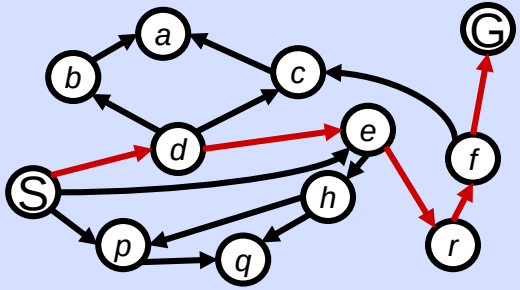


- states? discrete: dirt and robot location
- initial state? any
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action


L
R
L
R
S
S
S
L
R
R
L
S
S
R
L
L
R
S
S
R
L

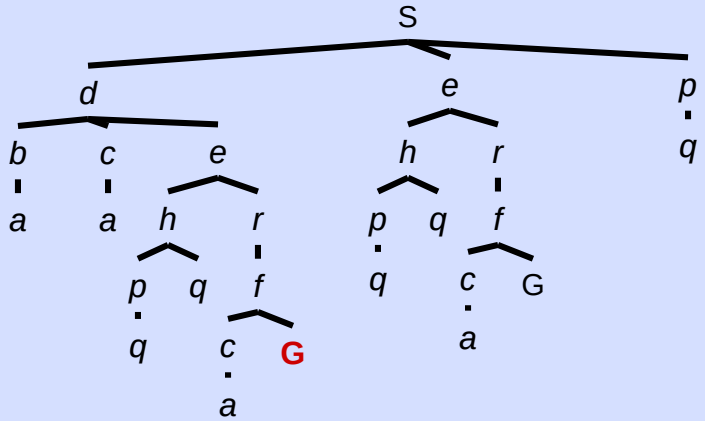
State Space Graphs vs. Search Trees

State Space Graph



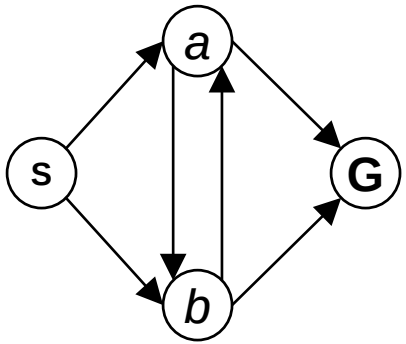
Each NODE in in the search tree is an entire PATH in the state space graph. We construct the tree on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

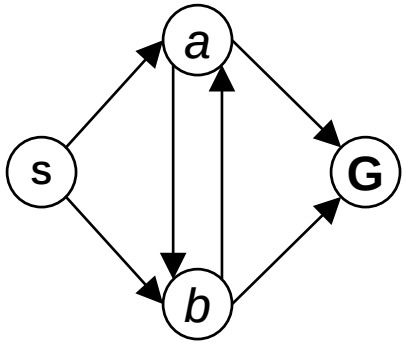
Consider this 4-state graph:



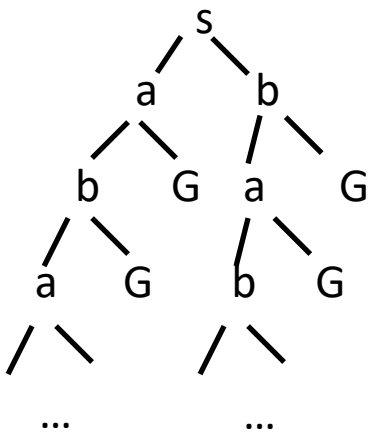
How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



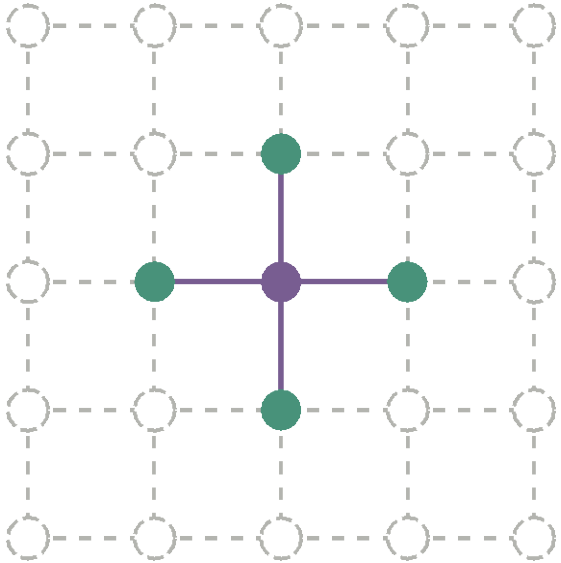
How big is its search tree (from S)?



Important: Those who don’t know history are doomed to repeat it!

Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:



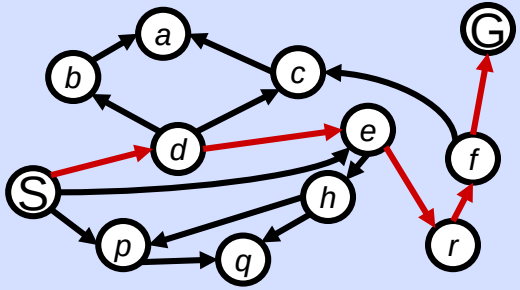
How many states within *d* steps of start?

How many states in search tree of depth *d*?

Search Problems

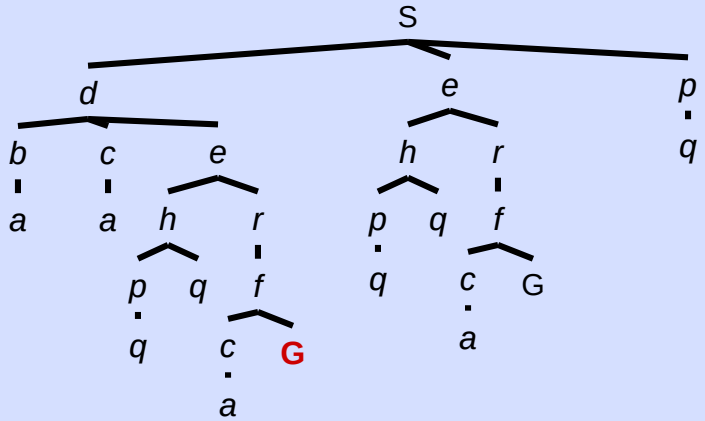
State Space Graphs vs. Search Trees

State Space Graph



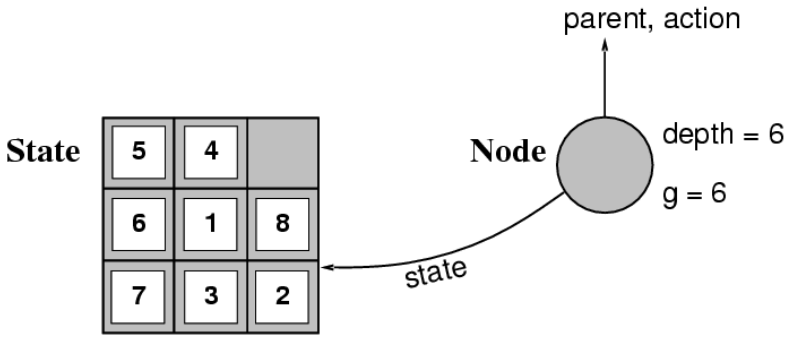
Each NODE in in the search tree is an entire PATH in the state space graph. We construct the tree on demand – and we construct as little as possible.

Search Tree



Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth

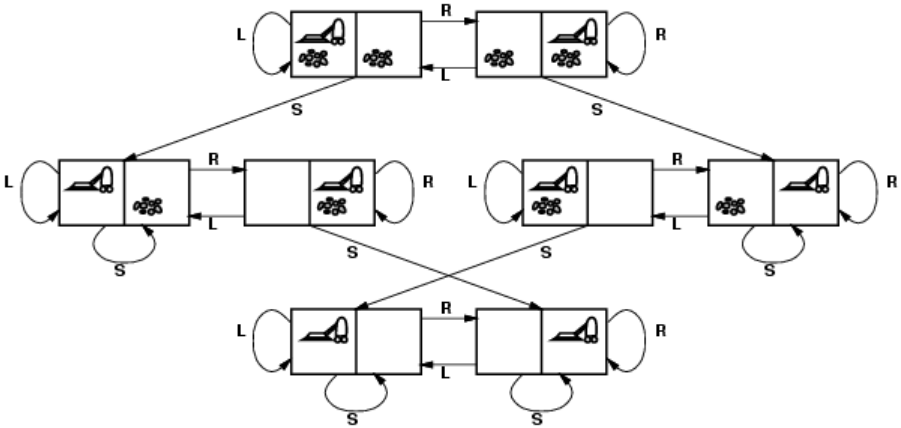


- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

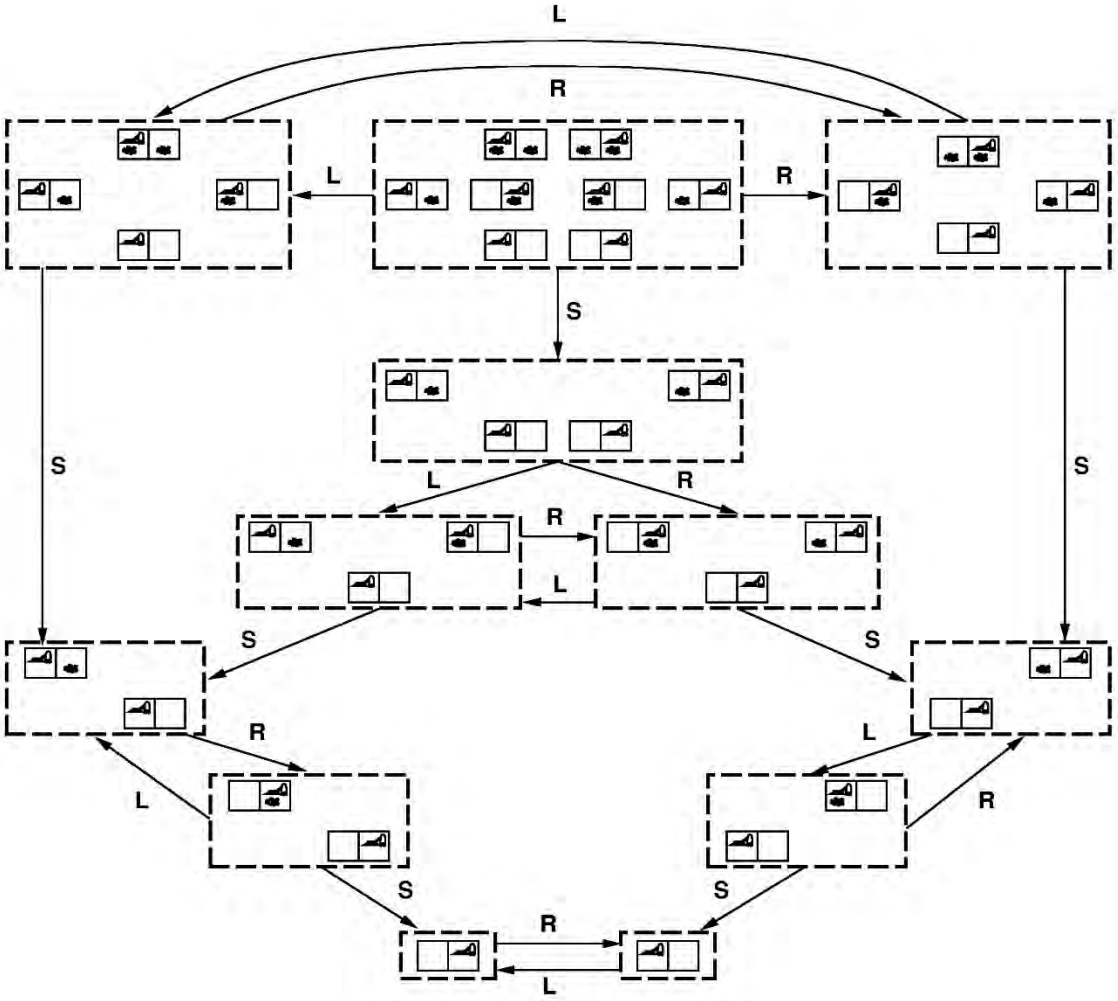
Search Problems

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model $Result(s,a)$ = s' (also SuccessorFunction)
  - A goal test $G(s)$ = T or F
    - s has no dots left
  - Action cost $c(s,a,s')$ = عدد
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost
- A solution is an action sequence that reaches a goal state
- An optimal solution has least cost among all solutions

Vacuum world state space graph



- states? discrete: dirt and robot location
- initial state? any
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

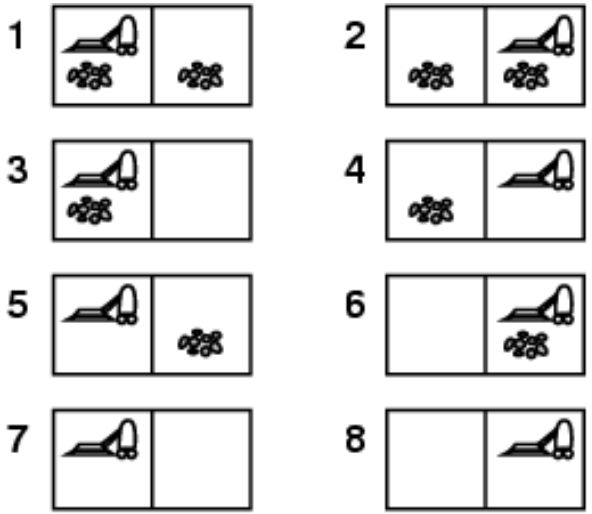

L
R
L
R
S
S
S
L
R
R
L
S
S
R
L
L
R
S
S
R
L

Example: vacuum world

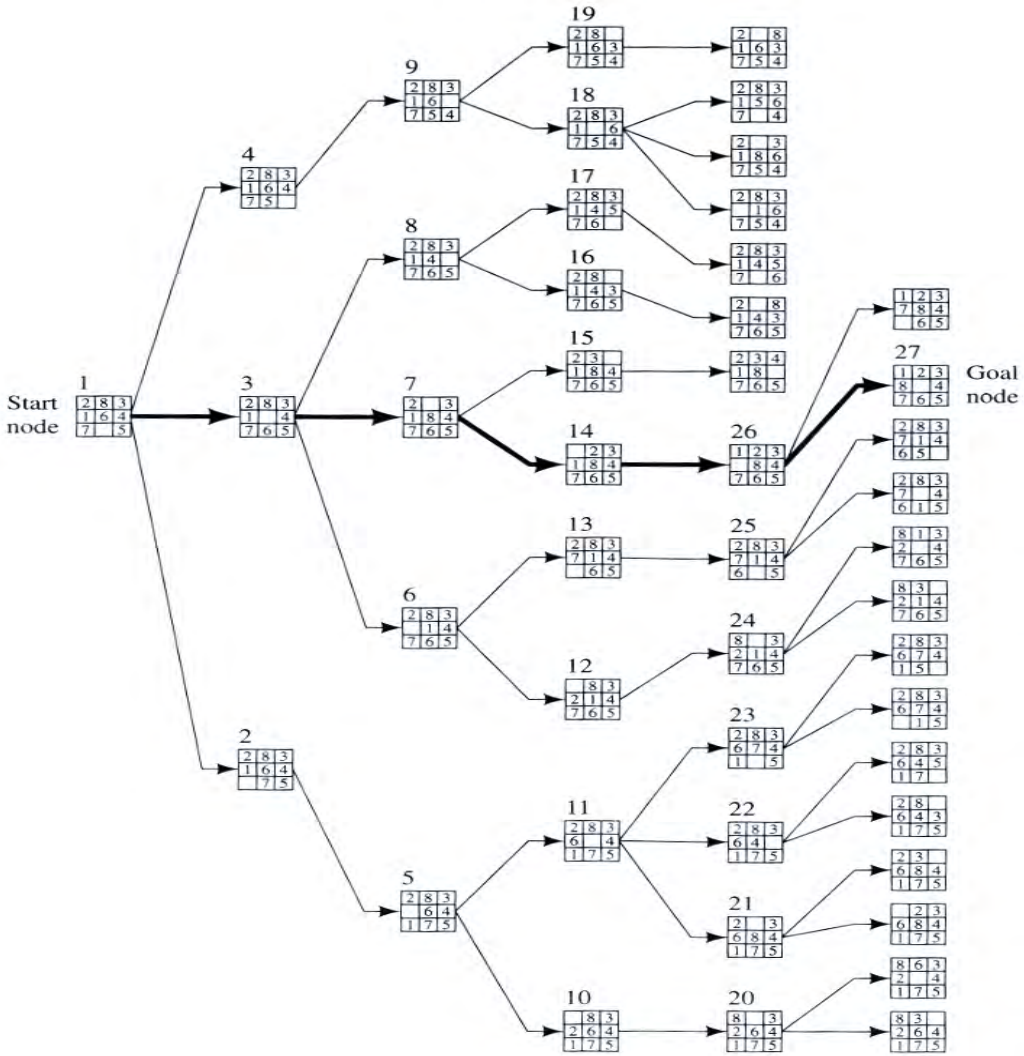
- Observable, start in #5.
  Solution? *[Right, Suck]*

- Unobservable, start in
  {*1,2,3,4,5,6,7,8*} e.g.,
  Solution?

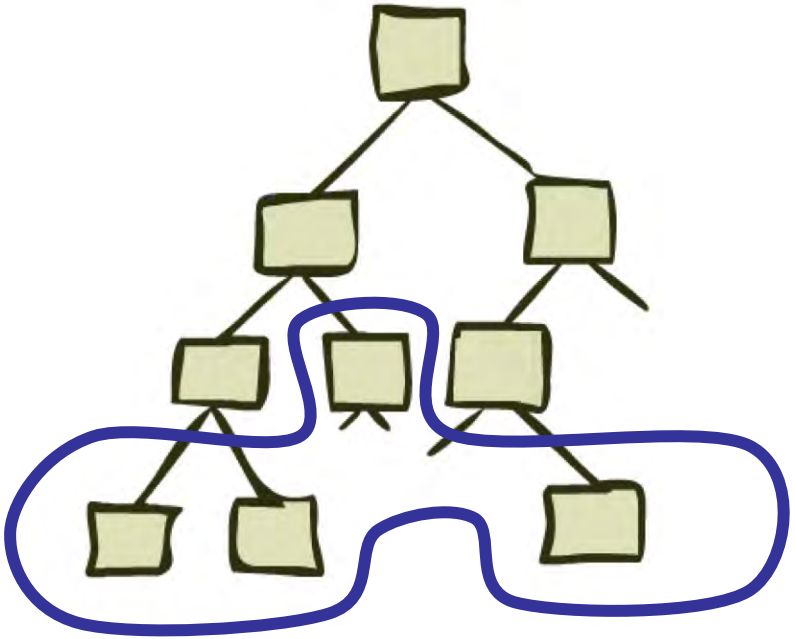
  [Right,Suck,Left,Suck]



