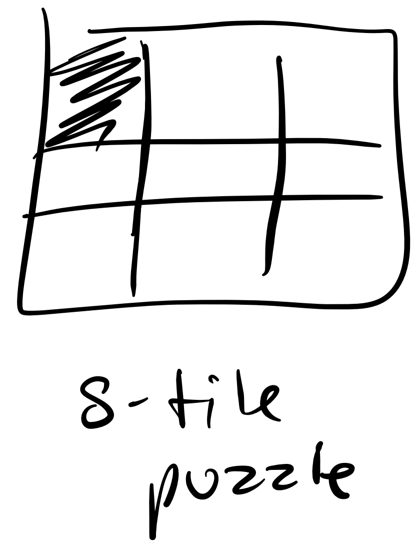
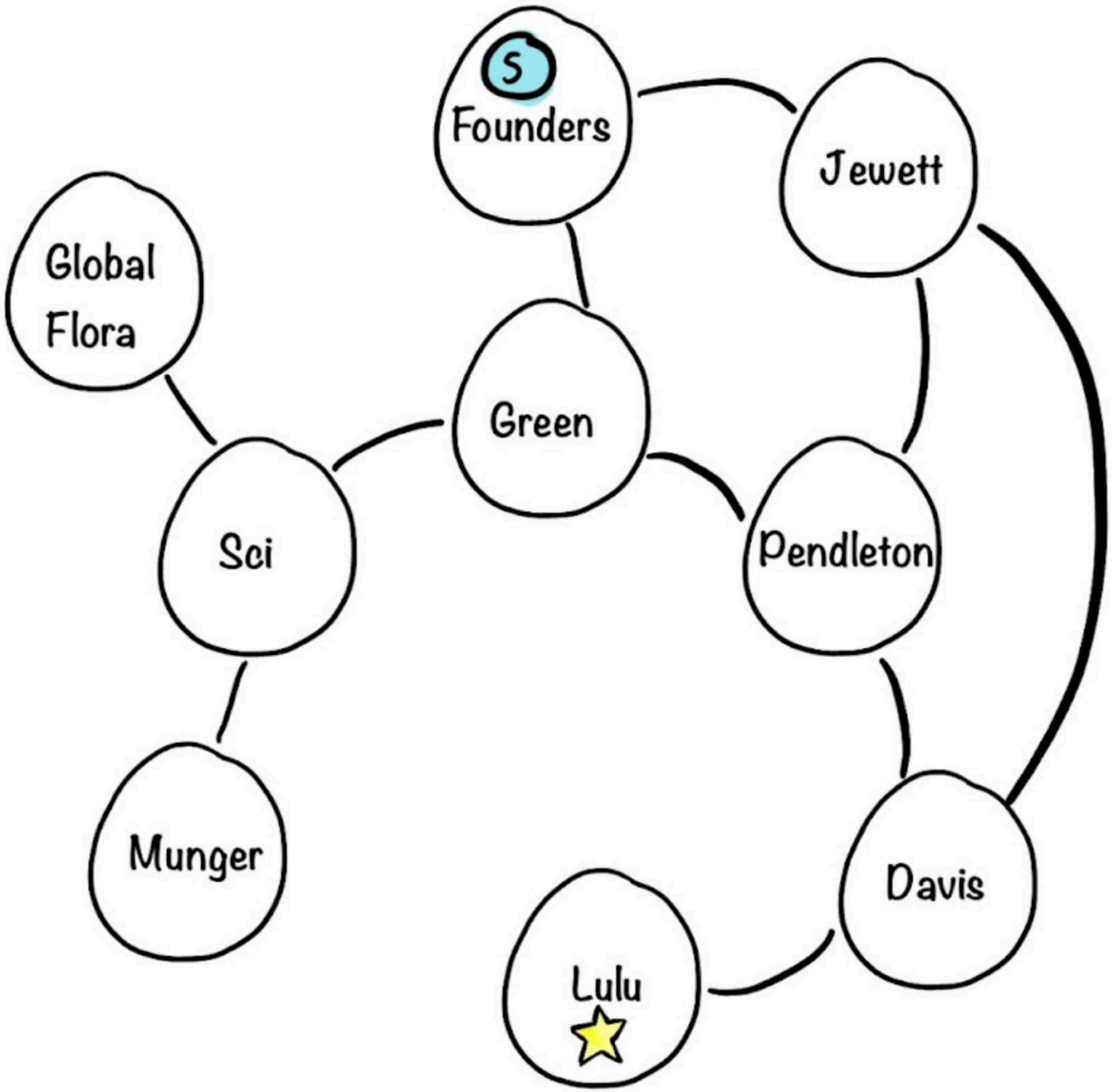
CS 232: Artificial Intelligence

Fall 2026

Prof. Carolyn Anderson
Wellesley College



Recap


S
Founders
Jewett
Global
Flora
Green
Sci
Pendleton
Munger
Davis
Lulu

Uninformed Search

Ignoring Cost

Uninformed Search

Uninformed Search: all non-goal nodes in frontier look equally good.

