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

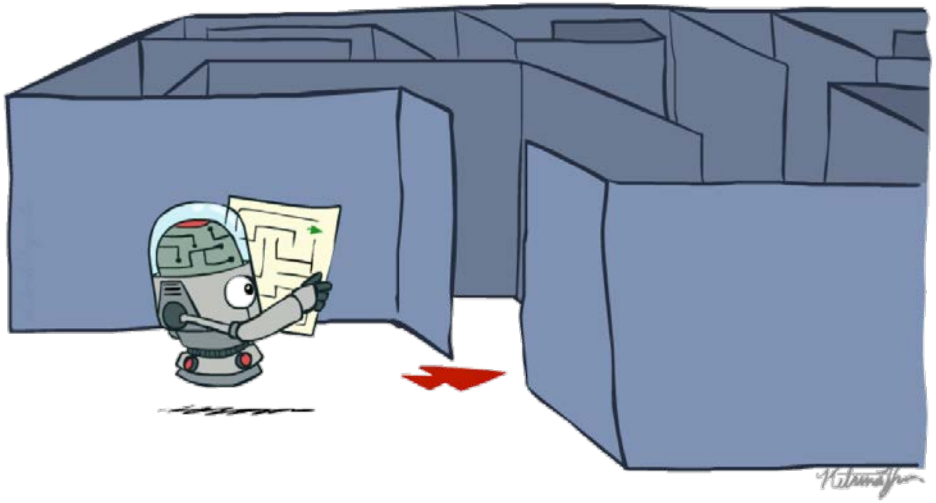
یکشنبه‌ها ۱۶–۱۷–۱۸، سه‌شنبه‌ها ۸–۱۰

lms.ui.ac.ir, **۳۶۰–۷**

نیمسال ۴۰۴۲

CS 188: Artificial Intelligence

Search



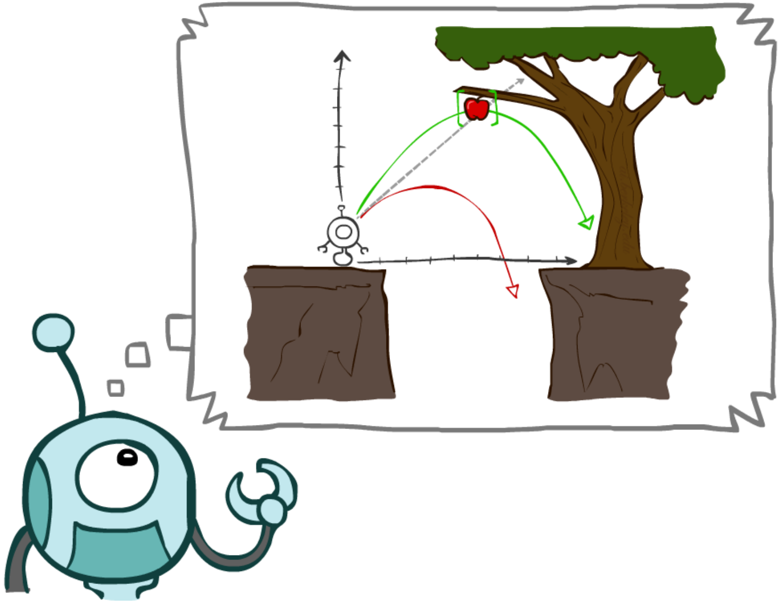
Instructors: Stuart Russell and Dawn Song

University of California, Berkeley

[slides adapted from Dan Klein, Pieter Abbeel]

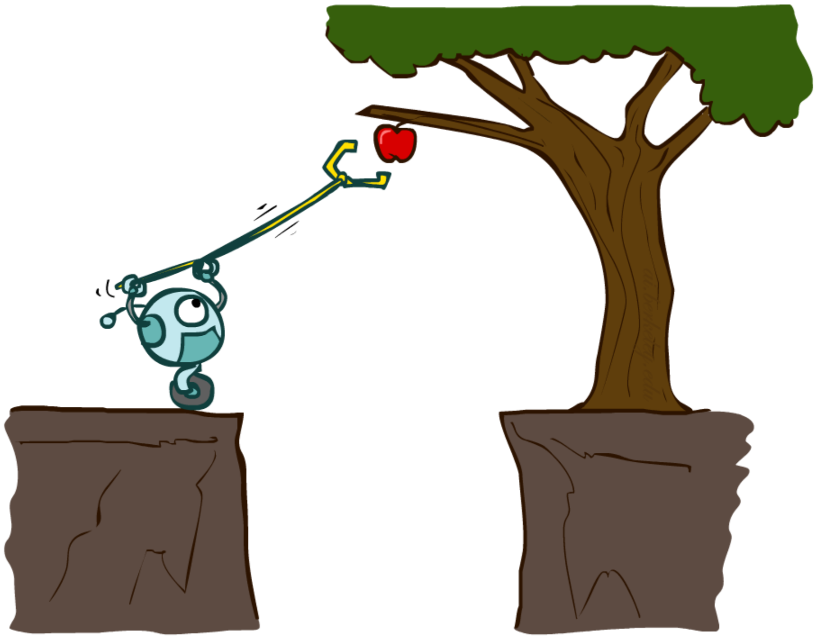
Today

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



Planning Agents

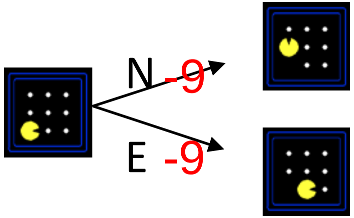
- Planning agents decide based on evaluating future action sequences
- Must have a model of how the world evolves in response to actions
- Usually have a definite goal
- Optimal: Achieve goal at least cost



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

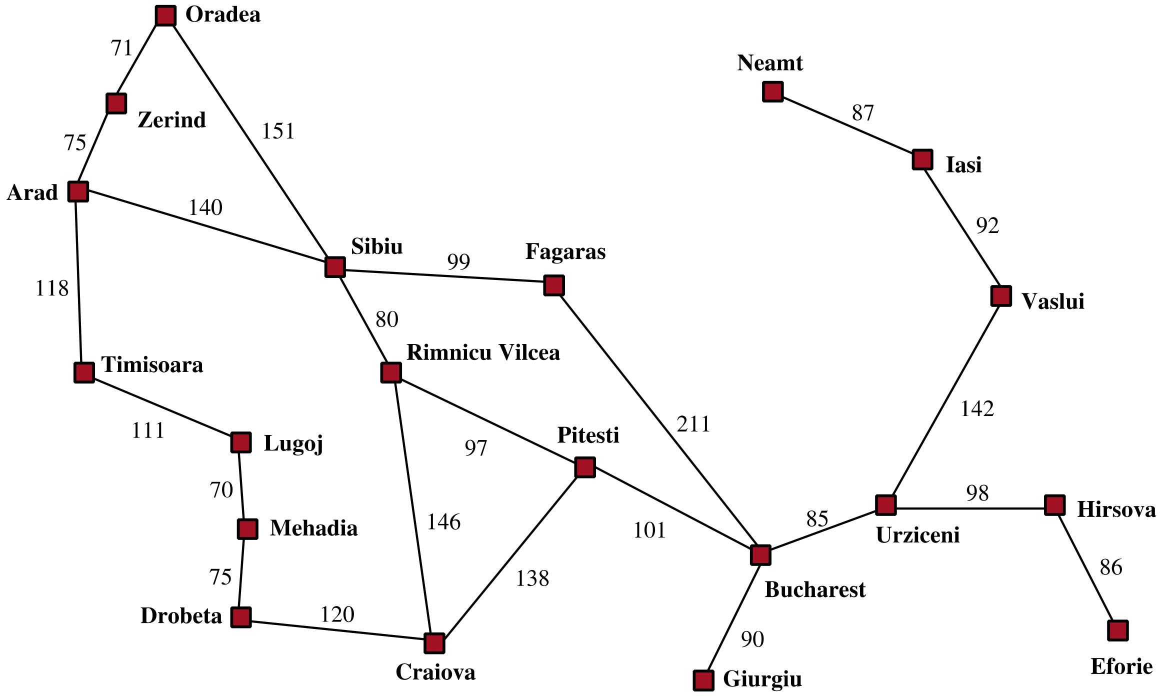
7:10am - 4:45pm

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

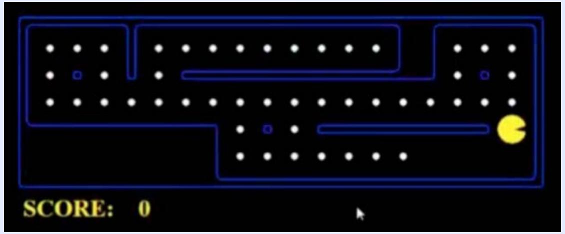
A *solution* is a sequence of actions
leading from the initial state to a goal state

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



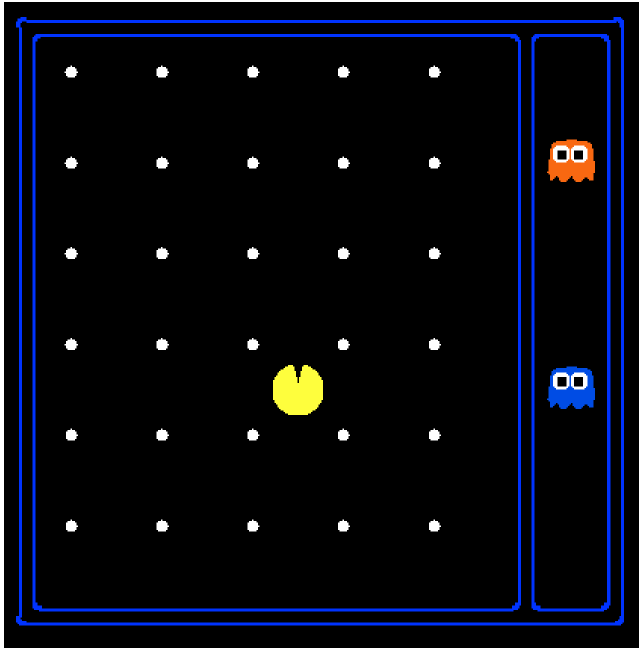
A search state keeps only the details needed for planning (abstraction)

- 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

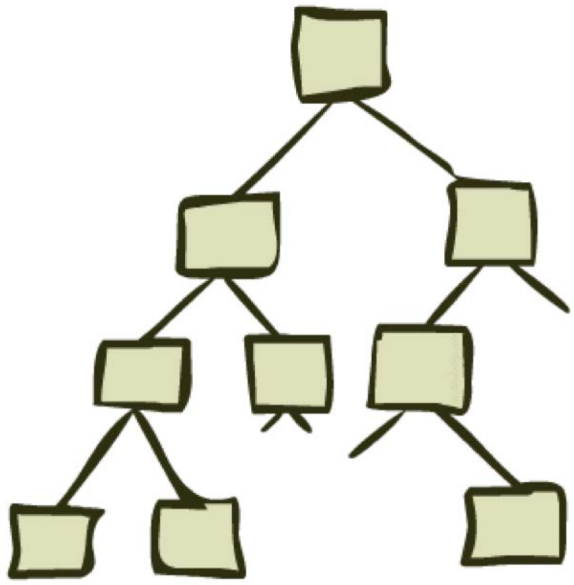
State Space Sizes

- 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

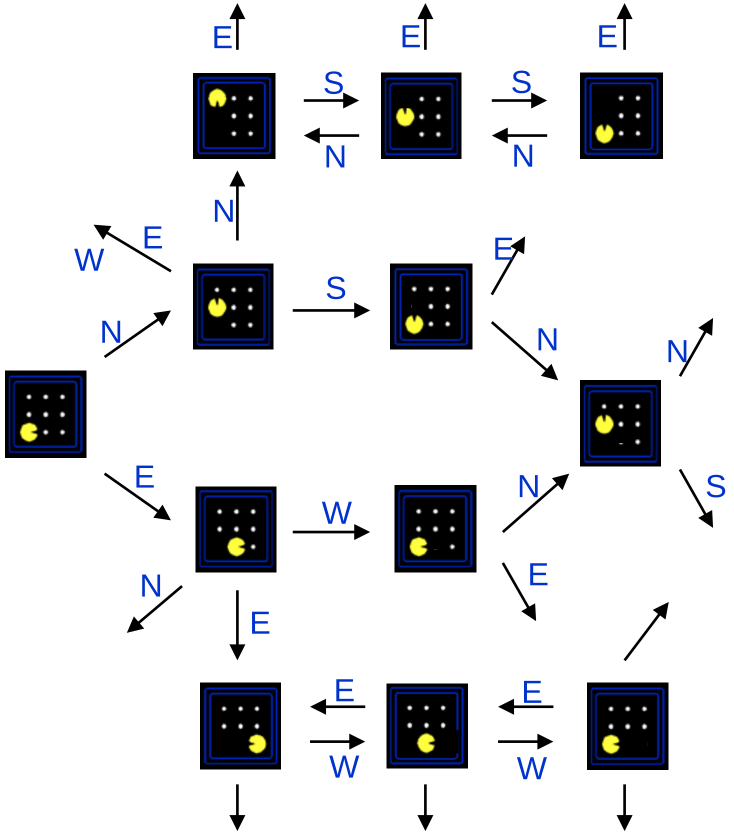


State Space Graphs

- 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)

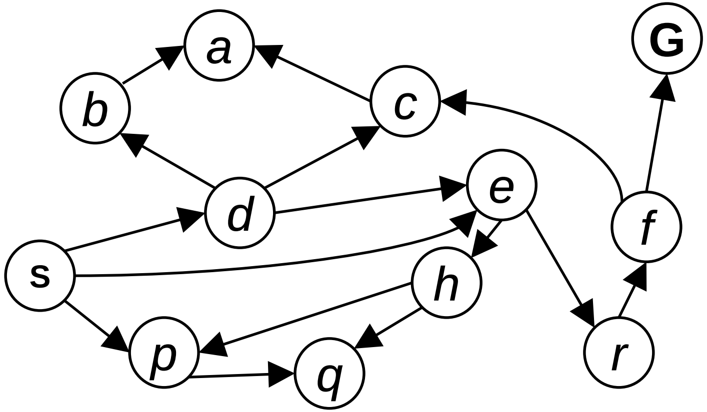
- In a state space graph, each state occurs only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea



