



Introduction To Artificial Intelligence

Isfahan University of Technology (IUT)
1402



Search

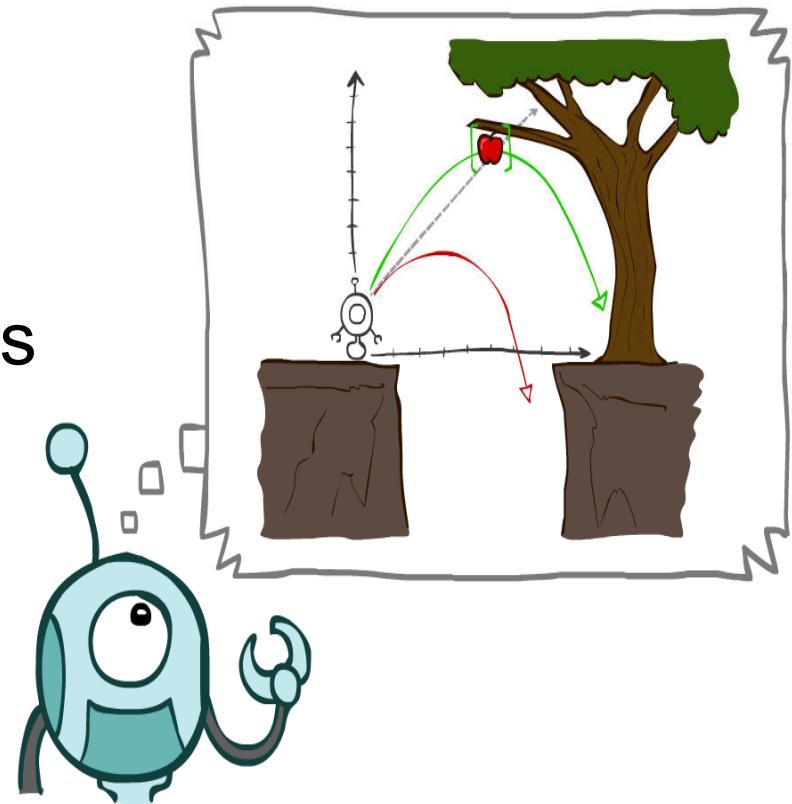
Dr. Hamidreza Hakim
hamid.hakim.u@gmail.com

[These slides were created by Dan Klein and Pieter
Abbeel for CS188 Intro to AI at UC Berkeley.]

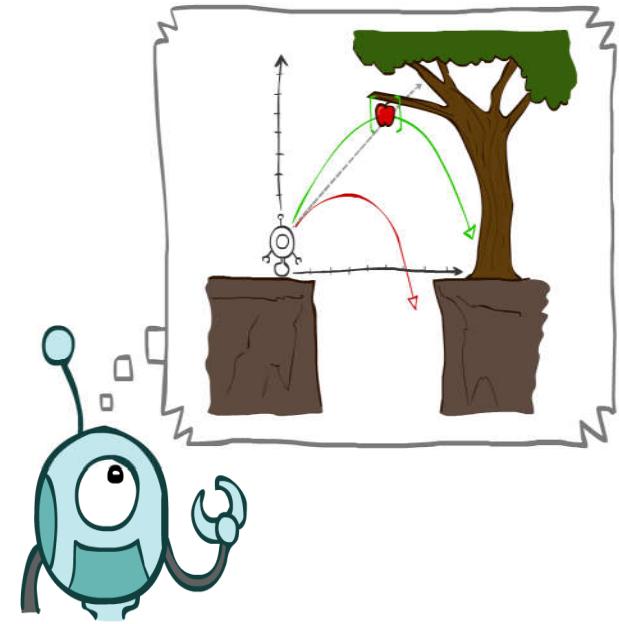
بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

Today

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



- عامل برای بعضی از تصمیم هاش میاد سرچ میکنه

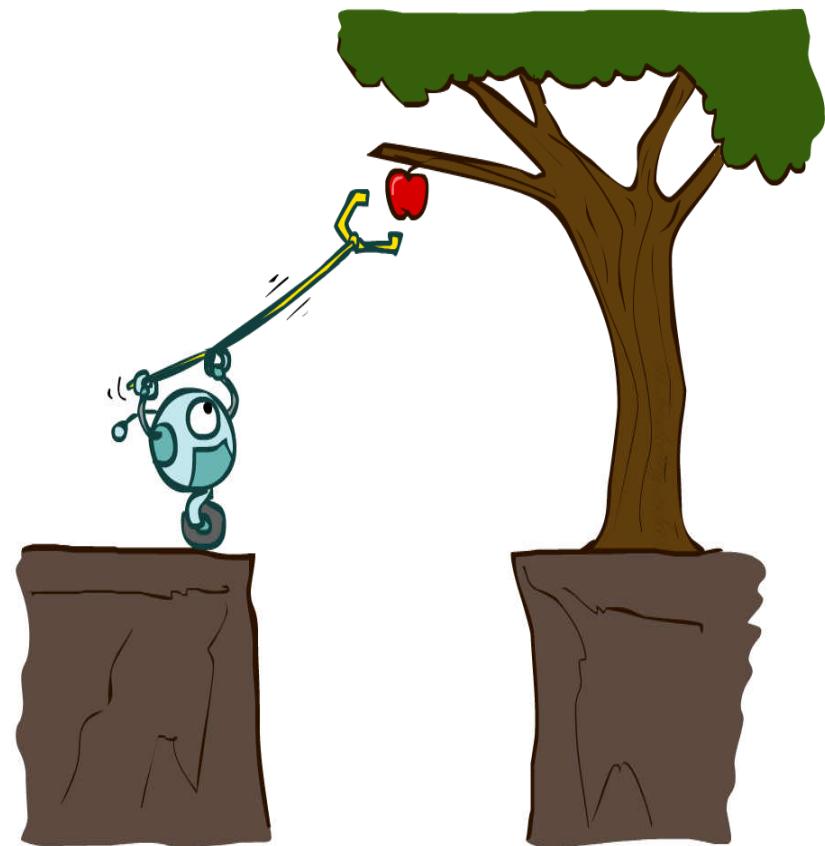


AGENTS THAT PLAN

عواملی که برنامه ریزی می کنند

Planning Agents

- Planning agents decide based on evaluating future action sequences
- Search algorithms typically assume
 - Known, deterministic transition model
 - Discrete states and actions
 - Fully observable
 - Atomic representation
- Usually have a definite goal
- Optimal: Achieve goal at least cost
- Planning vs. replanning (Demo)



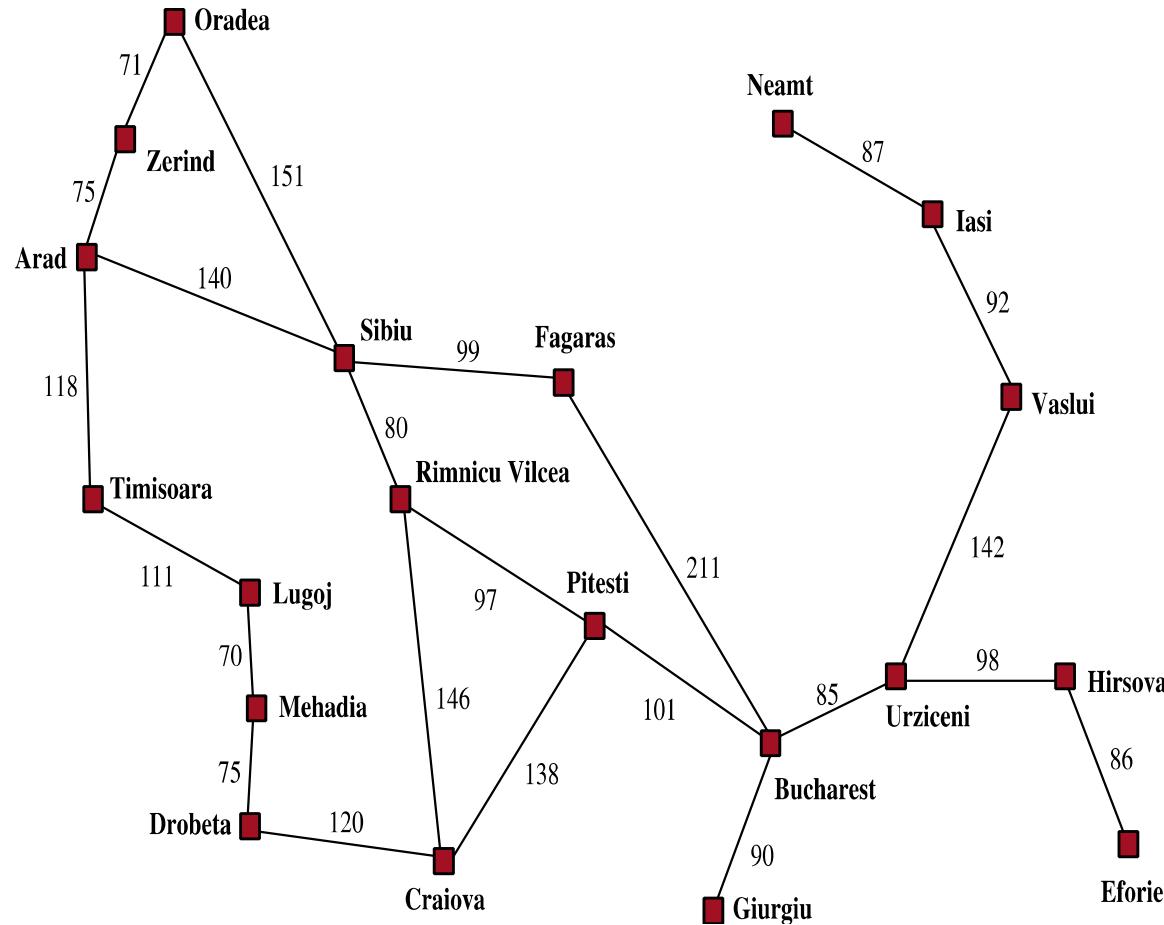
PROBLEM-SOLVING AGENTS

Example: Traveling in Romania



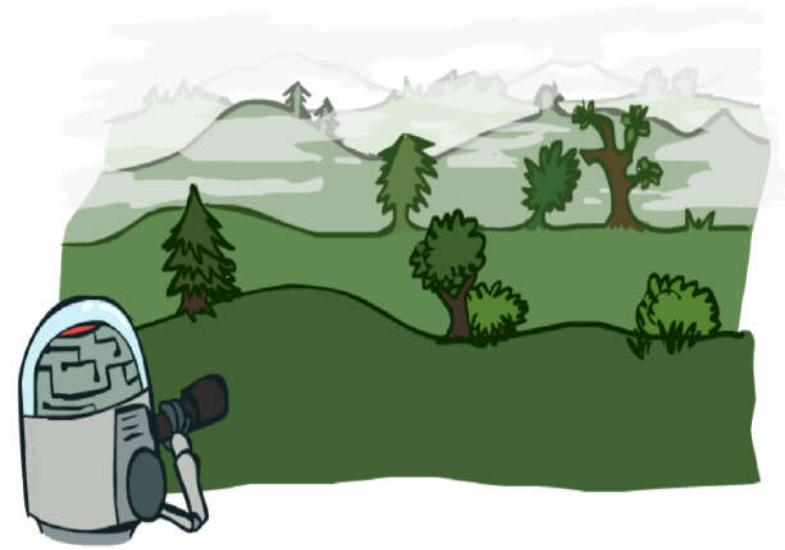
مثال: مسافرت در رومانی

Problem (travel from Arad to Bucharest)



Problem-Solving Agents

- agents always have access to information about the world
- With that information, the agent can follow four-phase problem-solving process
 - Goal formulation
 - Problem formulation
an abstract model of the relevant part of the world
 - Search
Simulation
 - Execution
- Open-loop/Close-Loop

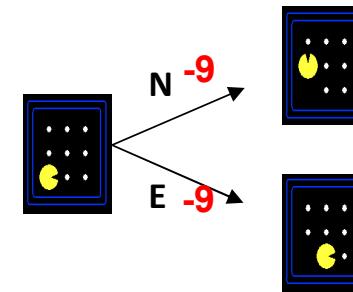
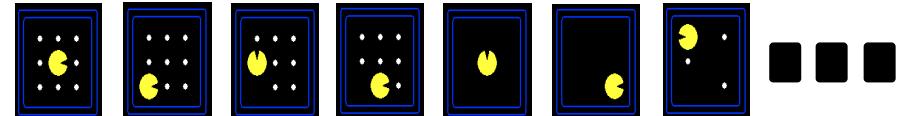


SEARCH PROBLEMS

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 $\text{Result}(s,a)$
- A **goal test** $G(s)$
 - ◆ s has no dots left
- Action cost $c(s,a,s')$ (Rationality!)
 - ◆ +1 per step; -10 food; -500 win; +500 die; -200 eat ghost