Informed Search: some non-goal nodes can be ranked above others.

Key Question: what strategy should we use to pick nodes from the frontier?

Breadth-First Search

Breadth-First Search

Strategy: expand shallowest nodes first.

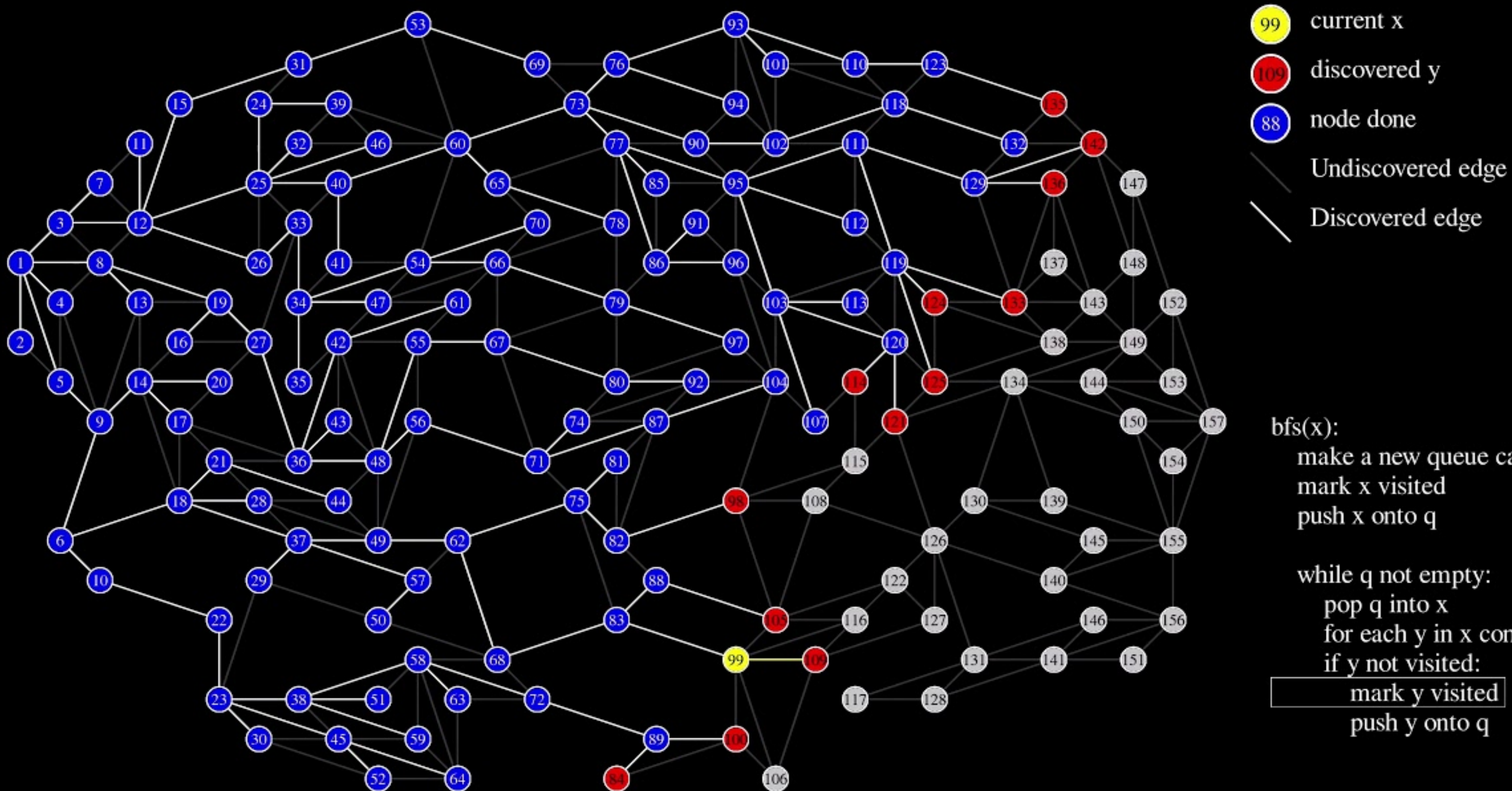
Implementation: frontier is first-in-first-out (FIFO) queue. Put successors at the end of the frontier list.