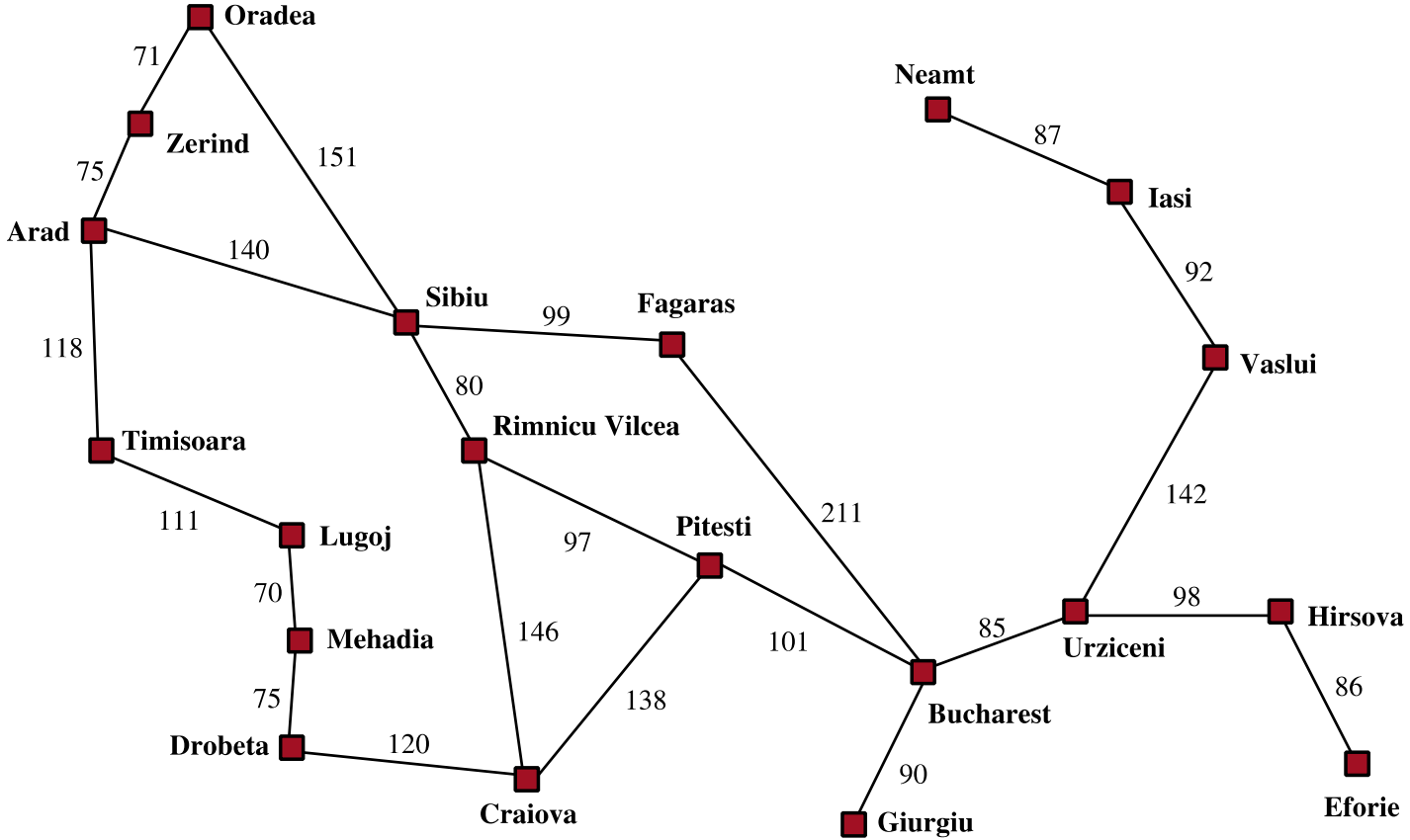
Example: The 8-puzzle



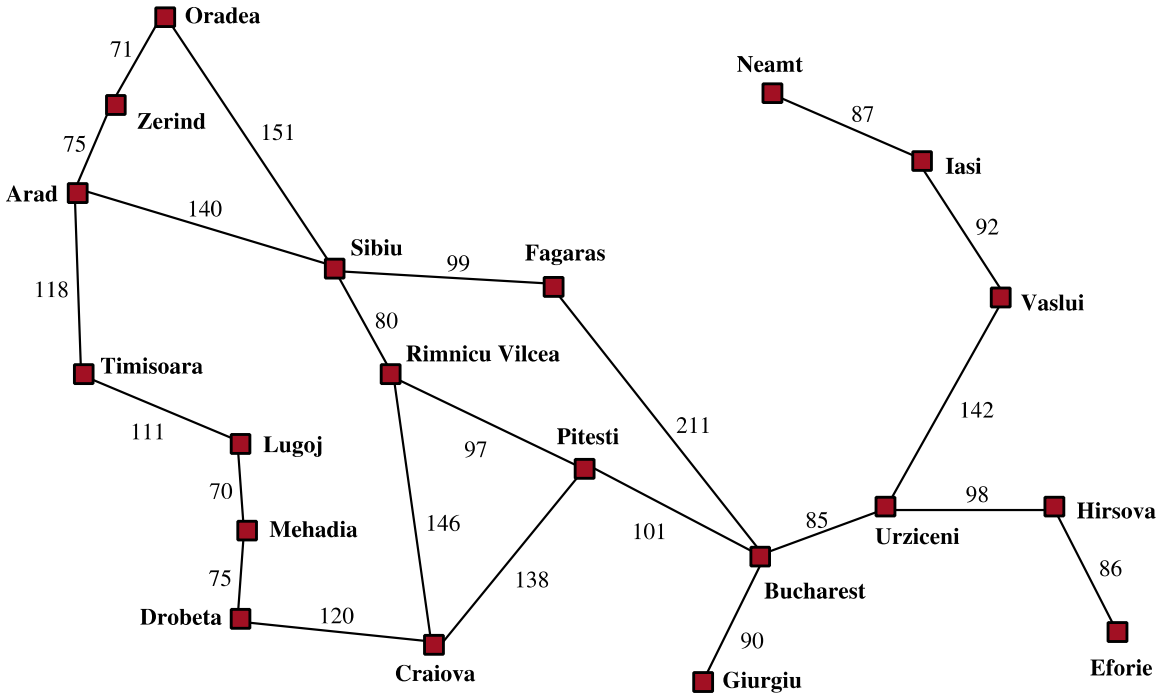
Tree Search



Search Example: Romania



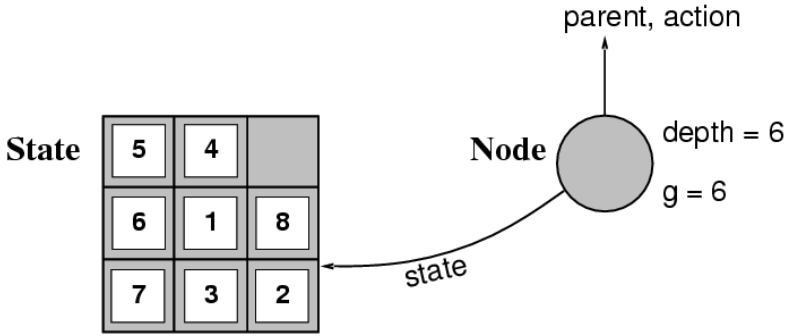
Example: Traveling in Romania



- State space:
  - Cities
- Initial state:
  - Arad
- Actions:
  - Go to adjacent city
- Transition model:
  - Reach adjacent city
- Goal test:
  - *s* = Bucharest?
- Action cost:
  - Road distance from *s* to *s'*
- Solution?

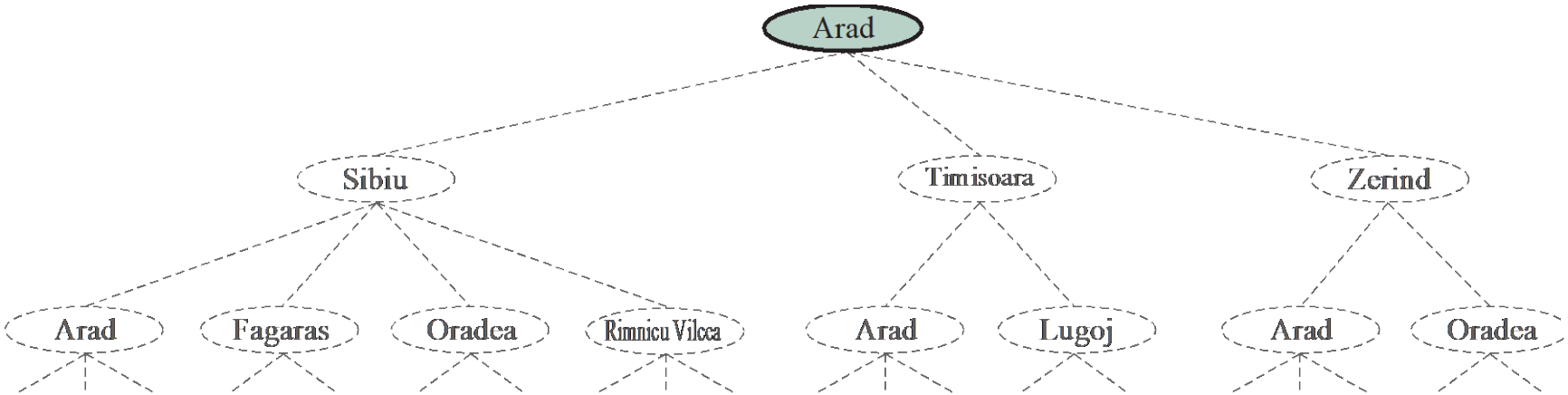
Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth



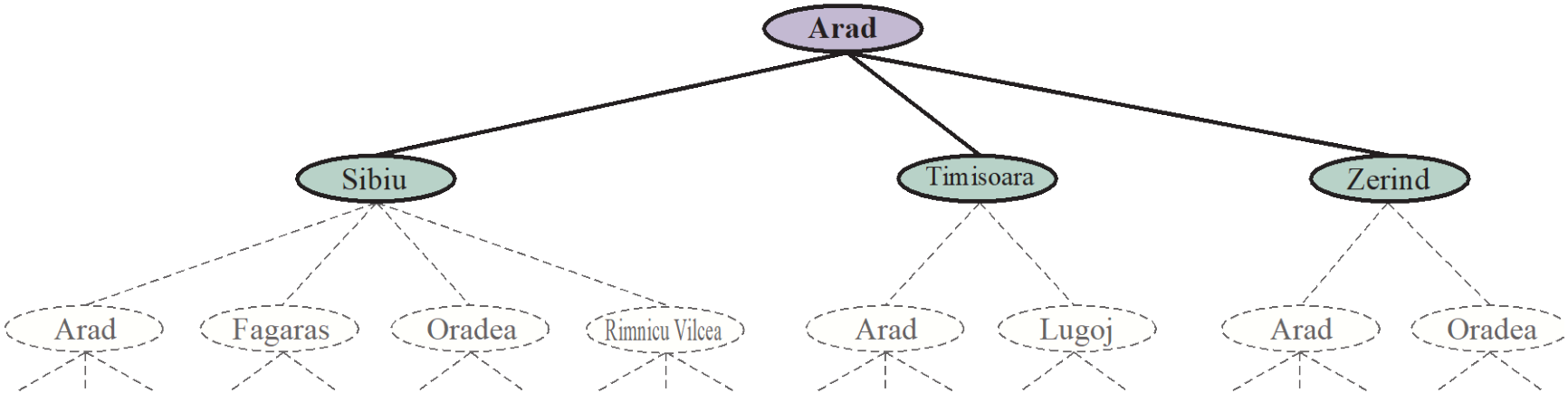
- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

Creating the search tree



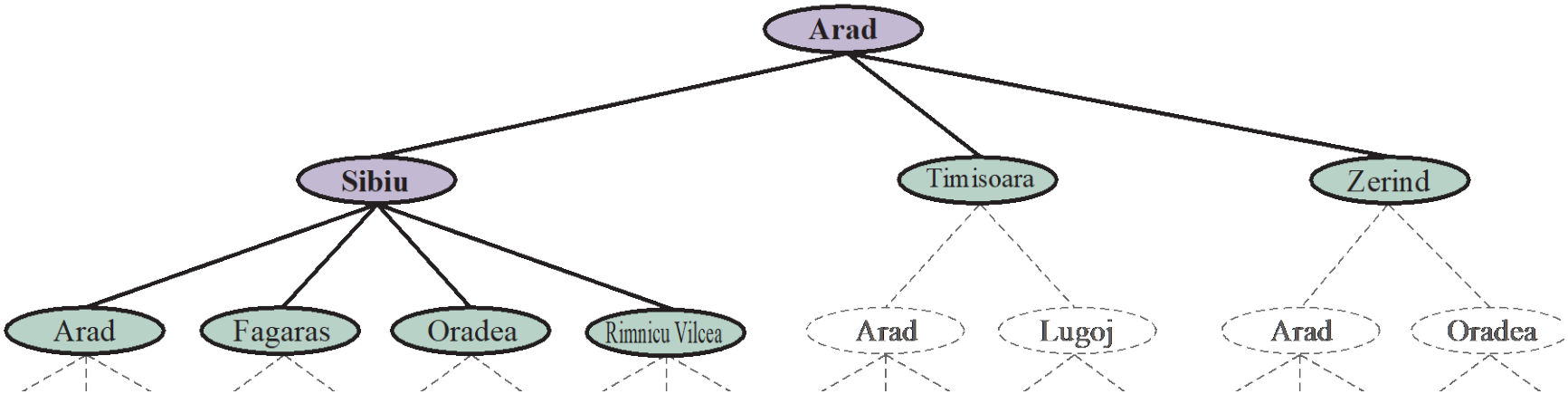
تابع Result را به حالت فعلی اعمال می‌کنیم تا مجموعه جدیدی از حالت‌ها (یعنی پسین‌ها) آشکار شود

Creating the search tree



اساس جستجو: فعلاً یک گزینه را دنبال کنید و چنانچه این گزینه منجر به جواب نشد، بقیه گزینه‌ها را بعداً در نظر بگیرید

Creating the search tree



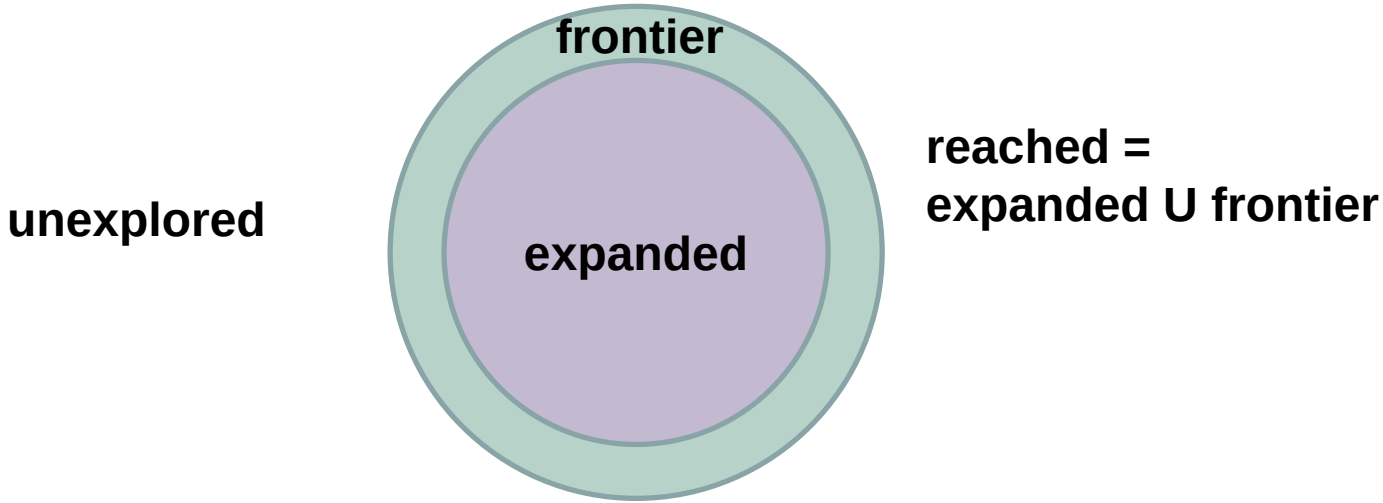
فرایند بسط و توسعه گره‌ها در مرز آنقدر ادامه می‌یابد تا به جوابی برسیم یا دیگر هیچ حالتی برای بسط وجود نداشته باشد

General Tree Search

```
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
    end
```

- **Main variations:**
  - Which leaf node to expand next
  - Whether to check for repeated states
  - Data structures for frontier, expanded nodes

Systematic search



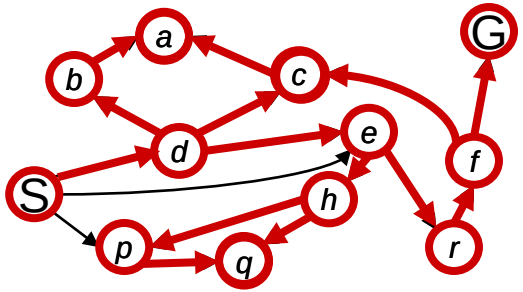
1. Frontier separates expanded from unexplored region of state-space graph
2. **Expanding a frontier node:**
   a. Moves a node from frontier into expanded
   b. Adds nodes from unexplored into frontier, maintaining property 1

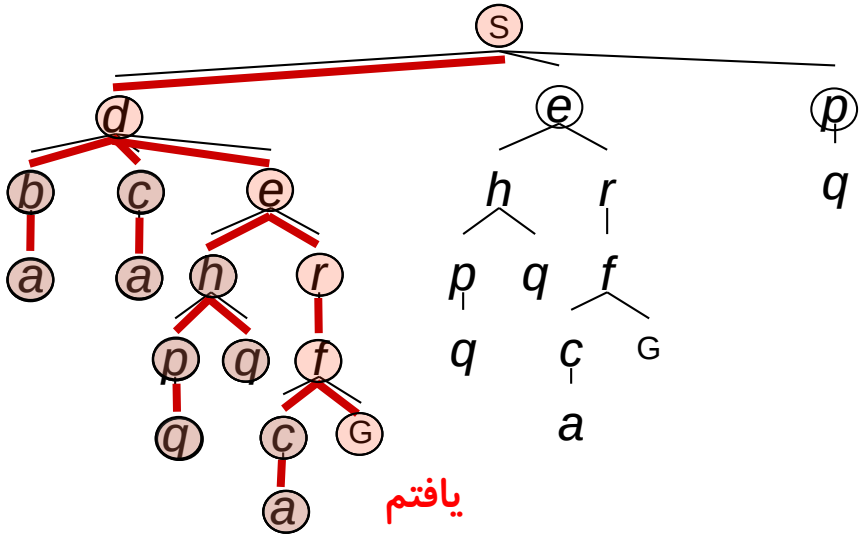
Depth-First Search

Depth-First Search

*Strategy: expand a **deepest** node **first***

*Implementation: Frontier is a **LIFO stack***

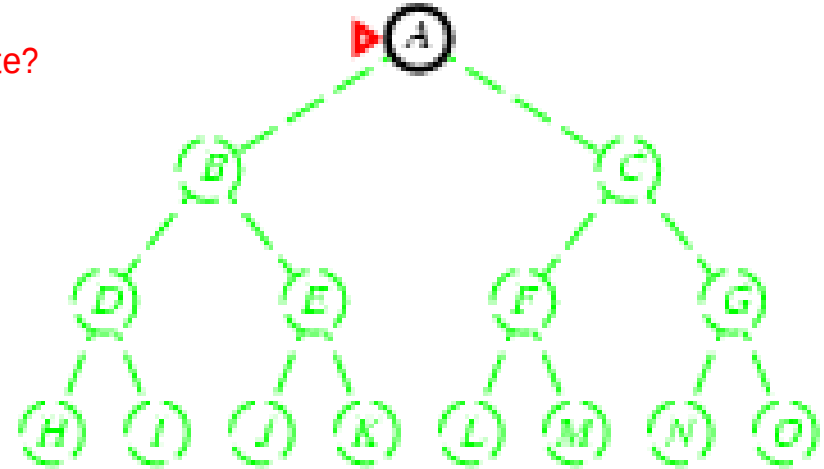




Depth-first search

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIFO) queue, i.e., put successors at front

Is A a goal state?

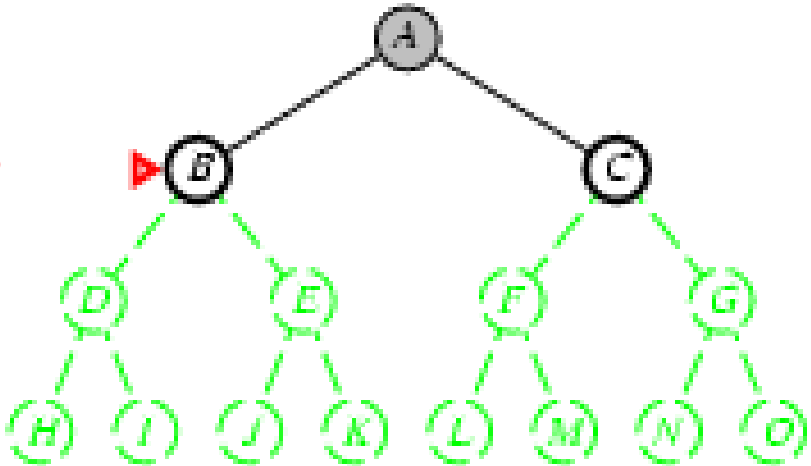


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[B,C]

Is B a goal state?

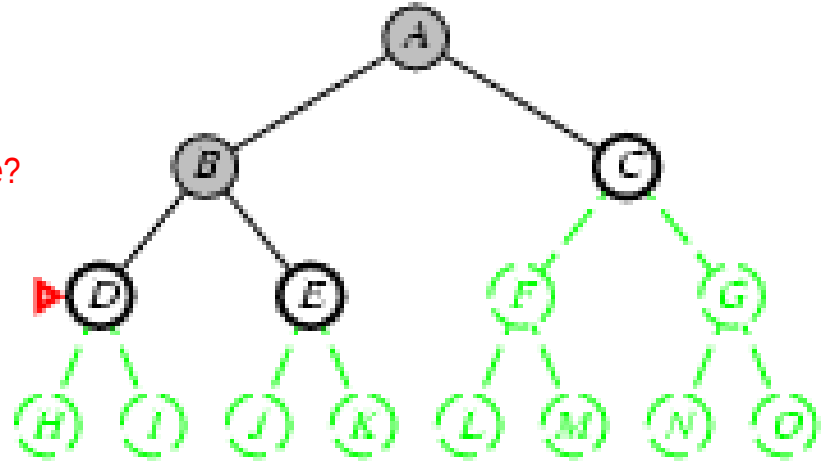


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[D,E,C]

Is D = goal state?