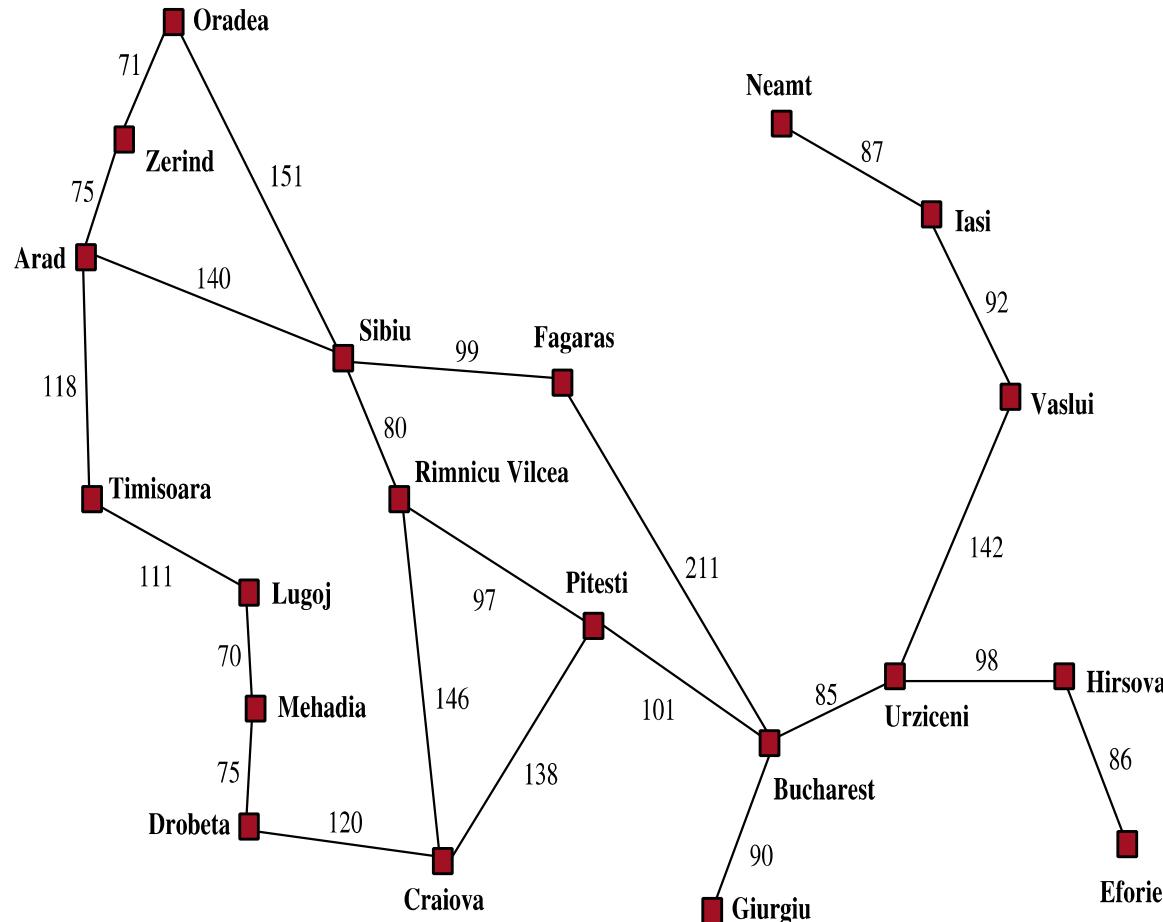
- A **solution** is an action sequence that reaches a goal state
- An **optimal solution** has least cost among all solutions

- مسئله سرچ چندتا قسمت داشت:

- 1- استیت استیت بود
- 2- حالت شروع
- 3- اکشن ها مثلا بالا بره و سمت راست بره
- 4- اگر این اکشن رو انجام دادیم وقتی این حالت بودیم به چه حالت دیگه ای می رسیم:
Transition model
- 5- به حالت هدف رسیدیم یا نه
- 6- هزینه؟؟؟؟؟

هدفمون پیدا کردن رسیدن به اون گل استیت است توی یک وضعیت optimal

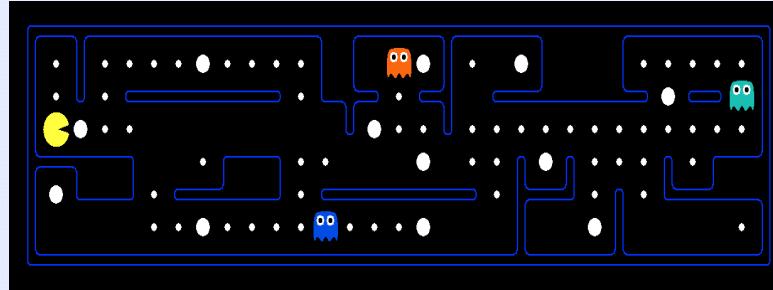
Example: Traveling in Romania



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

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 model:
update location only
 - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
 - States: ??

- زمین خوار از یک نقطه به یک نقطه دیگه منتقل میشے هدف اینه که به مقصد برسیم استیت ها:

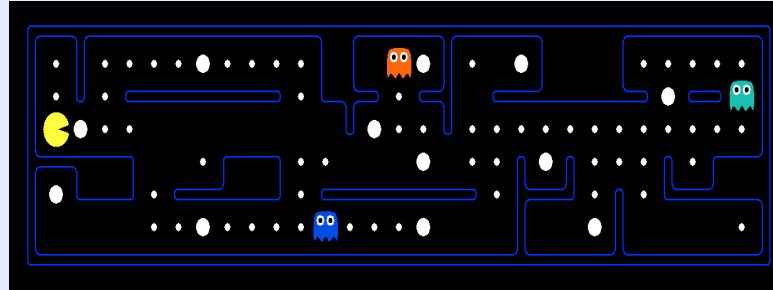
موقعیت هایی که داریم
اکشن: جهت هاش

فرض میکنیم الان هدف عوض بشه و این عامل بخواهد همه نقاط رو بخوره
نفهمیدم چی شد؟؟؟

- یک مسئله داری چند جزء می باشد:
 - مجموعه حالاتی که محیط می تواند در آن وضعیت باشد: فضای حالت
 - حالت اولیه
 - مجموعه اعمال ممکن (عملگرها)
 - مدل انتقال: توصیفی از آنچه هر عمل انجام می دهد.
 - Result(s, a)
 - مدل انتقال به همراه حالت اولیه و مجموعه اعمال فضای حالت را تشکیل می دهد.
 - نمایش فضای حالت با یک گراف
 - هدف
- هزینه انجام هر عمل در هر حالت Action-Cost(s, a, s')

What's in a State Space?

The **world state** includes every last detail of the environment

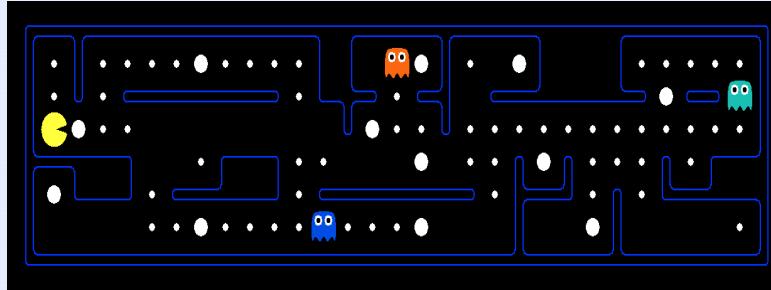


A **search state** keeps only the details needed for planning (abstraction)

- Problem: Pathing
 - States: (x,y) location
 - Actions: NSEW
 - Transition model: update location only
 - Goal test: is $(x,y)=\text{END}$
- Problem: Eat-All-Dots
 - States: $\{(x,y), \text{food locations}, \text{wall locations}, \text{ghost locations}, \text{etc.}\}$

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 model: update location only
 - Goal test: is $(x,y)=\text{END}$
- Problem: Eat-All-Dots
 - States: $\{(x,y), \text{dot booleans}\}$
 - Actions: NSEW
 - Transition model: update location and possibly a dot boolean
 - Goal test: dots all false

What's in a State Space?

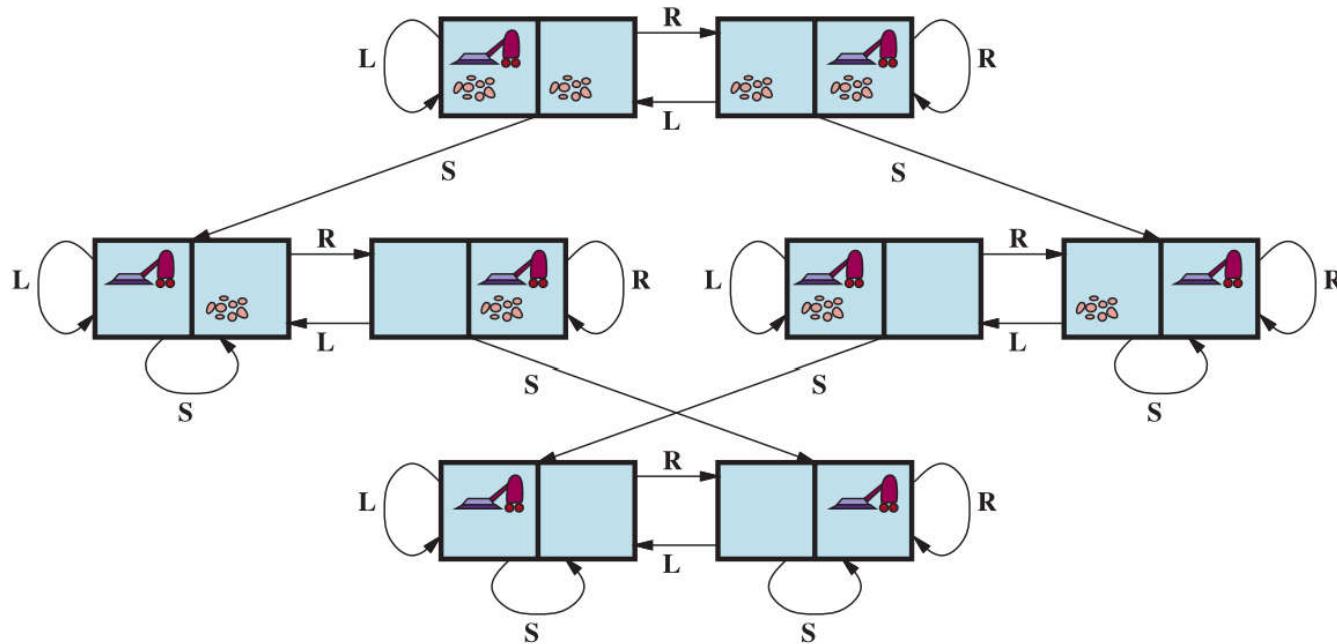


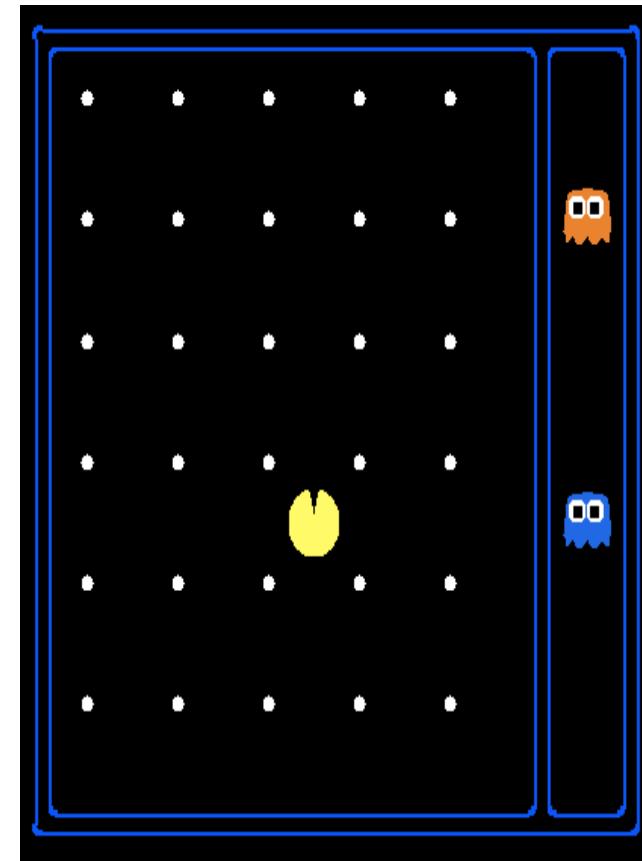
Figure 3.2 The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = Left, R = Right, S = Suck.

استیت: خود جاروبرقی کجاست؟ خونه اوله یا خونه دوم - خونه اول کثیفه یا .. - ...
نفهمیدم

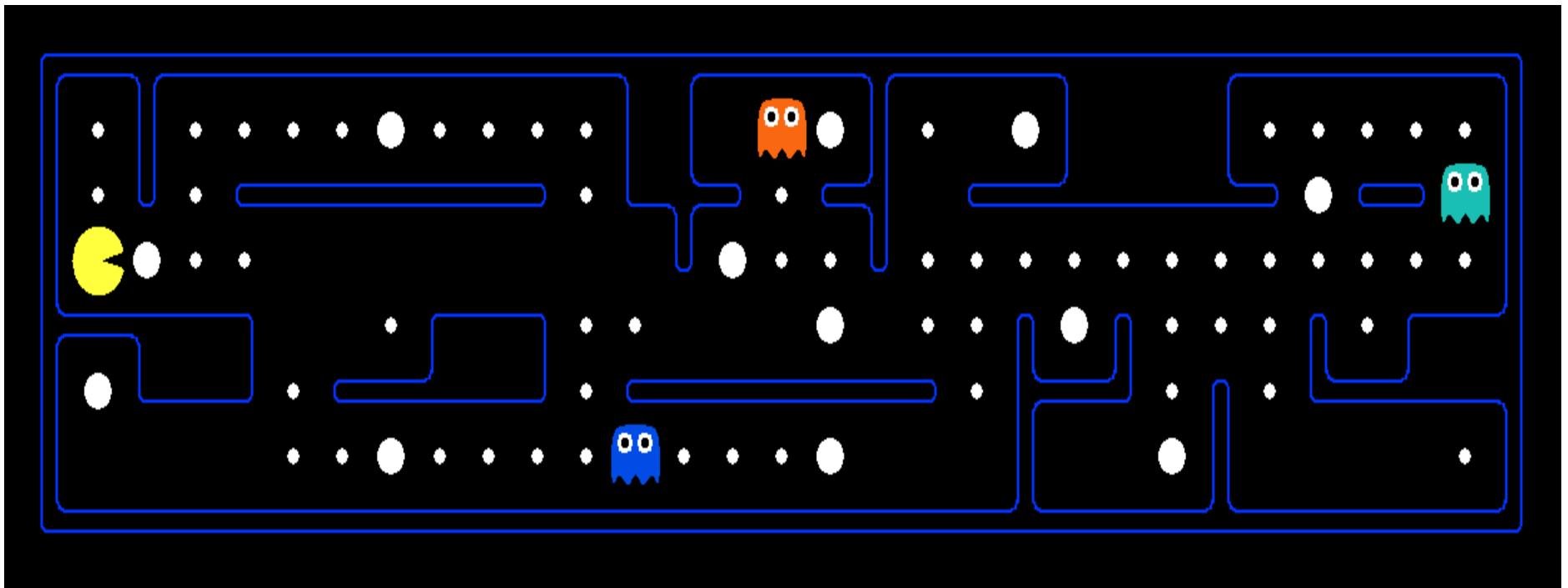
- حالات: ۸ حالت
- حالت اوّلیه: هر یک از ۸ حالت ممکن
- اعمال: چپ، راست، مکش
- مدل انتقال: اعمال کار مورد نظر شان را انجام می دهند بجز رفتن به چپ اگر در خانه چپ باشد، رفتن به راست اگر در خانه راست باشد، و مکش در حالتی که خانه تمیز باشد اثری ندارد.
- هدف: هر دو خانه تمیز
- هزینهٔ مسیر: هر عمل ۱

State Space Sizes?