Key Question: what strategy should we use to pick nodes from the frontier?

Tree Search, Breadth-First Search Version

function **GraphSearch**(problem, strategy) return solution or failure
  Initialize frontier to start state of problem
  Initialize *explored set* to be empty
  do
    if the frontier is empty then return failure
    choose leaf for expansion **according to strategy** and remove from frontier
    if node contains goal state then return solution
    add node to the explored set
    else expand the node and add resulting nodes to the frontier *only if not already in the explored set*

Breadth-first search: strategy = earliest nodes first




```
bfs(x):
  make a new queue called q
  mark x visited
  push x onto q

  while q not empty:
    pop q into x
    for each y in x connections
    if y not visited:
      mark y visited
      push y onto q
```

Breadth-first search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Position within queue of new items determines search strategy

Slides adapted from Chris Callison-Burch

Breadth-first search

```
function EXPAND(problem, node) yields nodes
    s ← node.STATE
    for each action in problem.ACTIONS(s) do
        s′ ← problem.RESULT(s, action)
        cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
        yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

Node data structure contains variables like the state, a pointer to its parent node, the action that was used to create this state, and the path cost.

The Python yield keyword means that we don't have to pre-compute a list of all successors.

Slides adapted from Chris Callison-Burch

Properties of breadth-first search

Complete? Yes (if b is finite)

Optimal? Yes, if cost = 1

Time Complexity? $1 + b + b^2 + b^3 \ldots O(b^d)$

Space Complexity? $O(b^d)$

m = maximum depth of the state space

Depth: d = depth of the least cost / closest solution

Branching Factor: b = maximum branching factor / maximum # of children a node can have