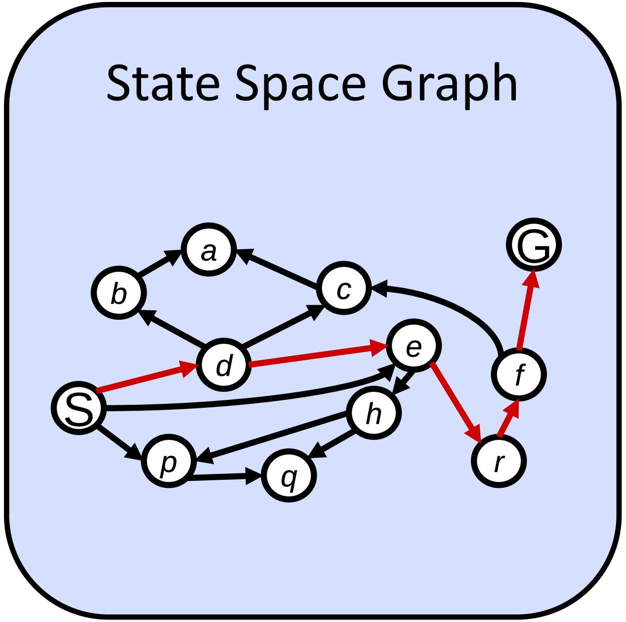
State Space Graphs

- 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



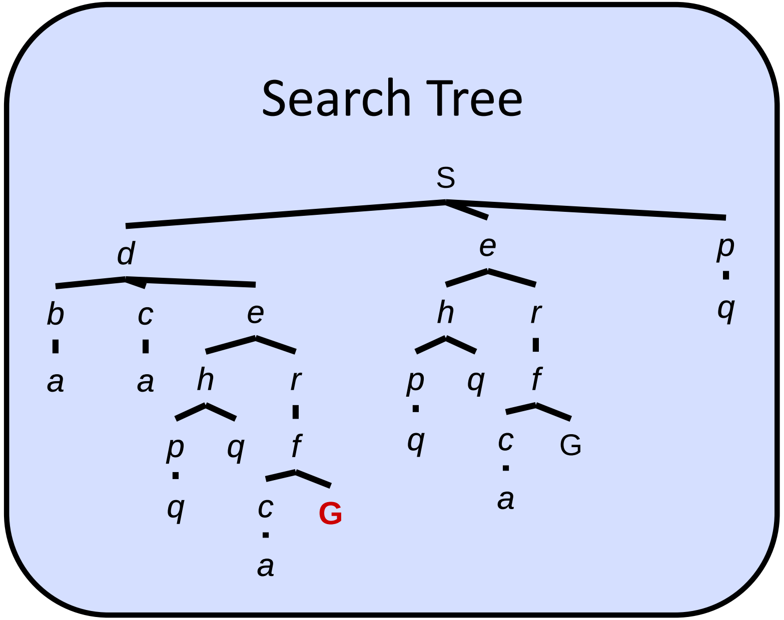
Tiny state space graph for a tiny search problem

State Space Graphs vs. Search Trees

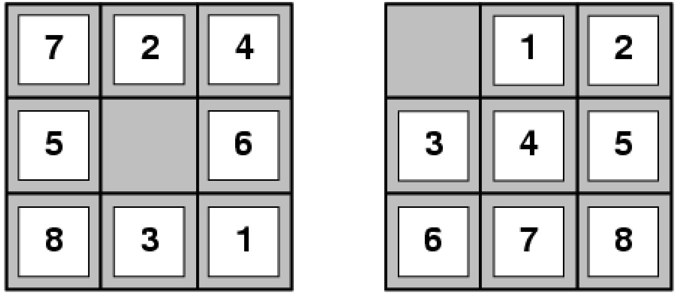


Each NODE in in the search tree is an entire PATH in the state space graph.

We construct the tree on demand – and we construct as little as possible.



Example: The 8-puzzle

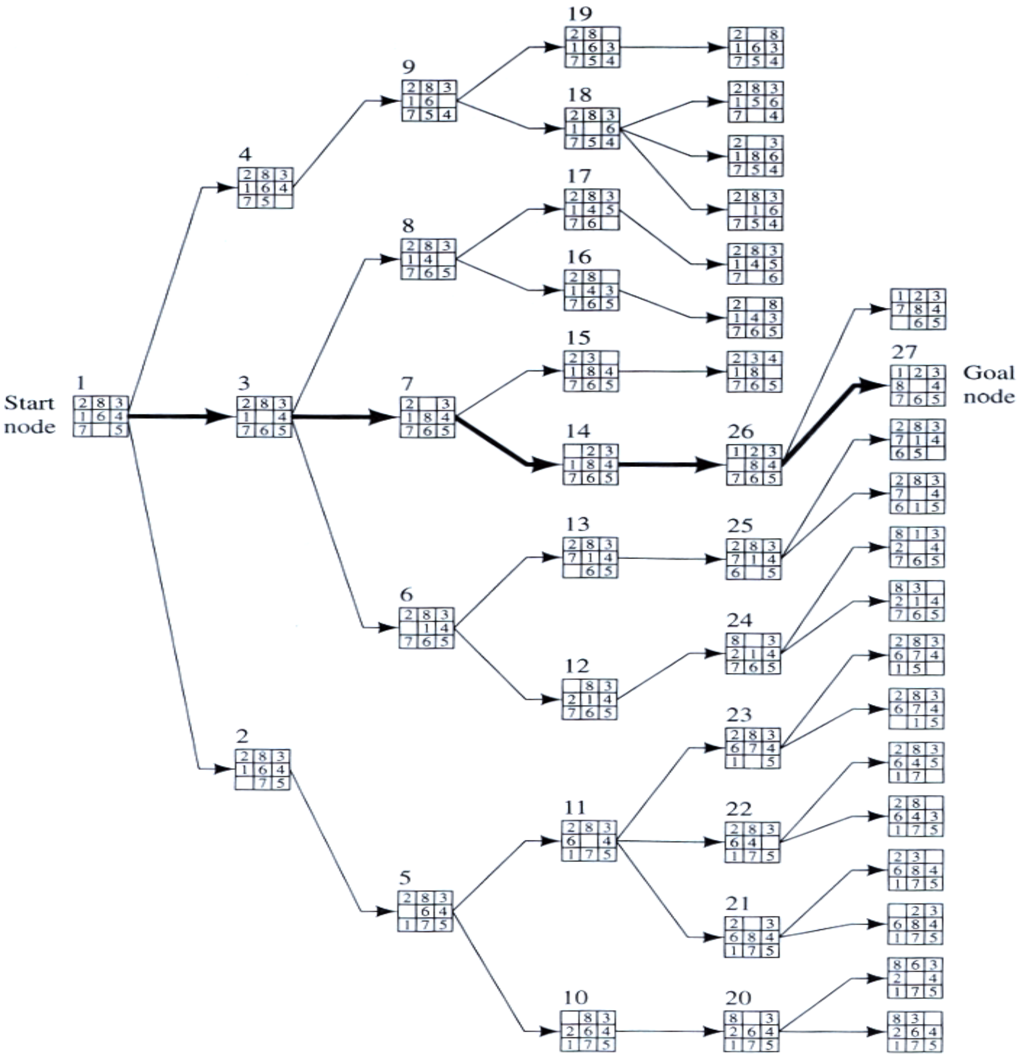


Start State Goal State

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

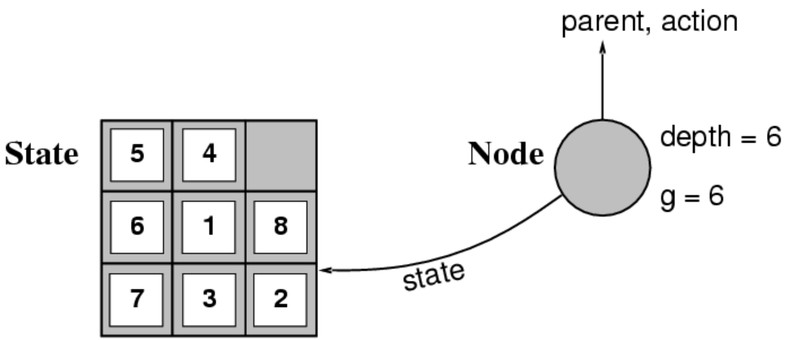
[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: The 8-puzzle



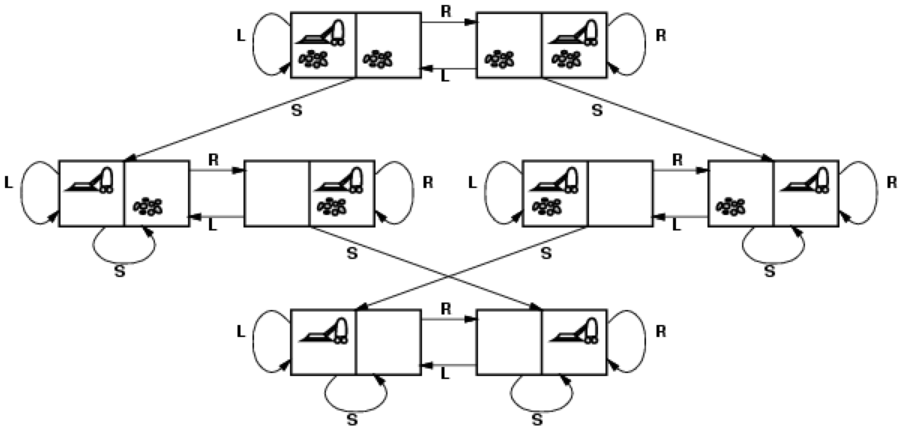
Implementation: states vs. nodes

- 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

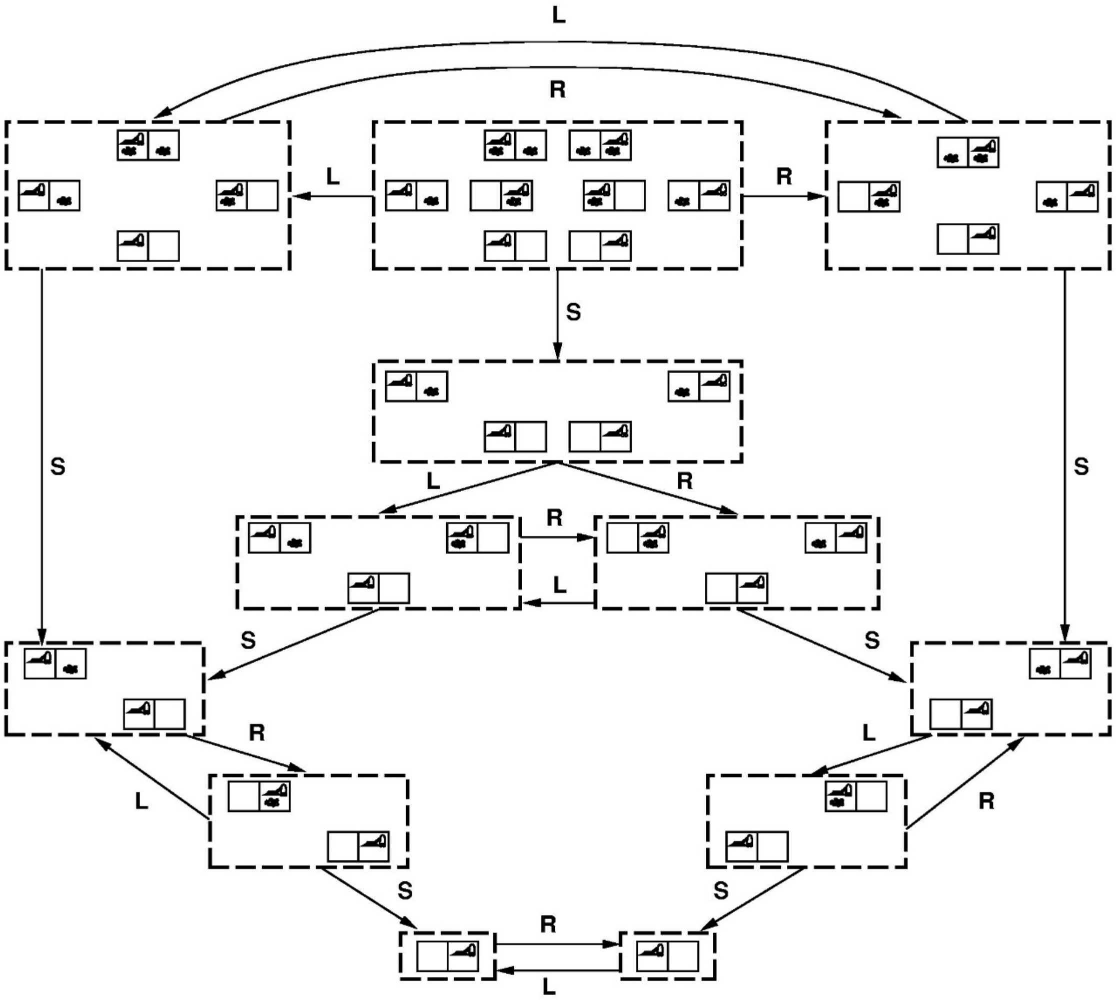


- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Vacuum world state space graph

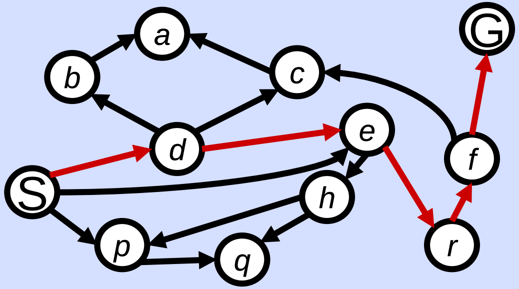


- states? discrete: dirt and robot location
- initial state? any
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action


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

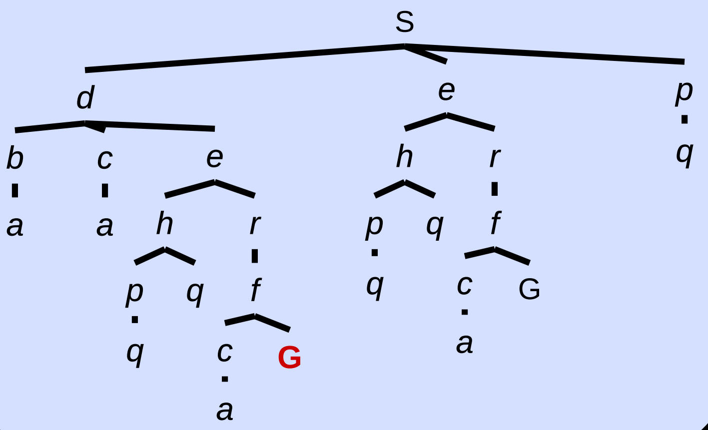
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

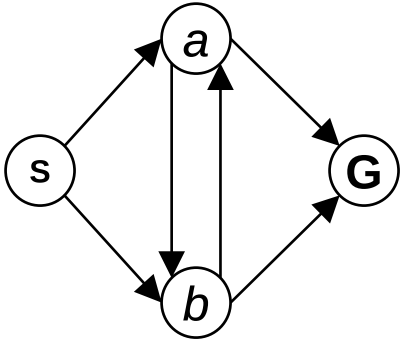
We construct the tree on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

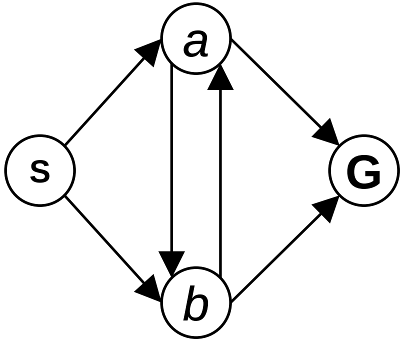


How big is its search tree (from S)?

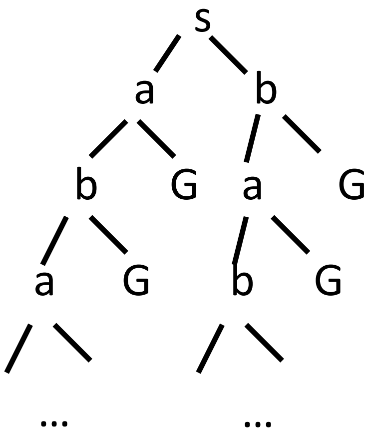
$$\infty$$

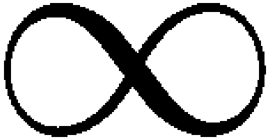
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?