- the computational runtime of solving a search problem
- 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?
 $120 \times (2^{30})$

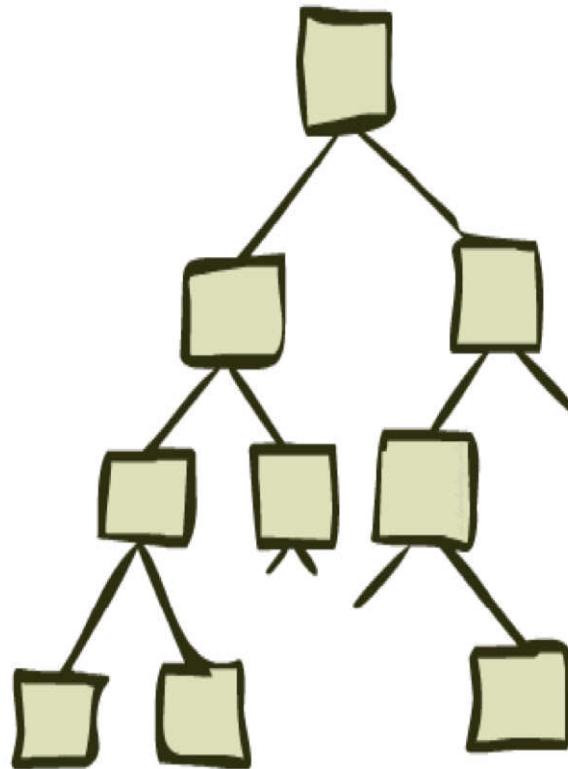


Quiz: Safe Passage



- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
 - (agent position, dot booleans, power pellet booleans, remaining scared time)

State Space Graphs and Search Trees

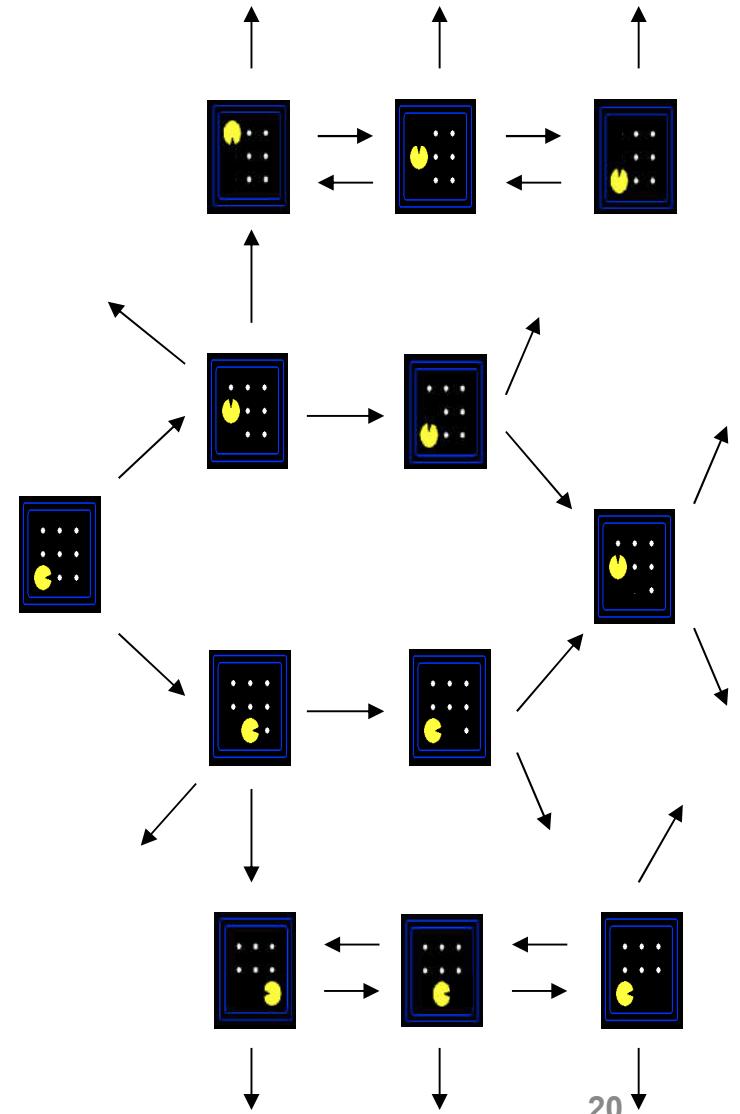


The final piece of the puzzle

برای اینکه بخوایم بین این حالت ها سرچ بکنیم با این گراف خیلی کار داریم

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

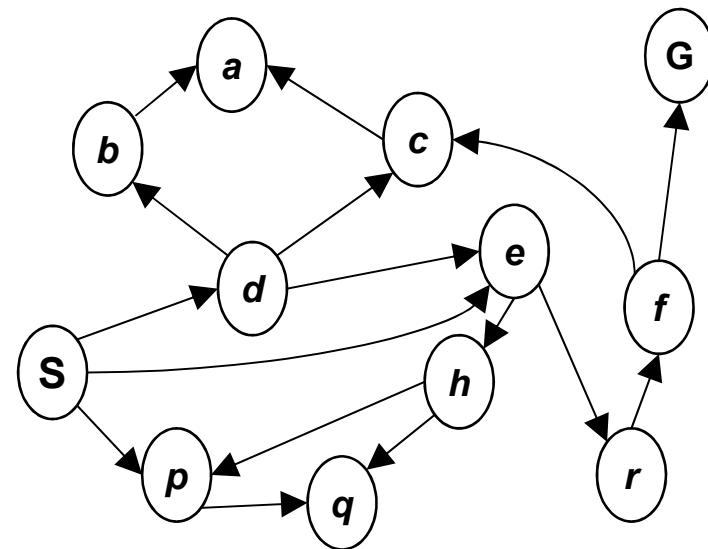


یکسری نود داریم و یکسری یا
؟؟

توی State Space Graph هر حالتی یکبار اتفاق می افته ینی قرار نیست هر حالت رو دوبار
ببینیم مثل

State Space Graphs

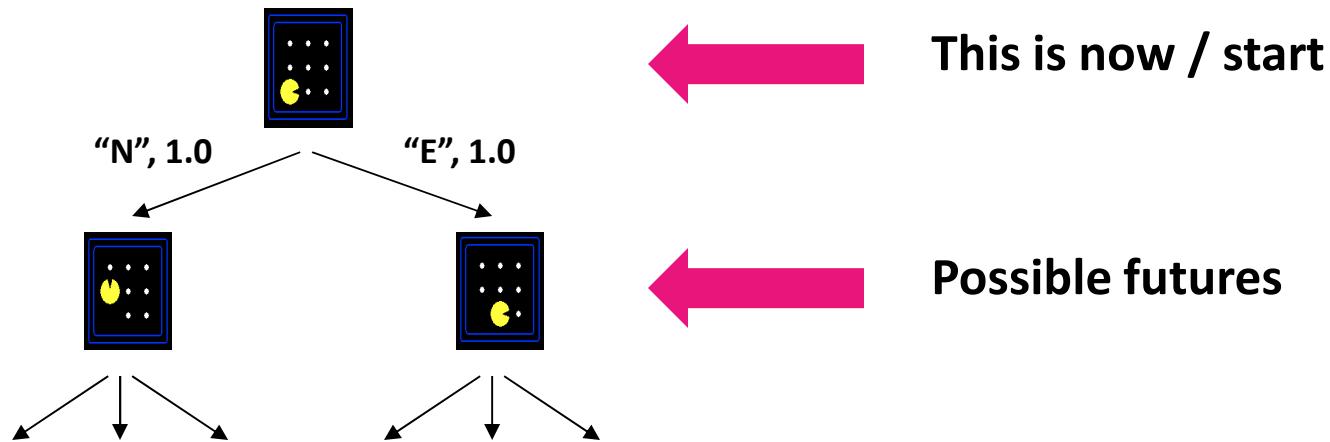
- State space graph: A mathematical representation of a search problem
 - Nodes are (abstracted) world configurations
 - Arcs represent successors (action results)
 - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this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



*Tiny state space graph for a
tiny search problem*

در کنار State Space Graph ما یک چیز دیگه ای داریم که صفحه بعدی است ..

Search Trees

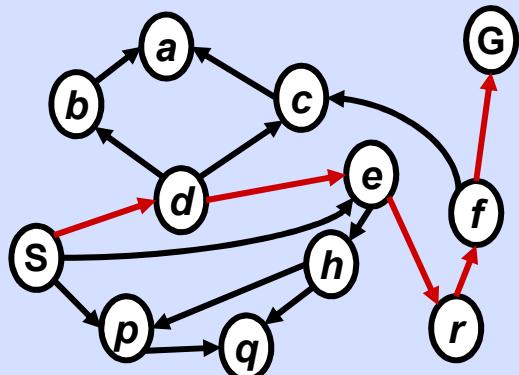


- A search tree:
 - A “what if” tree of plans and their outcomes
 - The start state is the root node
 - Children correspond to successors
 - Nodes show states, but correspond to PLANS that achieve those states
 - For most problems, we can never actually build the whole tree

این روش مثل همون گراف است ولی بر میگردد به درخت بودنش
توی روش State Space Graph همیشه یکی از نودها میشه استارت و یکی از نودها میشه
goal و بقیه حالت ها این وسط قرار داره ولی توی این روش همیشه حالت شروع توی ریشه قرار
داره

State Space Graphs vs. Search Trees

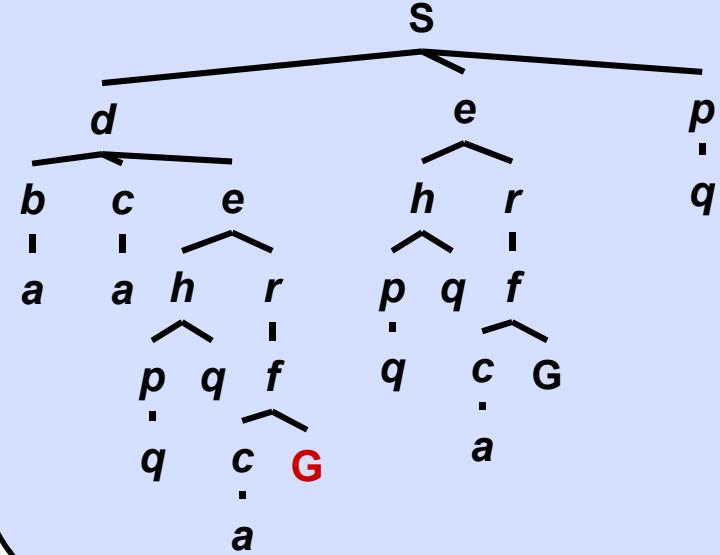
State Space Graph



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

We construct both on demand – and we construct as little as possible.

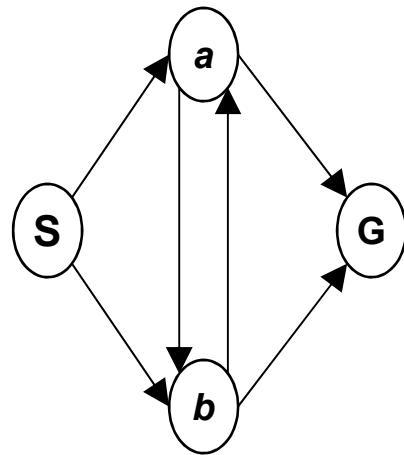
Search Tree



تبديل State Space Graph به درخت سرچ:

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



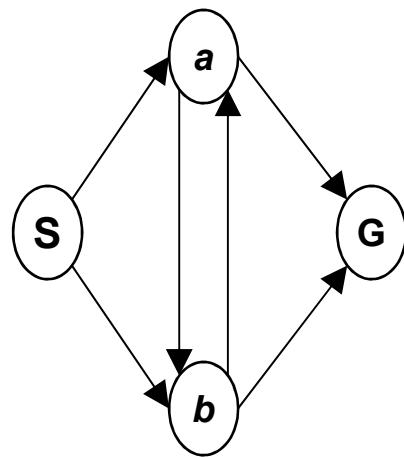
How big is its search tree (from S)?



