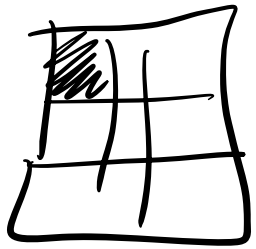
CS 232: Artificial Intelligence

Fall 2026

Prof. Carolyn Anderson
Wellesley College

Reminders

- Homework 2 is due Monday
- My help hours are shifted today: 3:30-4:15pm in H423
- My Monday help hours are normal (3-4pm in H323)
- Tutor hours:
  - Amber: today 1-2 in Whiteboard Alley
  - Coco: Monday 7-8 in Whiteboard Alley
  - Aeka: Thursday 7-8 in ~~L037~~ H103
- Reading for Tuesday: *Thinking Humans* Chapter 8-9
- Reference readings
- Reminder: no food in this classroom

Games —— 

8-tile

Recap

Navigation — Treasure Hunt around Wellesley

Uninformed Search

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?

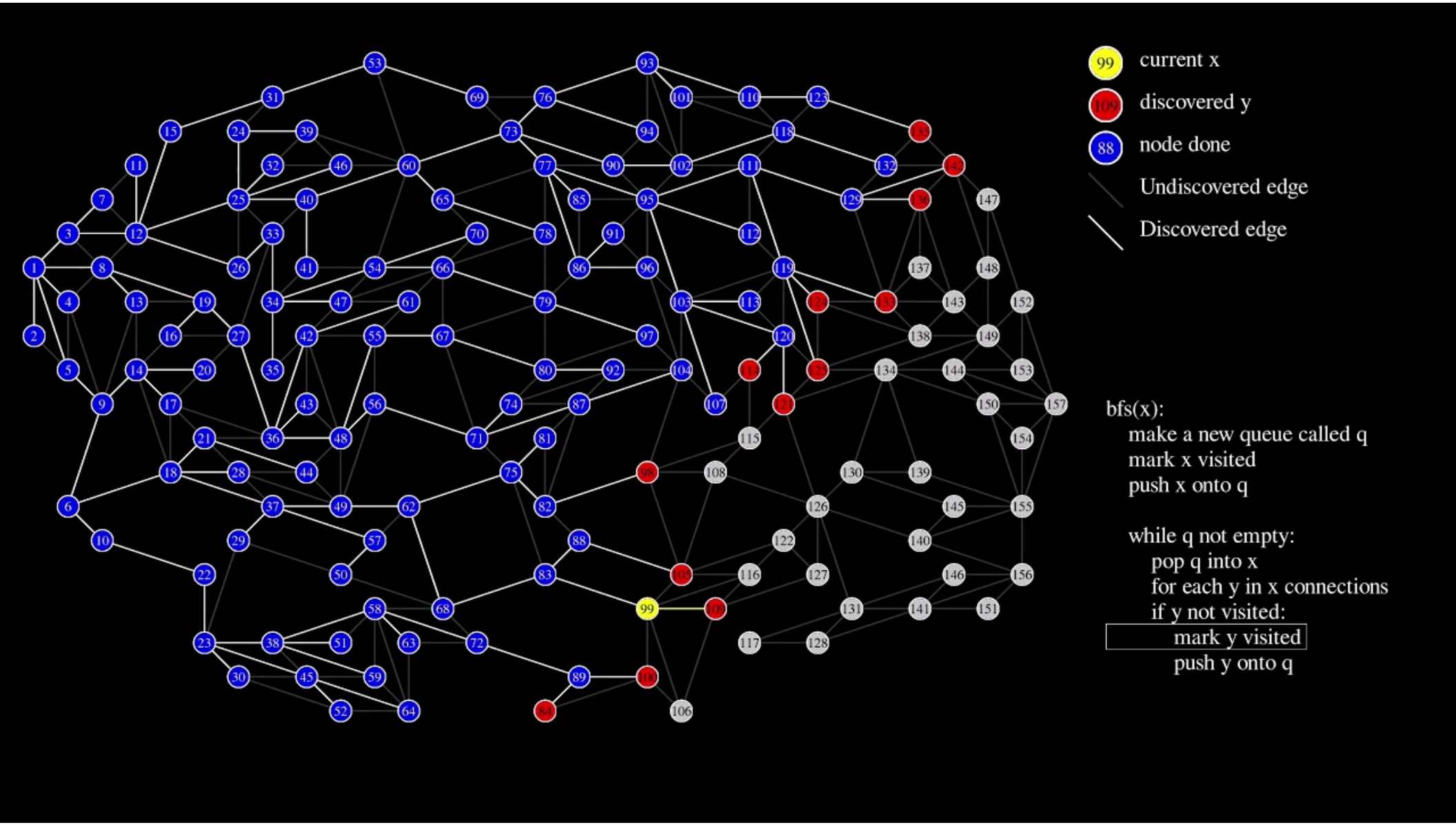
Breadth-First Search

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?