Exponential Space (and time) Is Not Good...

- Exponential complexity uninformed search problems *cannot* be solved for any but the smallest instances.
- *(Memory* requirements are a bigger problem than *execution* time.)

DEPTH	NODES	TIME	MEMORY
2	*110*	*0.11 milliseconds*	*107 kilobytes*
4	*11110*	*11 milliseconds*	*10.6 megabytes*
6	10^{6}	*1.1 seconds*	*1 gigabytes*
8	10^{8}	*2 minutes*	*103 gigabytes*
10	10^{10}	*3 hours*	*10 terabytes*
12	10^{12}	*13 days*	*1 petabytes*
14	10^{14}	*3.5 years*	*99 petabytles*

Assumes b=10, 1M nodes/sec, 1000 bytes/node

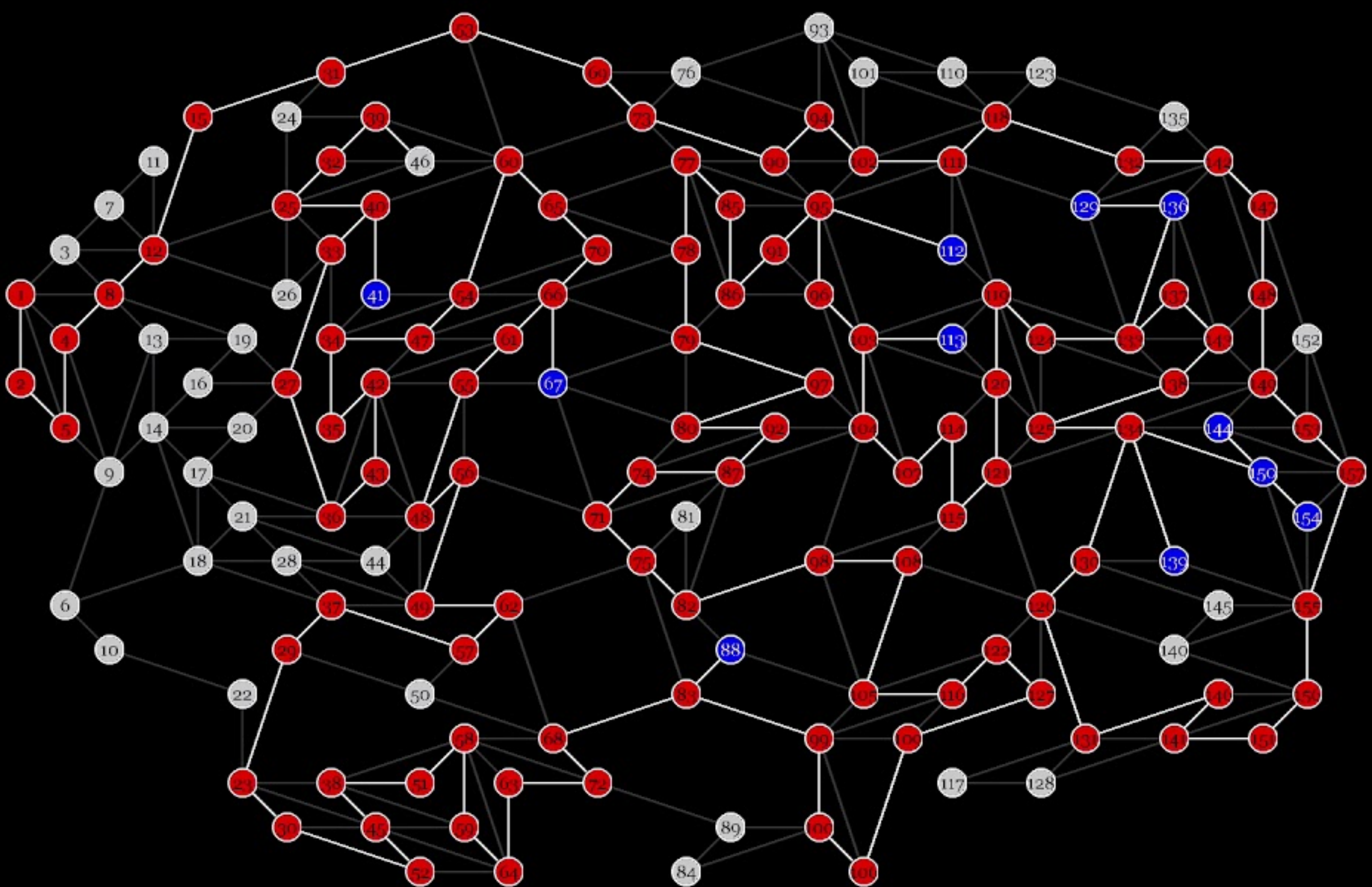
Depth-First Search

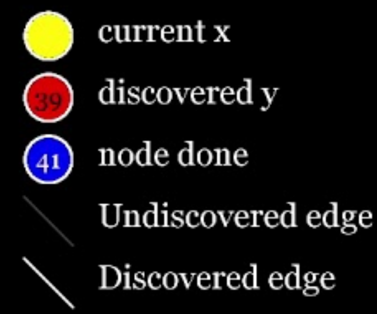
Depth-First Search

Strategy: expand deepest nodes first.

Implementation: frontier is last-in-first-out (LIFO) queue. Put successors at the front of the frontier list.

Key Question: what strategy should we use to pick nodes from the frontier?






```
x = start vertex(1)
dfs(x)

def dfs(x):
   mark x as visited
   for each y in x connections:
      if y not visited then
         dfs(y)
```

Please subscribe @youtube.com/gjenkinslbcc or with icon in lower right >>>

Properties of depth-first search

Complete? No. It fails in infinite-depth spaces.

Optimal? No.

Time Complexity? $O(b^m)$

Space Complexity? $O(b*m)$

m = maximum depth of the state space

Depth: d = depth of the least cost/closest solution

Branching Factor: b = maximum branching factor/ maximum # of children a node can have

Depth-first vs Breadth-first

Use depth-first if

- *Space is restricted*
- There are many possible solutions with long paths and wrong paths are usually terminated quickly
- Search can be fine-tuned quickly

Use breadth-first if

- *Possible infinite paths*
- Some solutions have short paths
- Can quickly discard unlikely paths

Search Conundrum

Breadth-first

- ☑ Complete,
- ☑ Optimal
- ☒ *but* uses $O(b^d)$ space

Depth-first

- ☒ Not complete *unless m is bounded*
- ☒ Not optimal
- ☒ Uses $O(b^m)$ time; terrible if m >> d
- ☑ *but* only uses *O(**b*m) space***