GRAPHS, part 2

cs2420 | Introduction to Algorithms and Data Structures | Spring 2015

administrivia…

-assignment 8 due tonight

-assignment 9 is out … due in 2.5 weeks

-midterm in two weeks!

-spring break next week

  -no class

  -no TA office hours

last time…

-trees are a *subset* of graphs

-a **graph** is a set of *nodes* connected by **edges**
  -an edge is just a link between two nodes
  -nodes don't have a parent-child relationship
  -links can be bi-directional

-graphs are used **EXTENSIVELY** throughout CS

-graphs have no root; must store all nodes

```
class Graph<E> {
  List<Node> nodes;
  …
}
```

-implementation is more general than a tree

```
class Node{
  E Data;
  List<Node> neighbors;
  …
}
```

-the order in which neighbors appear in the list is unspecified

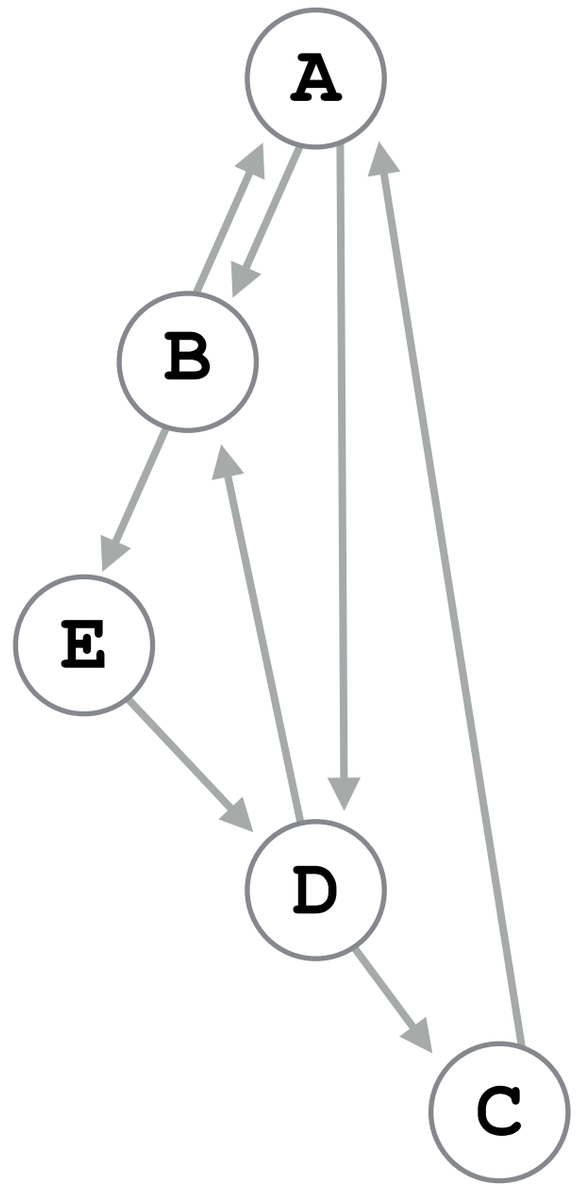
  -a different order still make the same graph!

pathfinding

-there may be more than one path from one node to another

-we are often interested in the *path length*

-finding the shortest (or cheapest) path between two nodes is a common graph operation



-depth-first search (just like a tree) — DFS

-breadth-first search — BFS

-if there exists a path from one node to another these algorithms will find it

  -the nodes on this path are the steps to take to get from point A to point B

-if multiple such paths exist, the algorithms may find different ones

topological sort

-consider a graph with no cycles

-a topological sort orders nodes such that…
    -if there is a path from node A to node B, then A appears before B in the sorted order

-example: scheduling tasks
    -represent the tasks in a graph
    -if task A must be completed before task B, then A has an edge to B