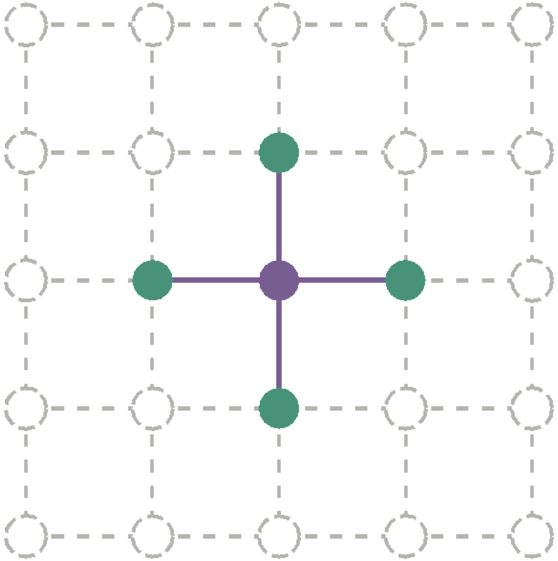




Important: Those who don’t know history are doomed to repeat it!

Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:



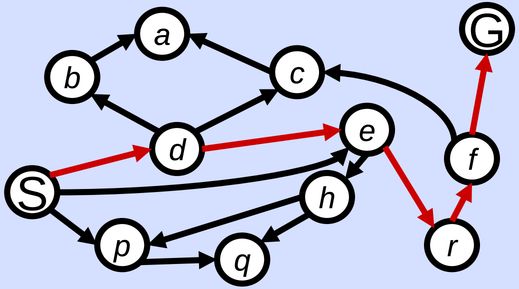
How many states within *d* steps of start?

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

Search Problems

State Space Graphs vs. Search Trees

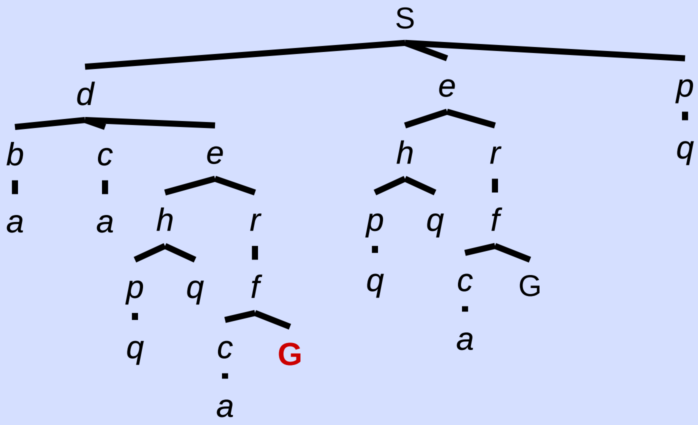
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

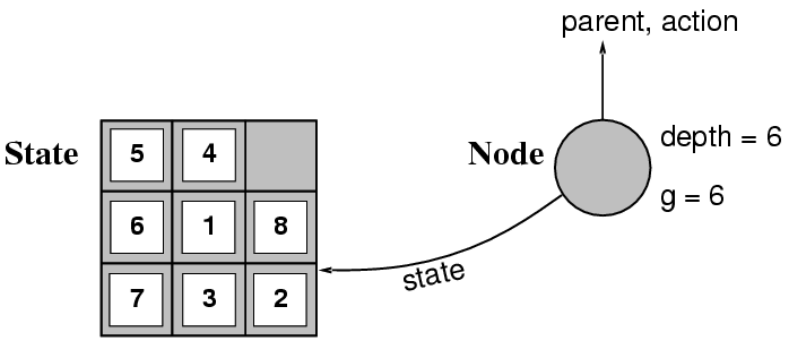
We construct the tree on demand – and we construct as little as possible.

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.

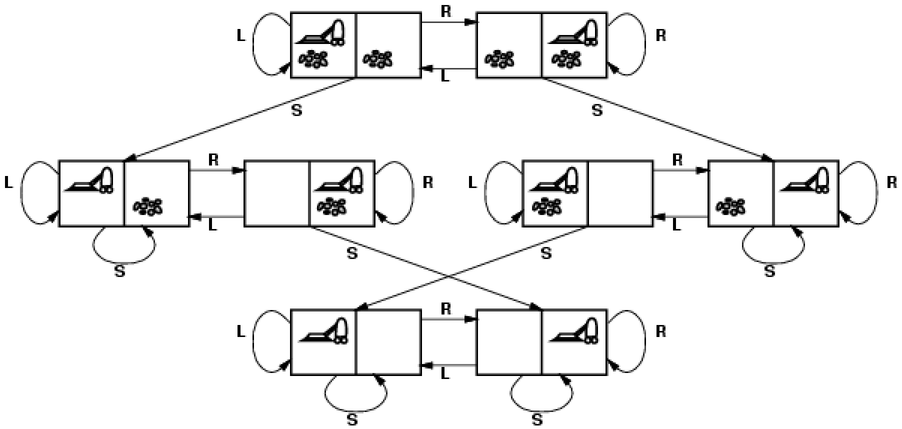
Search Problems

- A search problem consists of:

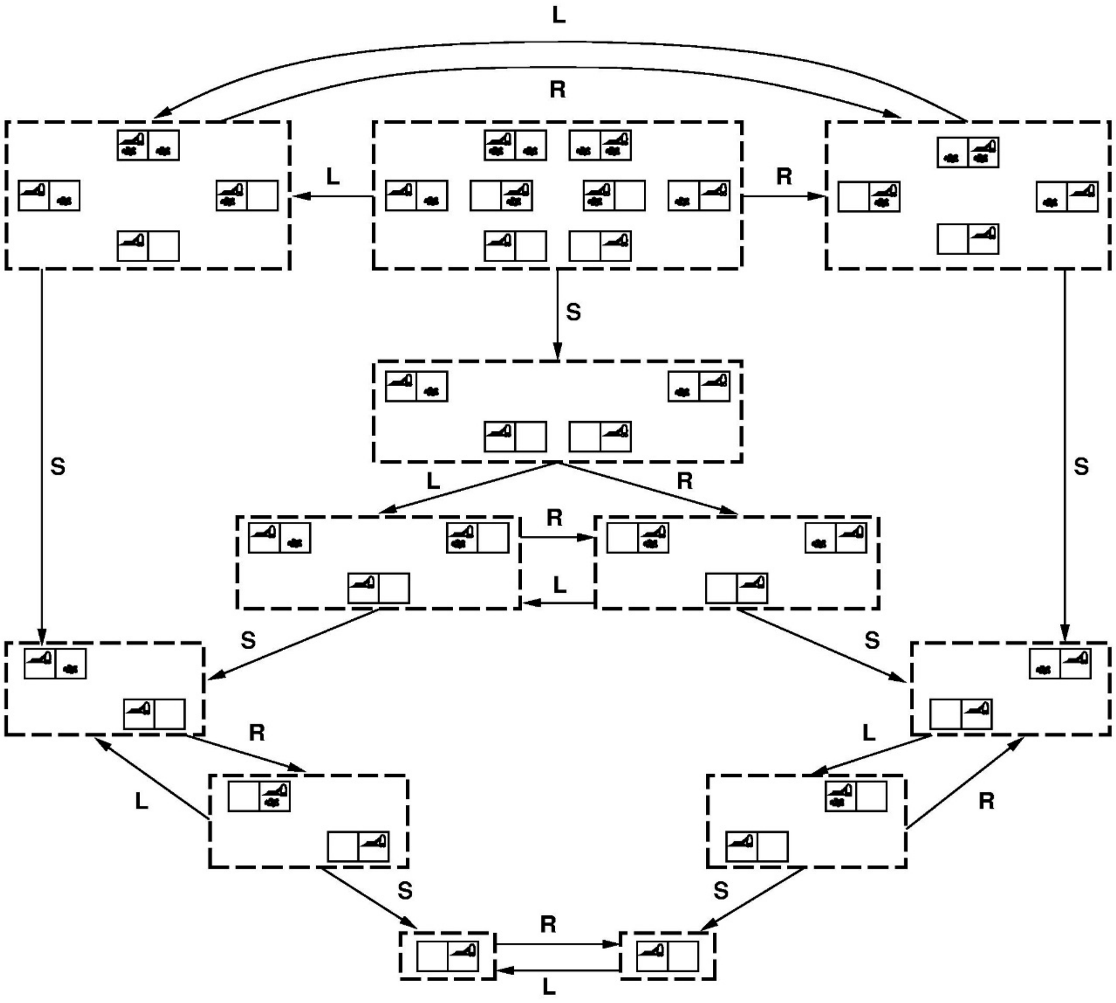
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model $Result(s,a)$ = s'   (also SuccessorFunction)
  - A goal test $G(s)$ = T or F
    - S has no dots left
  - Action cost $c(s,a,s')$ = عدد
    - +1 per step; -10 food; -500 win; +500 die; -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



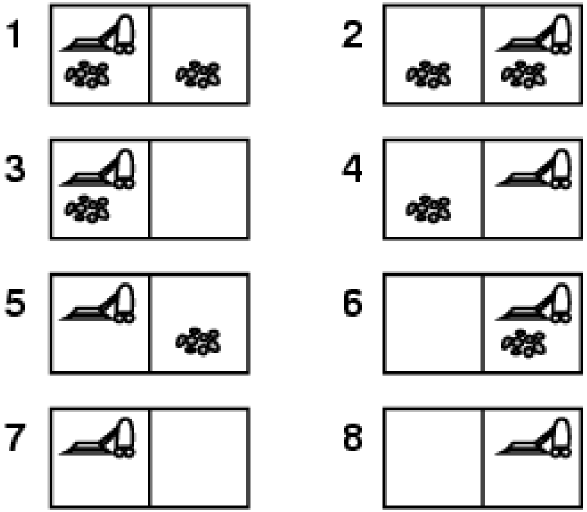
L
R
L
R
S
S
S
L
R
R
L
S
S
R
L
L
R
S
S
R
L

Example: vacuum world

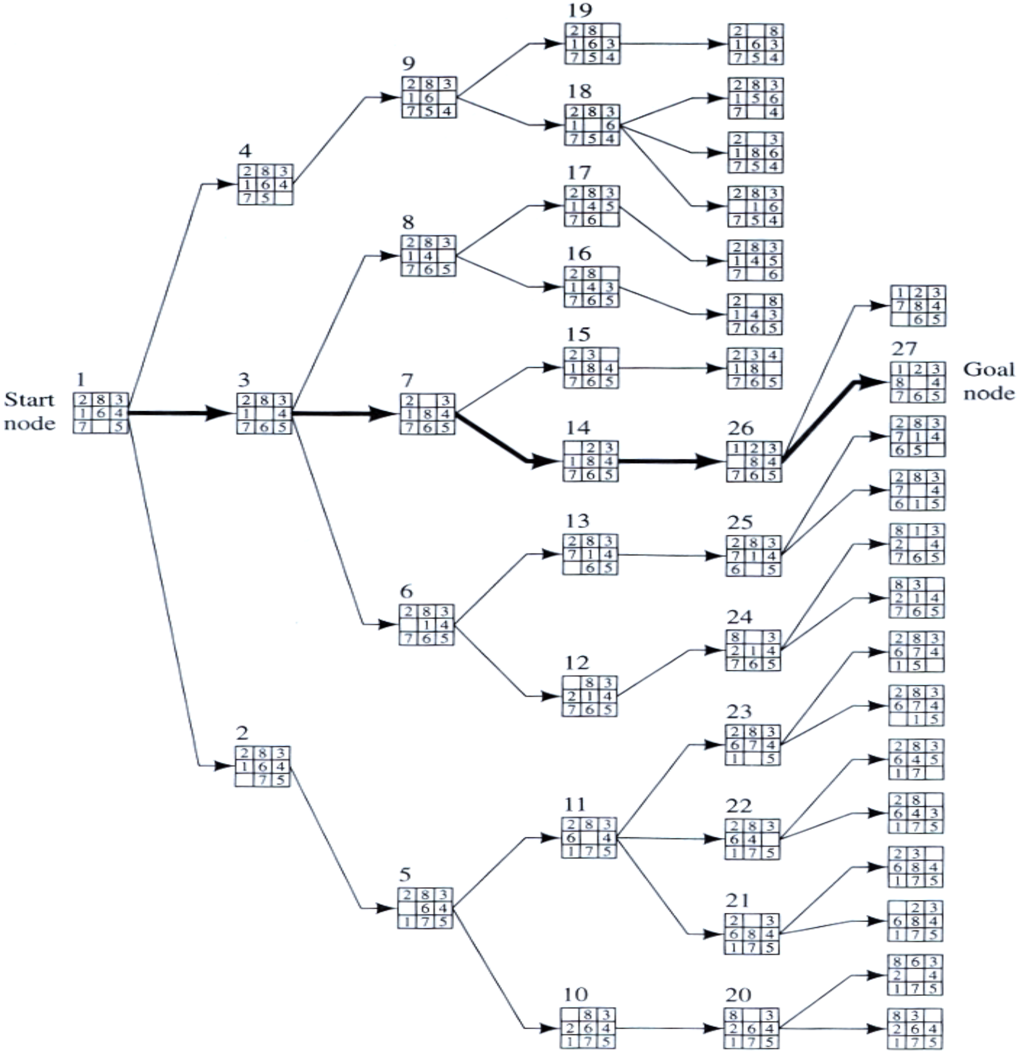
- Observable, start in #5.
  Solution? *[Right, Suck]*

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

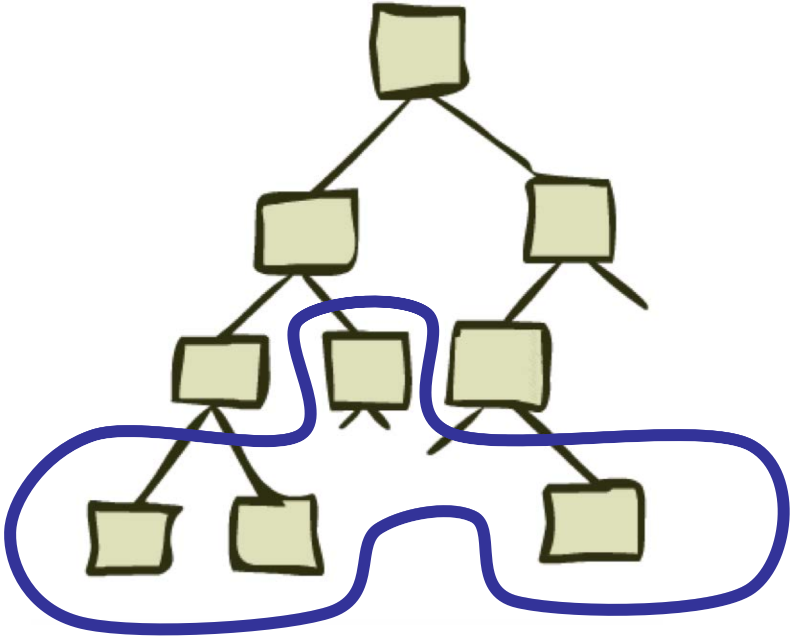
  [Right,Suck,Left,Suck]