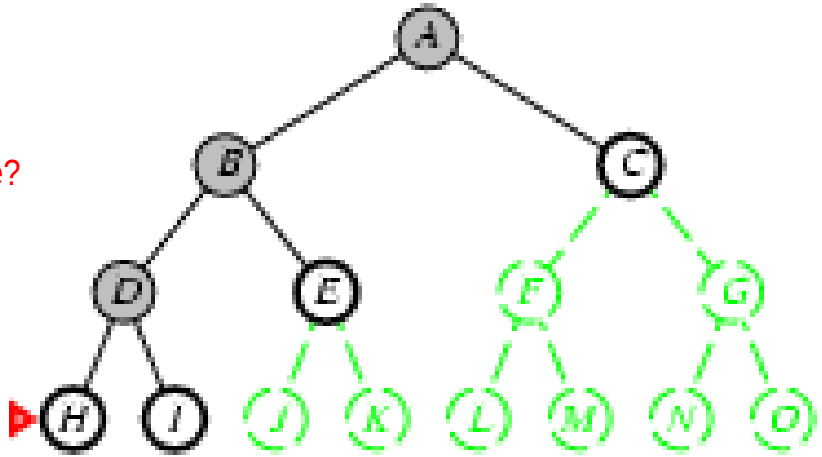


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[H,I,E,C]

Is H = goal state?

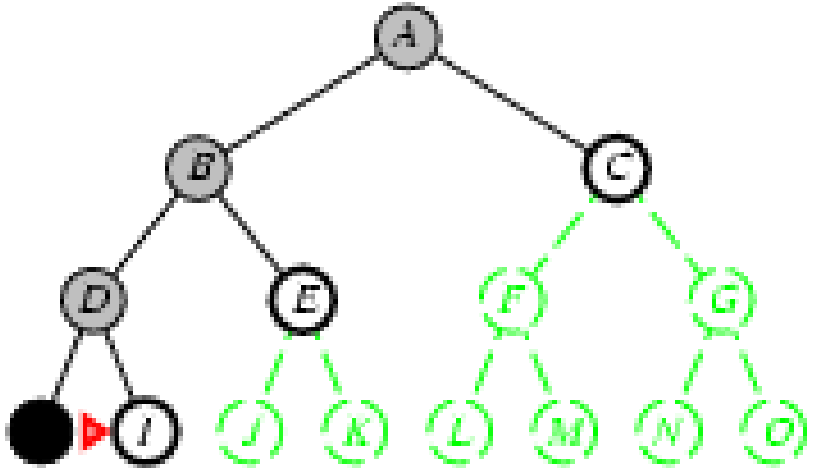


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[I,E,C]

Is I = goal state?

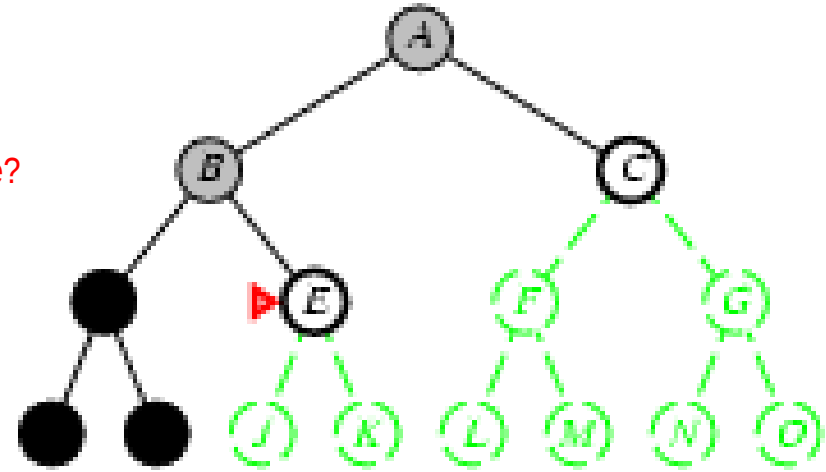


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[E,C]

Is E = goal state?

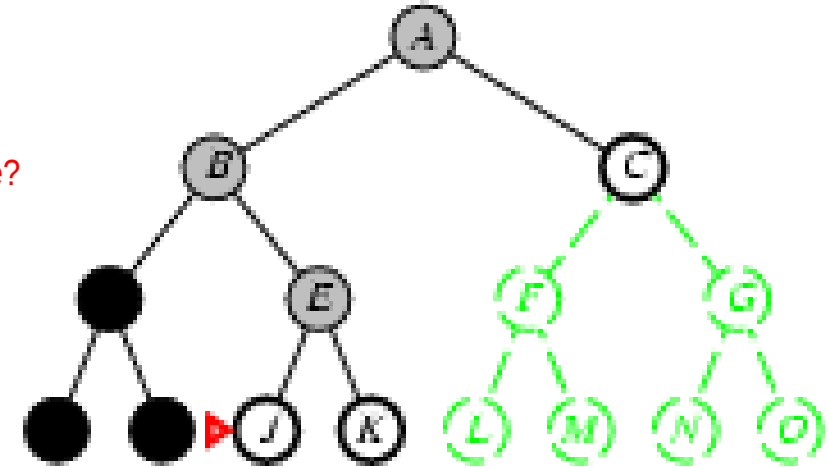


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[J,K,C]

Is J = goal state?

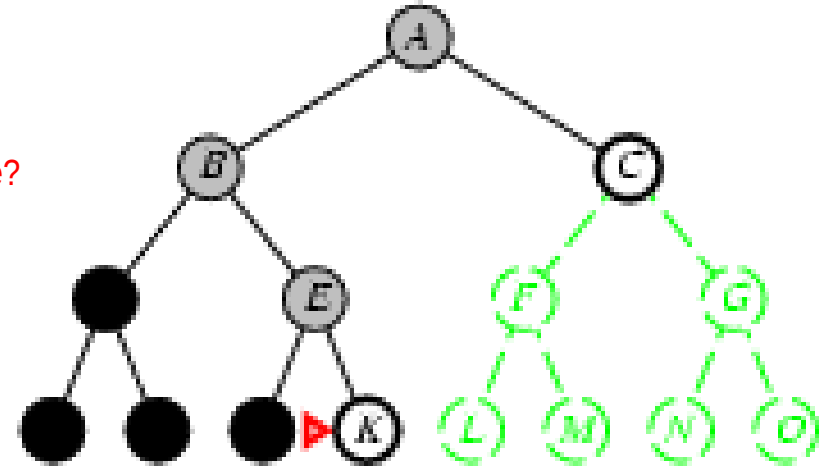


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[K,C]

Is K = goal state?

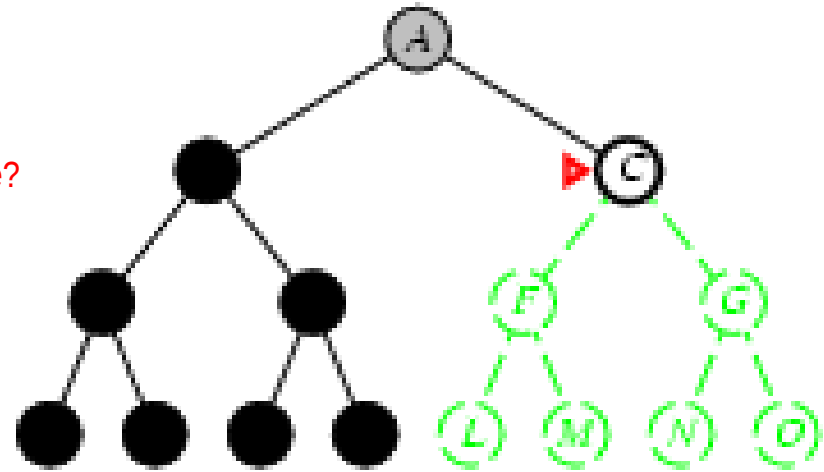


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[C]

Is C = goal state?

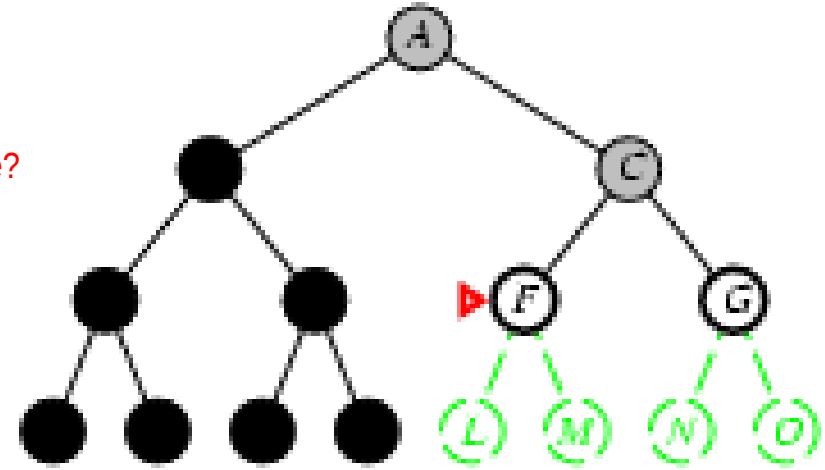


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[F,G]

Is F = goal state?

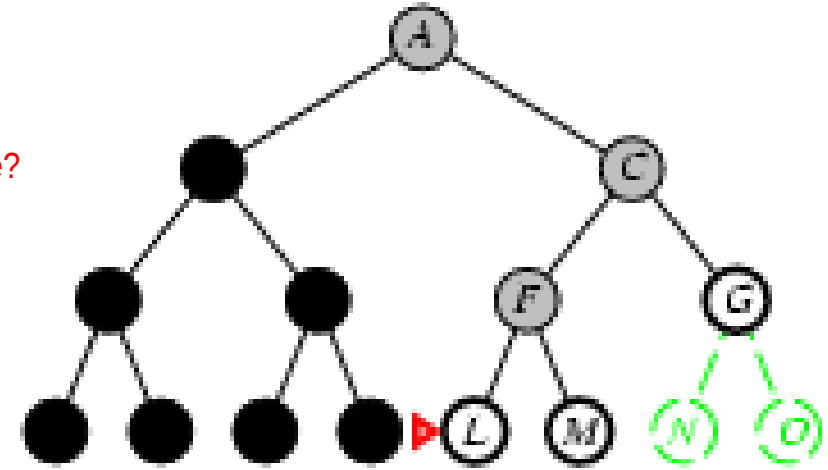


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[L,M,G]

Is L = goal state?

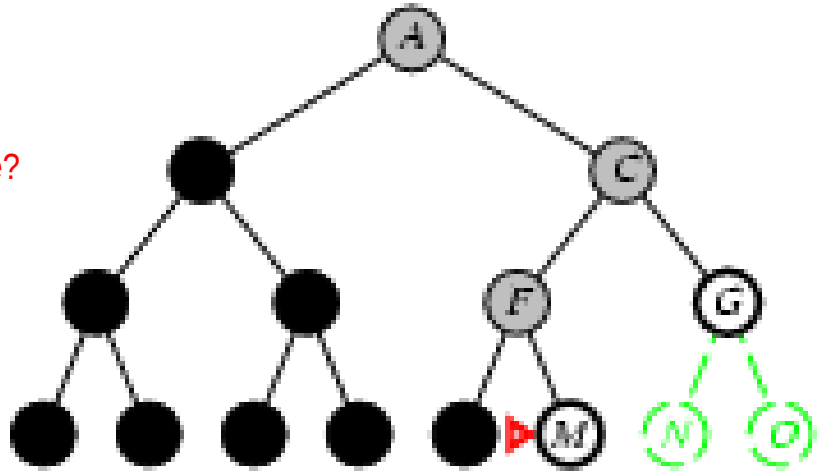


Depth-first search

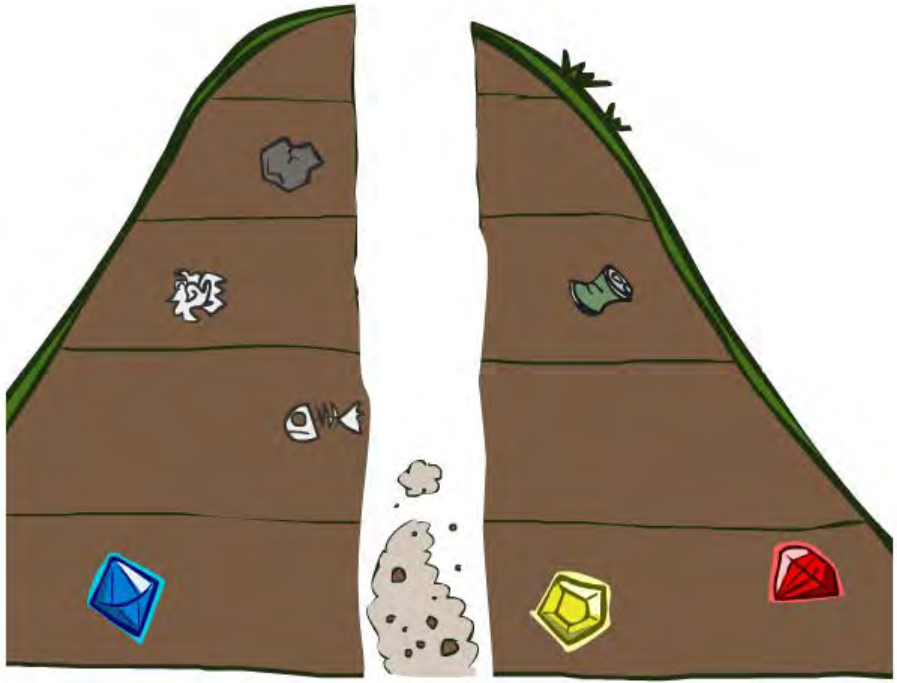
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?



Search Algorithm Properties

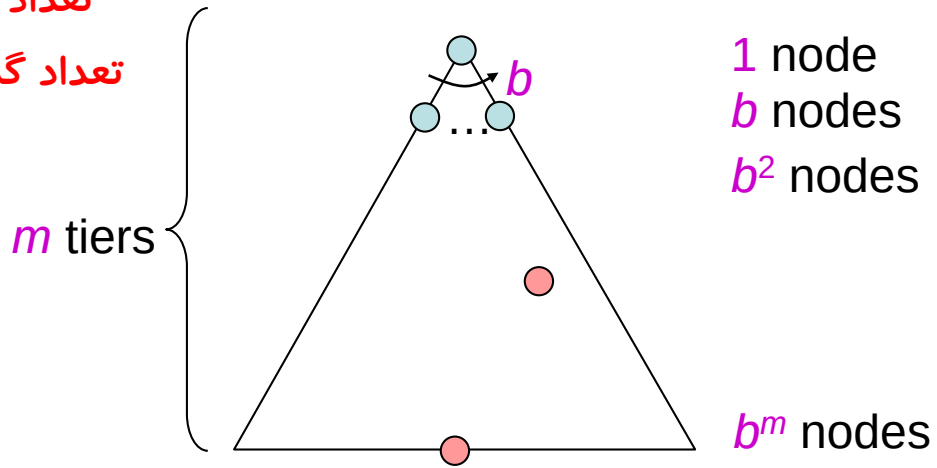


Search Algorithm Properties

- **Complete**: Guaranteed to find a solution if one exists?
- **Optimal**: Guaranteed to find the least cost path?
- **Time complexity?** تعداد گره‌های تولیدشده
- **Space complexity?** تعداد گره‌های ذخیره شده

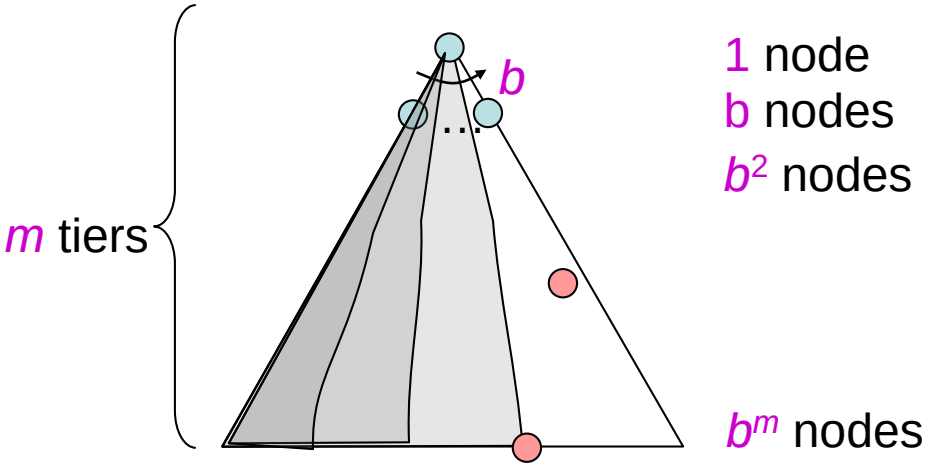
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

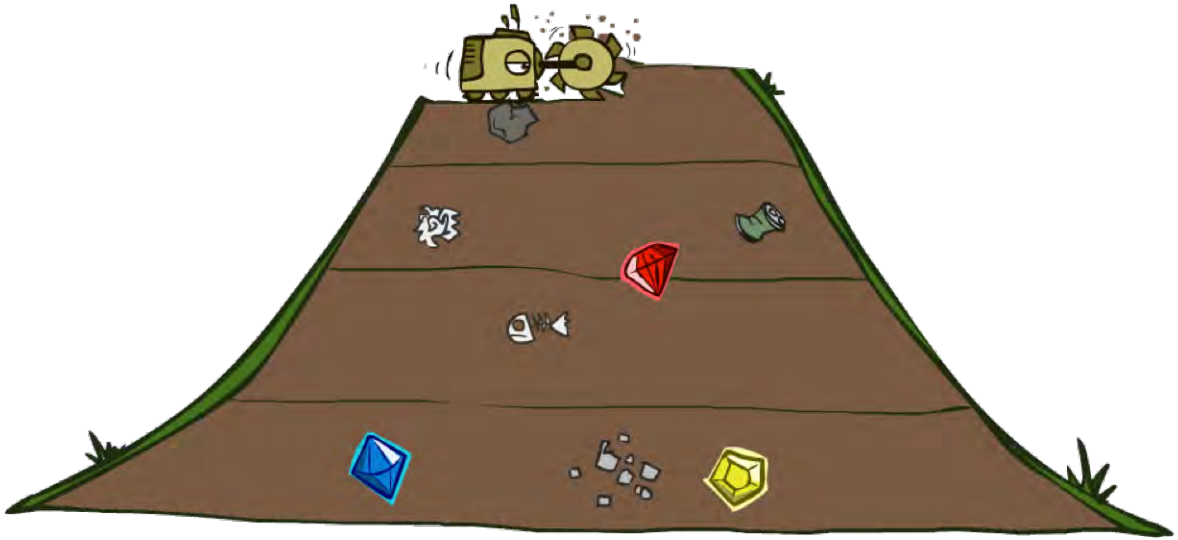


Depth-First Search (DFS) Properties

- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - *m* could be infinite
  - preventing cycles may help (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



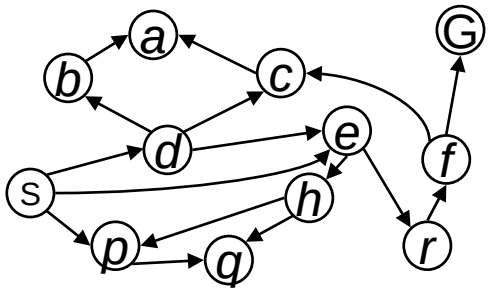
Breadth-First Search

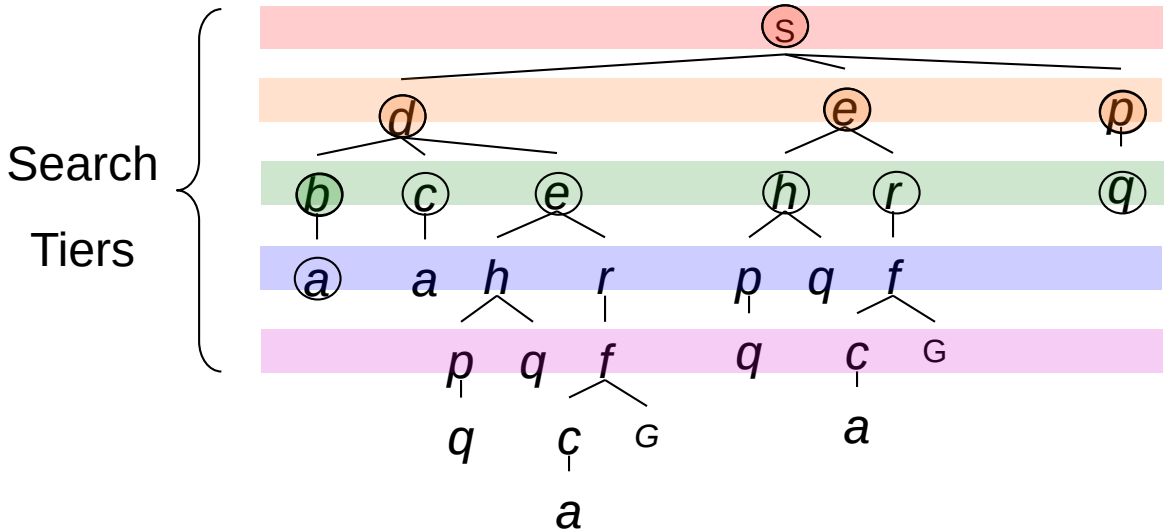


Breadth-First Search

*Strategy: expand a **shallowest** node **first***

*Implementation: Frontier is a **FIFO queue***



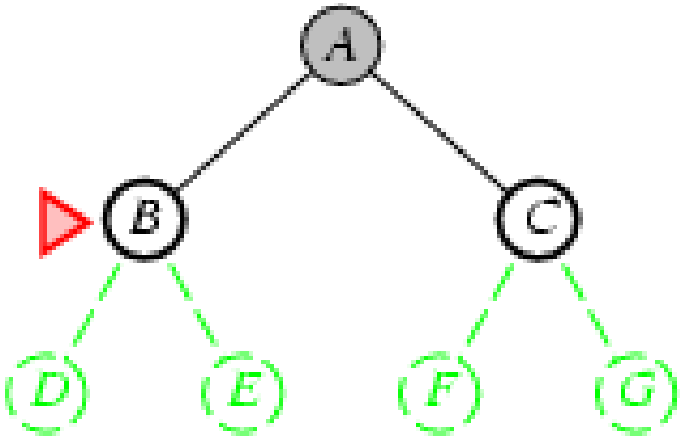


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe = [B,C]

Is B a goal state?

