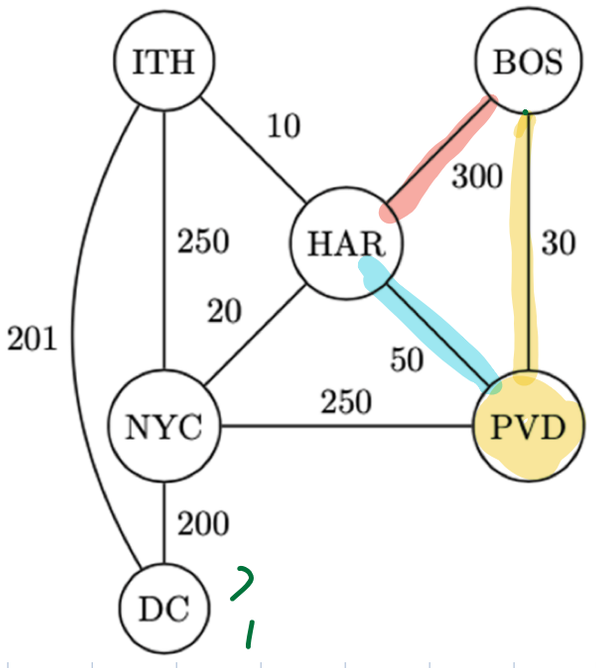
Dijkstra's algorithm example



FIND CHEAPEST PATH BOS → NYC?

NODE	BEST DISTANCE SEEN SO FAR	CAME FROM
BOS	0	PVD → BOS
DC	∞ 291	HAR → BOS PVD
HAR	∞ 300 80	NYC → PVD HAR
ITH	∞ 90	ITH → HAR
NYC	∞ 280 100	DC → ITH
PVD	∞ 30	

(1) PQ: Keep track of nodes we haven't checked yet in a priority queue—at each step, remove the node with the least cost

Order in which nodes are removed from PQ

BOS	PVD	HAR	ITH	NYC	DC
1	2	3	4	5	6

(2) When checking a node: for each of its neighbors, check if cost to reach neighbor through current node is less than known best cost.
If lower cost found, update costs, cameFrom
(Example shown here for checking if PVD->HAR is better than current cost for BOS-HAR)

EG. 30 + 50 < 300

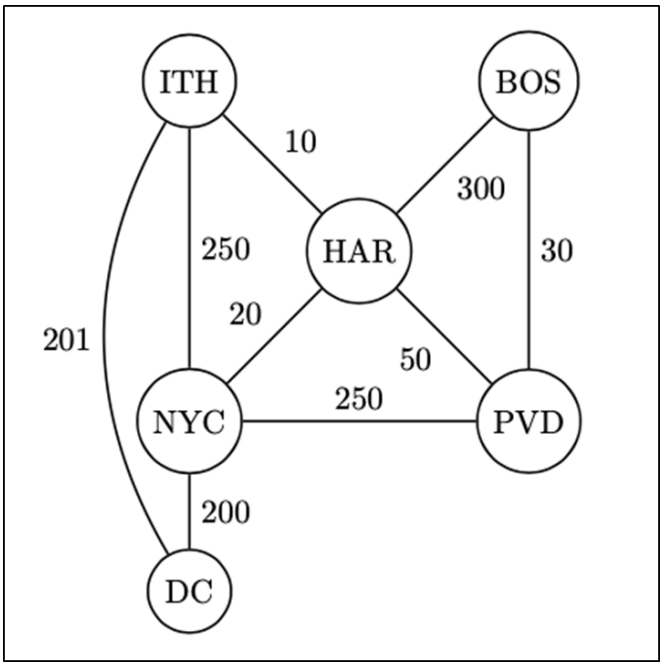
(3) **Results**
- "Table" of costs reflects the lowest cost from BOS (the source) to every other node
- Can use backtrack through canReach from dest -> source to find the least cost path (example on page 3)

Dijkstra is called a **shortest path algorithm** because it can find the least-cost path from source to any other node

See page 6 of this PDF (or the typed notes) for a pseudocode version of the algorithm

[Handout version for reference]

Dijkstra's algorithm



Find the cheapest route from Boston to NYC

toCheckPQ

Name	Route Distance
BOS	∞
DC	~~∞~~ 291
HAR	~~∞~~ ~~360~~ 80
ITH	~~∞~~ 90
NYC	~~∞~~ ~~280~~ 100
PVD	~~∞~~ 30

Order in which items were removed from toCheckPQ:

BOS, PVD, HAR, ITH, NYC, DC
1 2 3 4 5 6

cameFrom

PVD → BOS
HAR → ~~BOS~~ PVD
NYC → ~~PVD~~ HAR
ITH → HAR
DC → ITH

HOW TO USE CAME FROM TO GET SHORTEST PATH?