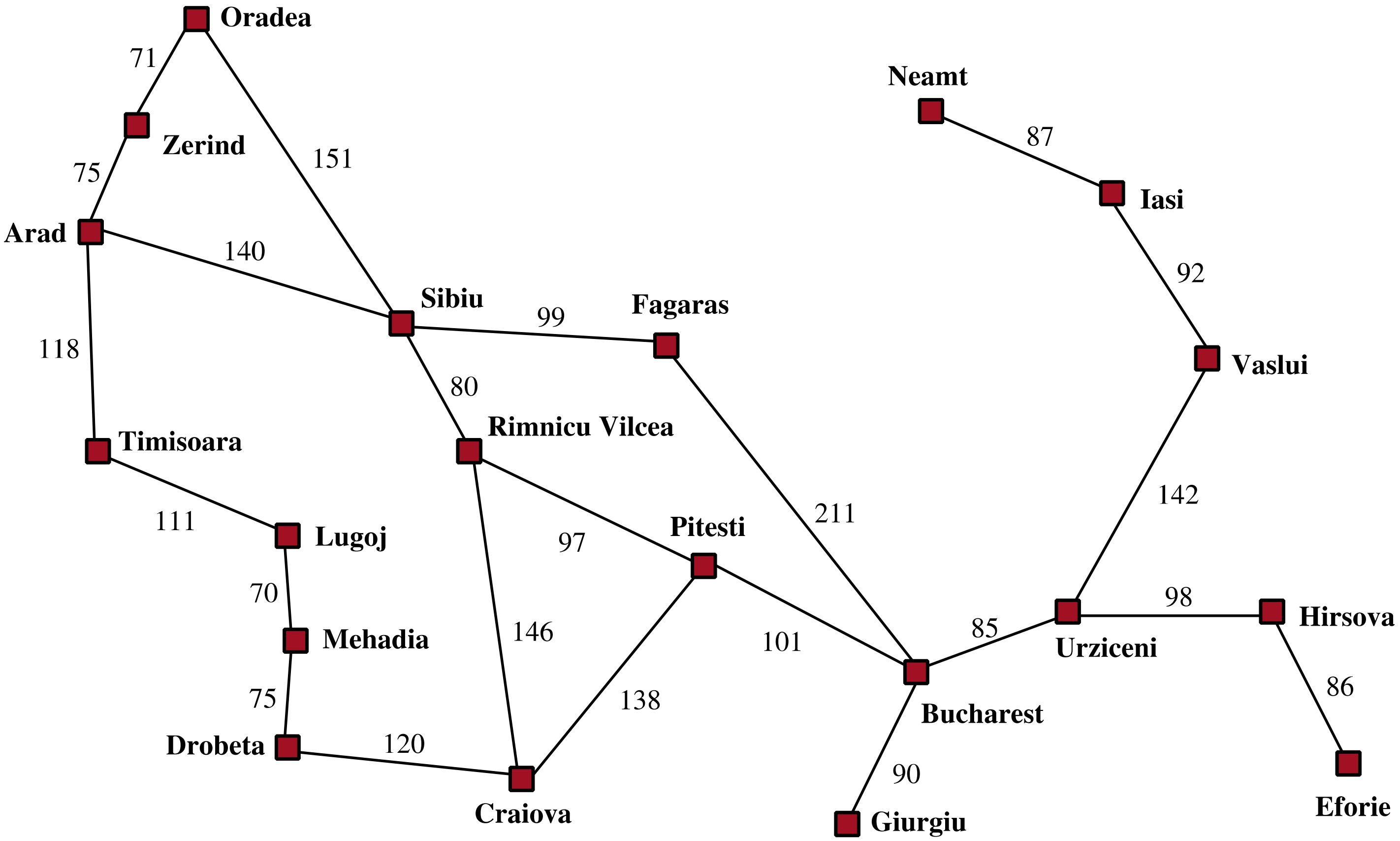
Example: The 8-puzzle



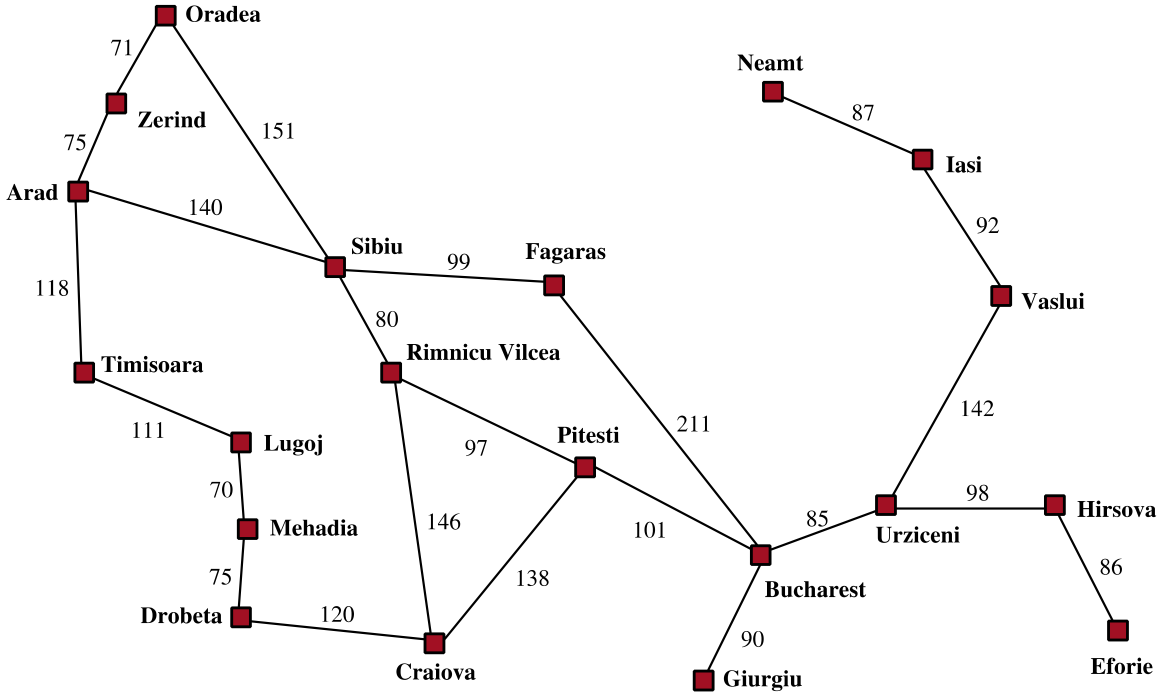
Tree Search



Search Example: Romania



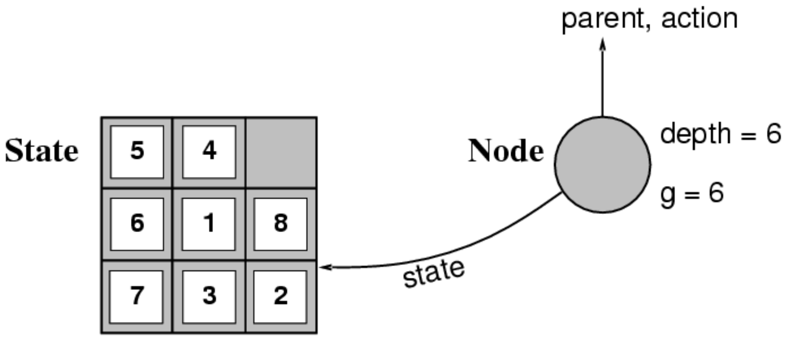
Example: Traveling in Romania



- 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?

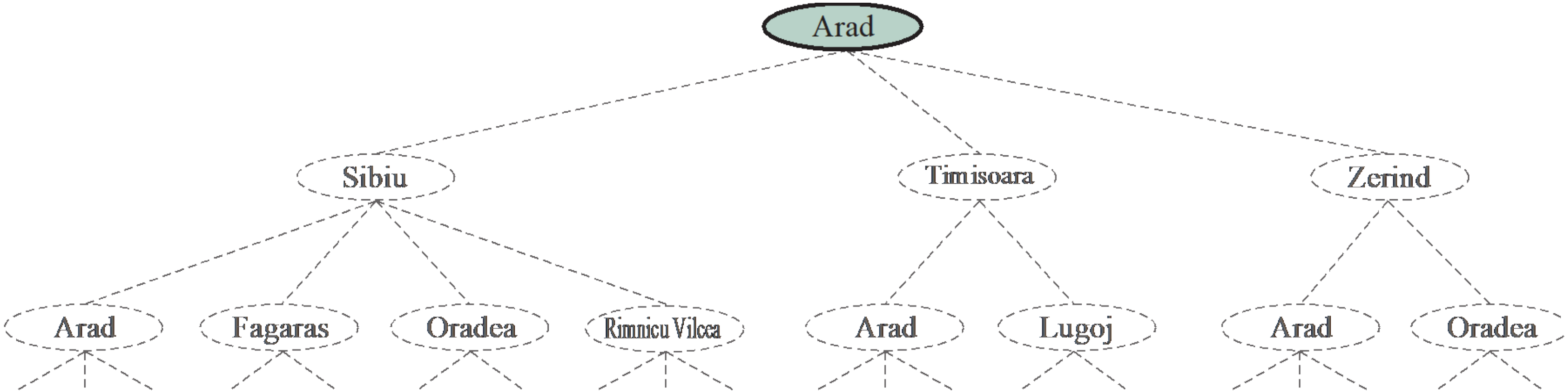
Implementation: states vs. nodes

- 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



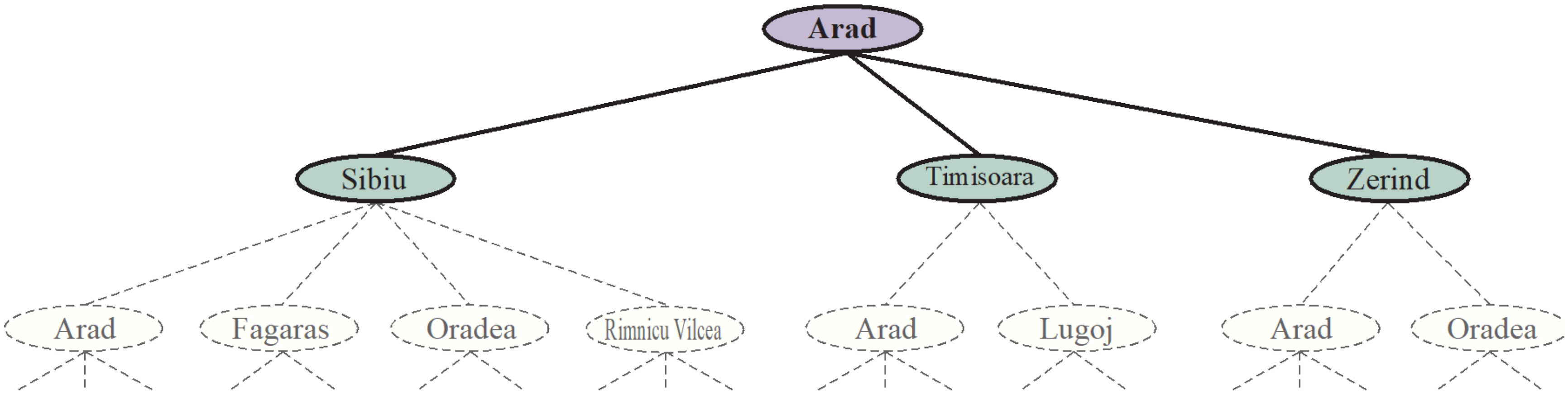
- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

Creating the search tree



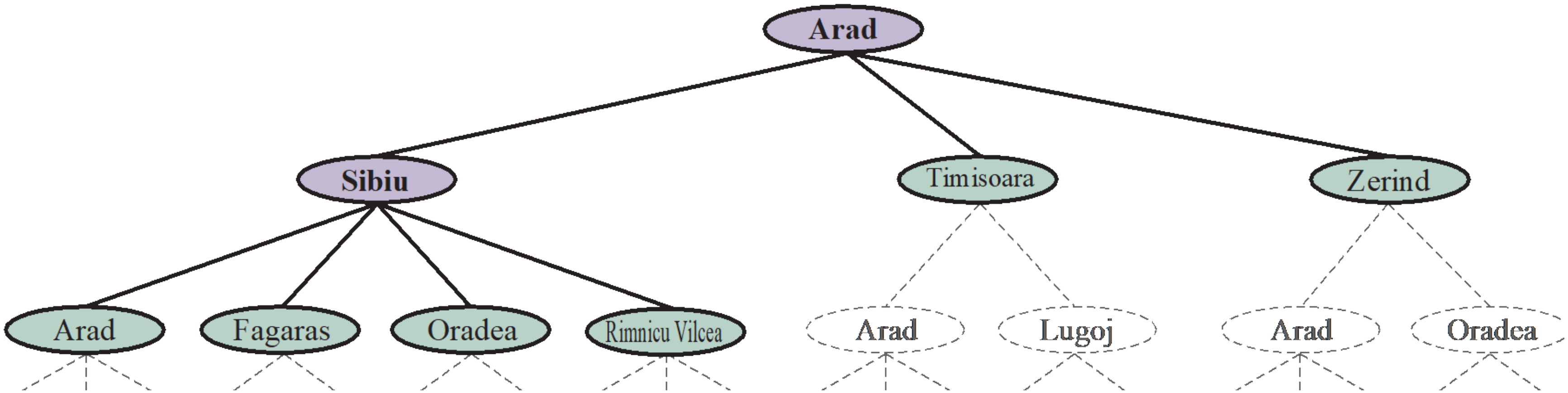
تابع Result را به حالت فعلی اعمال می‌کنیم تا مجموعه جدیدی از حالت‌ها (یعنی پسین‌ها) آشکار شود

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



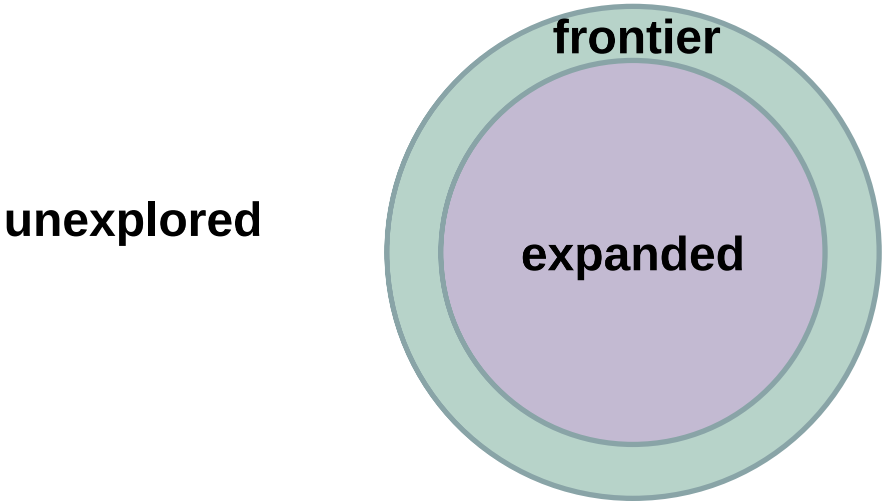
فرایند بسط و توسعه گره‌ها در مرز آنقدر ادامه می‌یابد تا به جوابی برسیم یا دیگر هیچ حالتی برای بسط وجود نداشته باشد