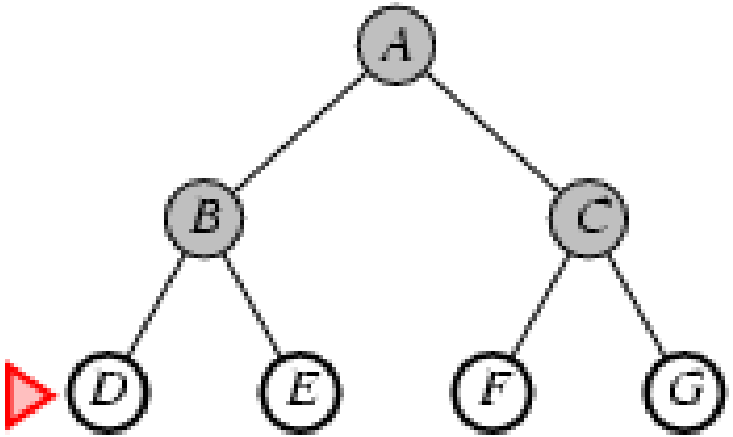


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

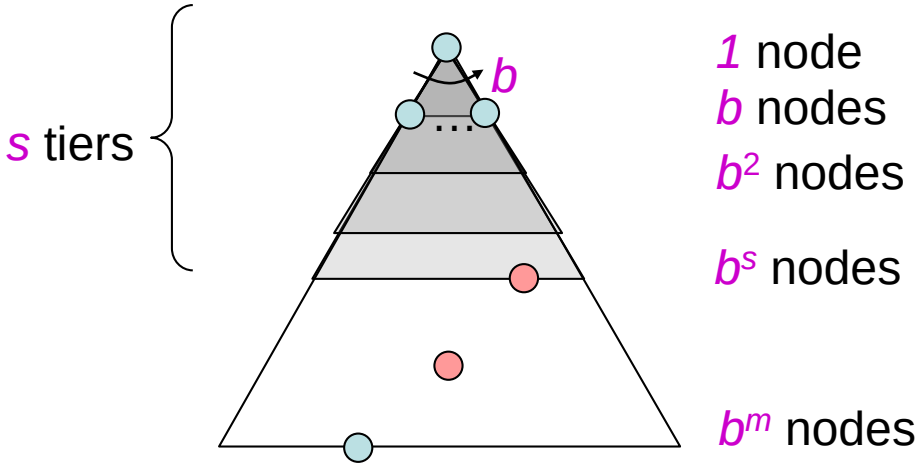
Expand:
fringe=[D,E,F,G]

Is D a goal state?



Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be *s*
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - *s* must be finite if a solution exists, so yes!
- Is it optimal?
  - If costs are equal (e.g., 1)



General Tree Search

```
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
    end
```

- **Main variations:**
  - Which leaf node to expand next
  - Whether to check for repeated states
  - Data structures for frontier, expanded nodes

DFS vs BFS

Quiz: DFS vs BFS

- When will BFS outperform DFS?
  - BFS best when there is a shallow solution, s << m
- When will DFS outperform BFS?
  - DFS best when solutions all at m, and fairly common at m

[Demo: dfs/bfs maze

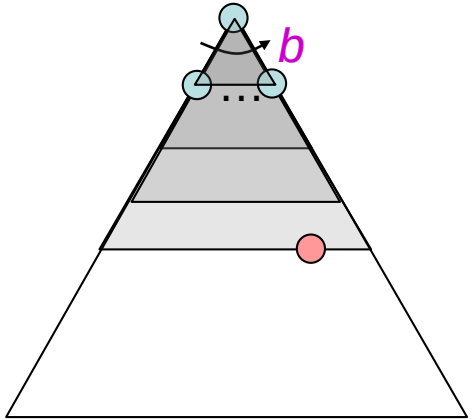
شکل ۳-۱۳ در کتابهای ۲۰۱۰ به قبل - (جستجوی عرضی)

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

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

Iterative Deepening

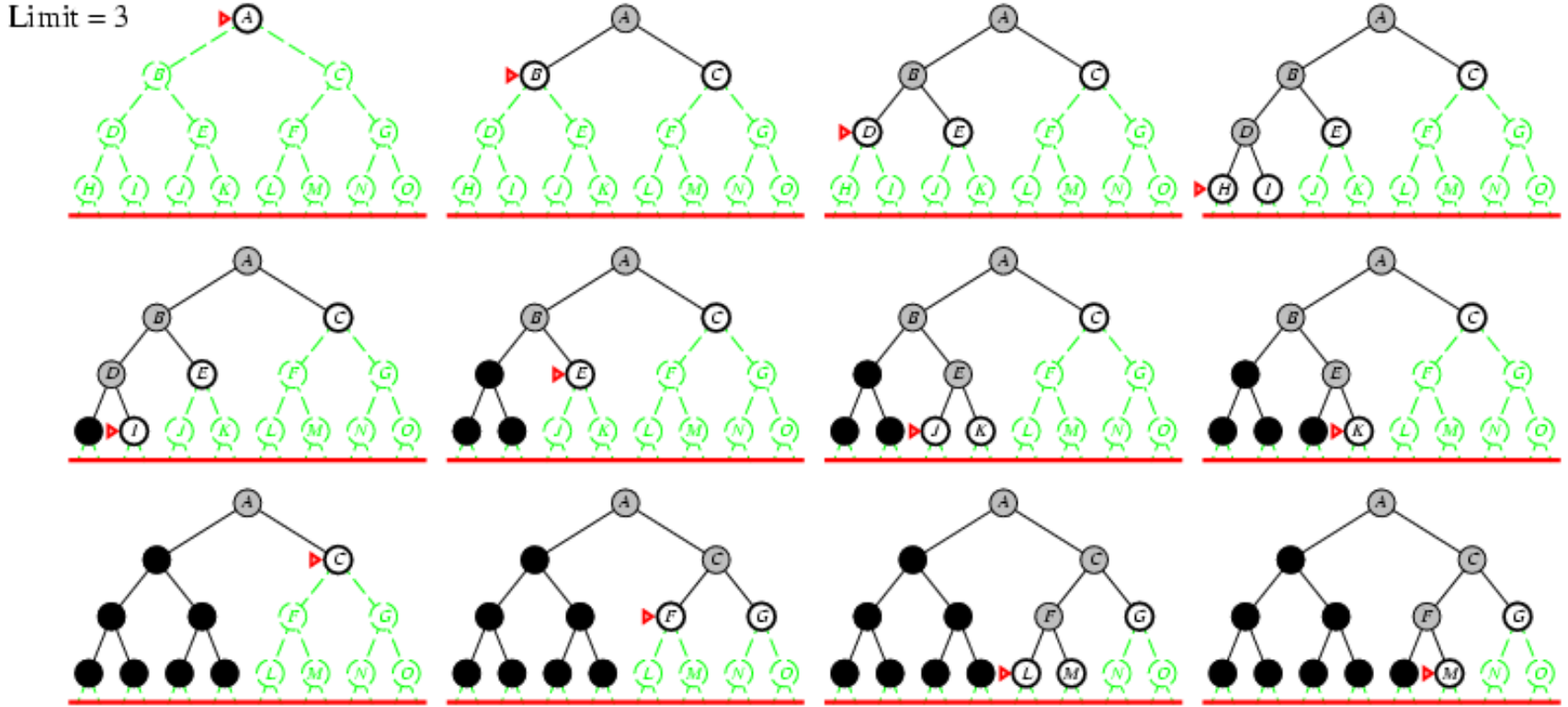
- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



Iterative deepening search *L*=1

Limit = 1

Iterative Deepening Search *L*=3

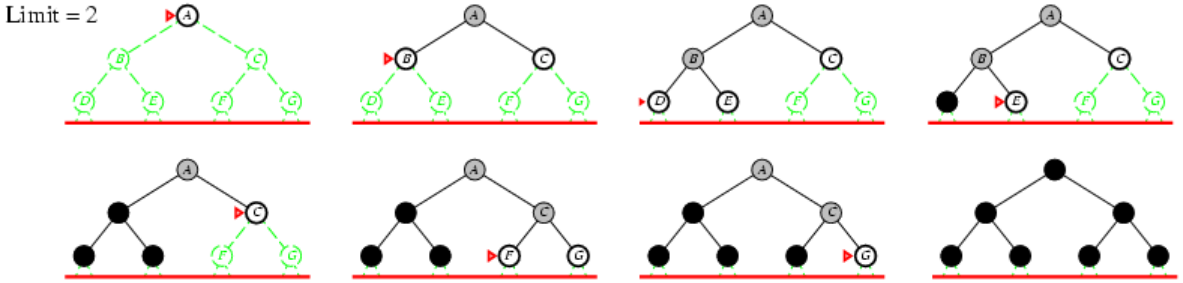


Iterative deepening search *L*=0

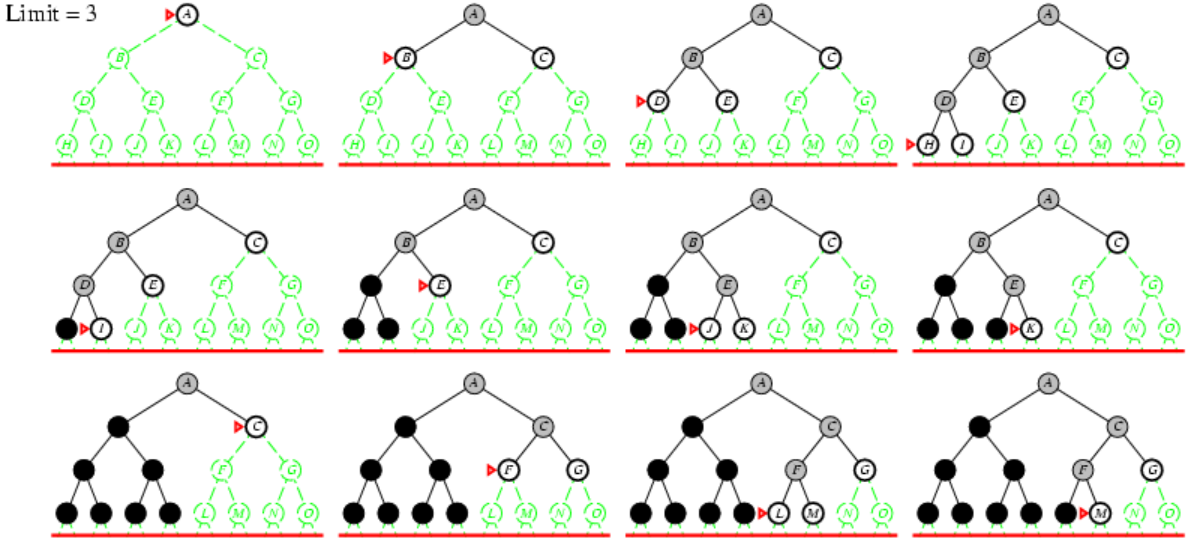
Limit = 0

Iterative deepening search *L*=1

Iterative deepening search *L*=2



Iterative Deepening Search *L*=3



Iterative deepening search

- 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*:

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

- For *b = 10, d = 5,*
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
  - $N_{BFS} = \ldots\ldots = 1{,}111{,}100$

$$O(b^d) \neq O(b^{d+1})$$

BFS

Note: BFS can also be adapted to be $O(b^d)$ by waiting to expand until all nodes at depth *d* are checked

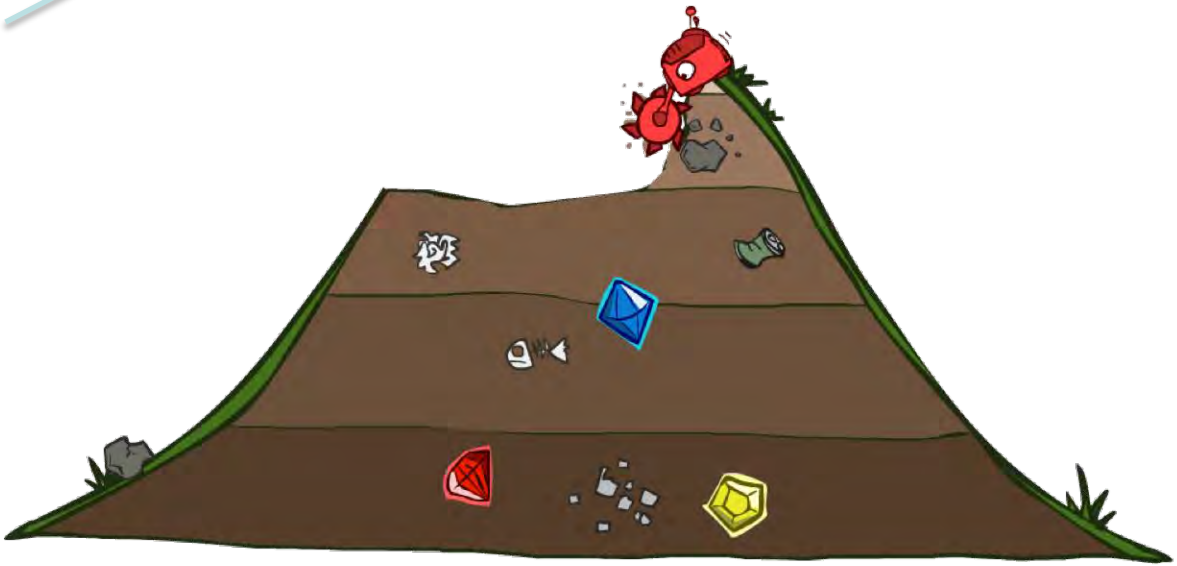
Uniform Cost Search

By the AI community

or

Dijkstra's algorithm

By the theoretical computer science community

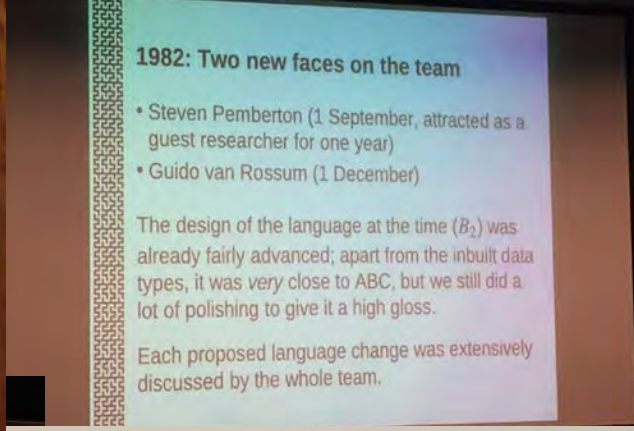


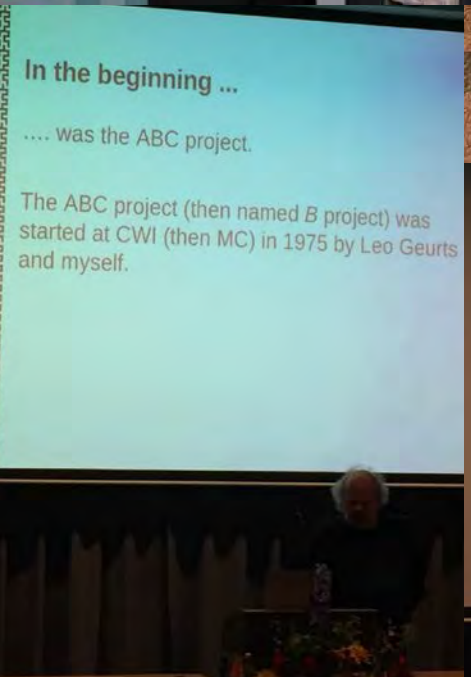
CWI

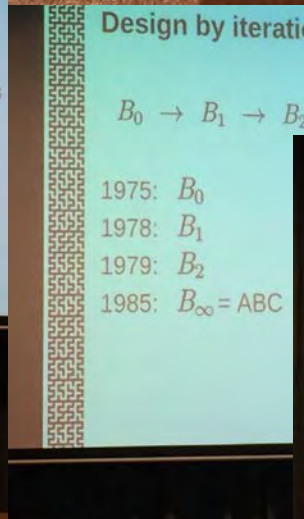
Programming language Python was made from this room in the early 1990s.

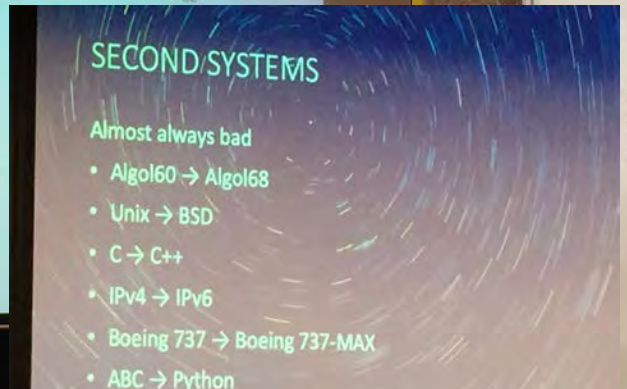
Programmeertaal Python is in deze kamer op het CWI ontstaan in de vroege jaren '90.

M353









CWI

The first open European Internet connection was made from this room at CWI on 17 November 1988.

De eerste open Europese internetverbinding werd vanuit deze ruimte op het CWI gelegd op 17 november 1988.

M382

Uniform-cost search

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant). Can we guarantee optimality for any step cost?

Uniform-cost Search: Expand node with smallest path cost g(n).

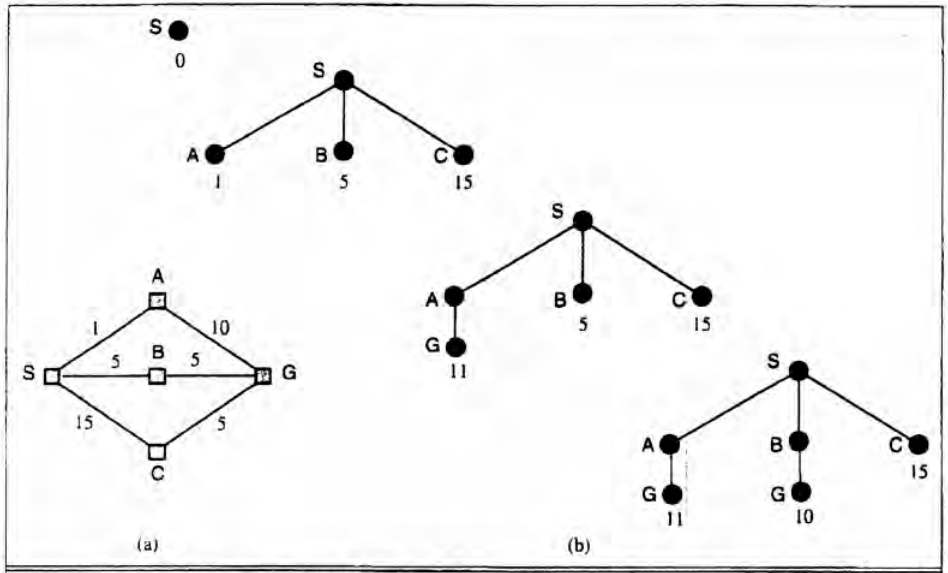


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Proof Completeness:

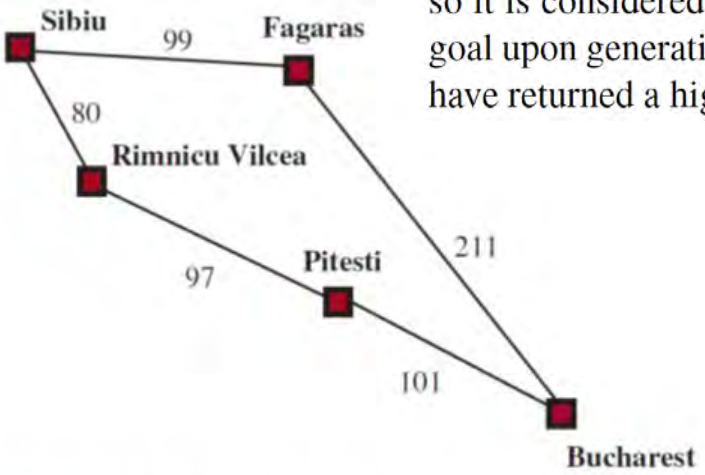
Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach it.