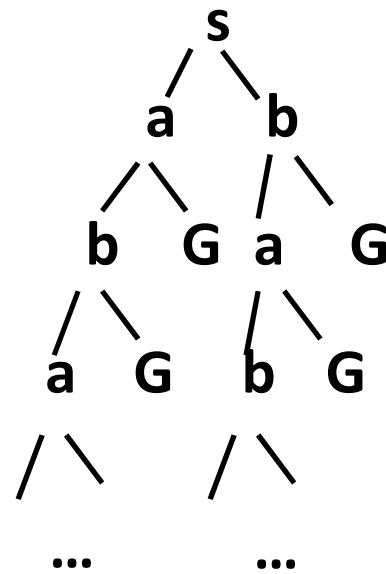
اگر بخوایم این گراف رو سرچ تیری بکنیم به شکل روبرو میشه ینی بی نهایت

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



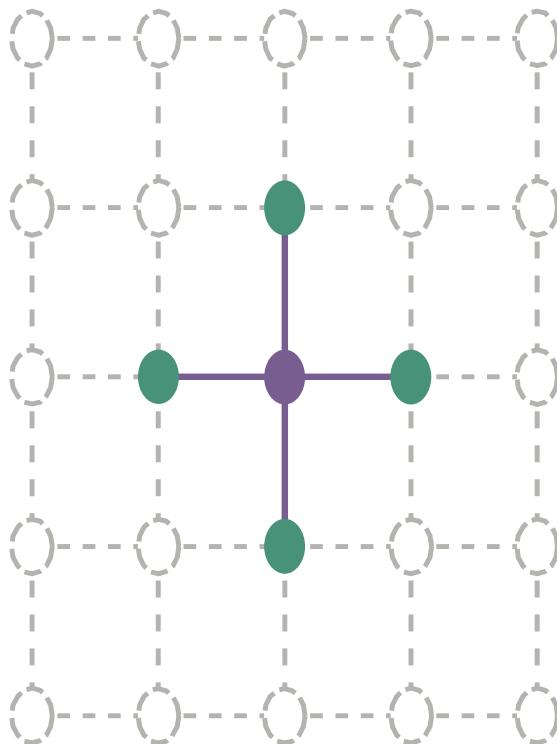
How big is its search tree (from S)?



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?

$$2d^2 + 2d + 1$$

How many states in search tree
of depth d ?

$$1 + b^1 + b^2 + \dots + b^d$$

گراف دیگه:

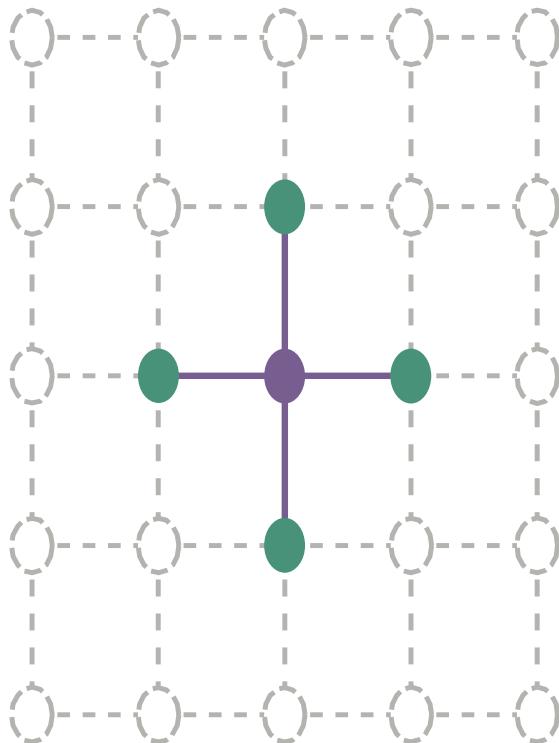
ما توی هر حالتی که هستیم دوباره میخوایم بریم حالت های بعدی یا همسایه هاش رو سرچ بکنیم تعداد استیت هایی که توی مرحله i ام داریم؟ فرمولش رو گفته

اگر بخوایم اینو با درخت بکشیم --> فرمولش رو گفته

نکته: وقتی که داریم روی گراف حرکت میکنیم عدد کوچیک است ولی وقتی که داریم درخت میسازیم عدد بزرگ میشه --> پس اینو داره میگه که وقتی که داریم به صورت درختی فضای حالت رو نشون میدیم و داریم روی سرچ تیری حرکت می کنیم تعداد حالت هایی که داریم بررسی میکنیم خیلی زیاده ولی توی گراف این اتفاق نمی افته چون اون حالت تکراری ها هم دیگر رو میگیرن

Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid: How many states within d steps of start?



$$2d^2 + 2d + 1$$

How many states in search tree of depth d ?

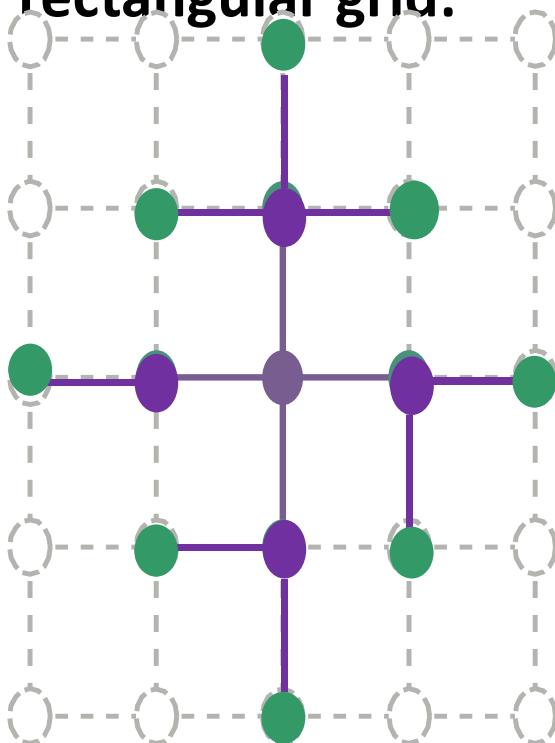
$$1 + b^1 + b^2 + \dots + b^d$$

There are:

- Closed/interior nodes (●)
- Open/frontier nodes (●)
- Unreached/exterior nodes (○)

Quiz: State Space Graphs vs. Search Trees

Consider a
rectangular grid:



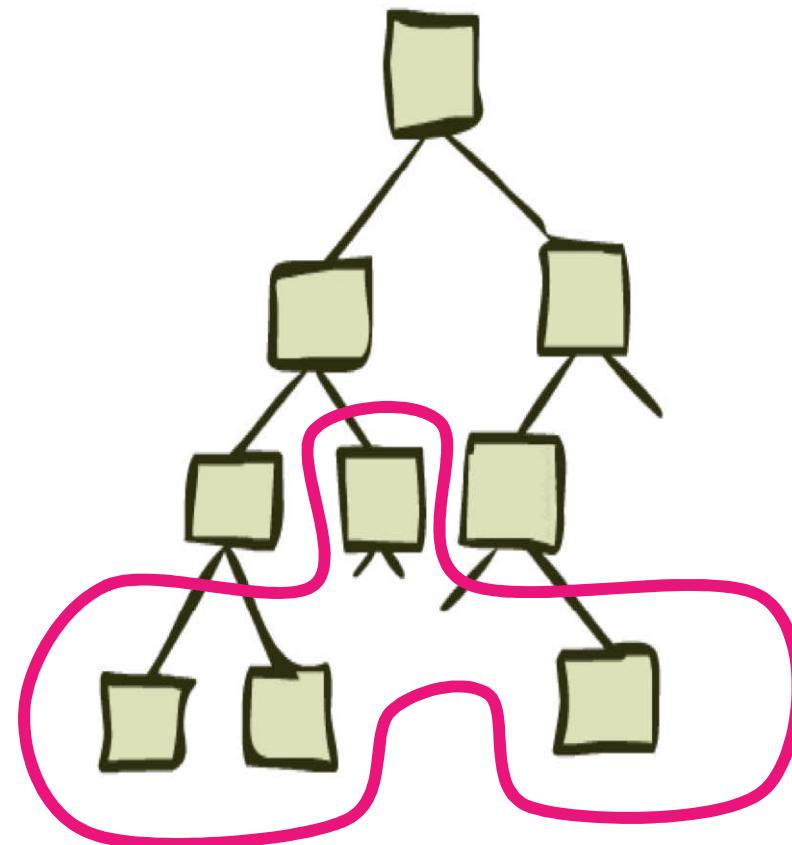
How many states within d steps of start?
 $2d^2 + 2d + 1$

How many states in search tree of depth d ?
 $1 + b^1 + b^2 + \dots + b^d$

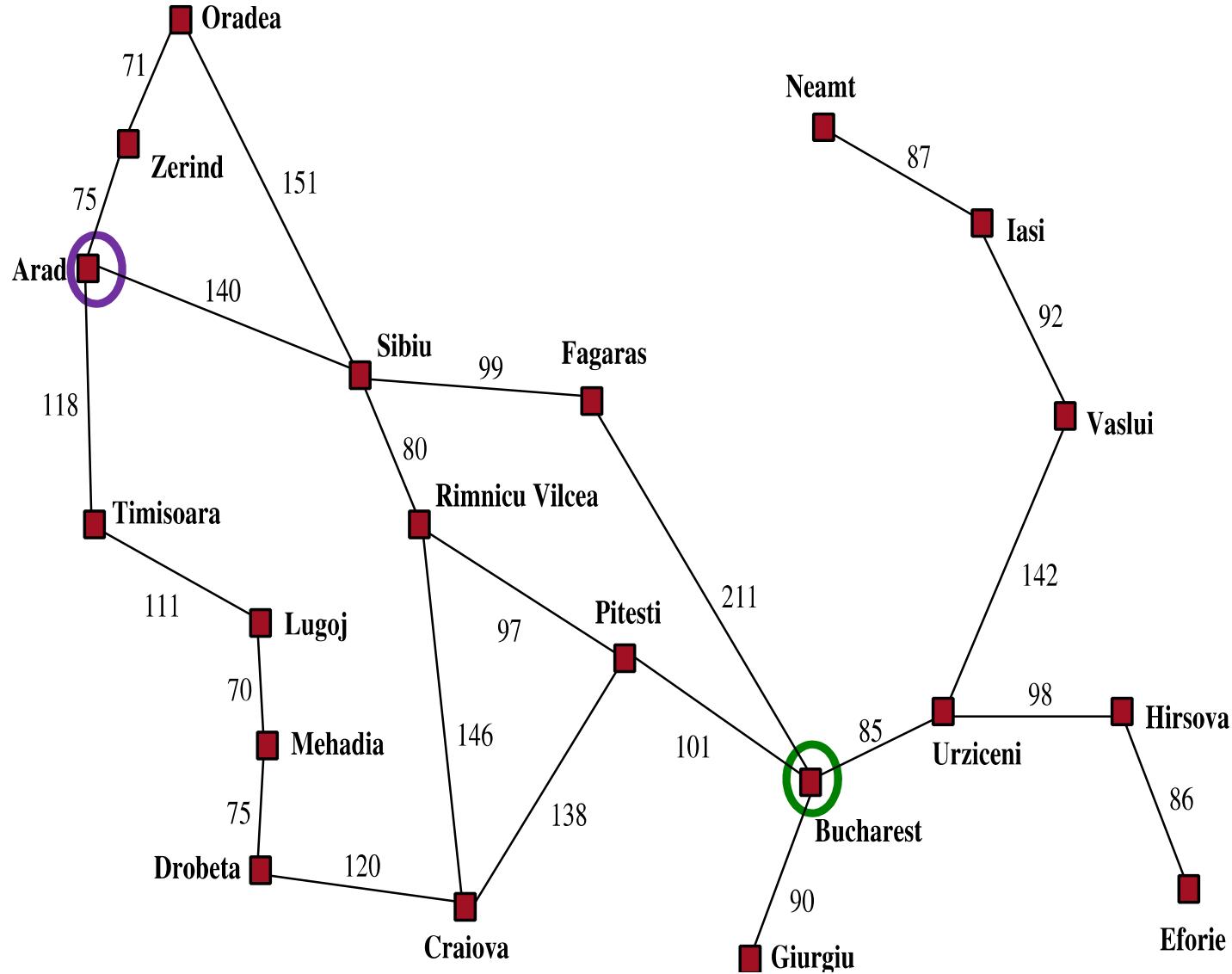
- There are:
- Closed/interior nodes (●)
 - Open/frontier nodes (●)
 - Unreached/exteriors nodes (●)

همینطوری که داریم سرچ می کنیم توی درخت یکسری نودها رو از قبل دیدیم
یکسری نود توی لبه سرچ ما هستن و ما اینارو باید باز بکنیم
یکسری نود هم هستن که سراغشون نرفتیم