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



reached = expanded U frontier

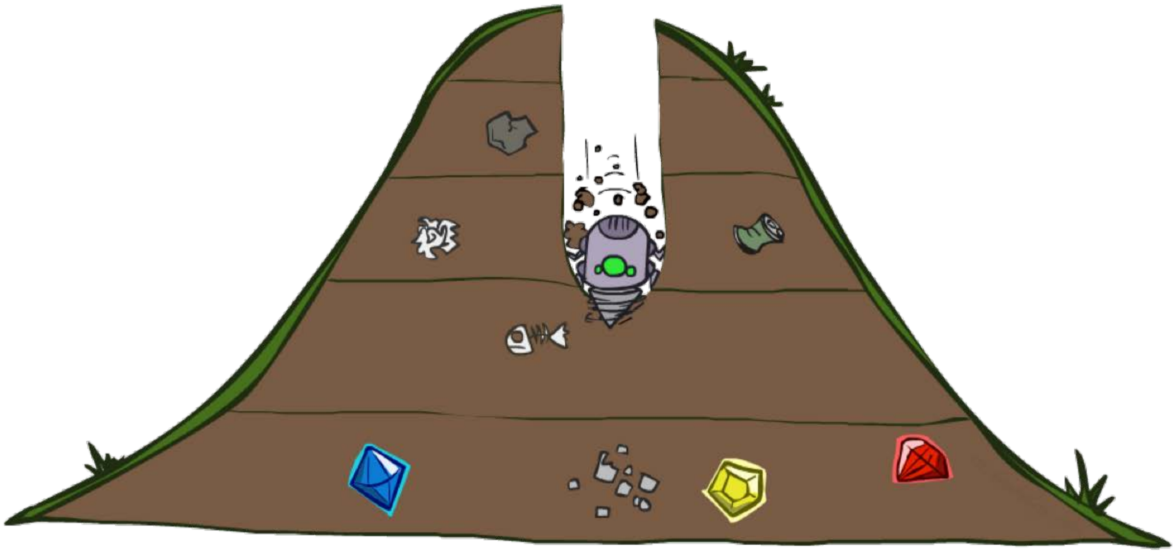
1. Frontier separates expanded from unexplored region of state-space graph

2. Expanding a frontier node:

   a. Moves a node from frontier into expanded

   b. Adds nodes from unexplored into frontier, maintaining property 1

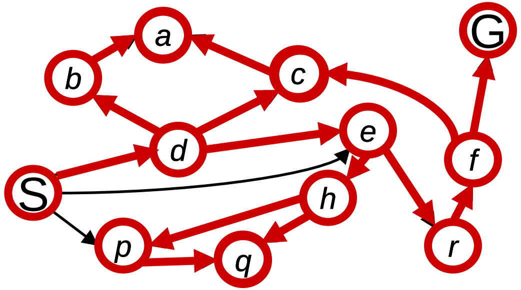
Depth-First Search

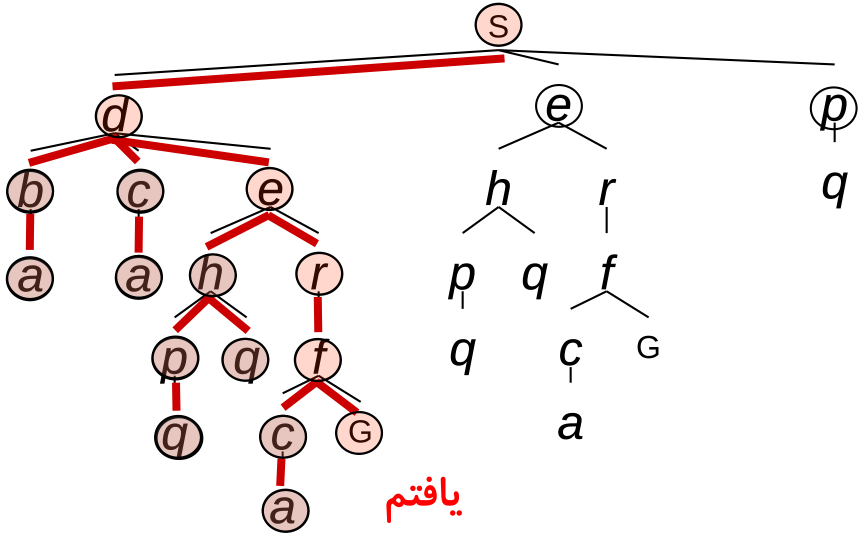


Depth-First Search

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

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

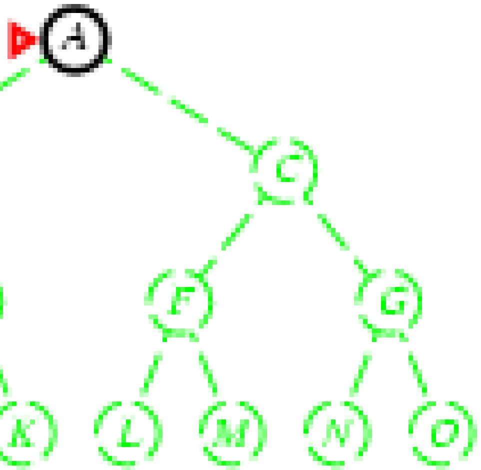




Depth-first search

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

Is A a goal state?

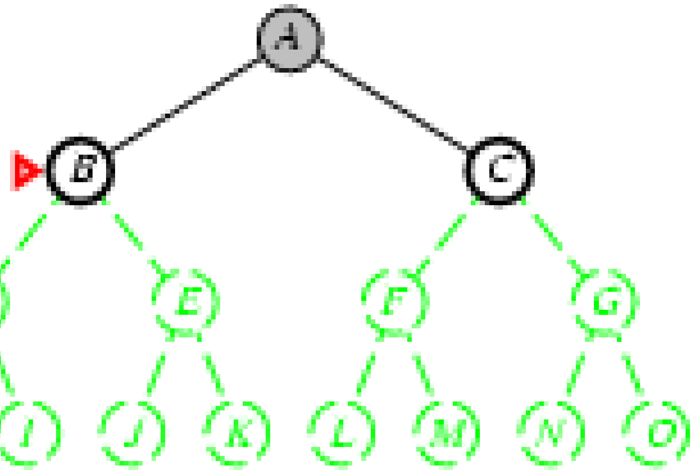


Depth-first search

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

queue=[B,C]

Is B a goal state?

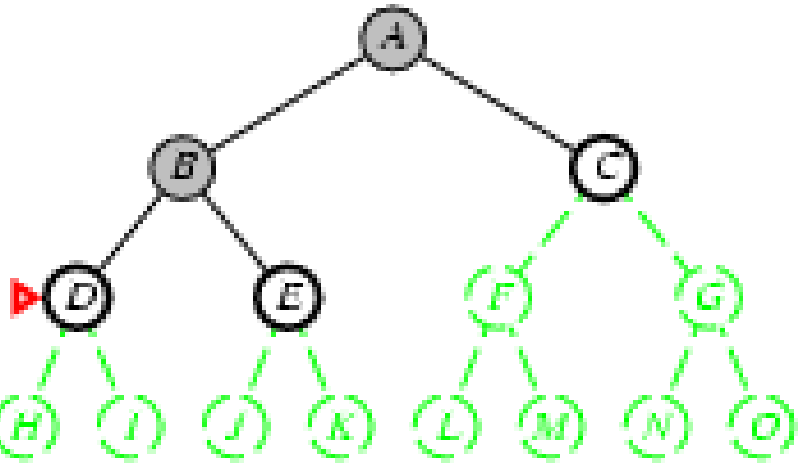


Depth-first search

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

queue=[D,E,C]

Is D = goal state?

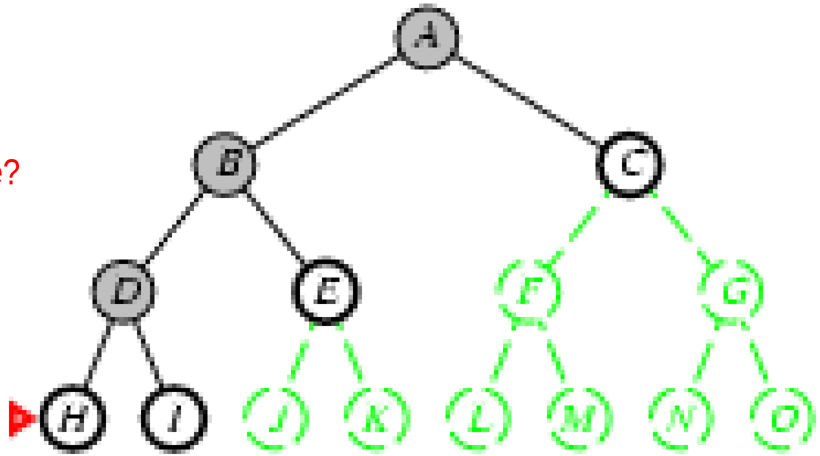


Depth-first search

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

queue=[H,I,E,C]

Is H = goal state?

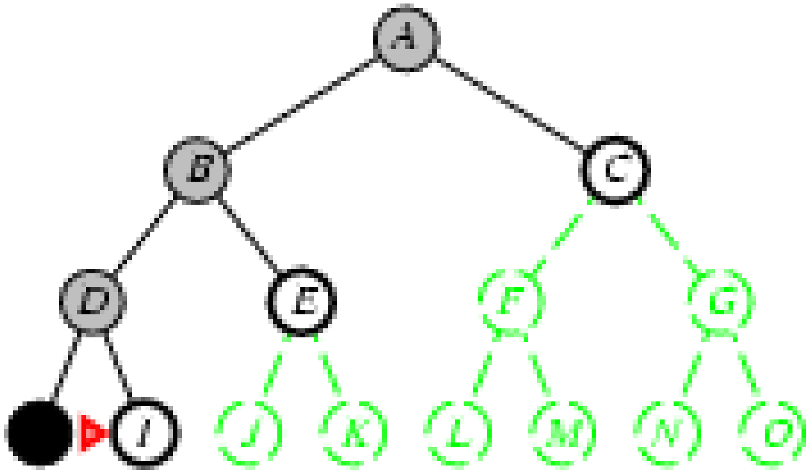


Depth-first search

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

queue=[I,E,C]

Is I = goal state?

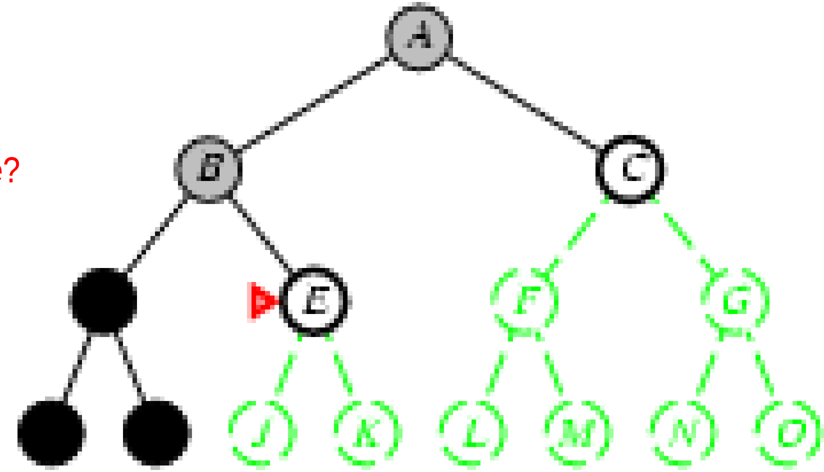


Depth-first search

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

queue=[E,C]

Is E = goal state?

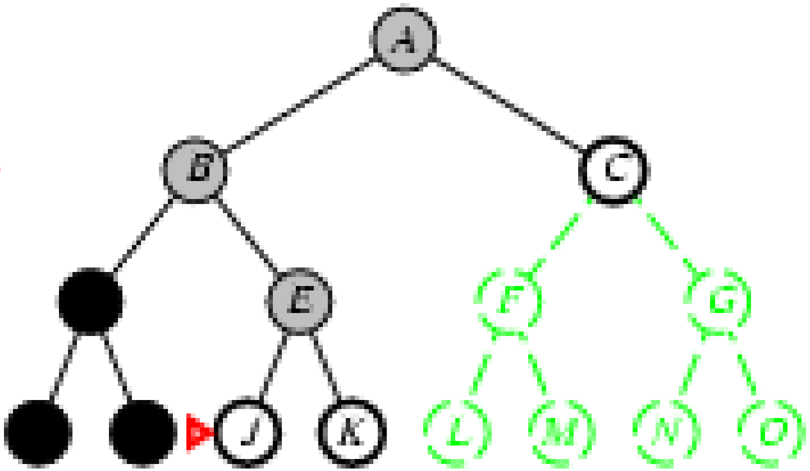


Depth-first search

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

queue=[J,K,C]

Is J = goal state?

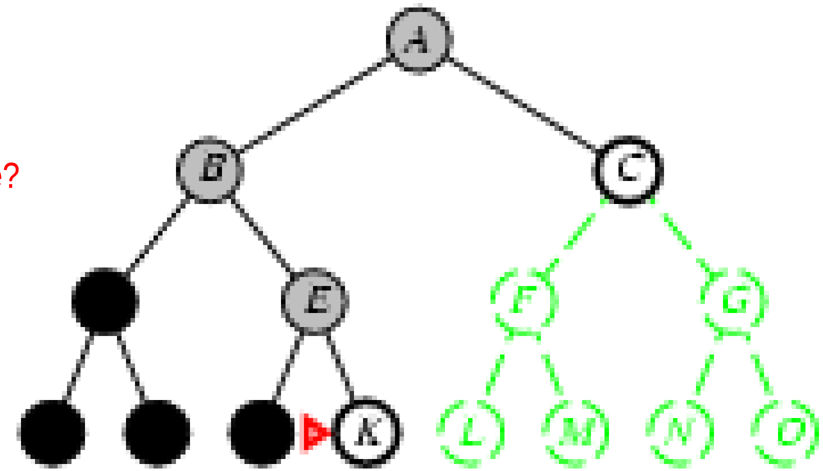


Depth-first search

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

queue=[K,C]

Is K = goal state?

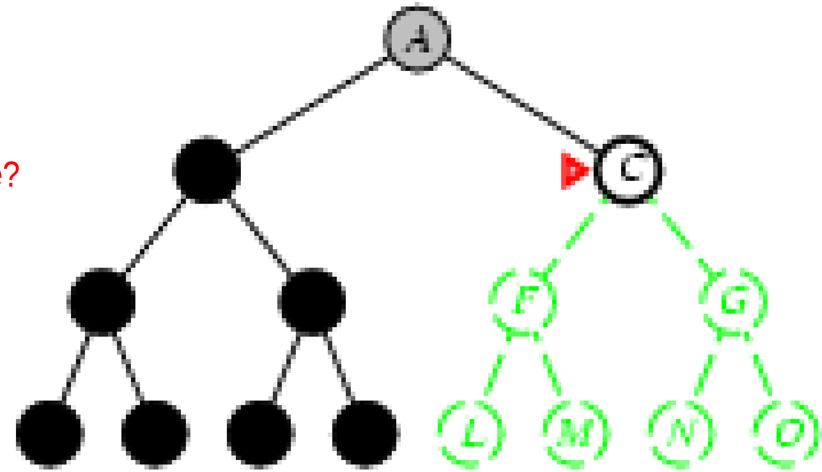


Depth-first search

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

queue=[C]

Is C = goal state?