Proof of optimality given completeness:

Assume UCS is not optimal.
Then there must be an (optimal) goal state with path cost smaller than the found (suboptimal) goal state (invoking completeness).
However, this is impossible because UCS would have expanded that node first by definition.
Contradiction.

Consider Figure 3.10, where the problem is to get from Sibiu to Bucharest. The successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively. The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost $80 + 97 = 177$. The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost $99 + 211 = 310$. Bucharest is the goal, but the algorithm tests for goals only when it expands a node, not when it generates a node, so it has not yet detected that this is a path to the goal.

The algorithm continues on, choosing Pitesti for expansion next and adding a second path to Bucharest with cost $80 + 97 + 101 = 278$. It has a lower cost, so it replaces the previous path in *reached* and is added to the *frontier*. It turns out this node now has the lowest cost, so it is considered next, found to be a goal, and returned. Note that if we had checked for a goal upon generating a node rather than when expanding the lowest-cost node, then we would have returned a higher-cost path (the one through Fagaras).



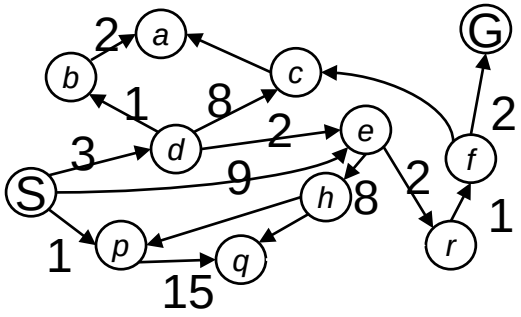
جستجوی عرضی بلافاصله بعد از تولید هدف متوقف می‌شود در حالی که جستجو با هزینه یکنوا تمام گره‌های موجود در عمق هدف را بررسی می‌کند تا گره‌ای با هزینه کمتر بیابد؛ یعنی با توسعه غیرضروری در عمق d کار بیشتری انجام می‌دهد.

Figure 3.10 Part of the Romania state space, selected to illustrate uniform-cost search.

Uniform Cost Search

***g(n)* = cost from root to n**

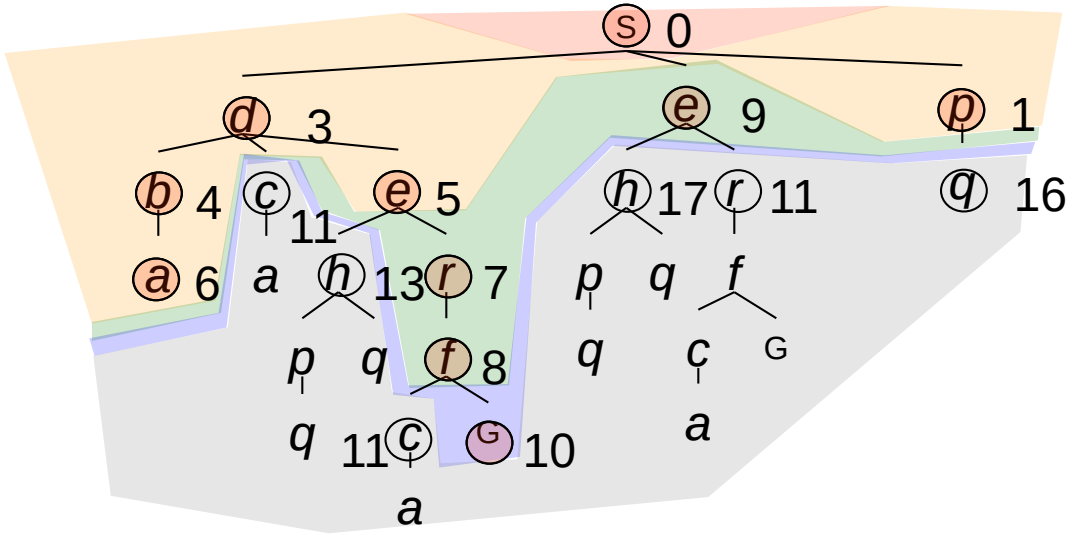
Strategy: expand ***lowest g(n)***



گره‌ای که کمترین هزینه در مسیر، یعنی کمترین g(n) را دارد توسعه می‌یابد

Frontier is a ***priority queue sorted by g(n)***

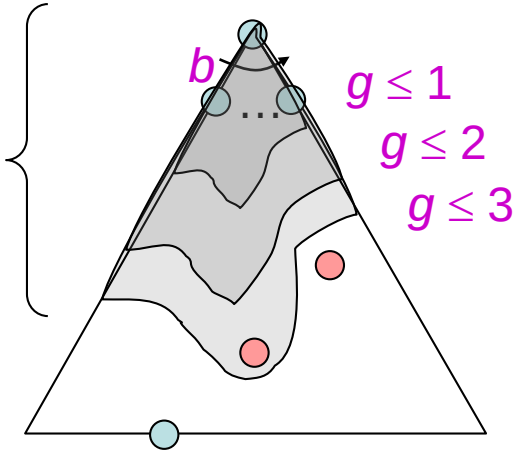
Cost contours



Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)

C^*/ε
"tiers"



- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming C^* is finite and $\varepsilon > 0$, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

The complexity of uniform-cost search is characterized in terms of C^*, the cost of the optimal solution,[8] and ϵ, a lower bound on the cost of each action, with $\epsilon > 0$. Then the algorithm's worst-case time and space complexity is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, which can be much greater than b^d. This is because uniform-cost search can explore large trees of actions with low costs before exploring paths involving a high-cost and perhaps useful action. When all action costs are equal, $b^{1+\lfloor C^*/\epsilon \rfloor}$ is just b^{d+1}, and uniform-cost search is similar to breadth-first search.

Uniform-cost search is complete and is cost-optimal, because the first solution it finds will have a cost that is at least as low as the cost of any other node in the frontier. Uniform-cost search considers all paths systematically in order of increasing cost, never getting caught going down a single infinite path (assuming that all action costs are $> \epsilon > 0$).

جستجوی عرضی بلافاصله بعد از تولید هدف متوقف می‌شود در حالی که جستجو با هزینه یکنوا تمام گره‌های موجود در عمق هدف را بررسی می‌کند تا گره‌ای با هزینه کمتر بیابد؛ یعنی با توسعه غیرضروری در عمق d کار بیشتری انجام می‌دهد.

Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost ≥ ε
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* ≤ cost of optimal solution.

Space? # of nodes with path cost ≤ cost of optimal solution.

Optimal? Yes, for any step cost ≥ ε

Summary of algorithms

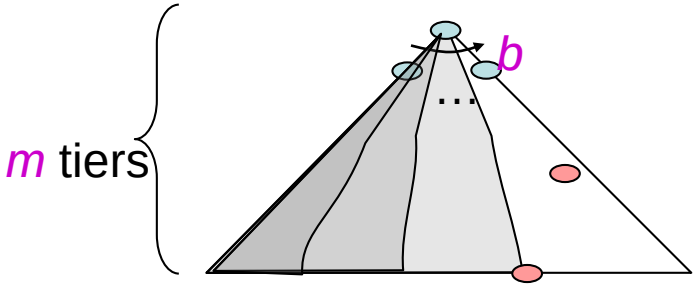
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$

even complete if step cost is not increasing with depth.

preferred uninformed search strategy

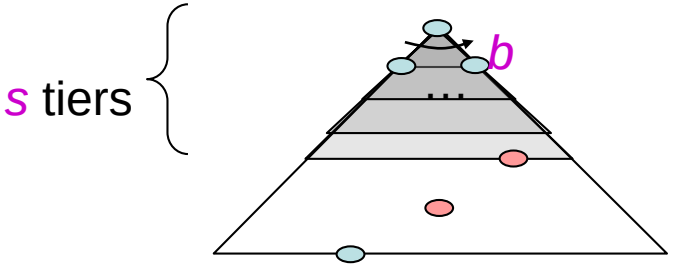
Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Uninformed Search: DFS vs BFS vs UCS vs IDS vs BiS



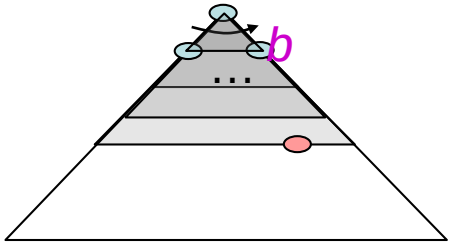
1 node
b nodes
b^2 nodes

b^m nodes

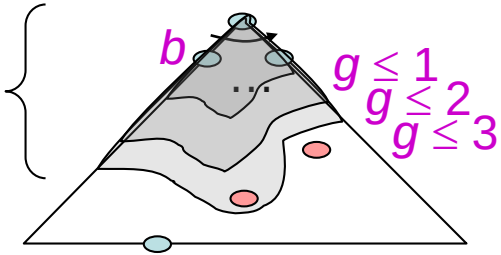


1 node
b nodes
b^2 nodes
b^s nodes

b^m nodes



C^*/ε
"tiers"



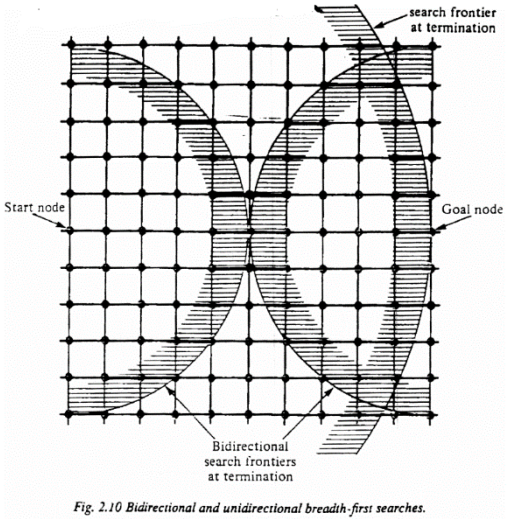


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both “meet in the middle”
  - need to keep track of the intersection of 2 open sets of nodes
- What does searching backwards from G mean
  - need a way to specify the predecessors of G
    - this can be difficult,
    - e.g., predecessors of checkmate in chess?
  - which to take if there are multiple goal states?
  - where to start if there is only a goal test, no explicit list?

Bi-Directional Search

Complexity: time and space complexity are: $O(b^{d/2})$

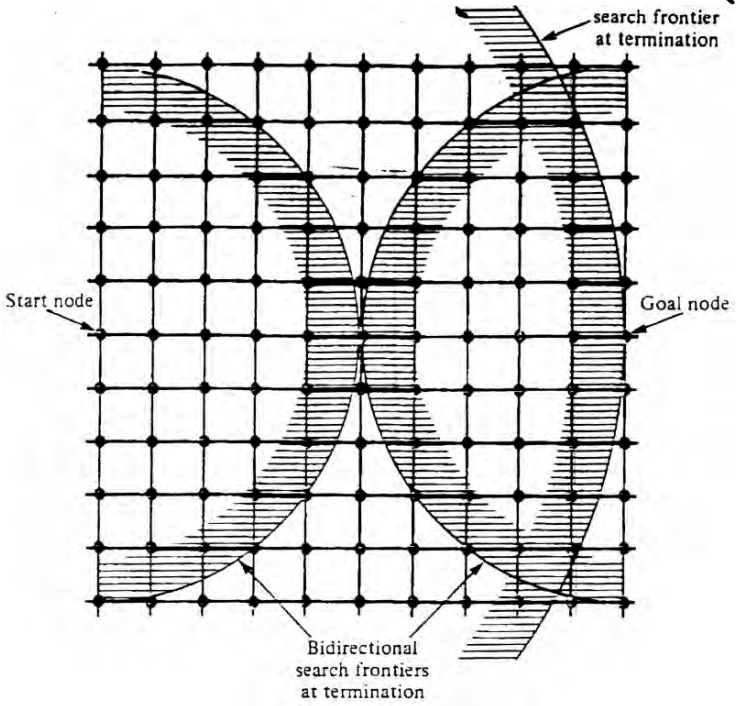


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

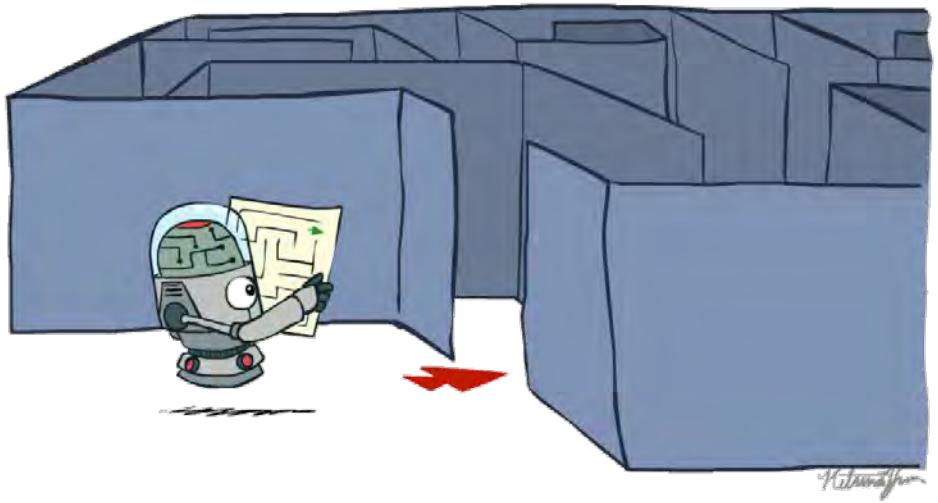
A Take-Home Quiz!

- شکل ۳-۱۴ شبه‌کد جستجوی دوطرفه را با استفاده از انواع استراتژی‌های جستجوی ناآگاهانه توسعه دهید و برای مسئله سفر از آراد به بخارست در کشور رومانی اجرا نموده، دموها و نتایج آن را تهیه نمایید.

- Demos
  - http://www.cs.rmit.edu.au/AI-Search/Product/
  - http://aima.cs.berkeley.edu/demos.html (for more demos)

CS 188: Artificial Intelligence

Search

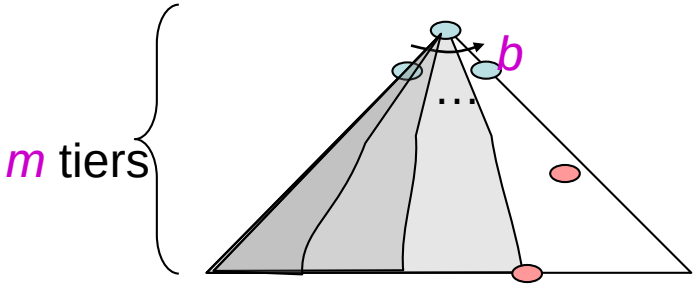


Instructors: Stuart Russell and Dawn Song

University of California, Berkeley

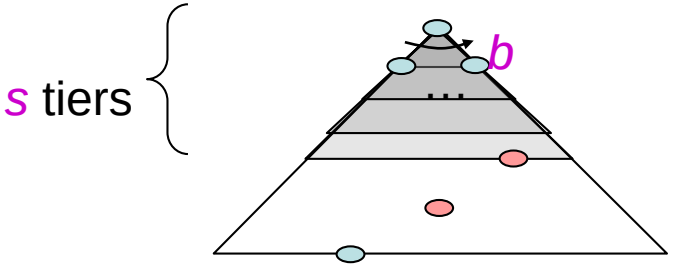
[slides adapted from Dan Klein, Pieter Abbeel]

Uninformed Search: DFS vs BFS vs UCS vs IDS vs BiS



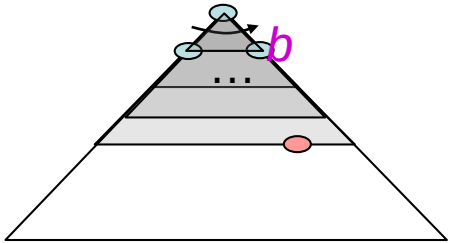
1 node
b nodes
b^2 nodes

b^m nodes

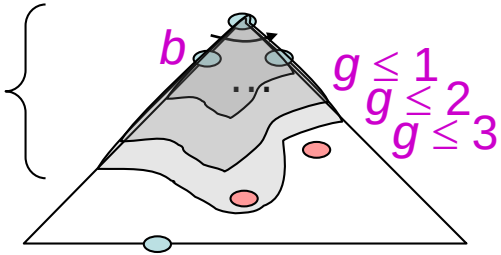


1 node
b nodes
b^2 nodes
b^s nodes

b^m nodes



C^*/ε
"tiers"



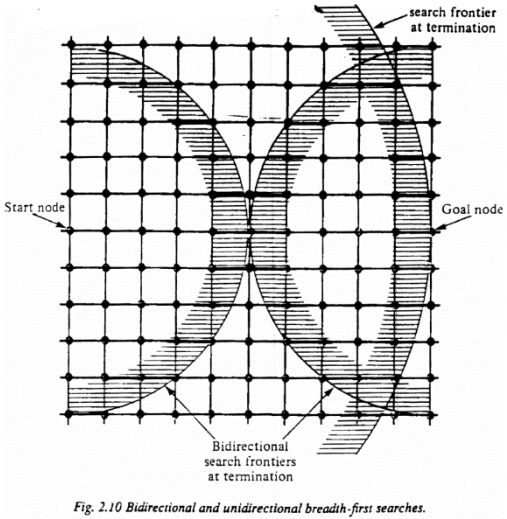


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

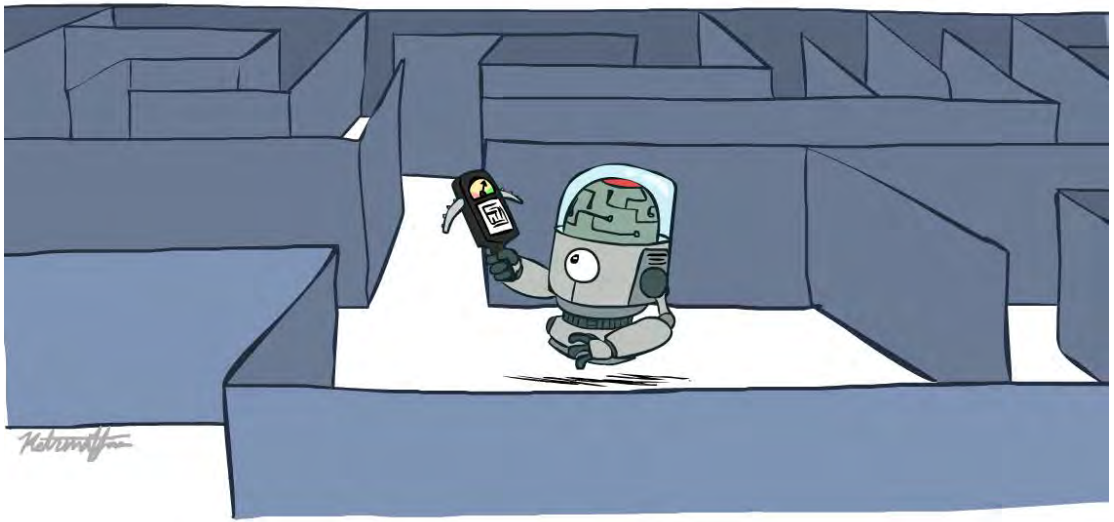
Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