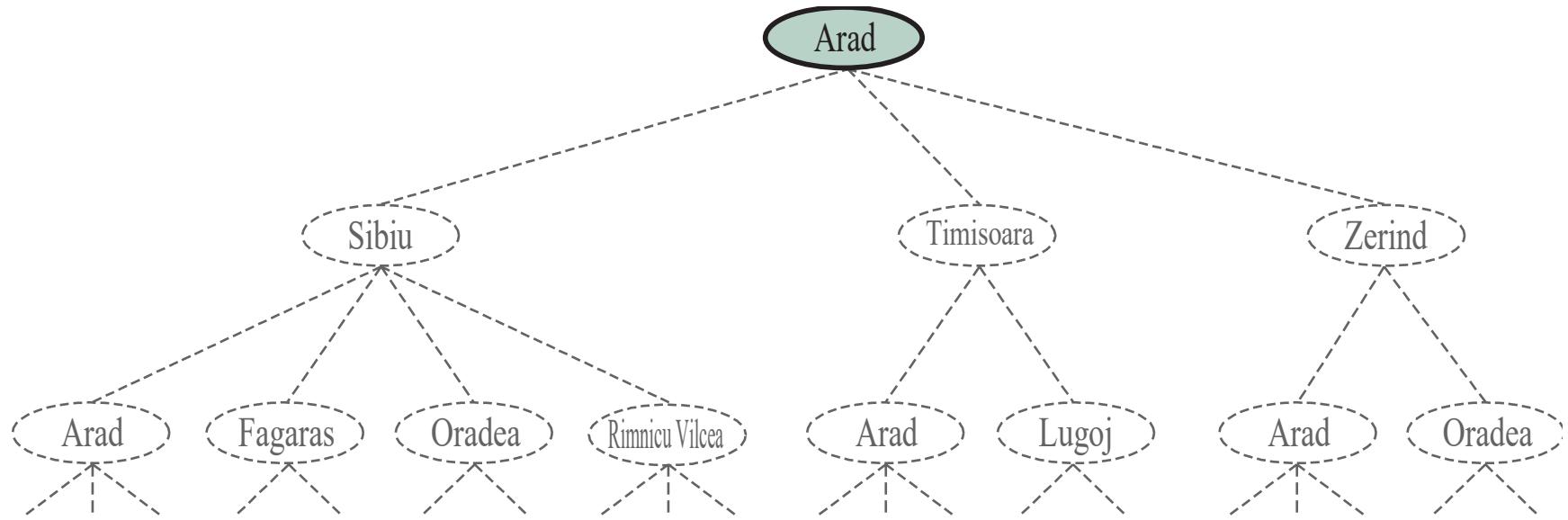
Tree Search



Search Example: Romania



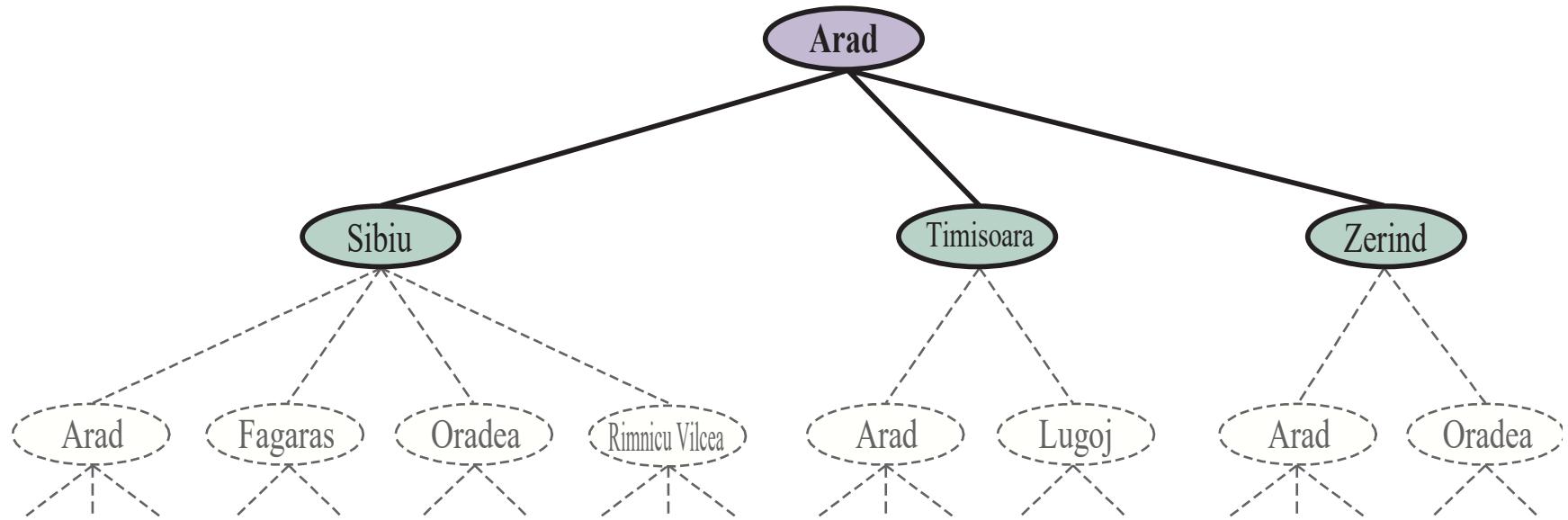
Creating the search tree



- **Search:**
 - Expand out potential plans (tree nodes)
 - Maintain a **frontier** of partial plans under consideration
 - Try to expand as few tree nodes as possible

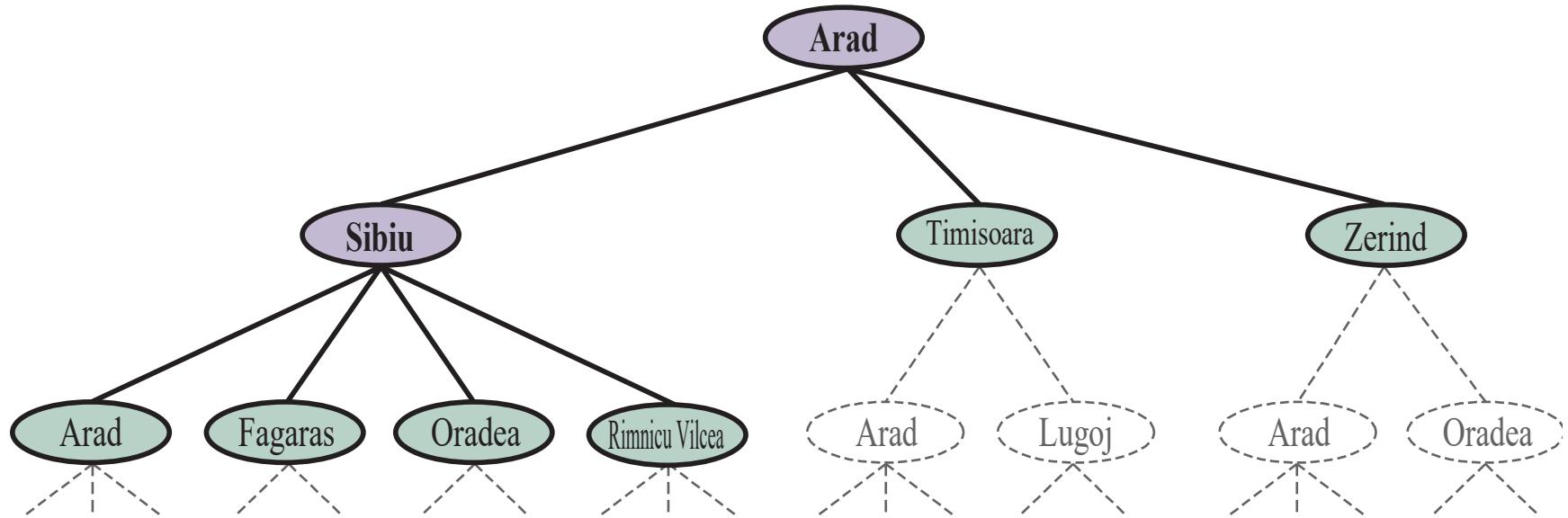
از a به یکدوم از این سه تا شهر می‌ریم
بعد اینکه کدوم شهر رو انتخاب بکنیم اینا باید دوباره سرچ بکنیم و یکی از اونارو انتخاب می‌کنیم
مثلًا اینجا S و همینطوری ادامه میدیم

Creating the search tree



- **Search:**
 - Expand out potential plans (tree nodes)
 - Maintain a **frontier** of partial plans under consideration
 - Try to expand as few tree nodes as possible

Creating the search tree



- **Search:**
 - Expand out potential plans (tree nodes)
 - Maintain a **frontier** of partial plans under consideration
 - Try to expand as few tree nodes as possible