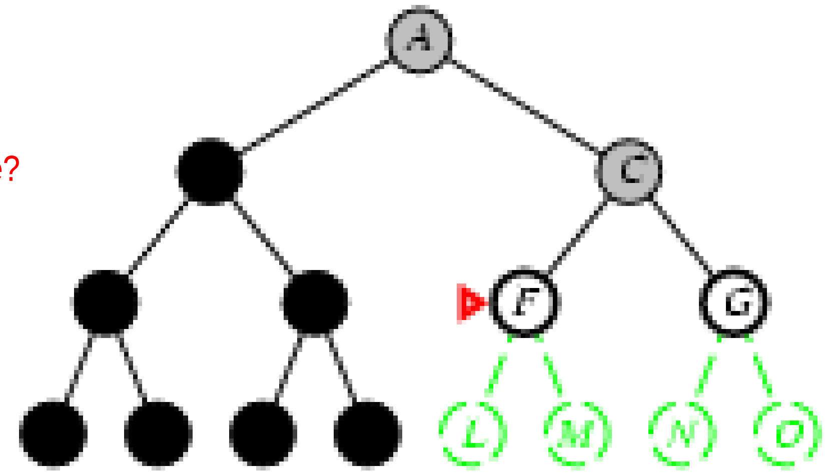


Depth-first search

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

queue=[F,G]

Is F = goal state?

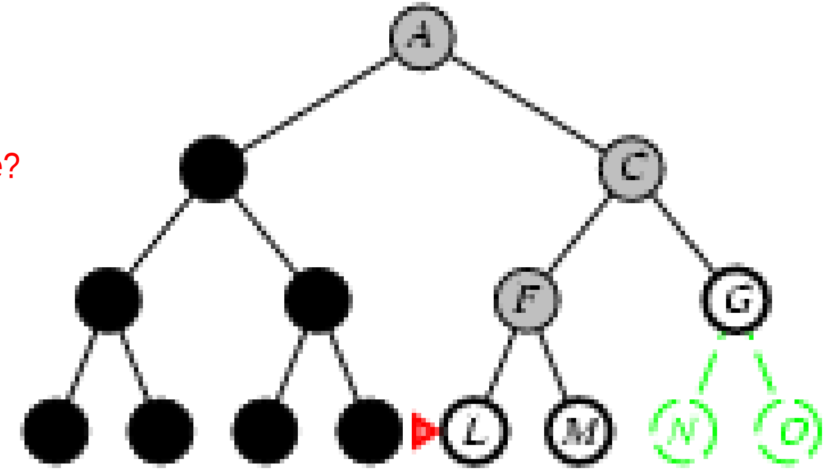


Depth-first search

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

queue=[L,M,G]

Is L = goal state?

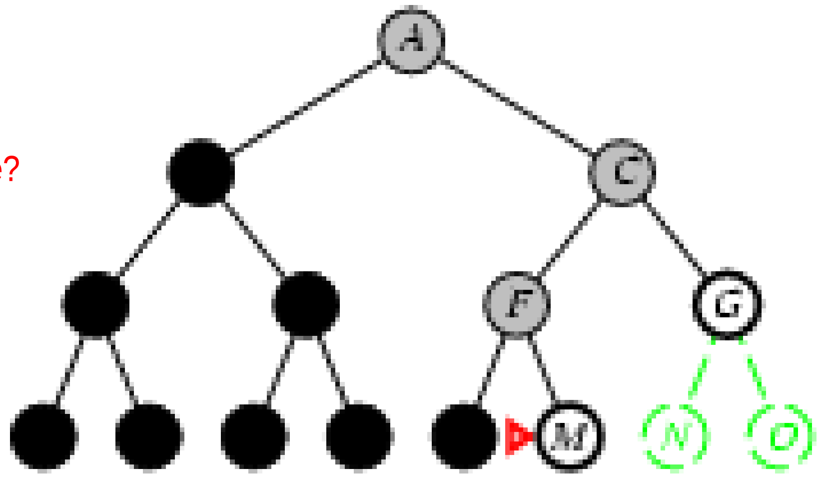


Depth-first search

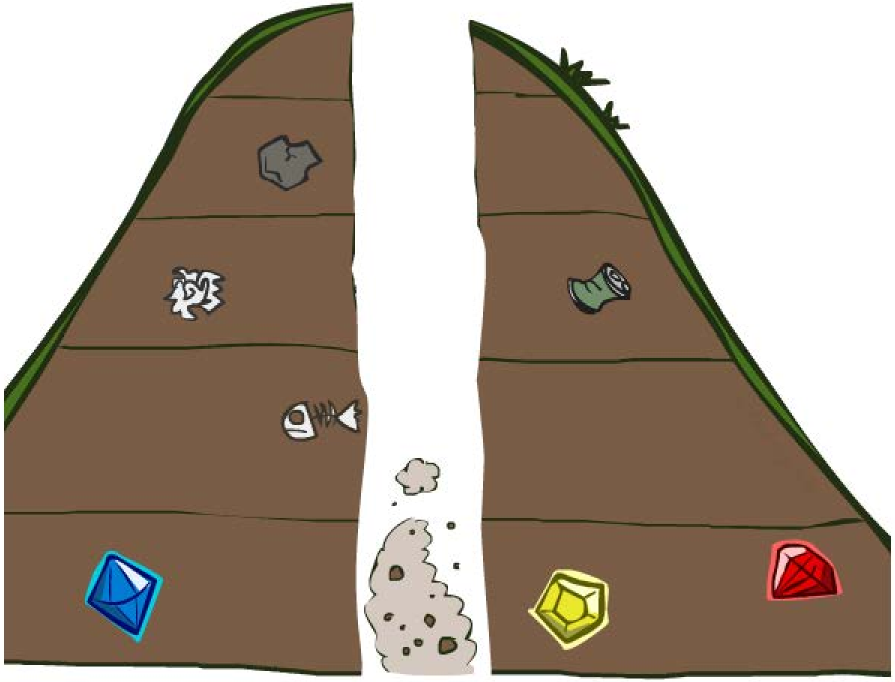
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

queue=[M,G]

Is M = goal state?



Search Algorithm Properties

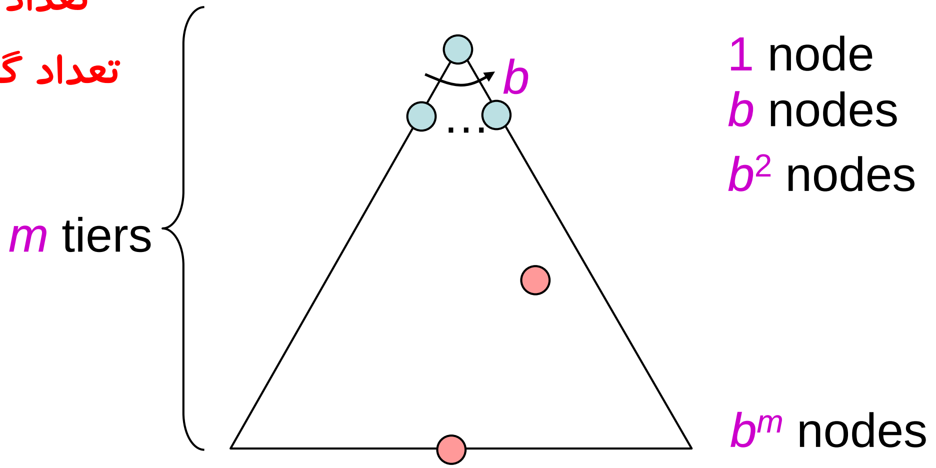


Search Algorithm Properties

- **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

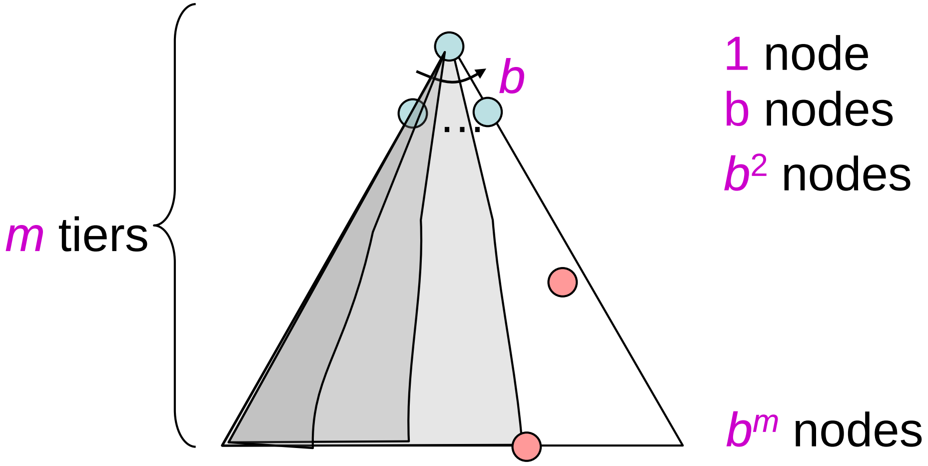
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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

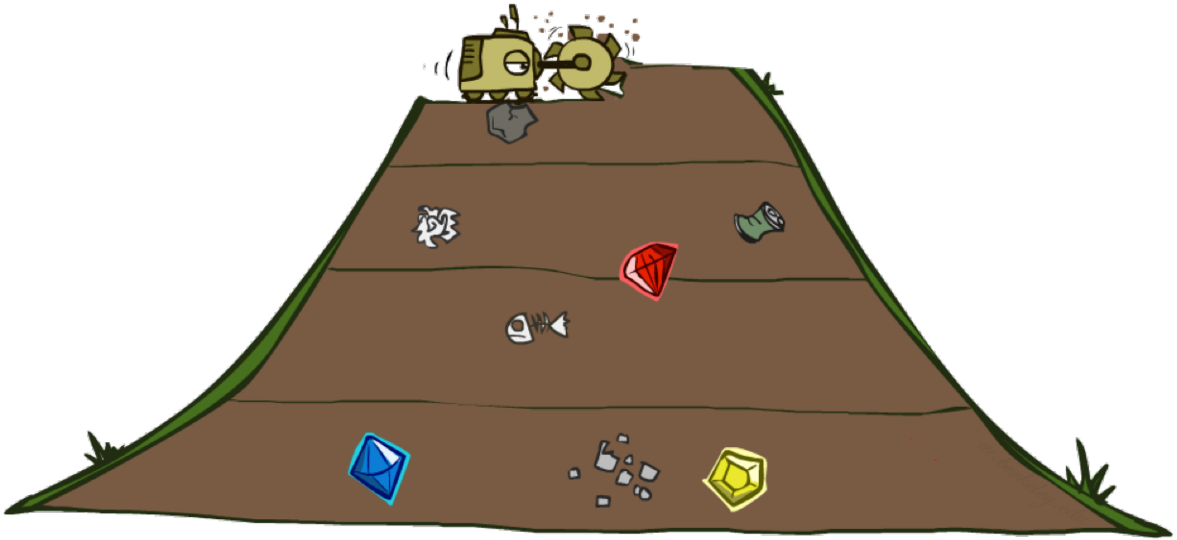


Depth-First Search (DFS) Properties

- 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



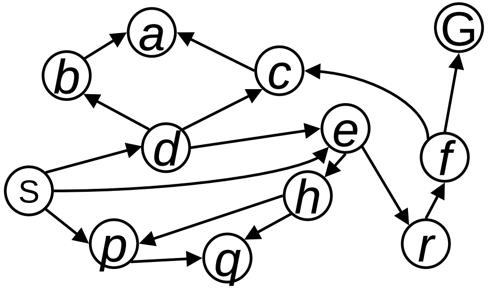
Breadth-First Search



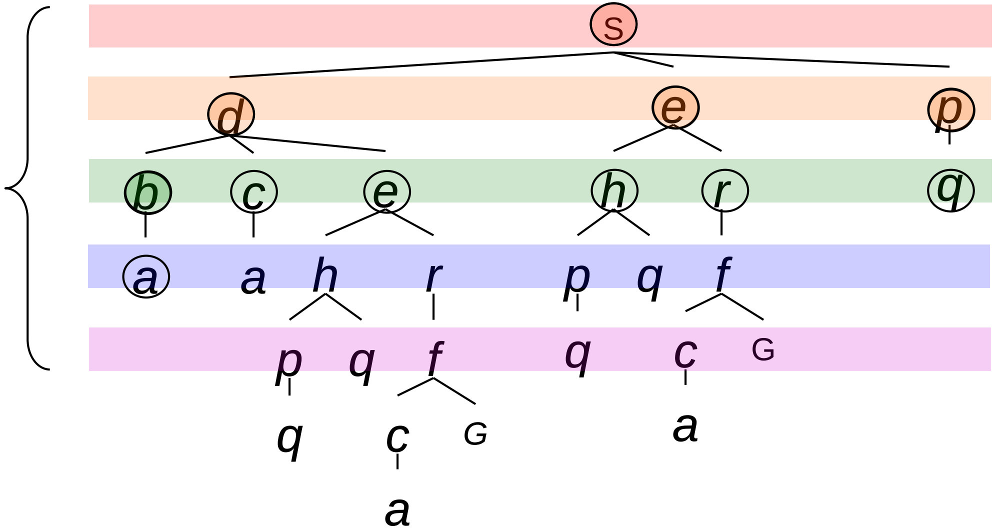
Breadth-First Search

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

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



Search Tiers

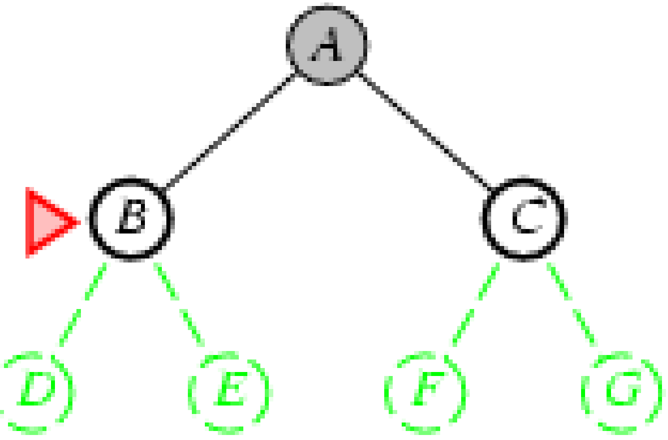


Breadth-first search

- 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?