CS 188: Artificial Intelligence

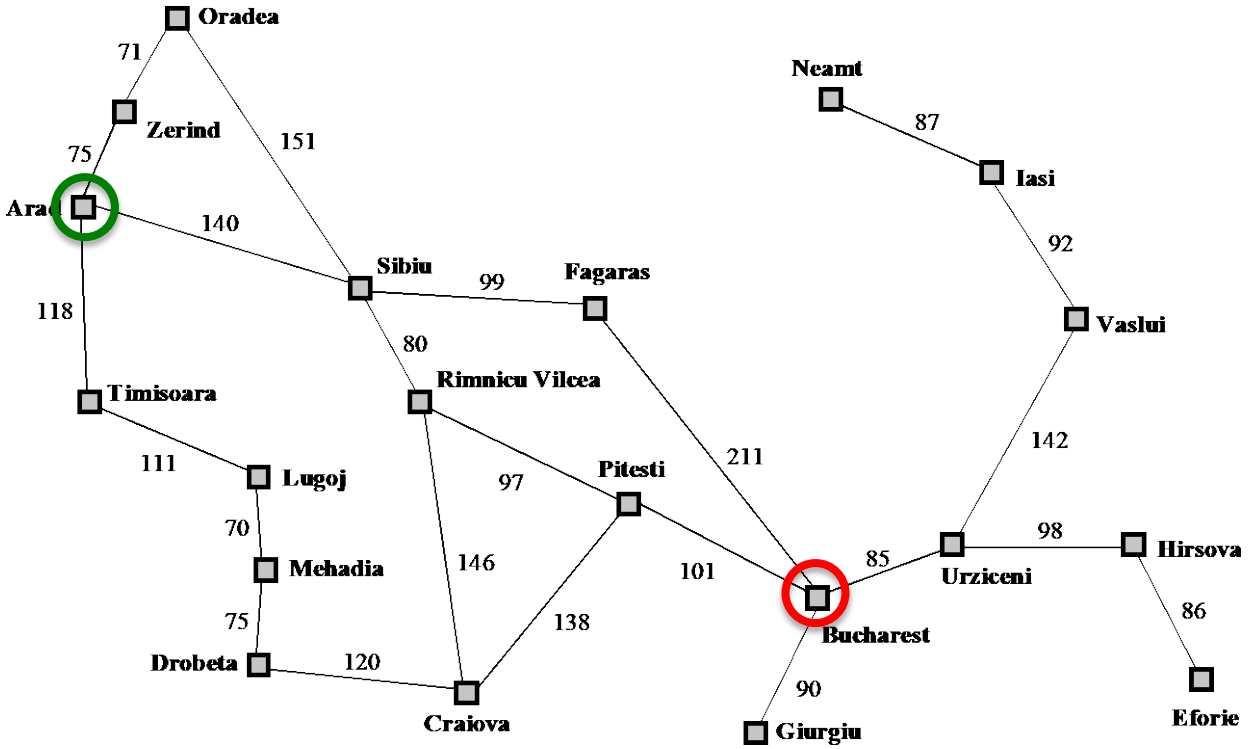
Informed Search



Instructors: Stuart Russell and Dawn Song

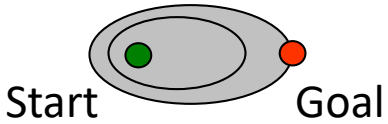
University of California, Berkeley

Example: route-finding in Romania

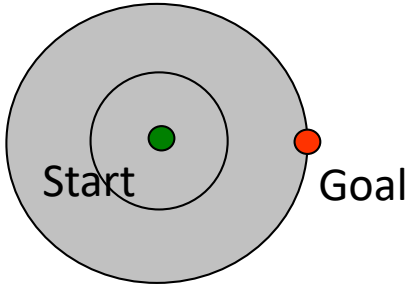


What we would like to have happen

Guide search ***towards the goal*** instead of ***all over the place***



Informed



Uninformed

Uninformed vs Informed Search

- استراتژی‌های جستجوی ناآگاهانه هیچ اطلاعی در مورد تعداد مراحل یا هزینه مسیر از حالت جاری به سوی هدف ندارند.
- و تمام چیزی که می‌توانند تشخیص دهند تشخیص حالت هدف از حالت غیر هدف است.
- استراتژی‌هایی که مفروضات مذکور را در نظر می‌گیرند، استراتژی‌های جستجوی آگاهانه یا مکاشفه‌ای نامیده می‌شوند.
- یعنی از دانش ویژه مسئله استفاده می‌کنند و می‌توانند راه‌حل‌های کاراتری پیدا کنند.
- تنها جایی که این دانش می‌تواند به‌کار برده شود در تابع مورد استفاده در صف (لبه، مرز) است که گره بعدی را برای بسط دادن تعیین می‌کند.

DFS vs BFS vs UCS

vs

A* Search, an informed search

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - f(n) provides an estimate for the total cost.
  - →Expand the node n with smallest f(n).

تابع ارزیابی، دانشی مبنی بر مطلوب بودن یا نبودن بسط گره ارائه می‌دهد.

گره‌ای که بهترین ارزیابی را داشته باشد ابتدا بسط داده می‌شود.

- Implementation:

  Order the nodes in fringe increasing order of cost.

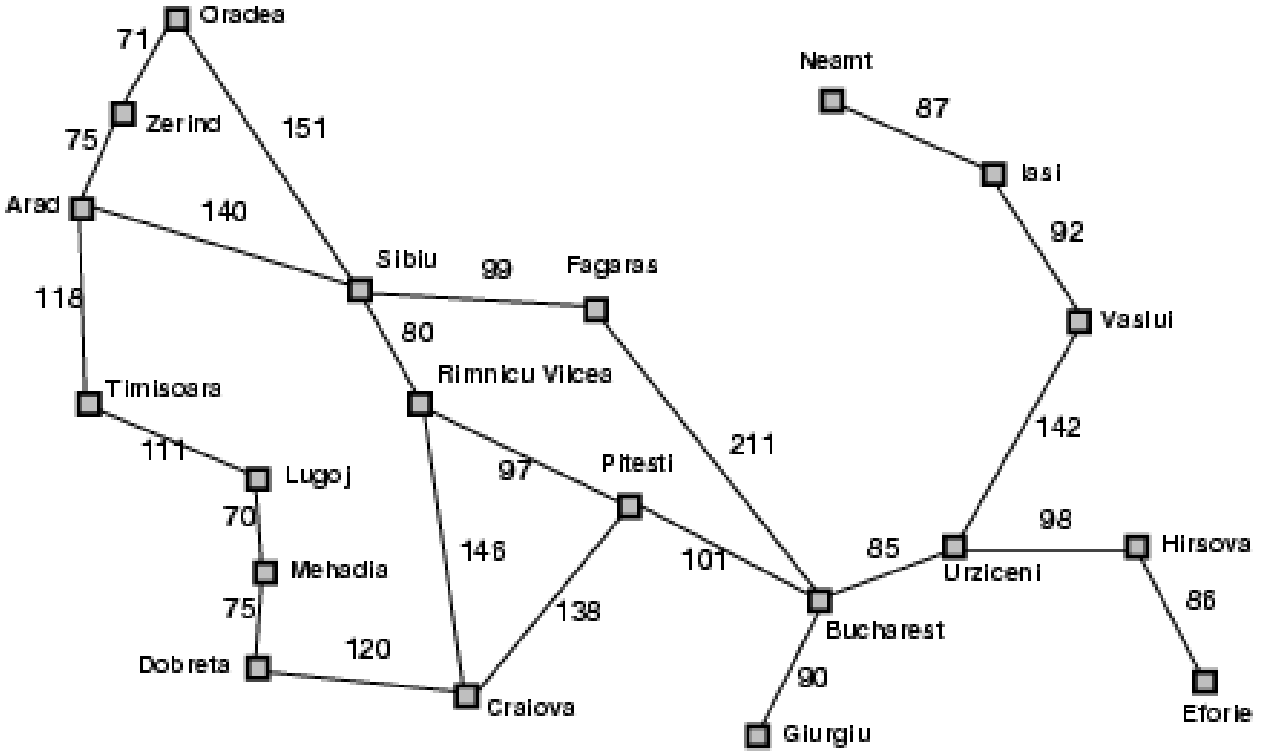
صرفاً استفاده از معیار هزینه مسیر (یعنی g) برای آنکه تصمیم بگیرد کدام مسیر گسترش یابد، جستجو را به سمت هدف هدایت نمی‌کند

بلکه باید تخمین‌هایی از هزینه مسیر از یک حالت به نزدیک‌ترین حالت هدف داشته باشد.

- Special cases:
  - greedy best-first search
  - A* search

دو راهکار:

Romania with straight-line dist.



Straight–line distance to Bucharest

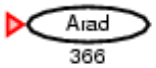
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

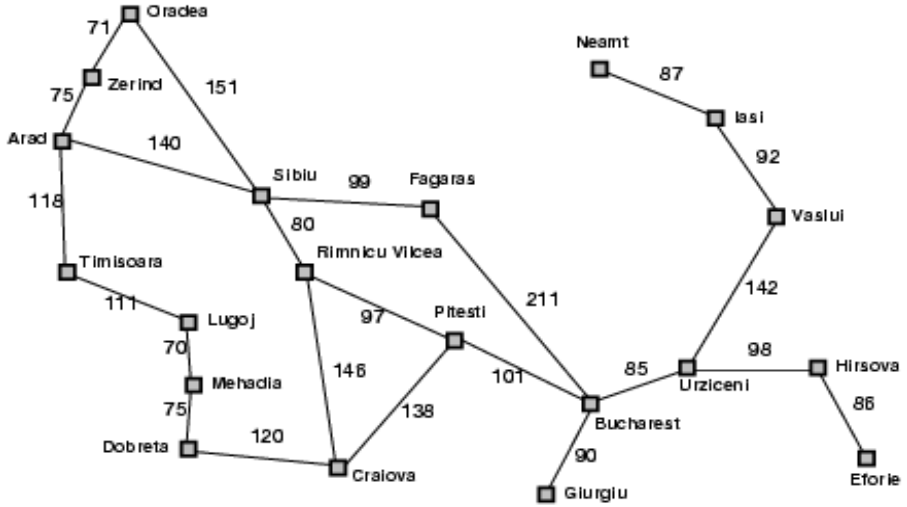
Greedy best-first search

- f(n) = estimate of cost from *n* to *goal*
- e.g., *f(n)* = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal.

- جستجوی حریصانه برای f از تابع h برای یافتن گره بعدی برای بسط استفاده می‌کند
- h(n) : هزینه تخمین زده شده از ارزان‌ترین مسیر از حالتی در گره n به یک حالت هدف

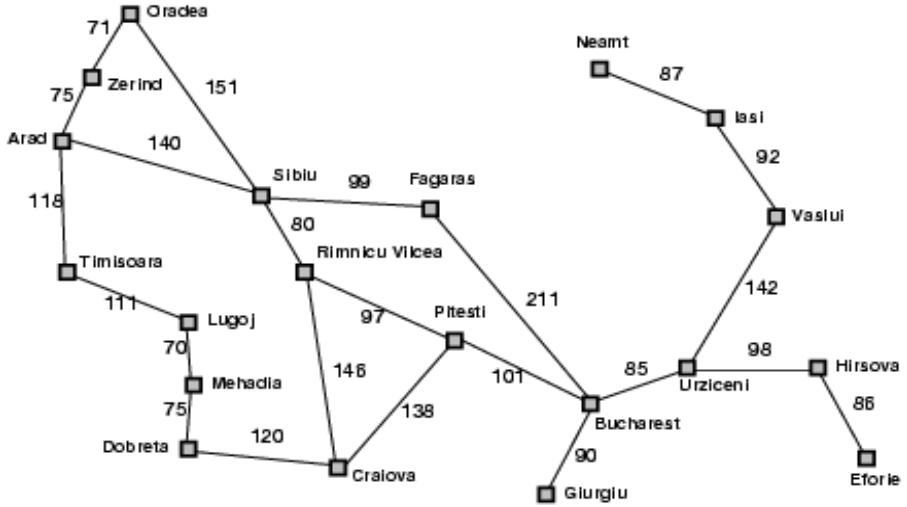
Greedy best-first search example





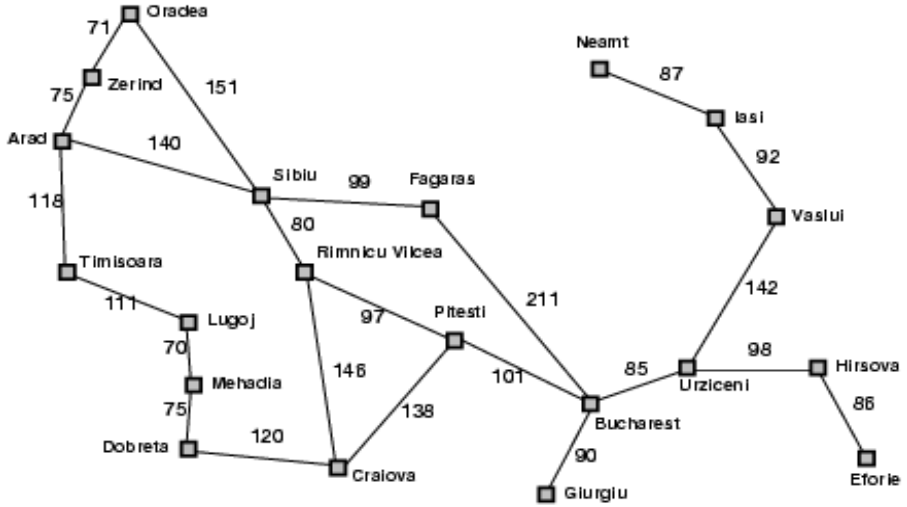
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

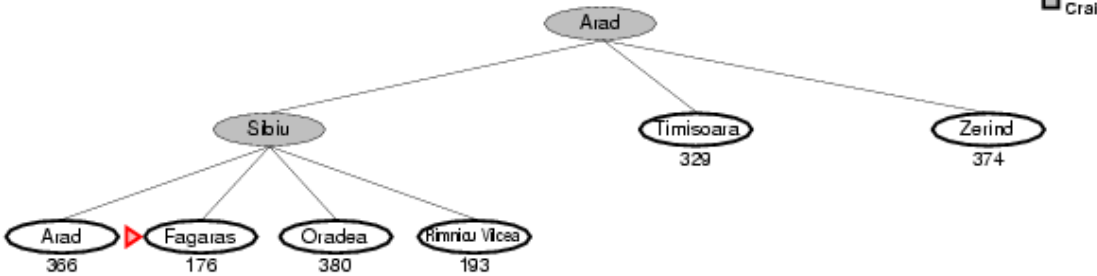


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

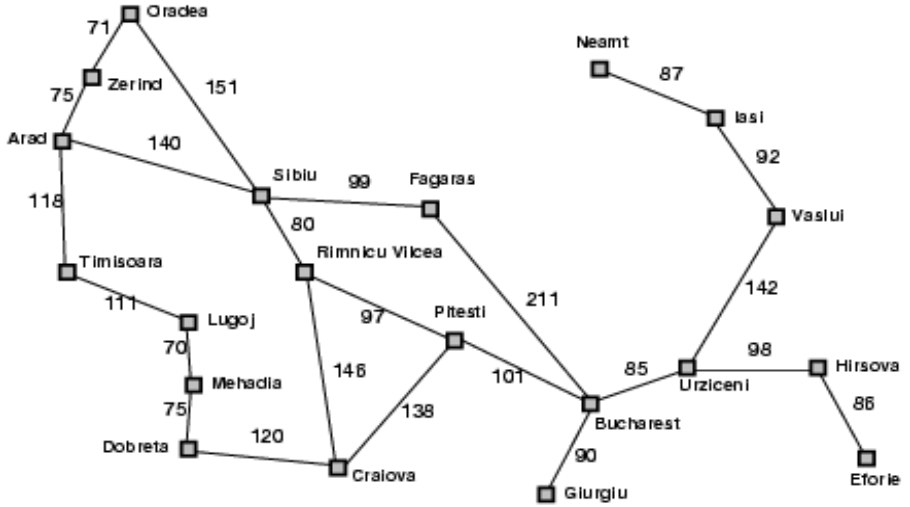
Greedy best-first search example



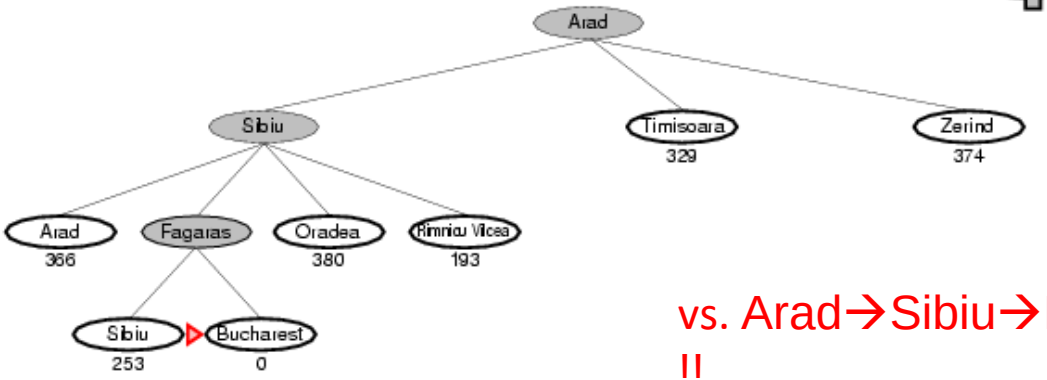
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Greedy best-first search example

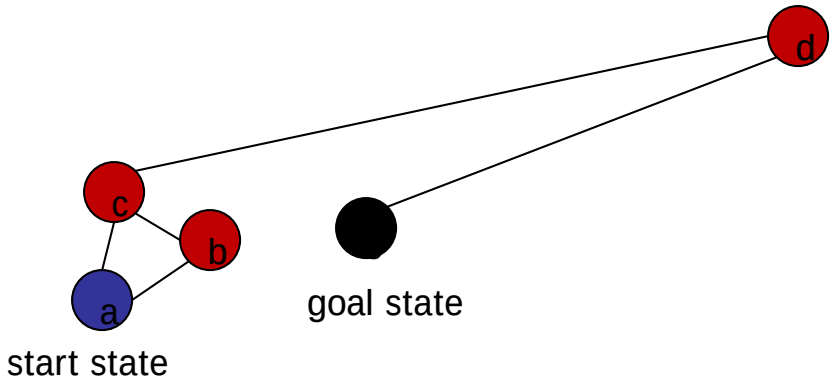


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



vs. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest !!

GBFS is not complete



f(n) = straightline distance

Properties of greedy best-first search

- Complete? No – can get stuck in loops.
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ - keeps all nodes in memory
- Optimal? No

  e.g. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest is shorter!

A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal
- Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$

A*: the core idea

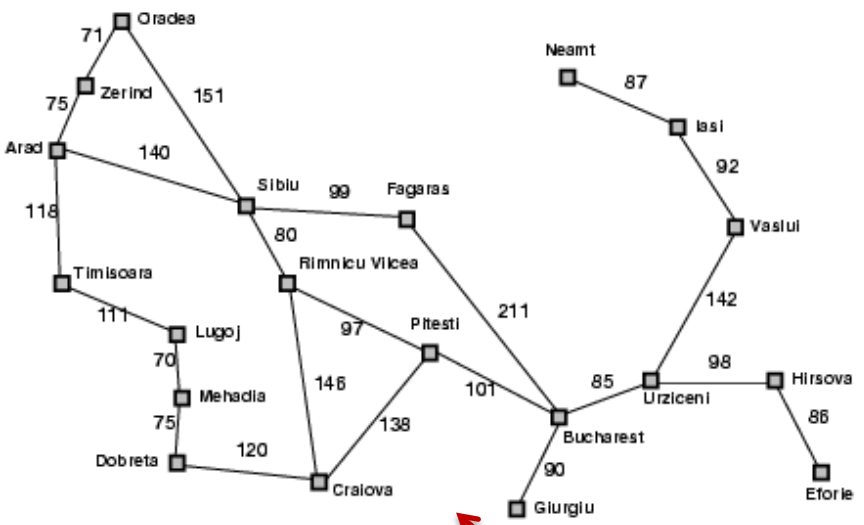
- Expand a node *n* most likely to be on the optimal path
- Expand a node *n* s.t. the cost of the best solution through *n* is optimal
- Expand a node *n* with lowest value of $g(n) + h^*(n)$
  - $g(n)$ is the cost from root to *n*
  - $h^*(n)$ is the optimal cost from *n* to the closest goal
- We seldom know $h^*(n)$ but might have a heuristic approximation $h(n)$
- A* = tree search with priority queue ordered by $g(n) + h(n)$

g(n) : هزینه مسیر از گره آغازین به گره n

h(n) : هزینه تخمین زده شده از ارزان‌ترین مسیر از n به هدف

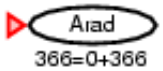
f(n)=g(n)+h(n) : هزینه تخمین زده شده از ارزان‌ترین راه‌حل از طریق n