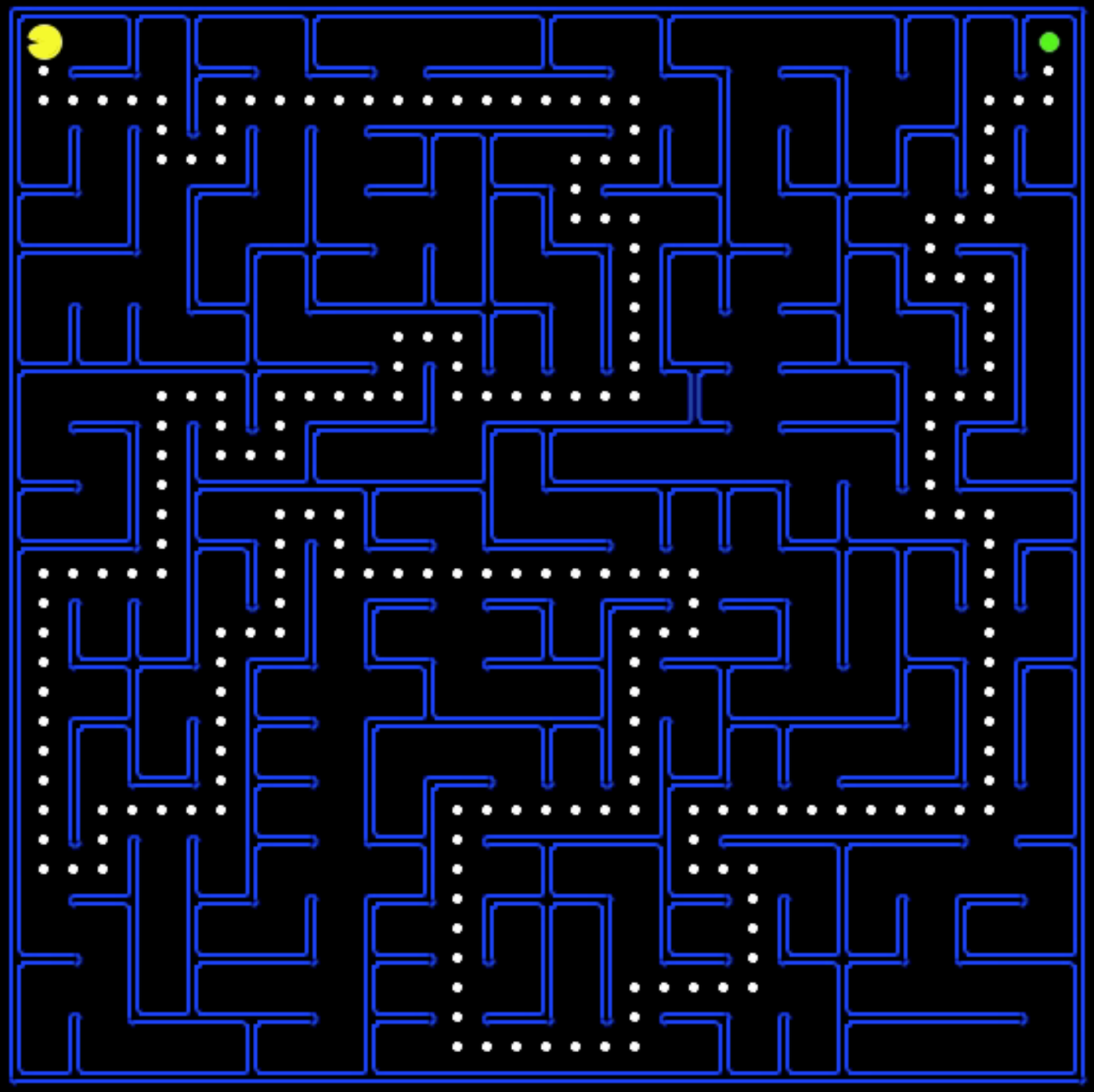
today…

-BFS and the homework

-weighted graphs

-dijkstra’s algorithm

-exam topics

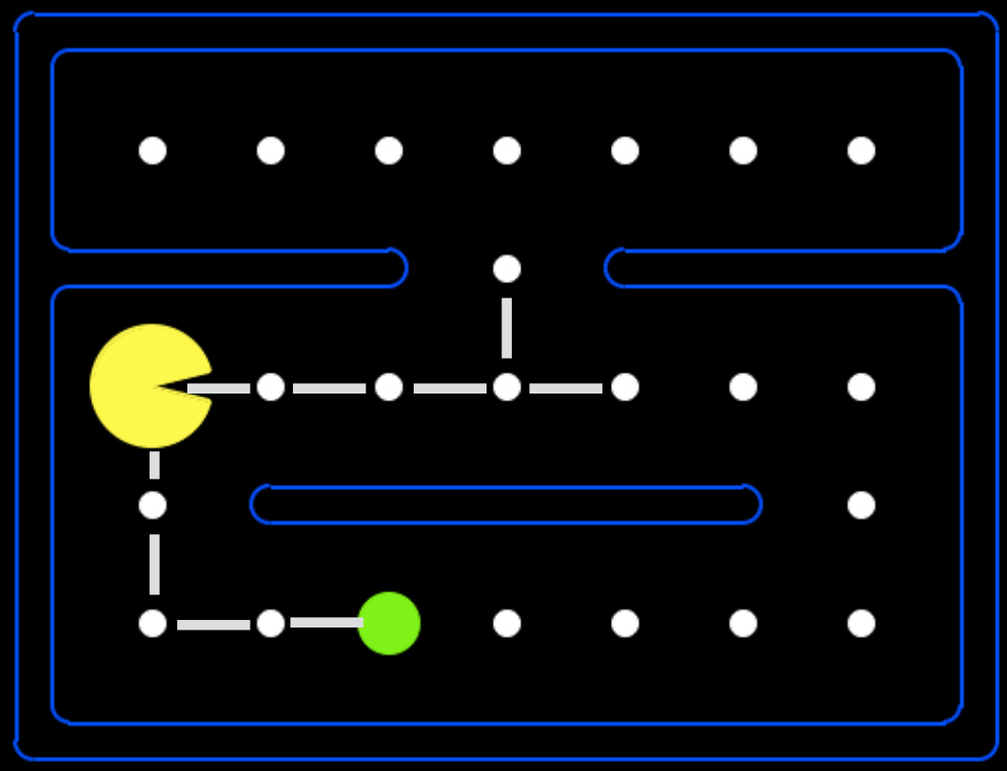


-in this graph, every spot is a node

-nodes have edges with adjacent spots

-expand out equally in all directions from starting node

-when any path reaches the goal node… done!



WHAT TYPE OF SEARCH ARE WE DOING HERE?