⇒ "BACKTRACKING". LOOK UP IN TO CHECK WITH DEST

GOAL: BOS → NYC

CAME FROM

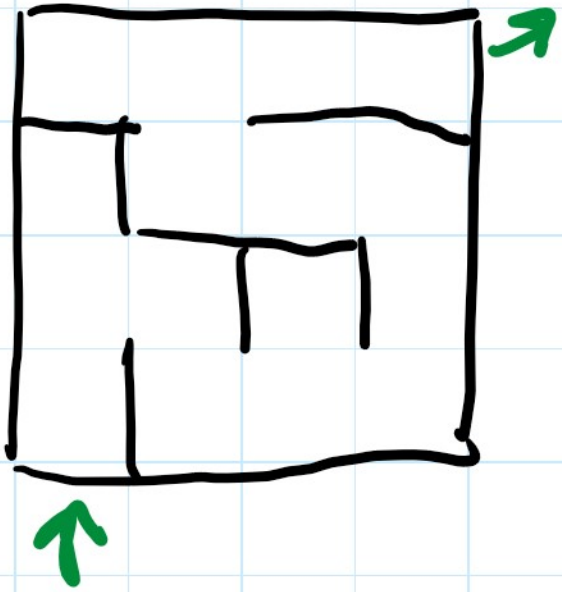
③ PVD → BOS
② HAR → ~~BOS~~ PVD
① NYC → ~~PVD~~ HAR
ITH → HAR
DC → ITH

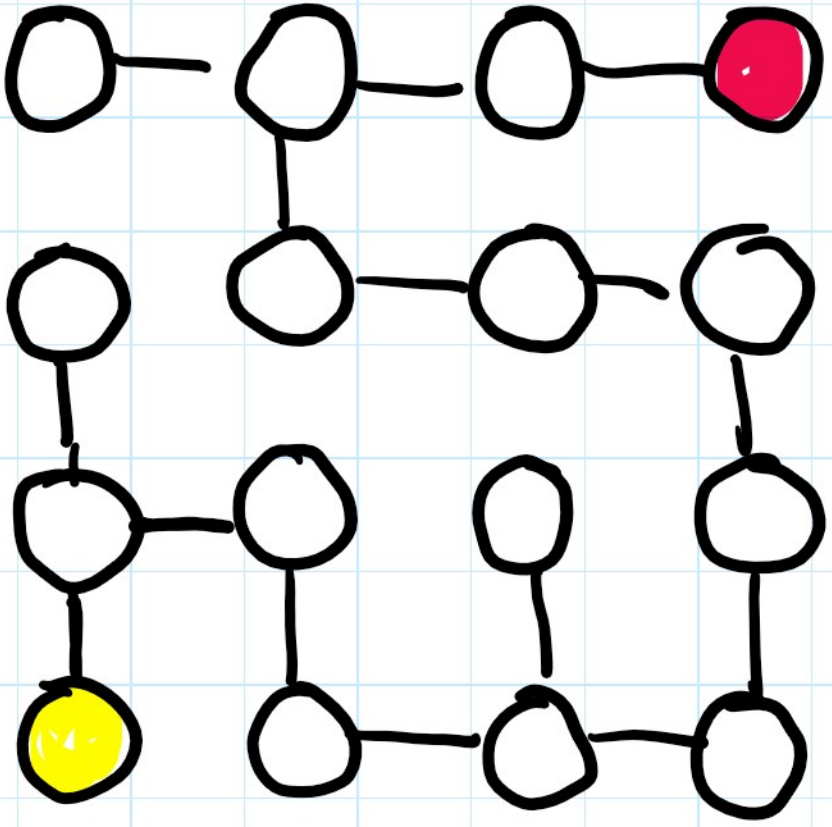
BOS → PVD → HAR → NYC
③ ② ①
↑ SRC
↓ DEST

[For more info, see typed/whiteboard notes from lecture 19]

Representing mazes as graphs

Sunday, October 23, 2022 4:54 PM



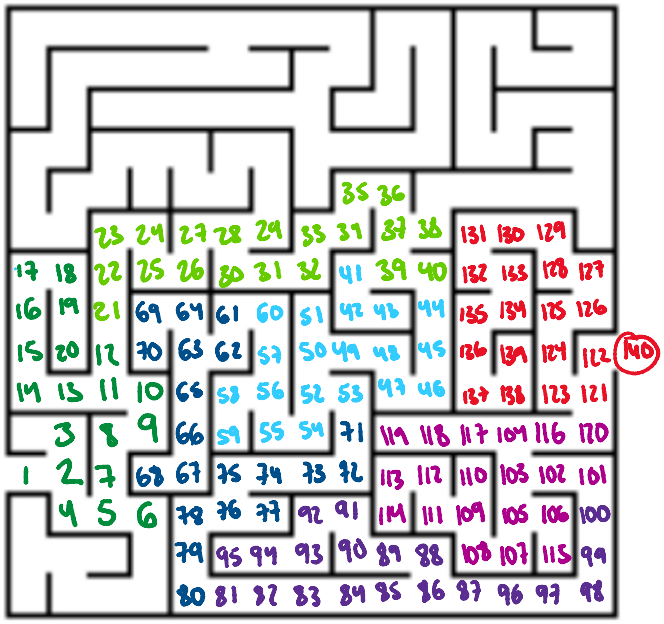


Solving the maze = finding route (DFS or BFS) from vertex that represents starting cell to vertex that represents ending cell