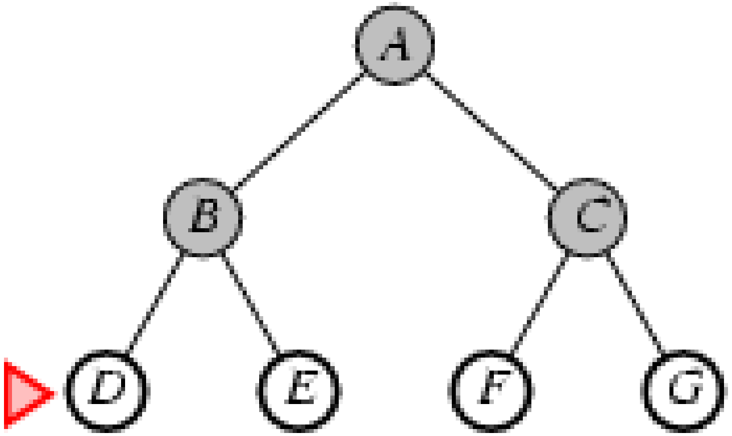


Breadth-first search

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

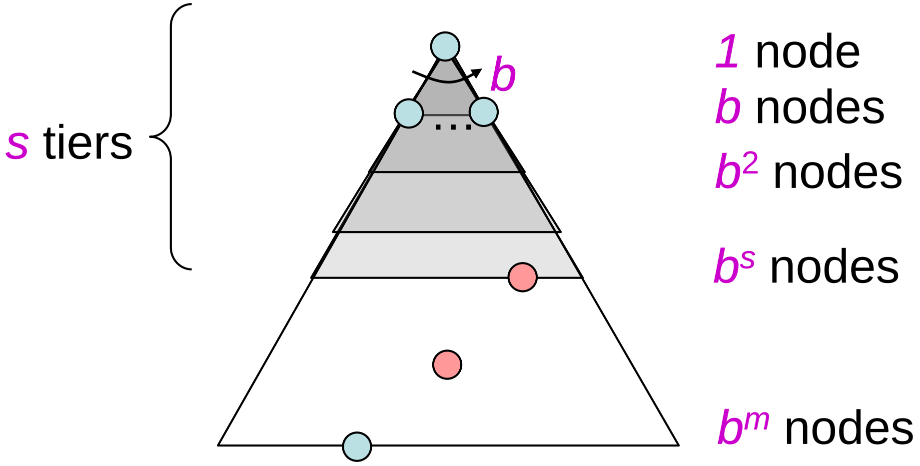
Expand:
fringe=[D,E,F,G]

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)