99 current x
109 discovered y
88 node done
Undiscovered edge
Discovered edge
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 (assuming a finite branching factor)

Optimal? Yes. (in # of steps)

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

Space Complexity? $O(b^d)$

Optimal = shortest solution not fastest search.

b: maximum branching factor or # of children any one node can have.

d: depth of the least cost/shortest solution

m: maximum depth of the state space

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

Slides adapted from Chris Callison-Burch

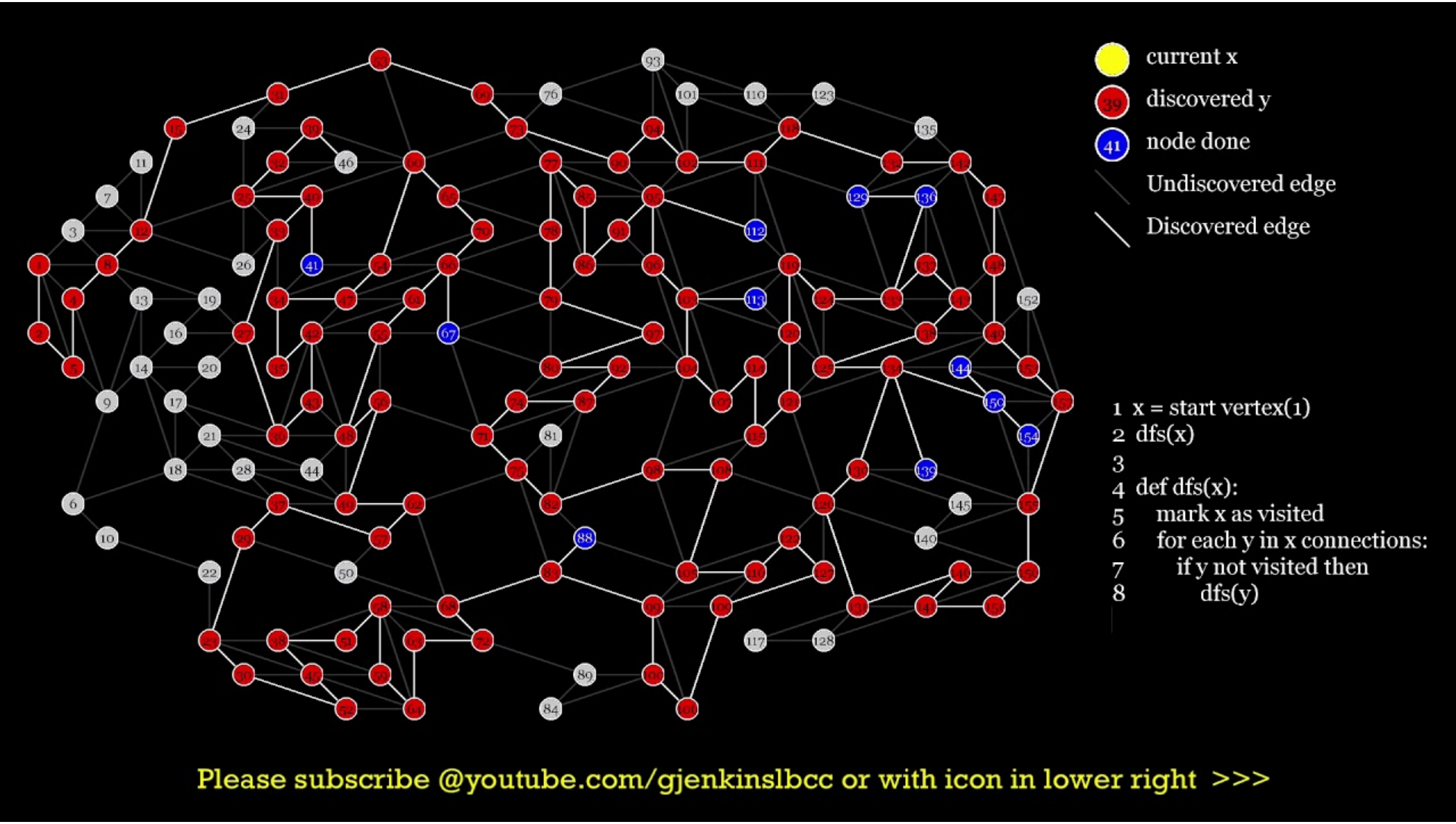
Depth-First Search

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?


current x
discovered y
node done
Undiscovered edge
Discovered edge
1 x = start vertex(1)
2 dfs(x)
3
4 def dfs(x):
5 mark x as visited
6 for each y in x connections:
7 if y not visited then
8 dfs(y)
Please subscribe @youtube.com/gjenkinslbcc or with icon in lower right >>>

Properties of depth-first search

Complete? No. Fails in infinite-depth.

Optimal? No.

Time Complexity? $O(b^m)$

Space Complexity? $O(b * m)$

Optimal = shortest solution not fastest search.

b: maximum branching factor of # of children any one node can have.

d: depth of the least cost/shortest solution

m: maximum depth of the state space

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

Slides adapted from Chris Callison-Burch

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***

Slides adapted from Chris Callison-Burch