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
```

- Important ideas:
 - Frontier
 - Expansion
 - Exploration strategy
- Main question: which frontier nodes to explore?

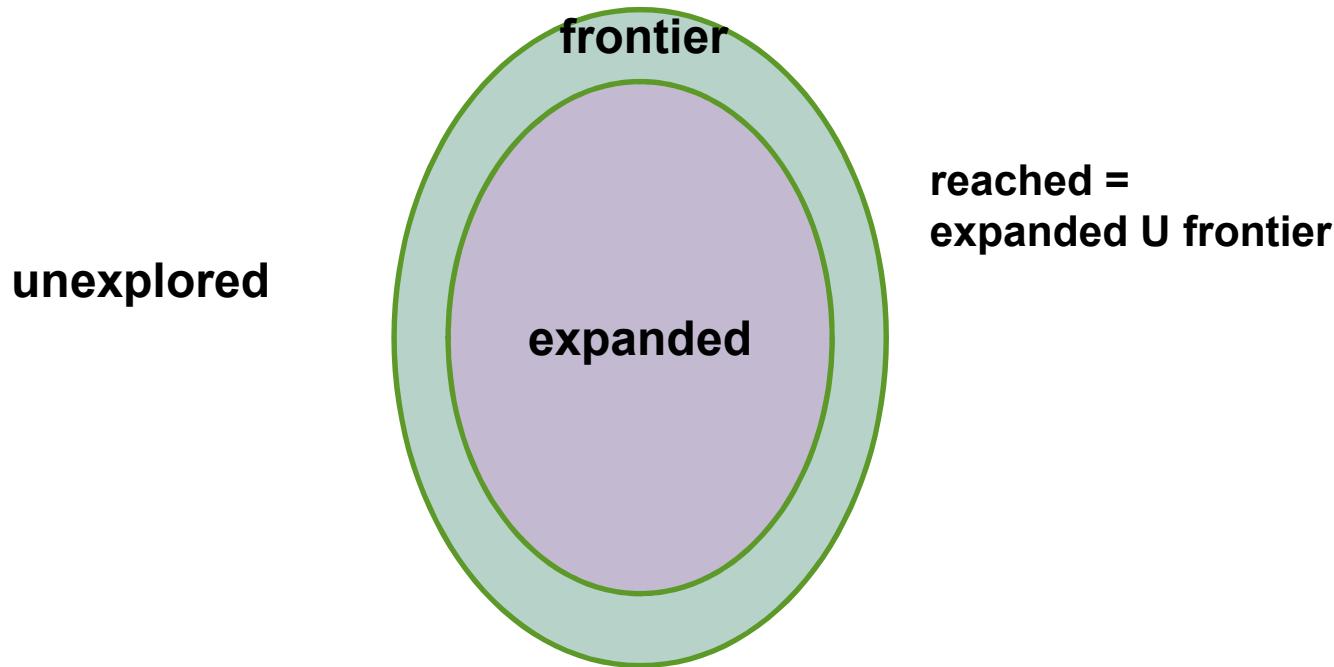
به حالت هدف برسیم:

اولش استیت گراف رو می سازیم و از نود استارت شروع میکنیم و توی هر مرحله یک نود یا چندتا نود رو انتخاب میکنیم برای **expand** شدن و اونارو **expand** میکنیم و براساس یکسری استراتژی اونارو انتخاب میکنیم اینکه کدام یکی از این شهرها رو انتخاب بکنیم برای **Expand** شدن یکسری استراتژی های پشتش داره که جلوتر می گیم بعد از اون لیست که قراره **expand** بشه یکسریشو انتخاب مکنیم و اونارو دوباره **expand** میکنیم و ...

سه تا مسئله اینجا وجود داره:

- 1- اون نود هایی که میخواه پیشگام باشه رو چطوری انتخاب میکنیم
- 2- بعد بر چه اساسی اونارو **expand** می کنیم
- 3- استراتژی جستجو به چه صورت باشه؟

Systematic search

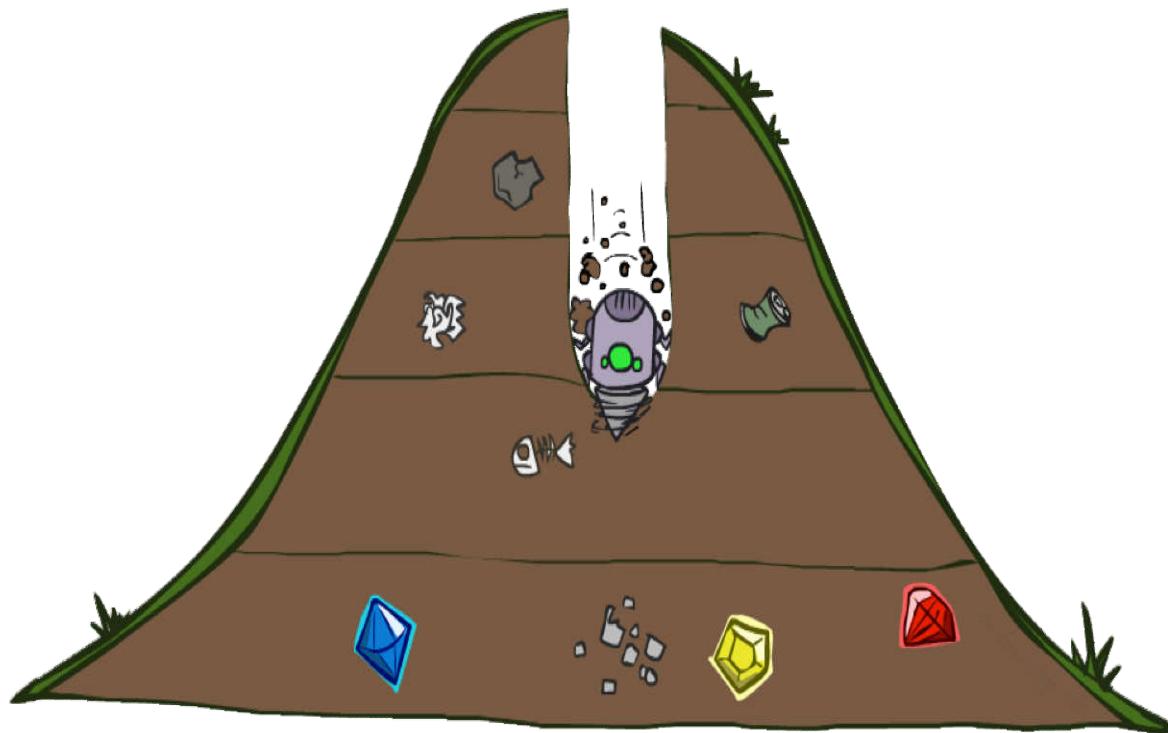


1. Frontier separates expanded from unexplored region of state-space graph
2. Expanding a frontier node:
 - a. Moves a node from frontier into expanded
 - b. Adds nodes from unexplored into frontier, maintaining property 1

توی این جستجو کردن استیت ها توی چند دسته قرار میگیرن:

بعضی از استیت ها رو از قبل دیدیم و نیاز نیست اوها **expand** پیدا بکنن اوها **Expand** شدن بعضی از استیت ها رو امادیم که اوها رو **Expand** بکنیم و به یکسری ها هم هنوز نرسیدیم

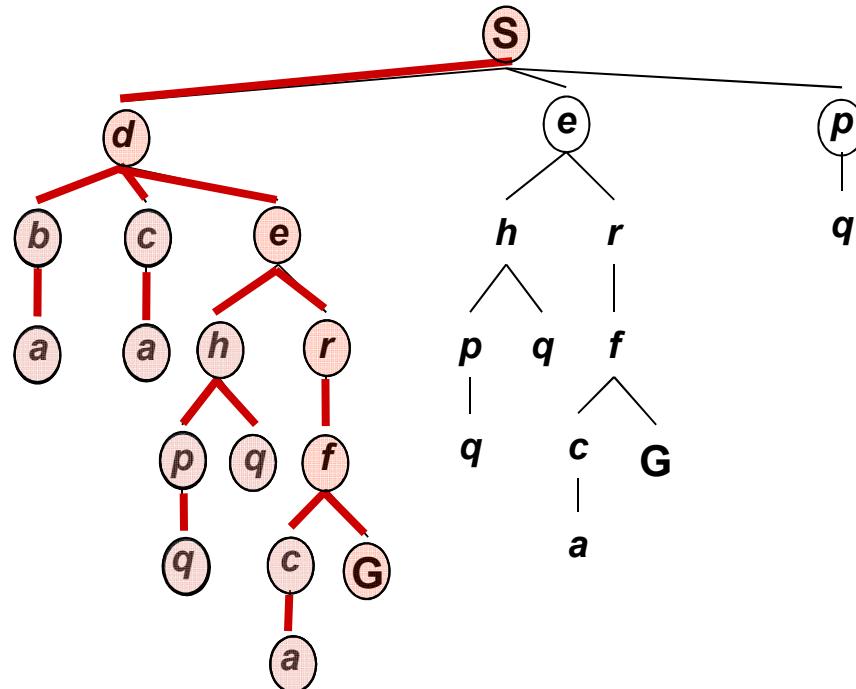
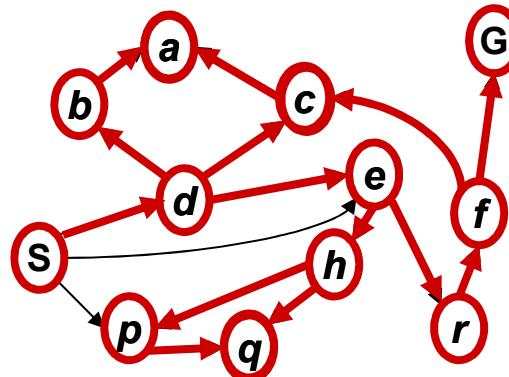
Depth-First Search



Depth-First Search

Strategy: expand a deepest node first

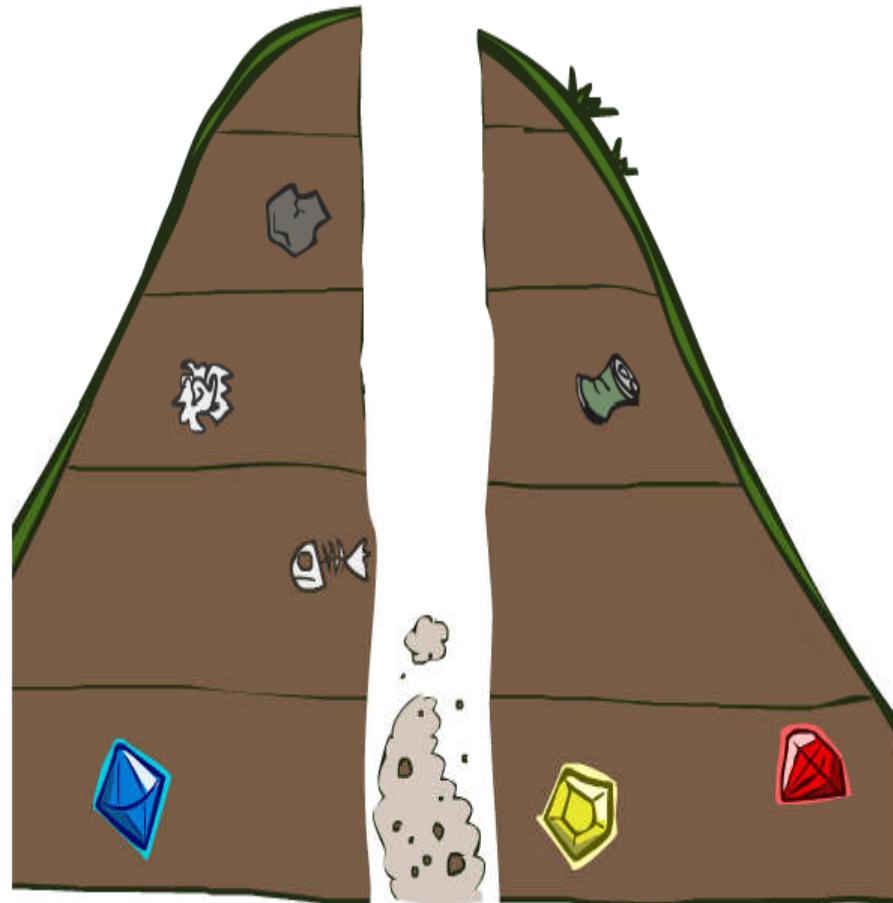
*Implementation:
frontier is a LIFO
stack*



جستجوی عمقی:

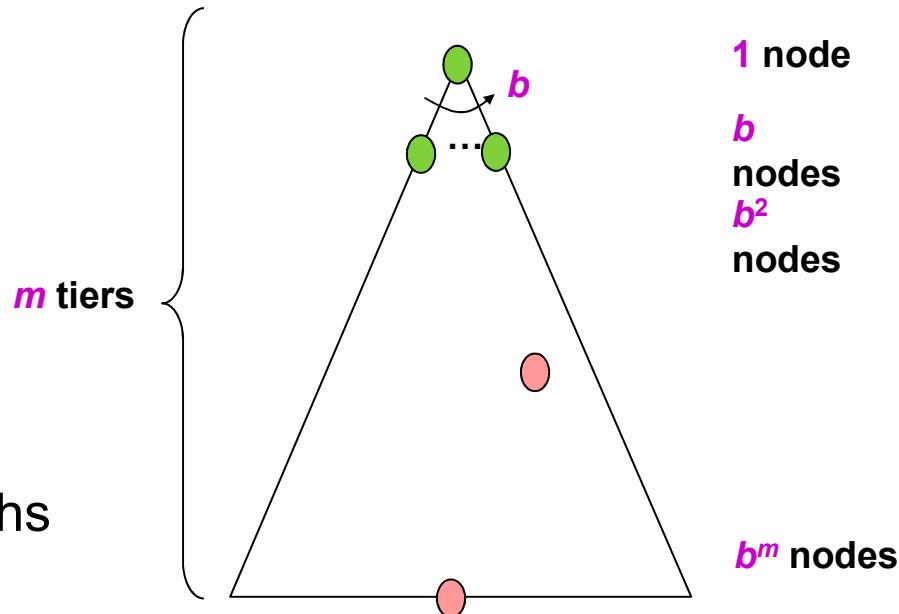
برای نود `s` ما سه تا همسایه داریم: `d` , `e` , `p` که این سه تا می رن توی لیست `expand` و حالا از بین این سه تا کدام رو انتخاب بکنیم؟ اونی رو انتخاب میکنیم که اولین نود است یعنی `d` و این نود رو `Expand` می کنیم و به همین صورت می ریم جلو در واقع اولین نود میشه همون نودی که برای اولین بار وارد لیست شده این نود رو انتخاب میکنیم

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?
- Space complexity?
- Cartoon of search tree:
 - b is the branching factor
 - m is the maximum depth
 - solutions at various depths
- Number of nodes in entire tree?
 - $1 + b + b^2 + \dots + b^m = O(b^m)$



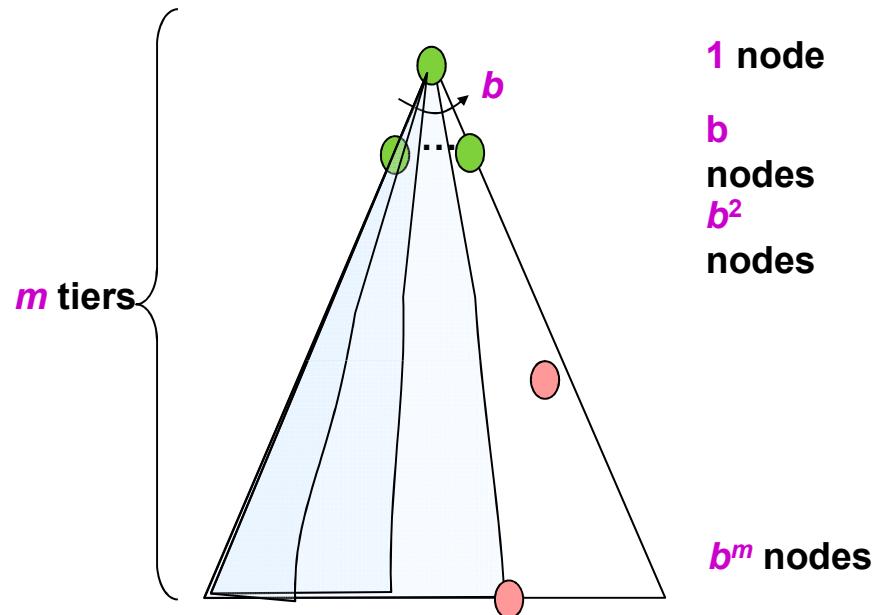
میخوام چندتا تکنیک سرچ دیگه هم بگیم:
برای اینکه تکنیک های سرچ های مختلف رو بخوایم با هم مقایسه کنیم چندتا شاخص برای مقایسه وجود داره:

- 1 Complete: زمانی میگیم یک الگوریتم سرچ Complete است که واقعا به هدف برسه
- 2 گارانتی بکنه که ما با کمترین هزینه به اون هدف برسیم
- 3 پیچیدگی زمانی خود الگوریتم
- 4

هر نودی b تا شاخه داره که این میشه branching factor ماکزیمم عمق چقدر است: m
ممکنه توی این درخت سرچ تیری چند جا به هدف برسیم

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



الگوریتم :dfs

توی یک برنج میره تا پایین اگر به جواب رسید که هیچ ولی اگه نرسید برمیگرده و عمق های بیشتری رو انالیز میکنه تا در نهایت به اون هدف برسه که الگوریتم تموم بشه

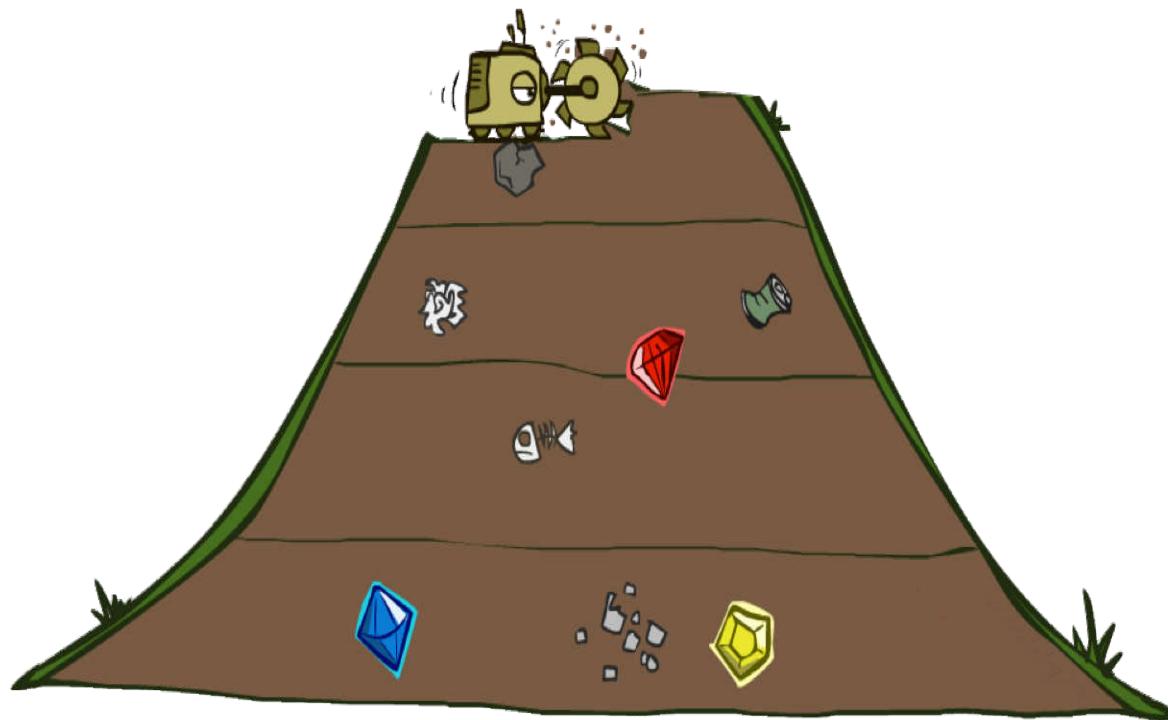
حالا مرتبه زمانی: توی بدترین حالت میشه b^m به توان m
چقدر فضا میگیره این b^m چرا؟

این الگوریتم حتما به هدف می رسه ینی **complete** است؟ بله حتما به هدف می رسه
ولی اگر لوب داشته باشیم هیچ وقت نمی رسه به هدف ولی در حالت کلی میشه نه

بهینه چی؟ نه

این تکنیک برای جایی که محدودیت حافظه داریم تکنیک خوبی است

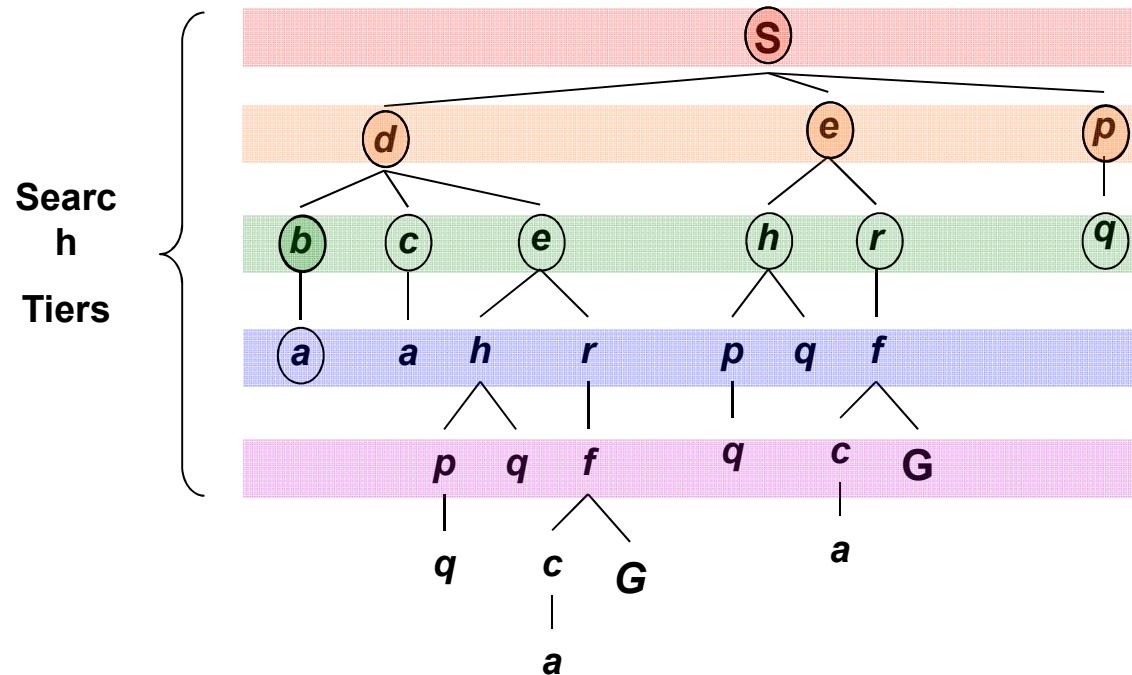
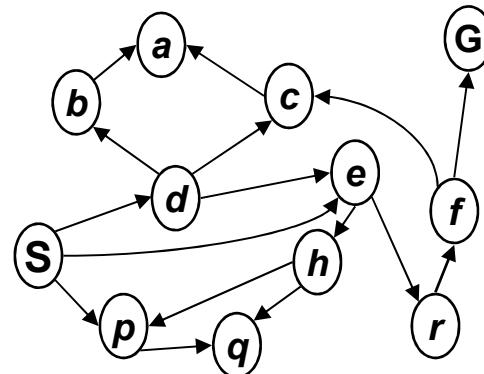
Breadth-First Search



Breadth-First Search

Strategy: expand a shallowest node first

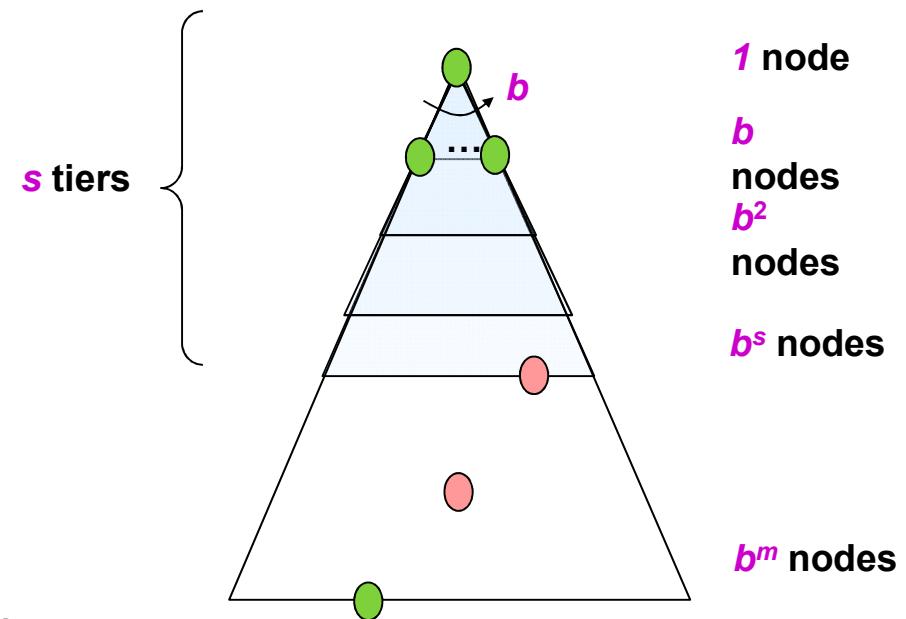
*Implementation:
frontier is a FIFO
queue*



الگوريتم : bfs
اينجا سطحي سرج ميكنه

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)



داریم لایه لایه درخت رو سرچ می کنیم

چقدر زمان می بره؟ b به توان S چون:

اینجا توی لوپ گیر نمی کنه چون داره منظم می ره پایین ینی اگر هدف ما توی لول S باشه حتما
اینجا پیداش می کنه پس به اندازه موقعیت خود هدف نسبت به مقدار اولیه ما نیاز داریم به مرتبه
زمانی که میشه b به توان S

فضا چطور؟ اینجا باید کل سطح رو نگه داره مثلًا اگر توی سطح b به توان S هستیم به تعداد بچه
های b به توان S داریم پس اینجا اردر حافظه نسبت به dfs خیلی بیشتر است
ایا $complete$ است؟ بله است. اینجا این مشکل رو نداره که توی لوپ گیر بکنه حتما بهش
می رسه

optimal چطور؟ با فرض اینکه استیت های ما هزینه مشخصی داشته باشن ینی هزینشون با هم
برابر باشه بله $optimal$ است

BFS; evaluation

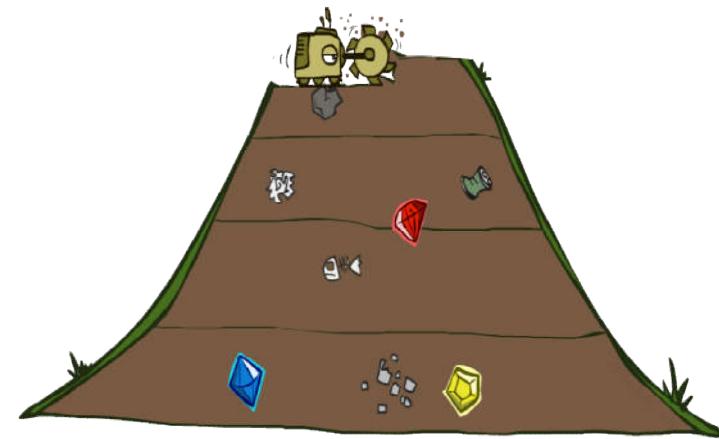
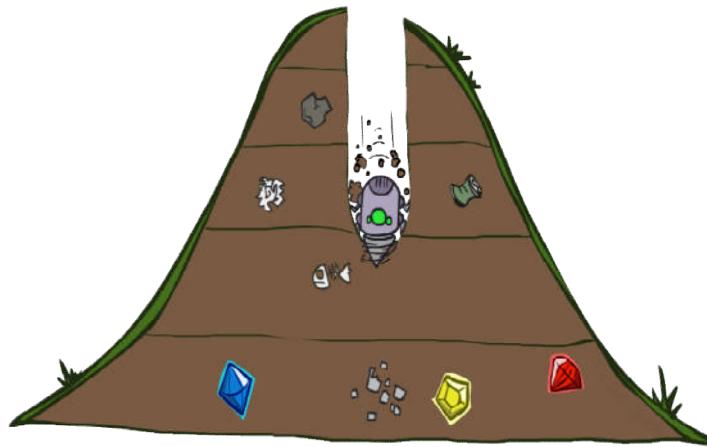
- **Two lessons:**
 - **Memory requirements are a bigger problem than its execution time.**
 - **Exponential complexity search problems cannot be solved by uninformed search methods for any but the smallest instances.**

DEPTH2	NODES	TIME	MEMORY
2	110	0.11 milliseconds	107 kilobytes
4	11110	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

Table is based on: $b = 10$; 10^6 nodes/second; 1000 bytes/node

مثلا برنچ فکتور توی یک مسئله ای 10 بوده و دیده حافظه چقدر می بره که به اون هدف بررسه اینجا مموری خیلی زیاد میشه چون نودهای زیادی رو داریم `expand` می کنیم

Quiz: DFS vs BFS



Quiz: DFS vs BFS

- When will BFS outperform DFS?
- When will DFS outperform BFS?

Depth-limited search

= depth-first search **with depth limit l** , i.e.,
nodes at depth l have no successors

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred?  $\leftarrow$  false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred?  $\leftarrow$  true
        else if result  $\neq$  failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

یک الگوریتم ارائه شده در توسعه اون dfs به اسم Depth-limited search توی این الگوریتم میگه نرو همه عمق رو بررسی بکن یک حد استانه ای برای عمق بذار

Iterative Deepening DFS

- Gradually increasing the limit in depth-limited search, until the solution is found:

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution
    inputs: problem, a problem
    for depth  $\leftarrow 0$  to  $\infty$  do
        result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
        if result  $\neq$  cutoff then return result
    end
```