(Attribution: Pages 4-5 drawn by Milda from Fall 2022 version)

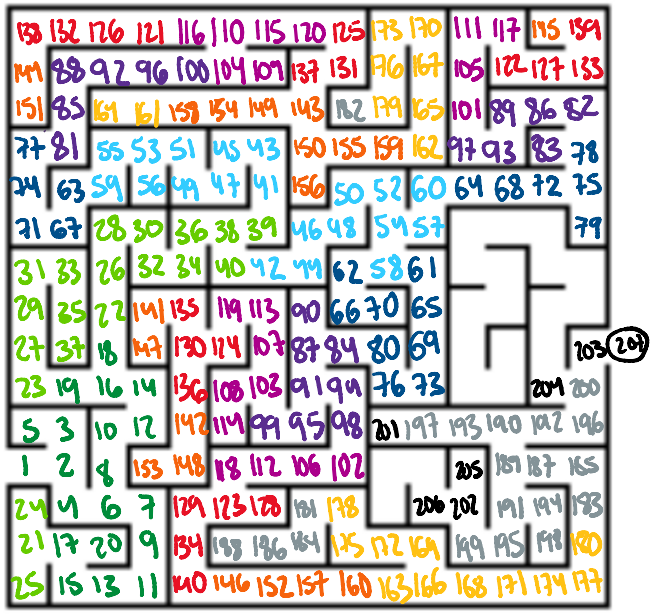
Bigger maze comparison

Monday, October 24, 2022 1:02 PM



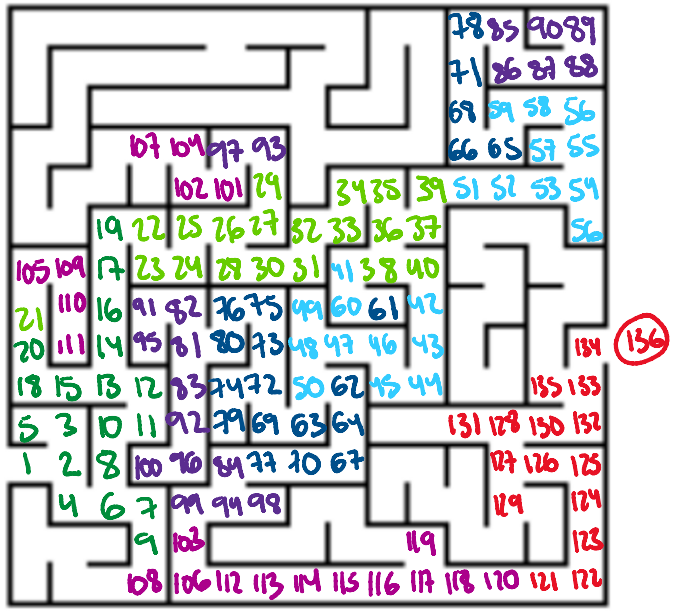
DFS (stack)

Will go down a path until it reaches a dead end and then search from last-seen branching-off point



BFS (queue)

Will "fan out" from the beginning of the maze (tracking many routes at once)



A* (priority queue)

Prioritizes based on distance to the end -- turns out to be fastest for most mazes

*A note on how these mazes were labeled: the number represents the timestep when that cell was *added* to the toCheck stack/queue/priority queue. Neighbors are checked in the order right, up, left, down (a different ordering can result in different numberings/traversals for the mazes). For A*, Manhattan distance is used and ties are broken by considering the cell that was added to the PQ earlier (has a lower timestep number). Colors change every 20 steps.*

Could we use Dijktra's algorithm to search the maze? BFS/DFS/A* are search algorithms (goal: find path to destination), whereas Dijkstra shortest path algorithm (ie, find shortest path to **any** node from source)—these are different types of algorithms and best-suited for different use cases! We'll talk about the runtime for BFS/DFS/Dijkstra in the next class.

BFS/DFS Pseudocode

```
while (! toCheck.isEmpty()) {
  Vertex<T> checkingVertex = toCheck.removeLast(); // removeFirst() for BFS
  if (dest.equals(checkingVertex)) {
    return true;
  }
  for (Vertex<T> neighbor : checkingVertex.getOutgoing()) {
    if (!visited.contains(neighbor)) {
      visited.add(neighbor);
      toCheck.addLast(neighbor);
    }
  }
}
```

Dijkstra Pseudocode

```
toCheckQueue = V (prioritized on routeDist)
cameFrom = empty map

for v in V:
  v.routeDist = inf
source.routeDist = 0

while toCheckQueue is not empty:
  checkingV = toCheckQueue.removeMin()
  for neighbor in checkingV's neighbors:
    if checkingV.routeDist + cost(checkingV, neighbor) < neighbor.routeDist:
      neighbor.routeDist = checkingV.routeDist + cost(checkingV, neighbor)
      cameFrom.add(neighbor -> checkingV)
      toCheckQueue.decreaseValue(neighbor)

backtrack from dest to source through cameFrom
```