A* search example



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

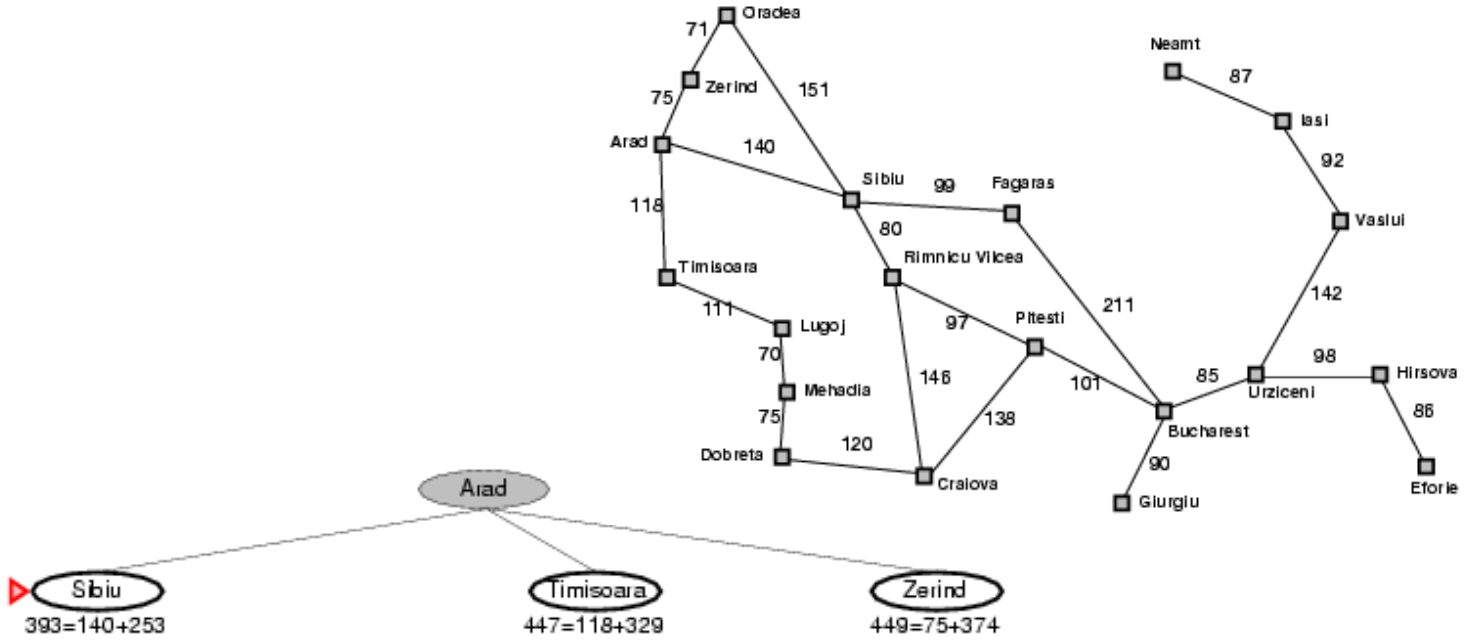
صرفاً برای صرفه‌جویی، نام نود
همان نام حالت



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

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

A* search example

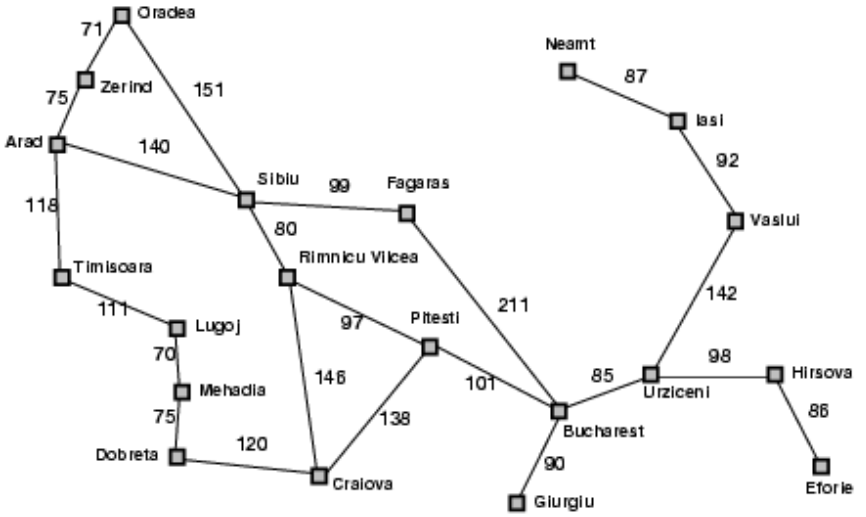


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

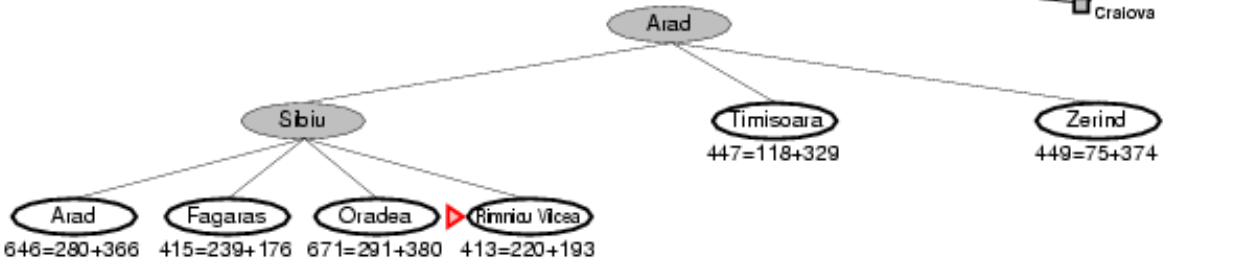
$f(s) = g(s) + h(s)$ $f(T) = g(T) + h(T)$ $f(Z) = g(Z) + h(Z)$

A* search example

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

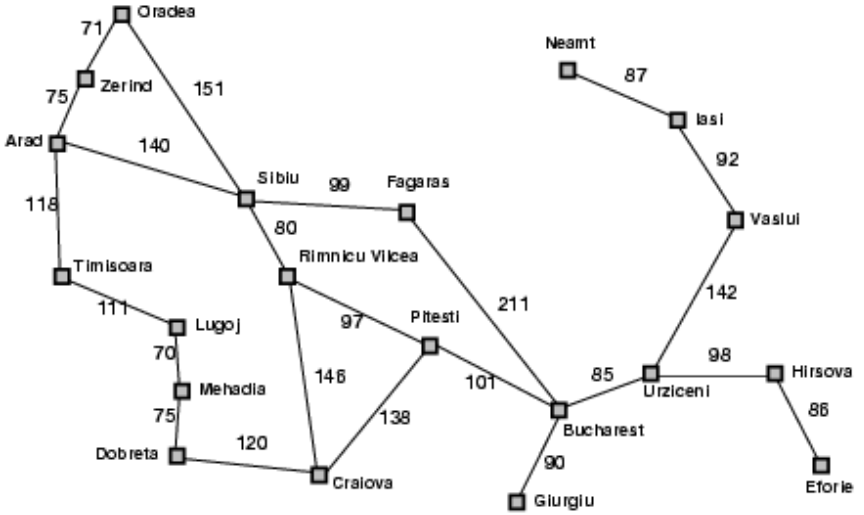


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

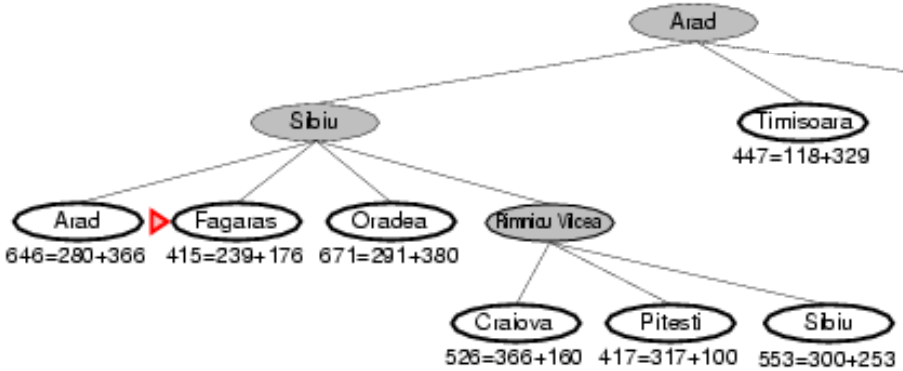


A* search example

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

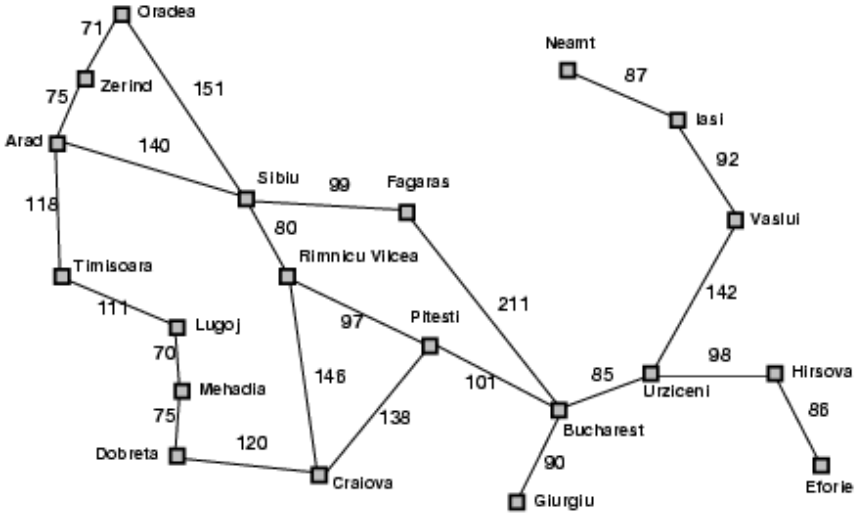


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

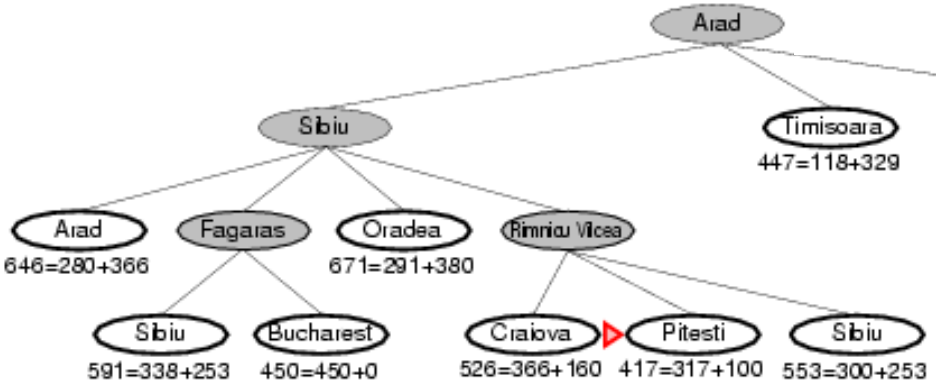


A* search example

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

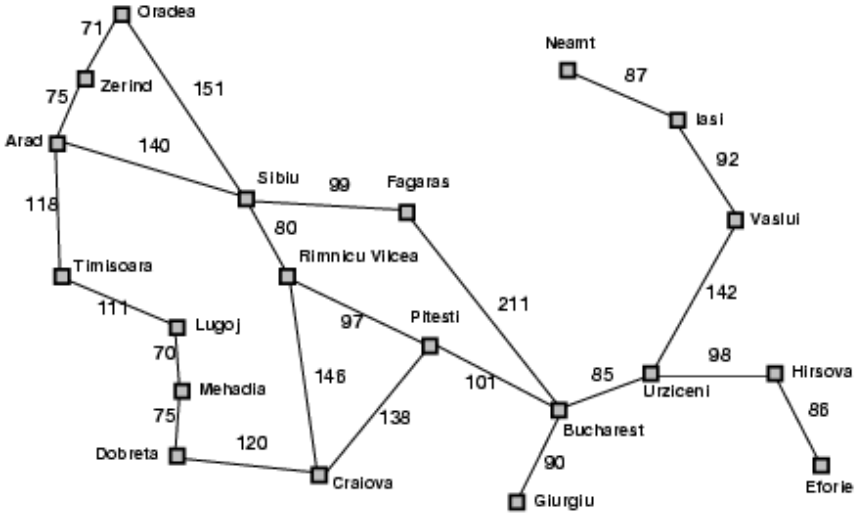


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

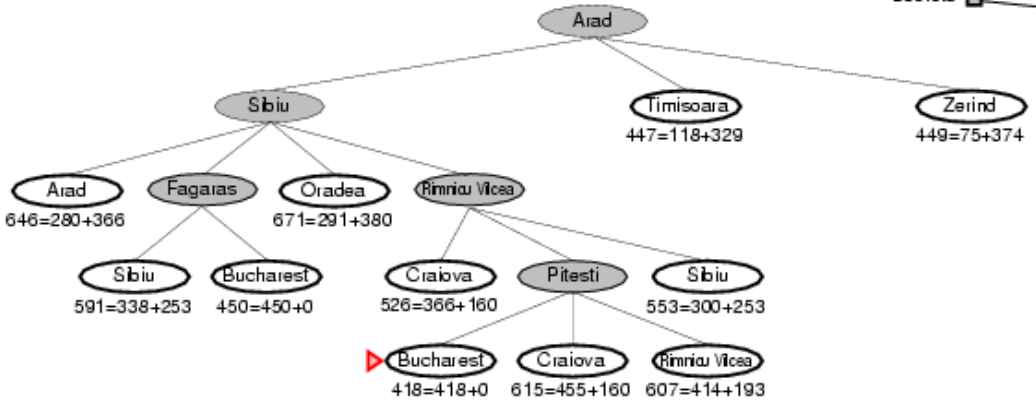


A* search example

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



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



Properties of A*

- Complete? Yes (unless there are infinitely many nodes with
- f ≤ *f(G)* , i.e. step-cost > ε)
- Time/Space? Exponential b^d

  except if: $|h(n) - h^*(n)| \leq O(\log h^*(n))$ (cf. Editions before 2020)
- Optimal? Yes
- *Optimally Efficient*: Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)

چند نکته

- بهینه کارامد: تضمینی نیست الگوریتم بهینه دیگری وجود داشته باشد که تعداد گره‌های کمتر از A* بسط دهد
- هر الگوریتمی که تمام گره‌ها را در نواحی بین ریشه و ناحیه هدف بسط ندهد در معرض خطر گم کردن راه حل بهینه است.
- A* برخی گره‌های دیگر هم هزینه با هدف (در همان کانتور هدف) را نیز ممکن است قبل از انتخاب گره هدف بسط دهد
- در اکثر مسائل، تعداد گره‌ها در کانتور هدف نسبت به طول راه حل مرتبه نمایی دارند مگر آن‌که خطا در تابع کشف‌کننده رشدی سریع‌تر از لگاریتم هزینه مسیر واقعی (در اسلاید قبل) نداشته باشد

Optimality of A* Tree Search

- **Admissibility**
- **Consistency ~ Monotonicity**

- اولین شرط مورد نیاز برای تضمین بهینگی: قابل‌قبول‌بودن یا مجاز بودن
  یعنی تابع h باید هرگز هزینه‌ی رسیدن به هدف را زیاد برآورد نکند

- دومین شرط مورد نیاز برای تضمین بهینگی: سازگاری یا یکنوایی
  یعنی در طول هر مسیری از ریشه، هرگز هزینه f کاهش پیدا نکند.

یکنوایی، نسبت به قابل‌قبول‌بودن، شرط دقیق‌تری است.
تمام ابتکارهای سازگار و یکنوا، این خاصیت قابل‌قبول‌بودن را دارند
و به سختی بتوان هیوریستیک قابل‌قبول پیدا کرد که سازگار نباشد

Admissible heuristics

تابع h باید هرگز هزینه‌ای بیش از تخمین برای رسیدن به هدف نداشته باشد

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

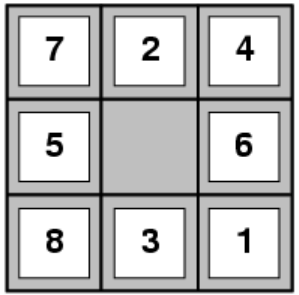
Admissible heuristics

E.g., for the 8-puzzle:

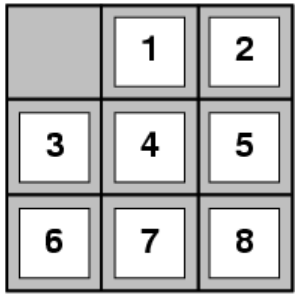
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

- $h_1(S) = ?$
- $h_2(S) = ?$



Start State



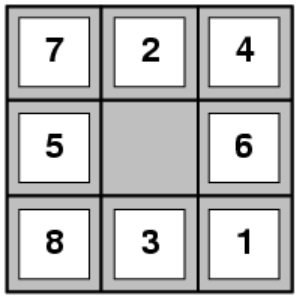
Goal State

Admissible heuristics

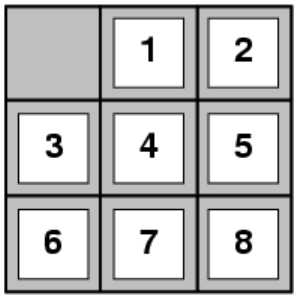
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1
- **h_2 is better for search: it is guaranteed to expand less or equal nr of nodes.**

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

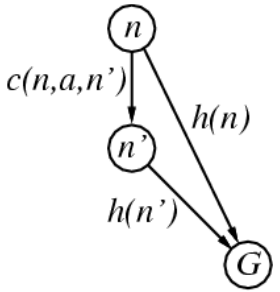
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Consistent or monotone heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

$$h(n) \leq c(n,a,n') + h(n')$$



- If *h* is consistent, we have

f(n') = g(n') + h(n') (by def.)
= g(n) + c(n,a,n') + h(n') (g(n')=g(n)+c(n.a.n'))
≥ g(n) + h(n) = f(n) (consistency)
f(n') ≥ f(n)

It's the triangle inequality !

- i.e., *f(n)* is non-decreasing along any path.

تمام هیوریستیک‌های قابل‌قبول، این خاصیت یکنوایی را دارند.

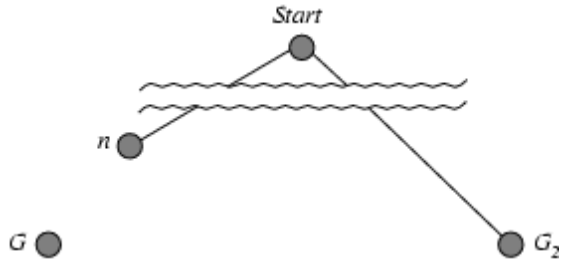
- Theorem:
  If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal

keeps all checked nodes in memory to avoid repeated states

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.

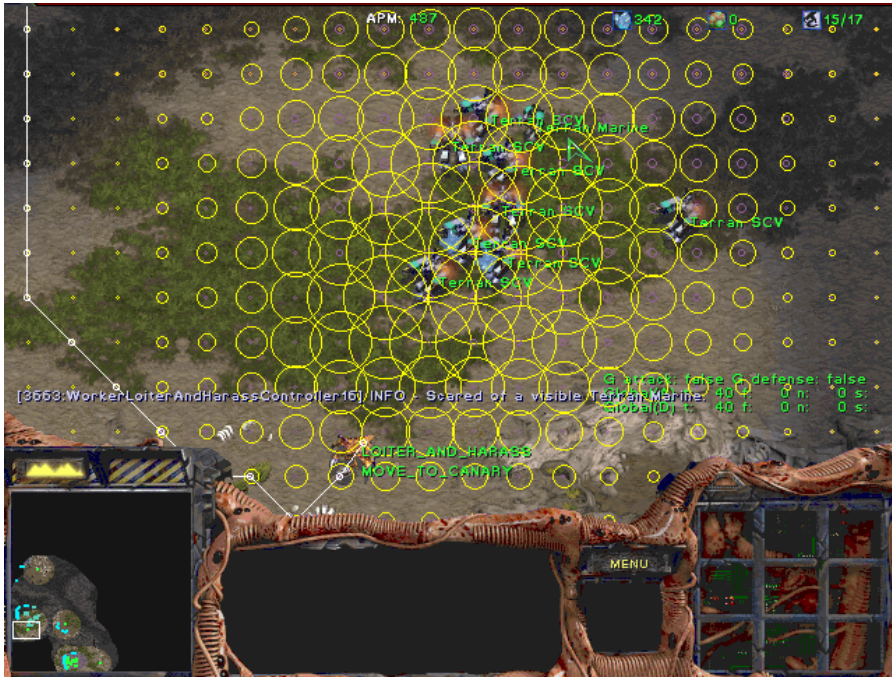
We want to prove:
f(n) < f(G2)
(then A* will prefer n over G2)



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

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Protein design
- Chemical synthesis
- …



مشکلات A*

- A* در کنار مزایای خاص، دارای مشکلاتی است:

1. پیچیدگی زمانی: زمان اجرا به صورت تابعی از عمق جواب بهینه (d) است.
2. وقتی چندین حالت هدف وجود داشته باشد، جستجو می‌تواند منجر به انحراف از مسیر بهینه شود و هزینه‌ای اضافی متناسب با تعداد آن اهداف ایجاد می‌شود.
3. پیچیدگی فضایی: چون تمام گره‌های تولید شده را در حافظه نگه می‌دارد، قبل از اتمام زمان اجرا، با کمبود فضا (حافظه) مواجه می‌شود.