اینجا bfs , dfs را با هم ترکیب کرده

Iterative Deepening DFS (cont.)

- 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?

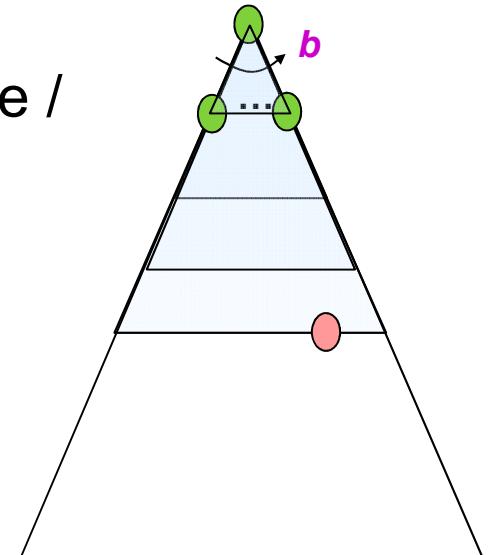
- Generally most work happens in the lowest level searched, so not so bad!

- Time Complexity:

$$1 + (1+b) + (1+b+b^2) + \dots + (1+b+\dots+b^s) = (s+1) + sb + (s-1)b^2 + \dots + b^s = O(b^s)$$

- Is it optimal?

- Yes, If step costs are equal (e.g., 1)



مشکل این: اینارو داره خیلی تکراری می بینه ینی هر دفعه که این `for` ران میشه ینی هر دفعه عمق رو میداره یه حدی و دوباره از اول ران میشه ولی این خیلی مسئله ای ایجاد نمی کنه چون معمولاً توی عمق های کم این هدف پیدا میشه

پیچیدگی زمانی: میشه b به توان s

optimal ؟ بله است

Iterative Deepening DFS (cont.)

- **Completeness:**
 - YES (no infinite paths)
- **Time complexity:** $O(b^s)$
 - Algorithm seems costly due to repeated generation of certain states.
 - Node generation vs Node expansion

Level	Generated Nodes (IDS)	Generated Nodes (BFS)	Expanded Nodes (IDS)	Expanded Nodes (BFS)
0	-	-	$(s+1)$	1
1	$(s)b^1$	b^1	$(s)b^1$	b^1
2	$(s-1)b^2$	b^2	$(s-1)b^2$	b^2
...
$s-1$	$(2)b^{s-1}$	b^{s-1}	$(2)b^{s-1}$	b^{s-1}
s	$(1)b^s$	b^s	$(1)b^s$	b^s
$s+1$	-	$b^{s+1} - b$	-	-

Iterative Deepening (cont.)