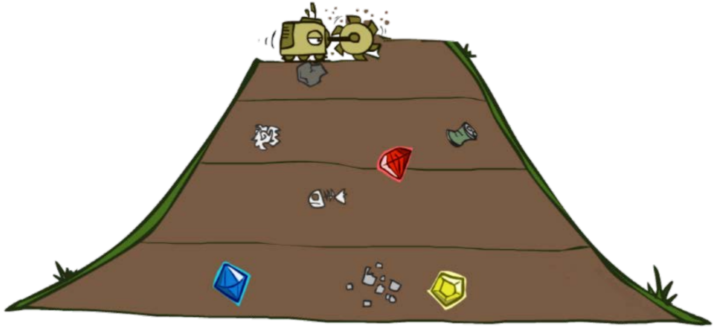


General Tree Search

```
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)

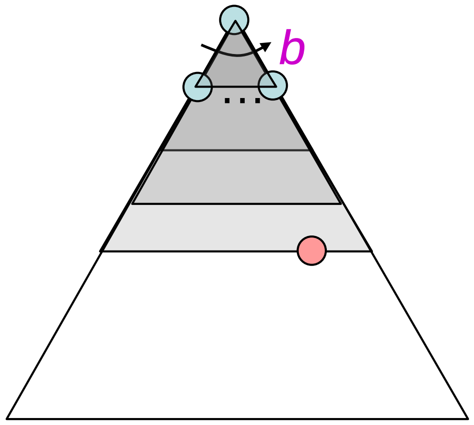
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

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.

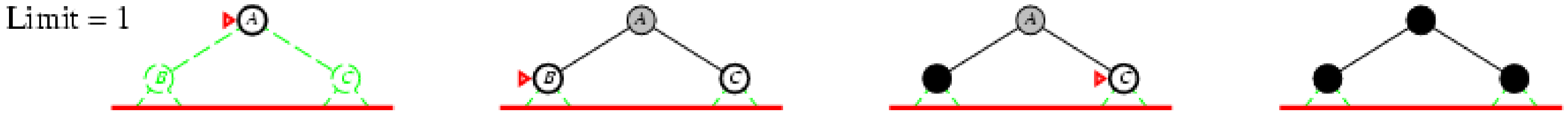
Iterative Deepening

- 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. …..

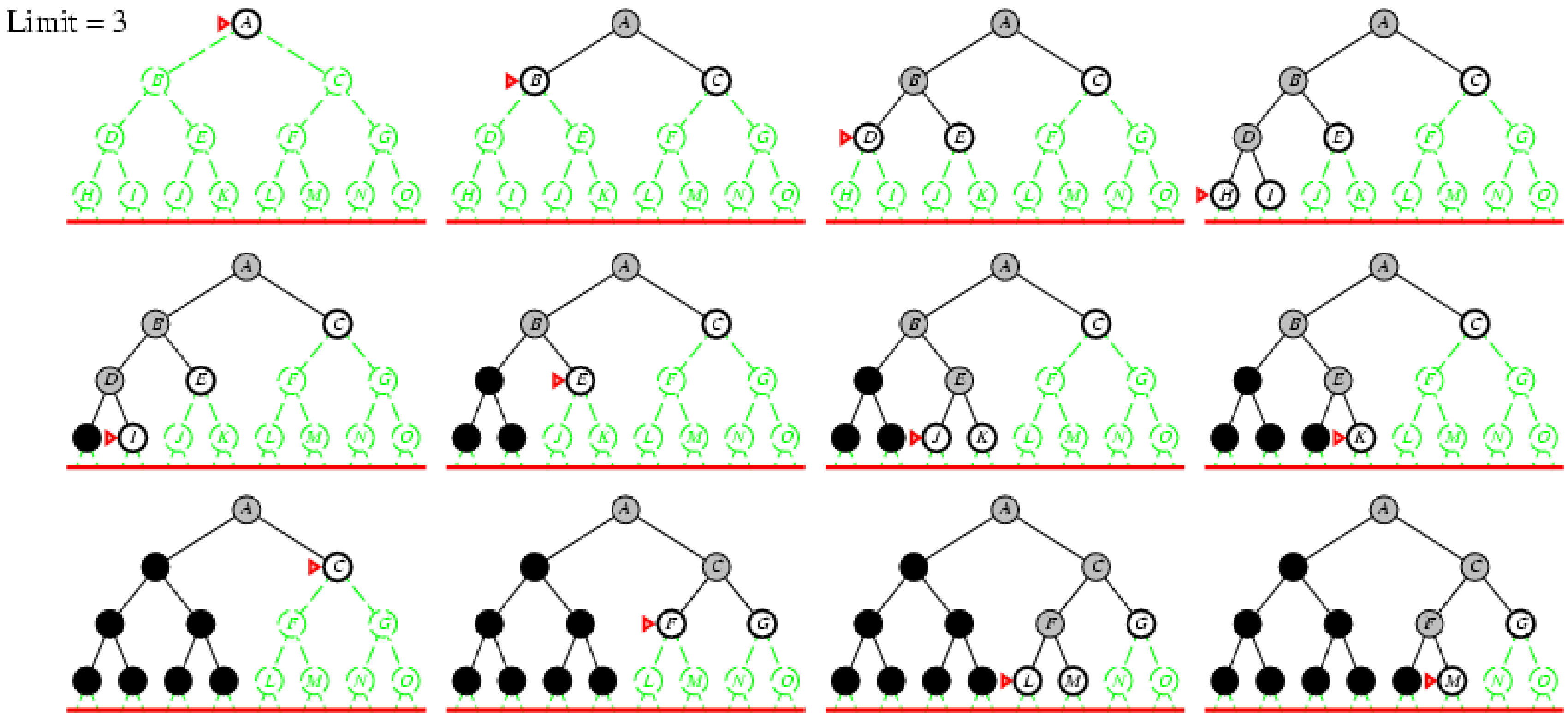
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



Iterative deepening search L=1

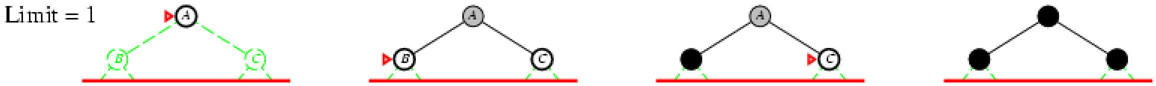


Iterative Deepening Search *L*=3

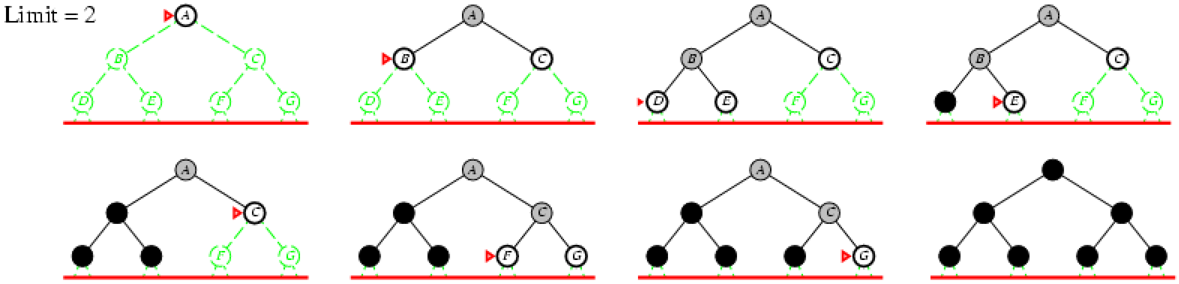


Iterative deepening search $L=0$

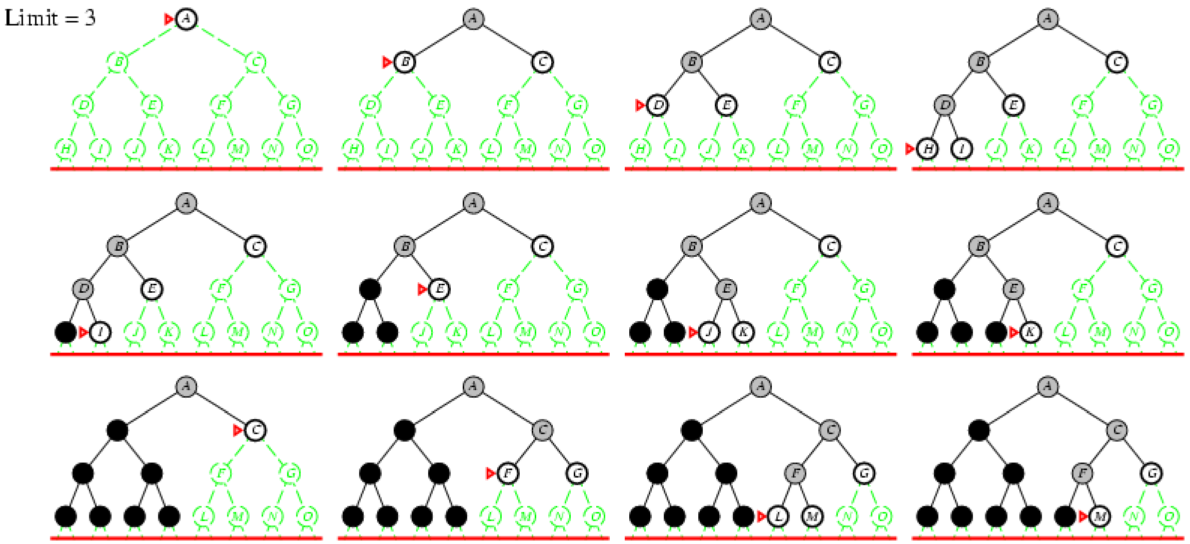
Iterative deepening search L=1



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*:

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

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

BFS

- 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} = ... = 1,111,100

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

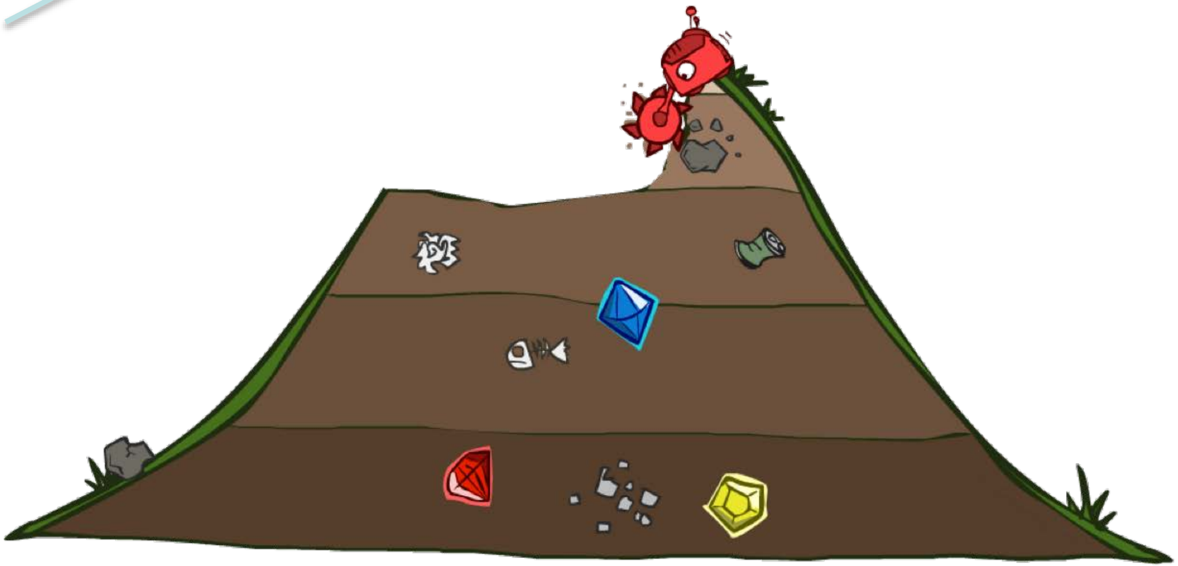
Uniform Cost Search

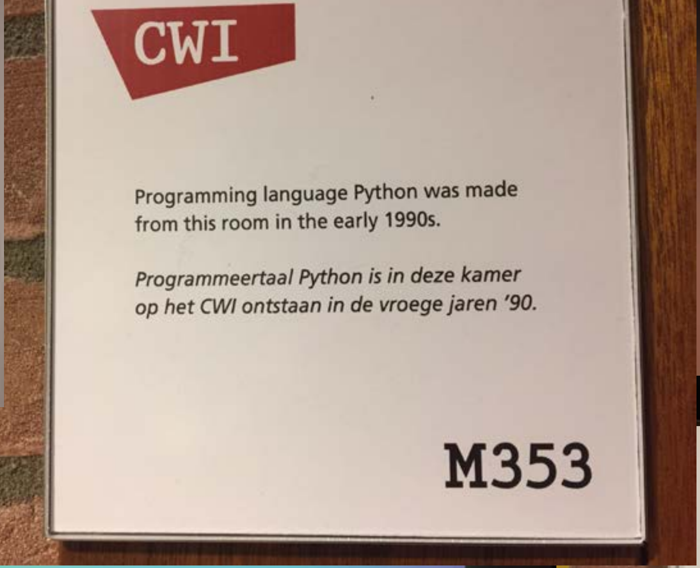
By the AI community

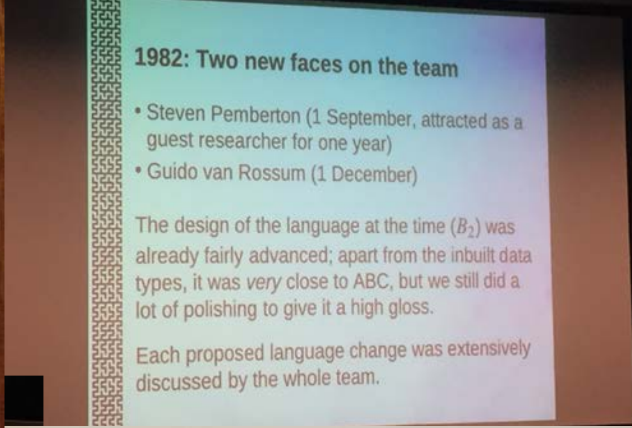
or

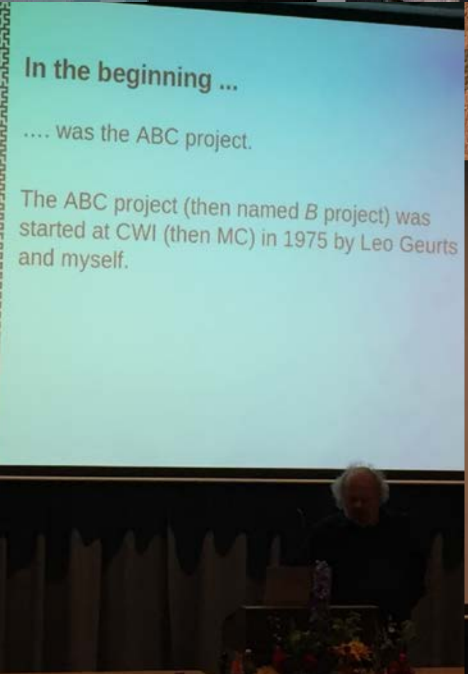
Dijkstra's algorithm

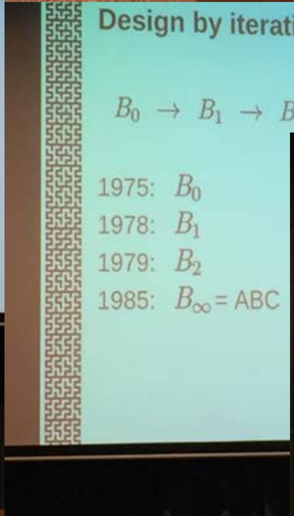
By the theoretical computer science community

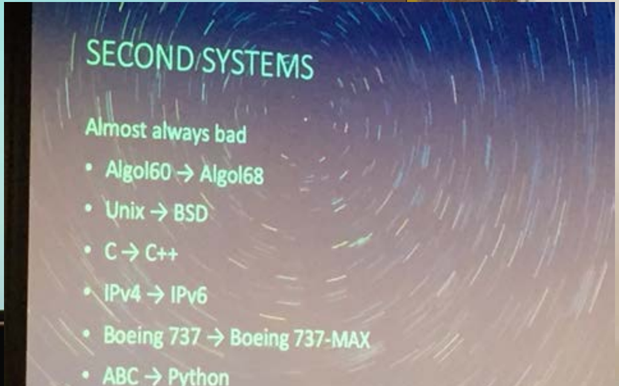



CWI
Programming language Python was made from this room in the early 1990s.
Programmeertaal Python is in deze kamer op het CWI ontstaan in de vroege jaren '90.
M353


1982: Two new faces on the team
• Steven Pemberton (1 September, attracted as a guest researcher for one year)
• Guido van Rossum (1 December)
The design of the language at the time (B_2) was already fairly advanced; apart from the inbuilt data types, it was very close to ABC, but we still did a lot of polishing to give it a high gloss.
Each proposed language change was extensively discussed by the whole team.


In the beginning ...
.... was the ABC project.
The ABC project (then named B project) was started at CWI (then MC) in 1975 by Leo Geurts and myself.


Design by iteration
$B_0 \rightarrow B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_\infty$.
1975: B_0
1978: B_1
1979: B_2
1985: B_∞ = ABC


SECOND SYSTEMS
Almost always bad
• Algol60 → Algol68
• Unix → BSD
• C → C++
• IPv4 → IPv6
• Boeing 737 → Boeing 737-MAX
• ABC → Python


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).

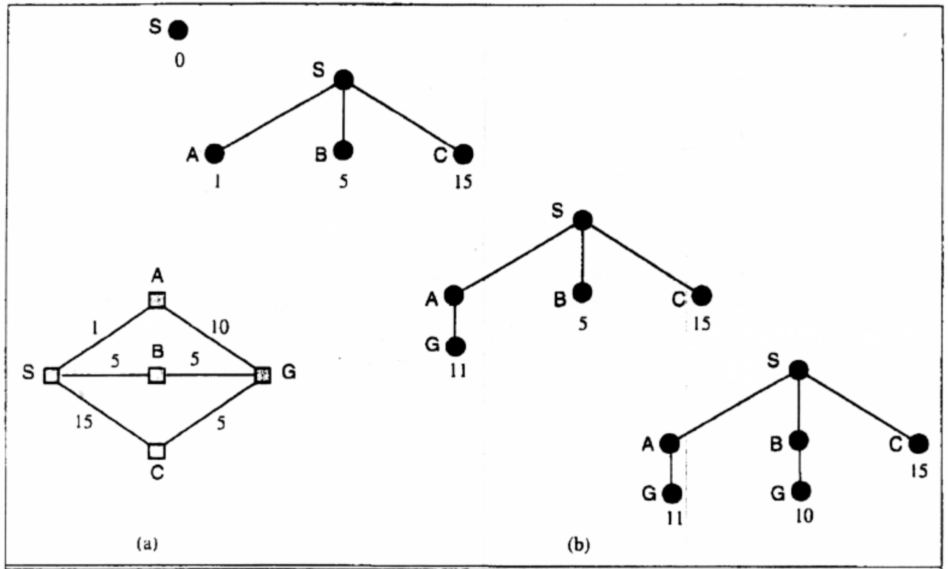


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Proof Completeness:

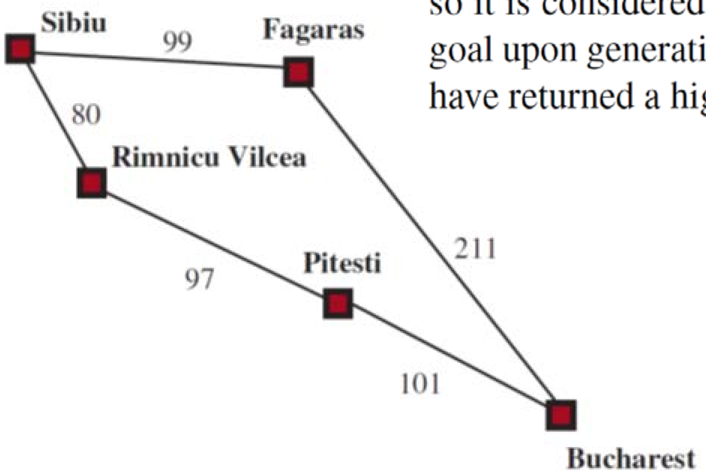
Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach it.

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).



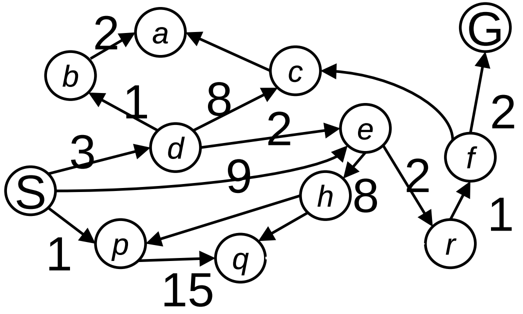
جستجوی عرضی بلافاصله بعد از تولید هدف متوقف می‌شود در حالی که جستجو با هزینه یکنوا تمام گره‌های موجود در عمق هدف را بررسی می‌کند تا گره‌ای با هزینه کمتر بیابد؛ یعنی با توسعه غیرضروری در عمق d کار بیشتری انجام می‌دهد.

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**

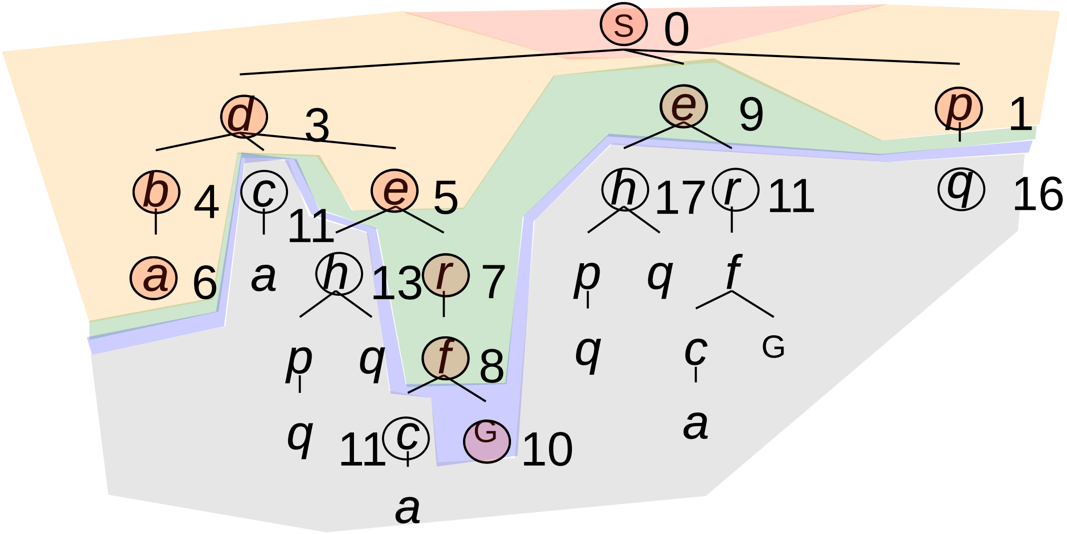
Strategy: expand ***lowest g(n)***



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

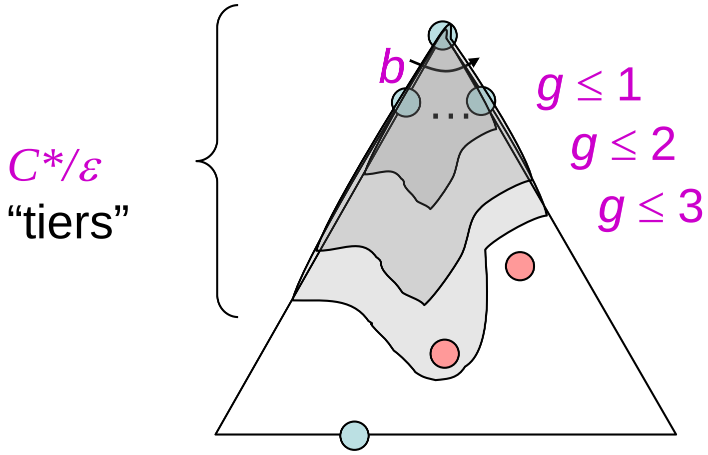
Frontier is a ***priority queue sorted by g(n)***

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)



- 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

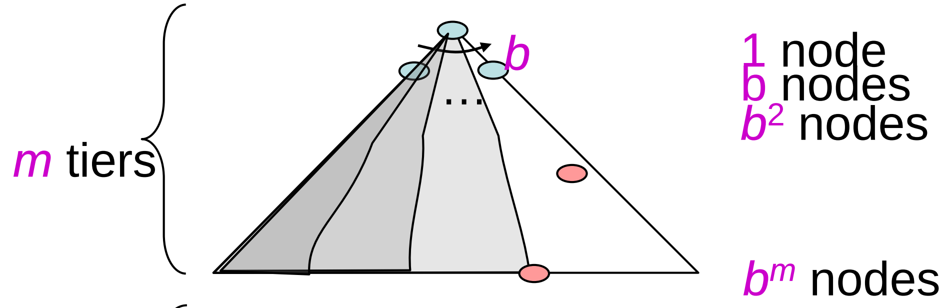
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$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

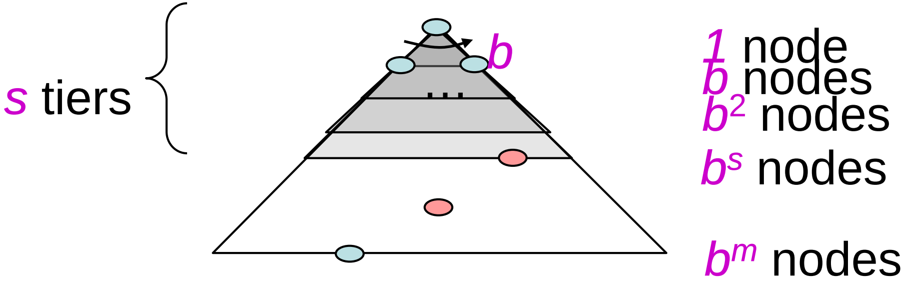
even complete if step cost is not increasing with depth.

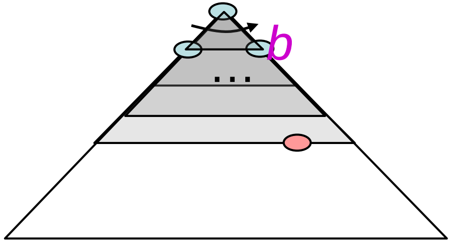
preferred uninformed search strategy

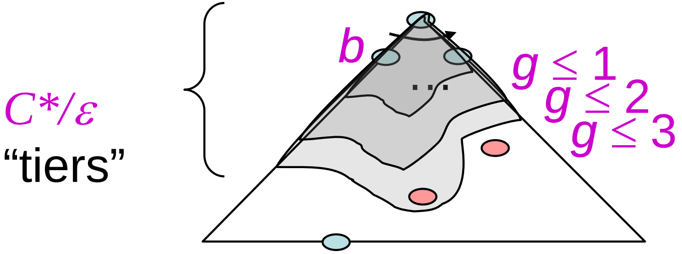
Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either 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









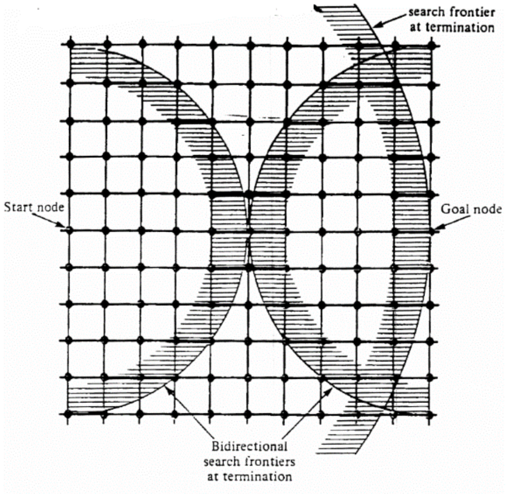


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

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: $O(b^{d/2})$

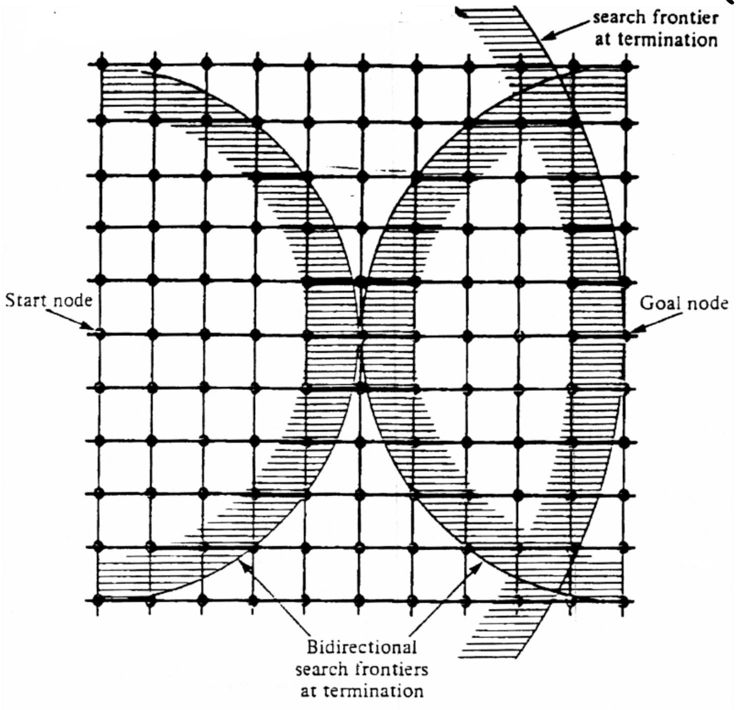


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

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)

با سپاس از توجه شما

دانشجوی گرامی