جستجوی حافظه محدود شده

- A* کامل و بهینه است اما به دلیل آنکه تمام گره‌های تولید شده را در حافظه ذخیره می‌کند، دچار کمبود حافظه می‌شود.
- یادآوری: وقتی فضای جستجو بزرگ و عمق راه حل مجهول باشد، استراتژی جستجوی عمیق‌کننده تکراری، روش جستجوی برتر است و تکنیکی برای کاهش درخواست حافظه است
- ایده: تلفیق جستجوی A* و جستجوی عمیق‌کننده تکراری
- دو دسته الگوریتم:
  - IDA*, RBFS: Iterative Deepening A*, Recursive Best-First Search
  - SMA*: Simplified Memory-Bounded A*

Memory Bounded Heuristic Search: Recursive BFS

- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let's not forget everything about the branches we have partially explored.
- ***We remember the best f-value we have found so far in the branch we are deleting*.**

IDA*

- هر تکرار، تغییر یافته‌ی یک جستجوی عمقی DFS است به گونه‌ای که به جای یک محدوده عمقی، از یک محدوده f_cost استفاده کند و در هر تکرار، تمام گره‌های داخل محدوده f_cost جاری را بسط می‌دهد و بعد شروع به جستجوی یافتن ناحیه بعدی می‌کند.
- افزایش محدوده f_cost توسط یک مقدار ثابت ϵ روی هر تکرار

RBFS

- به جای آنکه در مسیر فعلی به طور بینهایت به سمت پایین بیاید، برای نگهداری مقدار f مربوط به بهترین مسیر آلترناتیوی که از جد گره فعلی وجود دارد، از متغیر f_limit استفاده می‌کند.
- اگر گره فعلی از این حد تجاوز کند، بازگشتی به عقب برمی‌گردد تا مسیر دیگری را طی کند.
- هنگام برگشت از بازگشتی‌ها، بهترین مقدار فرزندانش را به عنوان یک مقدار پشتیبان به جای آن f قرار می‌دهد؛ یعنی، مقدار f مربوط به بهترین برگ موجود در زیردرخت فراموش شده را به یاد دارد، به منظور تصمیم برای اینکه آیا این زیردرخت در آینده ارزش دوباره بسط دادن دارد یا خیر.

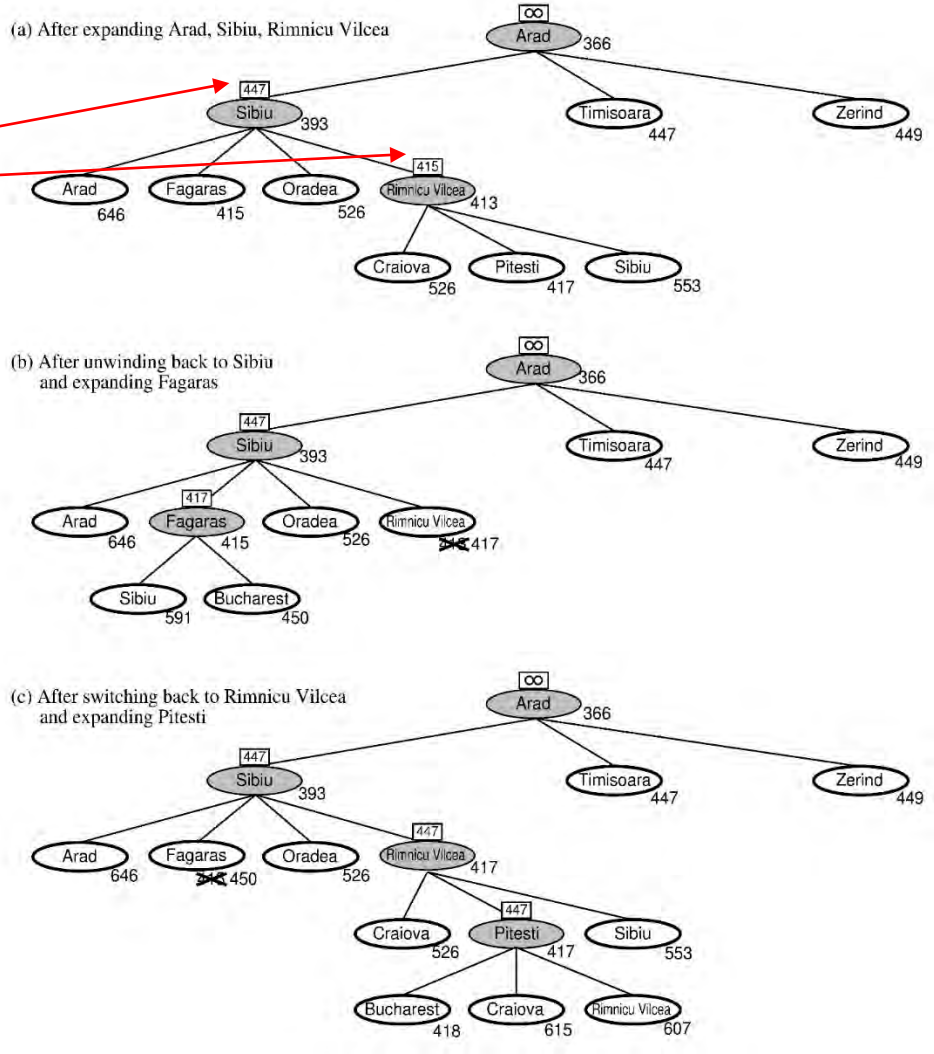
RBFS:

best alternative over fringe nodes, which are not children:
i.e. do I want to back up?

RBFS changes its mind very often in practice.

This is because the **f=g+h become more accurate** (less optimistic) **as we approach the goal**. Hence, higher level nodes have smaller f-values and will be explored first.

Problem: We should keep in memory whatever we can.



```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution or failure
    solution, fvalue ← RBFS(problem, NODE(problem.INITIAL), ∞)
  return solution

function RBFS(problem, node, f_limit) returns a solution or failure, and a new f-cost limit
  if problem.IS-GOAL(node.STATE) then return node
  successors ← LIST(EXPAND(node))
  if successors is empty then return failure, ∞
  for each s in successors do          // update f with value from previous search
      s.f ← max(s.PATH-COST + h(s), node.f))
  while true do
      best ← the node in successors with lowest f-value
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result, best.f
```

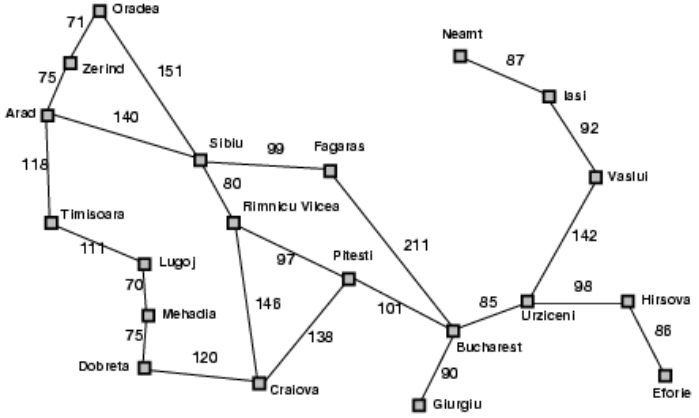
Ed.2020

Figure 3.22 The algorithm for recursive best-first search.

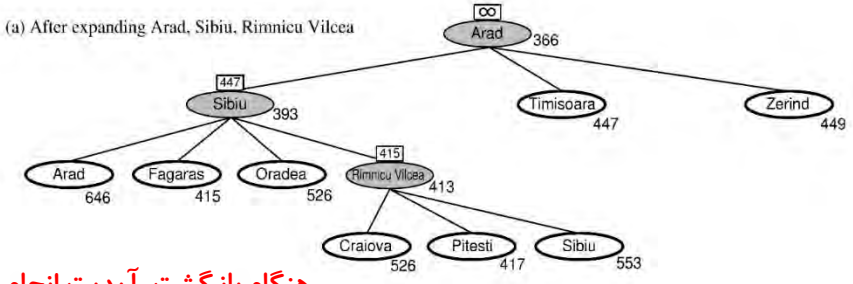
هنگام بازگشت، آپدیت انجام می‌دهد. f هیچ نودی نباید از پدرش کمتر باشد.

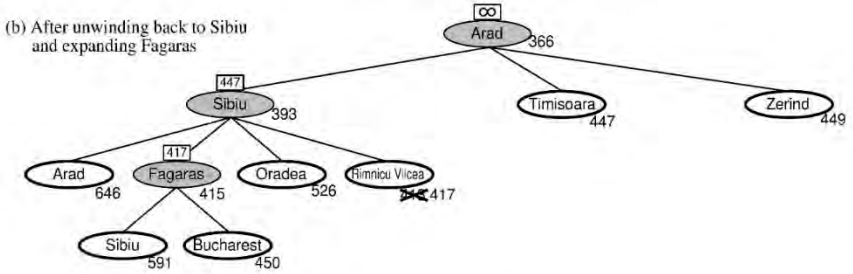
مقداردهی f فرزندان با max خودش و f پدرش.

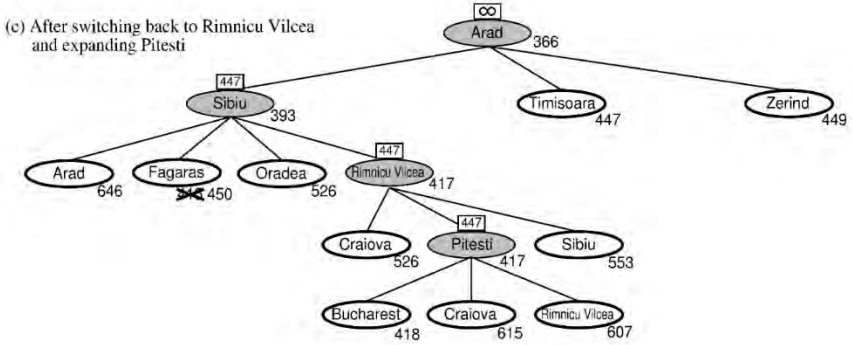
به عنوان یک cutoff جدید



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







f = g + h

SMA*: Simple Memory Bounded A*

- اگر حافظه بیشتری هم فراهم بود، RBFS نمی‌توانست از آن استفاده کند.
- اغلب کارهایی که انجام داده‌است را فراموش می‌کند و ممکن است حالت‌هایی را چندین بار بسط دهد.

- **معمولاً بهتر است گره را به یاد بیاوریم تا اینکه مجبور باشیم آن را دوباره تولید کنیم.**
- **ویژگی‌های SMA*:**
  - **از تمام حافظه موجود برای اجرای جستجو استفاده می‌کند.**
  - **از حالت تکراری تا جایی که حافظه اجازه می‌دهد جلوگیری می‌کند.**
  - **کامل است به شرط آنکه حافظه برای ذخیره کم‌عمق‌ترین مسیر راه‌حل، کافی باشد.**
  - **بهینه است اگر حافظه کافی برای ذخیره کردن کم‌عمق‌ترین مسیر راه‌حل بهینه کافی باشد. در غیراینصورت راه‌حلی را بر می‌گرداند که بتواند با حافظه موجود مطابقت داشته باشد.**
  - **زمانی که حافظه موجود برای درخت جستجوی کامل کافی باشد، جستجو optimally efficient است.**

طراحی SMA*

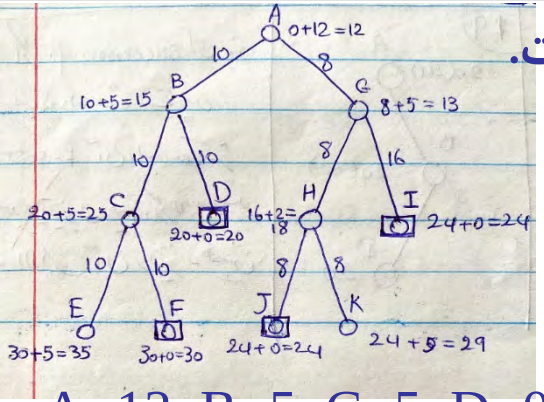
- زمانی که نیاز به تولید فرزند داشته باشد ولی حافظه‌ای بر روی صف نداشته باشد، یک گره را حذف می‌کند و این گره‌های فراموش شده آن‌هایی هستند که f_cost آنها زیاد است.
- برای اجتناب از جستجوی مجدد زیردرخت‌های حذف‌شده از حافظه، اطلاعاتی در مورد کیفیت بهترین مسیر در زیردرخت فراموش‌شده، در گره‌های اجدادی نگه داشته می‌شود.
- پس فقط زمانی زیردرخت‌ها دوباره تولید می‌شوند که ثابت شود تمام مسیرهای دیگر بدتر از مسیر فراموش‌شده باشند.

SMA*: Simple Memory Bounded A*

- This is like A*, but **when memory is full we delete the worst node** (largest f-value).
- Like RBFS, we **remember the best descendent in the branch we delete.**
- If there is **a tie** (equal f-values) we **delete the oldest** nodes first.
- simple-MBA* finds the optimal *reachable* solution given the memory constraint.
- Time can still be exponential.

A Solution is not reachable
if a single path from root to goal
does not fit into memory

مثال SMA*

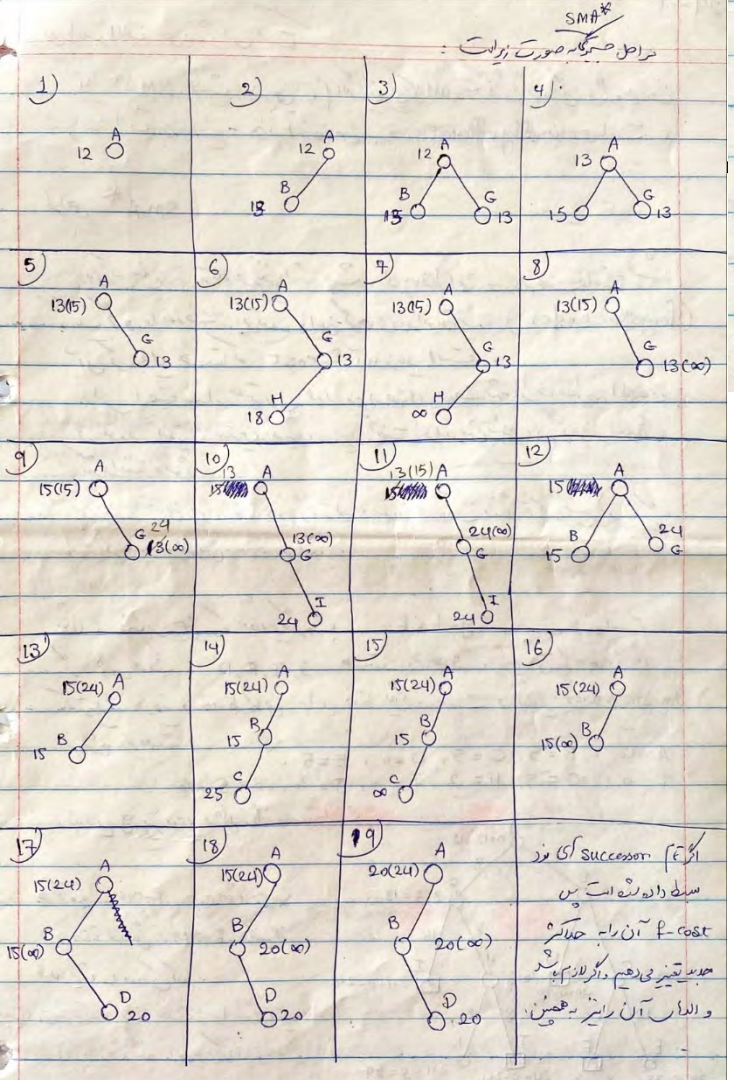


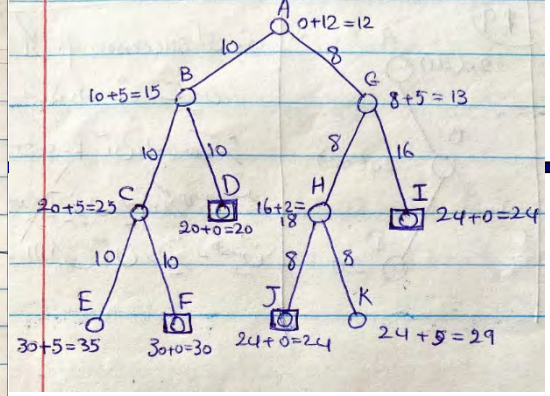
- مثال: فضای جستجو که مقادیر هر گره با g+h=f برچسب خورده است.
- گره‌های هدف، J,I,F,D هستند. هدف، یافتن گره هدف با کمترین هزینه و حافظه کافی برای فقط ۳ گره است و مقادیر heuristic values به شرح زیر هستند و مقادیر g روی شکل نشان داده شده است:

A=12, B=5, C=5, D=0, E=5, F=0, G=5, H=2, I=0, J=0, K=5.

- هر گره با f_cost جاری‌اش برچسب می‌خورد تا حداقل f_cost هر یک از نسل‌هایش را منعکس کند.
- در جستجو، مقادیری که در پرانتز هستند ارزش بهترین نسل فراموش شده را بر می‌گردانند (اسلاید بعد).

مثال SMA*





1. در هر مرحله یک فرزند به عمیق‌ترین گره که دارای حداقل f_cost باشد اضافه می‌شود و فرزندانی دارد که در حالت جاری در درخت نیست. فرزند چپ که B است به ریشه اضافه می‌شود.
2. اکنون هنوز f(A)=12 بنابراین فرزند سمت راست را اضافه می‌کنیم G(f=13). حالا که ما تمام فرزندان A را دیدیم می‌توانیم f_cost آن را به حداقل فرزندانش تغییر دهیم که مقدار ۱۳ است. حافظه اکنون پر است.
2. G در حال حاضر آماده بسط دادن است اما ابتدا باید گره‌ای را حذف کنیم تا فضای کافی موجود باشد. برگی که کم عمق‌ترین است و بیشترین f_cost را دارد که همان B است را حذف می‌کنیم. زمانی که این عمل را انجام دادیم متوجه می‌شویم که بهترین نسل فراموش شده A، f=15 دارد که در پرانتز نشان داده شده است. سپس H را با f(H)=18 اضافه می‌کنیم. متاسفانه H گره هدف نیست اما مسیری که به H می‌رود تمام حافظه موجود را استفاده می‌کند. از این‌رو راهی برای یافتن یک راه‌حل از طریق H نیست. بنابراین f(H)=infinity قرار می‌دهیم.
3. G دوباره بسط داده می‌شود. H را حذف کردیم و I را اضافه نمودیم. f(I)=24. حالا هر دو فرزند G را دیدیم. با مقادیری از infinity و 24 بنابراین f(G) برابر 24 می‌شود. f(A) برابر ۱۵ می‌شود که همان مینیمم ۱۵ (مقدار قرزند فراموش شده) و ۲۴ است. I یک گره هدف است اما بهترین راه حل نیست چون f_cost مربوط به A فقط 15 است.
4. A دوباره گره‌ای است که بیشترین تعهد را دارد. بنابراین B برای بار دوم تولید می‌شود. دریافتیم که مسیر از طریق G چندان جالب نبود.
5. C اولین فرزند B، یک گره غیرهدفی در عمق حداکثر است. بنابراین f(c)=infinity است.
6. در نگاه کردن به فرزند دوم D ابتدا C را حذف می‌کنیم. سپس f(D)=20 می‌شود و این مقدار توسط A و B به ارث می‌رسد.
7. اکنون عمیقترین و کمترین f_cost را گره D دارد. بنابراین D انتخاب می‌شود و چون گره هدف است جستجو خاتمه می‌یابد.

توضیحات اضافه‌تر در مراحل جستجو (در تصویر چپ اسلاید قبل)

- پس از خانه 4: تمام فرزندان A رؤیت شد. A را به حداقل f_cost فرزندان تغییر می‌دهیم و حافظه هم پر شده است.
- خانه 5: حذف B و حفظ f_cost آن در پرانتز.
- خانه 7: H گره هدف نیست و حافظه هم پر شده است. پس راهی از H برای رسیدن به هدف نیست. پس f(H)=infinity قرار می‌دهیم.
- خانه 8: H حذف می‌شود و f_cost آن در پرانتز می‌آید. G می‌شود 13(infinity).
- خانه 10: G را بسط می‌دهیم لذا بین f(I)=24 و f(H)=infinity مقدار f(G)=24 می‌شود و سپس بین 24 و (15)، f(A)=15 می‌شود.
- خانه 11: I گره هدف است اما f_cost (24) آن از f_cost گره A (15) بیشتر است لذا بهترین راه‌حل نیست. لذا مسیر از طرف G جالب نبوده و حذف می‌شود . 24 در پرانتز برای A قرار می‌گیرد و حال B برای بار دوم بسط داده می‌شود.
- خانه 15: در اینجا نیز مثل H همان توضیحات را برای C داریم.
- خانه 16: باز مثل H، اینجا C را حذف می‌کنیم و B می‌شود 15(infinity). حال B را بسط می‌دهیم و بین f(D)=20 و f(c)=infinity، f(B)=20 می شود و
- خانه 17: سپس بین f(A)=15 و A(B)=20 مقدار f(A)=20 می‌شود و D یک گره هدف نیز هست . خاتمه می‌یابد.
- ولی اگر J هزینه 19 به جای 24 داشت چون راه‌حل باید شامل ۴ گره می‌بود که J را بیابد، *SMA قادر نبود این مسیر حل بهینه را پیدا کند.
- پس *SMA راه حل قابل دسترس را می‌دهد اما الزاماً بهینه نیست.

با سپاس از توجه شما
دانشجوی گرامی