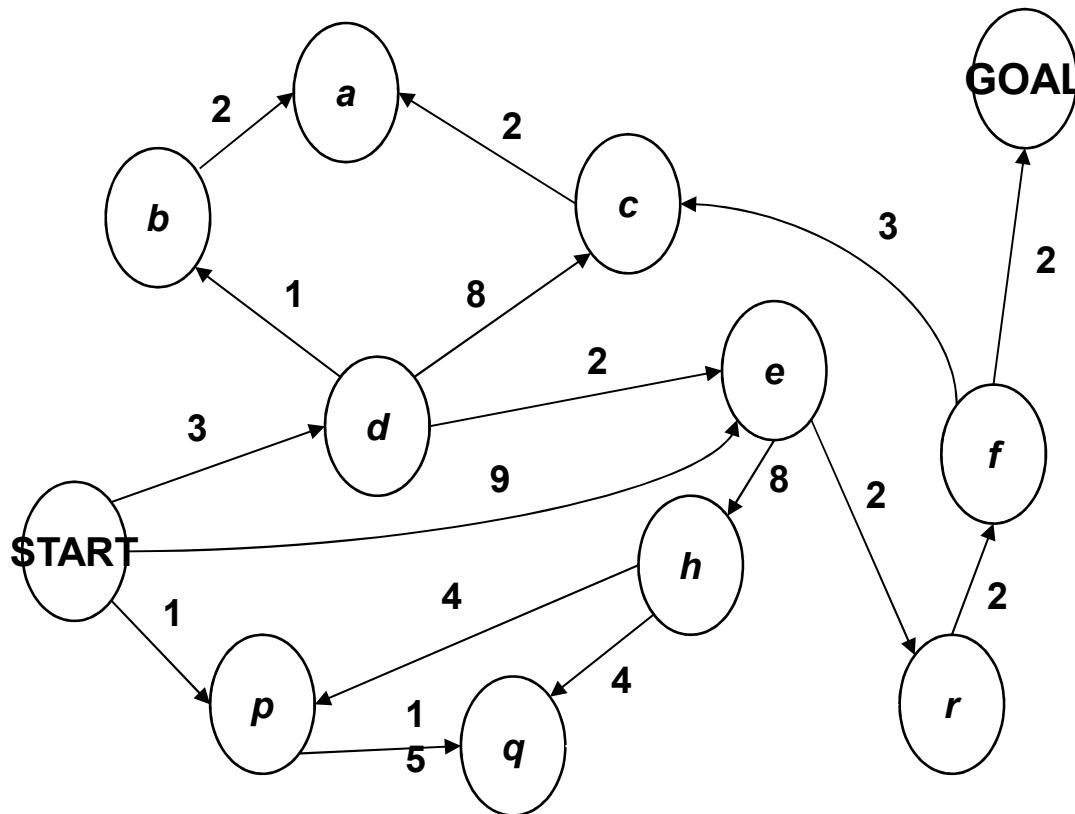
- Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:
- IDS does better because other nodes at depth d are not expanded
- BFS can be modified to apply goal test when a node is generated

Level	Generated Nodes (IDS)	Generated Nodes (BFS)	Expanded Nodes (IDS)	Expanded Nodes (BFS)
0	-	-	6	1
1	50	10	50	10
2	400	100	400	100
4	3,000	1,000	3,000	1,000
4	20,000	10,000	20,000	10,000
5	100,000	100,000	100,000	100,000
6	-	999,990	-	-
Total	123,450	1,111,100	123,456	111,110

Cost of iterative deepening

b	ratio ID to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Cost-Sensitive Search

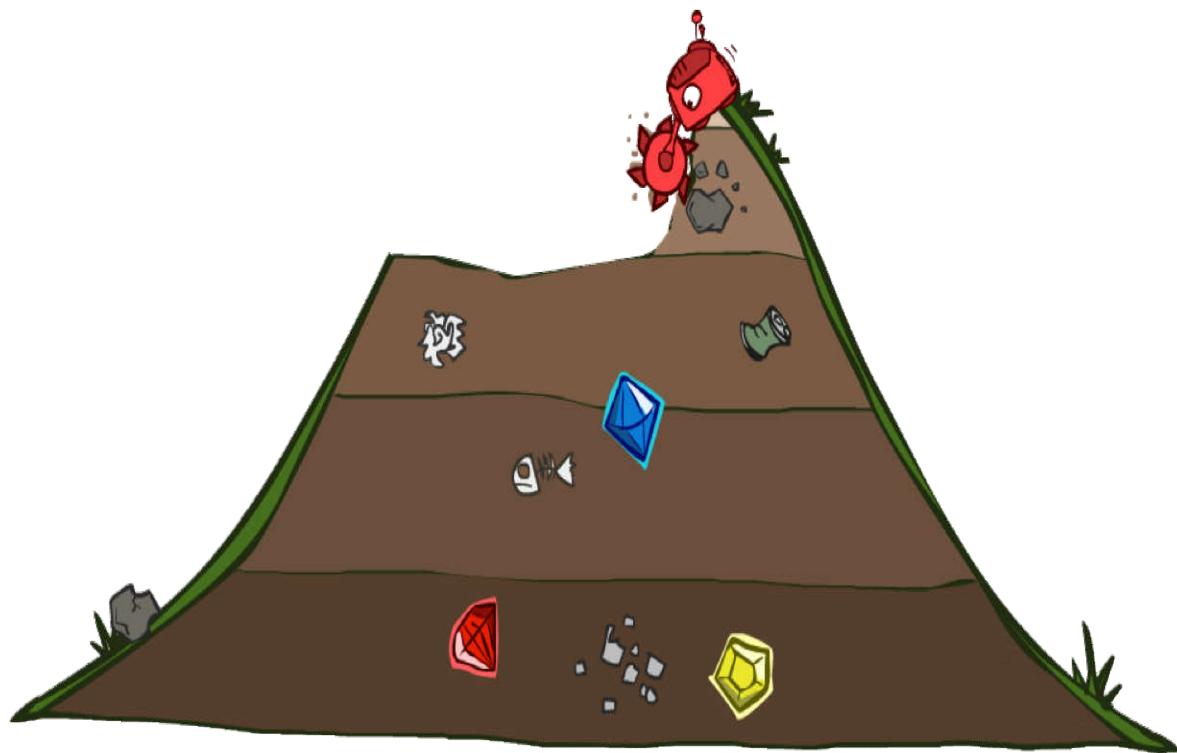


BFS finds the shortest path in terms of number of actions.

It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.

وزن اینکه از یک استیت به یک دیگه بریم فرق میکنه و اینجا میخوایم این وزن هم در نظر بگیریم:
توی این حالت تکنیک های قبلی جوابگو نیستن پس از تکنیک Cost-Sensitive Search می ریم

Uniform Cost Search

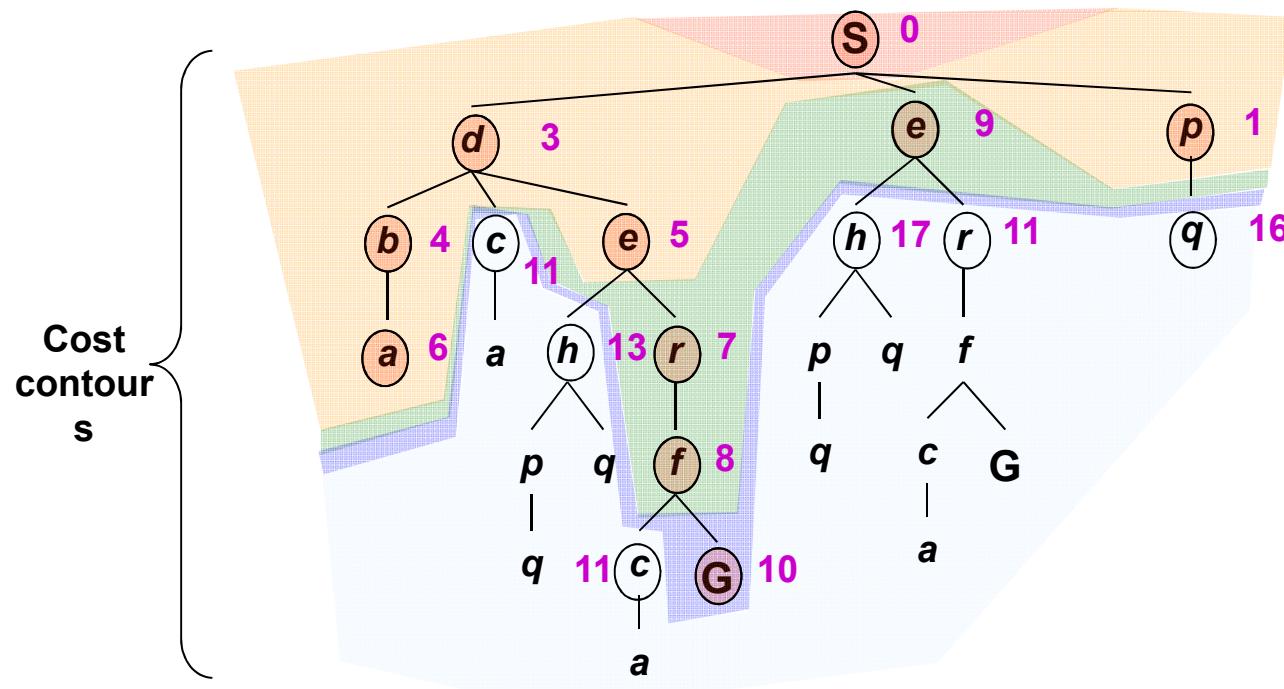
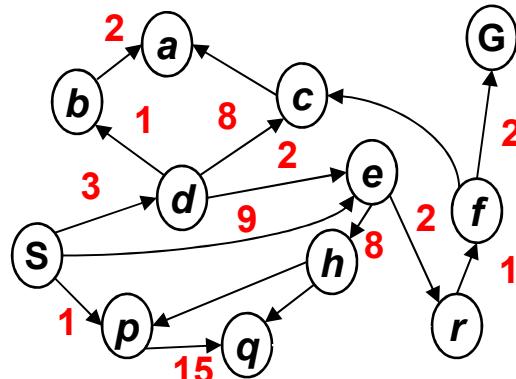


Uniform Cost Search

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

Strategy: expand lowest $g(n)$

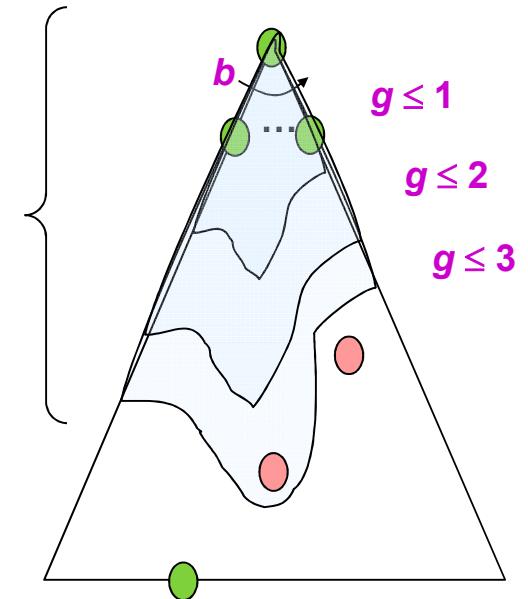
Frontier is a priority queue sorted by $g(n)$



هزینه بالای هر یال نوشته شده مثلا اگر از نود b بخوایم بریم به نود a هزینه اش میشه 2
توی این حالت نودی رو انتخاب می کنیم که هزینه اش کمتر است ینی مثلا p اینجا بعدش میشه d
چون کمتره بعدش میشه b بعد e بعد r بعد f بعد a و بعد دید این 10 هدف است

Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
 - Expands 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 “tiers” 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*)



رویه به این صورته --> می ره توی اون جهتی که هزینه کمتر است هی عمیق تر میشه و همینطوری می ره تا به هدف بررسه هزینه رسیدن به جواب نهایی C^* باشه و اگر اپسیلون یال روطی کرده باشیم میزان عمق درمیاد پس عمقی که به طور متوسط باید طی بشه میشه C^* بر اپسیلون

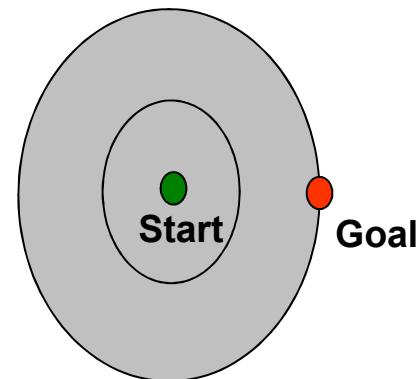
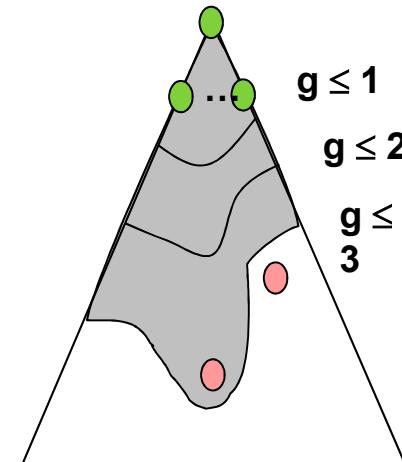
چقدر فضا می بره؟ باید تا اون عمق رو بریم که عمق اینجا C^* بر اپسیلون است؟؟؟

این کامل است؟ بله چون نهایتا اون جواب بهینه هم داره از این هزینه کمینه ها انتخاب میشه

این حتی optimal هم هست یعنی جواب بهینه هم میده

Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
 - Explores options in every “direction”
 - No information about goal location
- We'll fix that soon!



یک مسئله راجع به این روش هست:

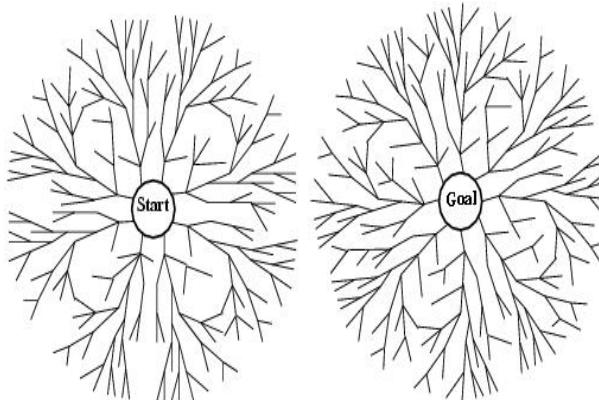
برای اینکه بخوایم به اون هدف بررسیم خیلی حالت رو داریم بررسی می کنیم --> درسته که مارو به هدف می رسونه ولی بیش از اندازه داریم سرچ میکنیم اطراff رو

در واقع اگر یکسری اطلاعاتی از هدف داشته باشیم ینی فاصله از هدف رو بسنجیم می تونیم خیلی هوشمندانه تر عمل بکنیم

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Bidirectional search



- Two simultaneous searches from start and goal.
 - Motivation:
$$b^{s/2} + b^{s/2} \neq b^s$$
- Check whether the node belongs to the other fringe before expansion.
- Space complexity is the most significant weakness.
- Complete and optimal if both searches are BFS.
- The predecessor of each node should be efficiently computable.
 - When actions are easily reversible.

اولین پیشنهاد برای بحث هوشمندانه تر:
الان اینجا یک درخت داریم سمت استارت
و یک درخت هم داریم سمت هدف
و اینجا الان میایم؟؟؟

زمانی می ریم سراغ این روش که جمع این دو تا درخت نسبت به یک درخت یعنی b به توان s بصرفه

بحث حافظه داریم اینجا