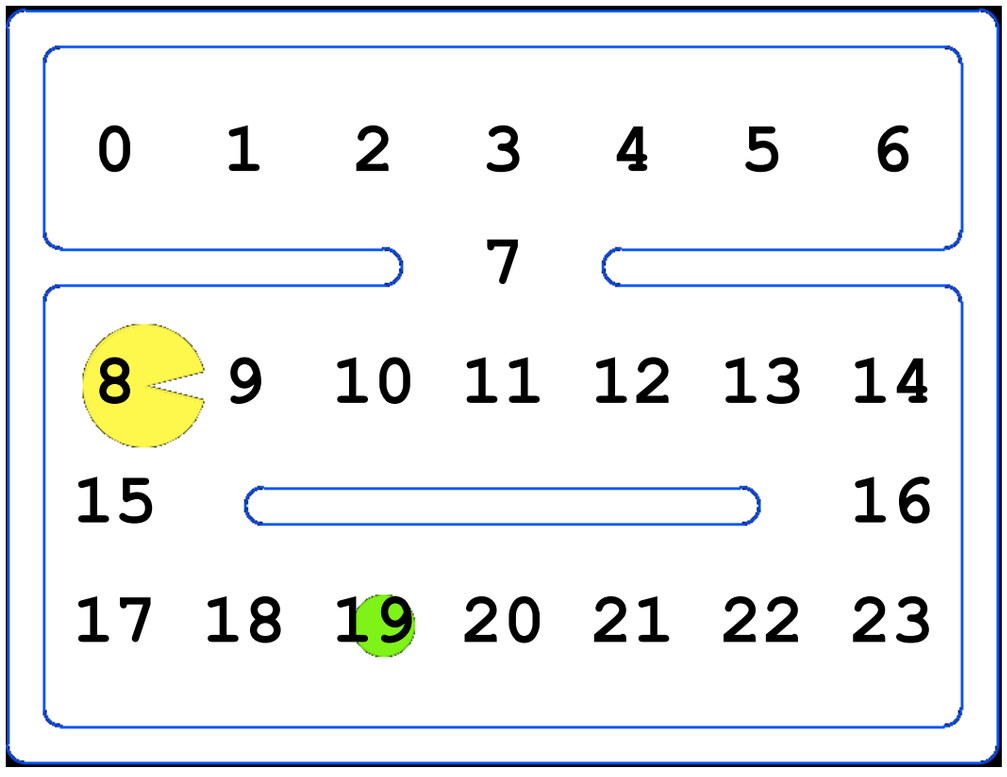
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue



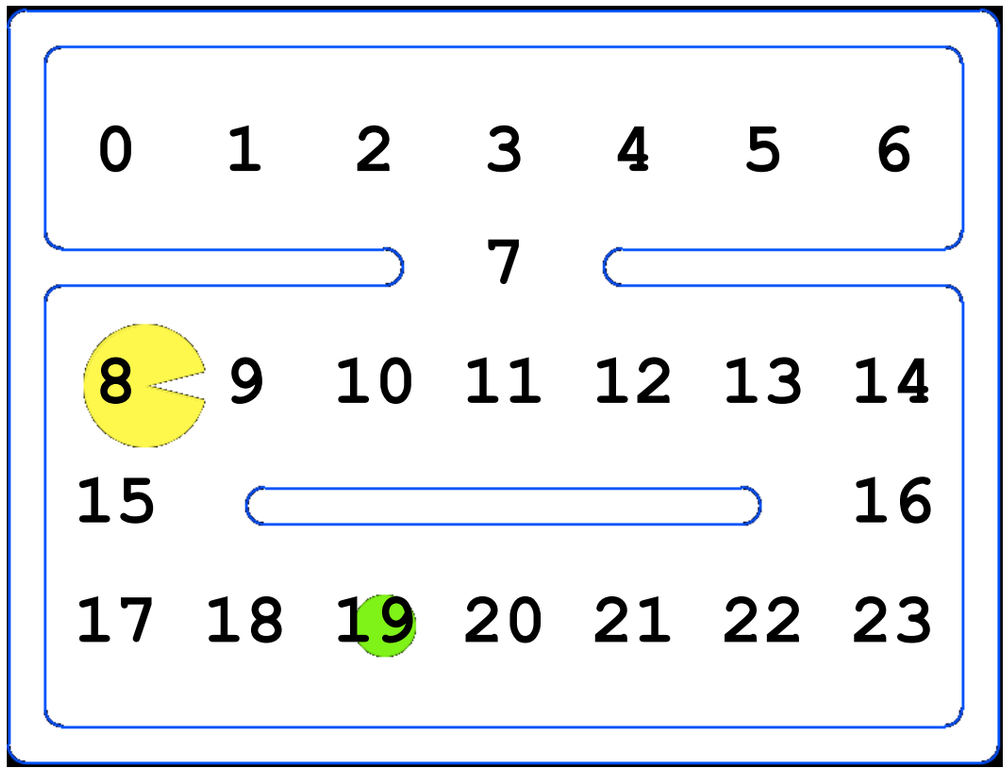
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue



current:

queue:

VISITED

CAME FROM

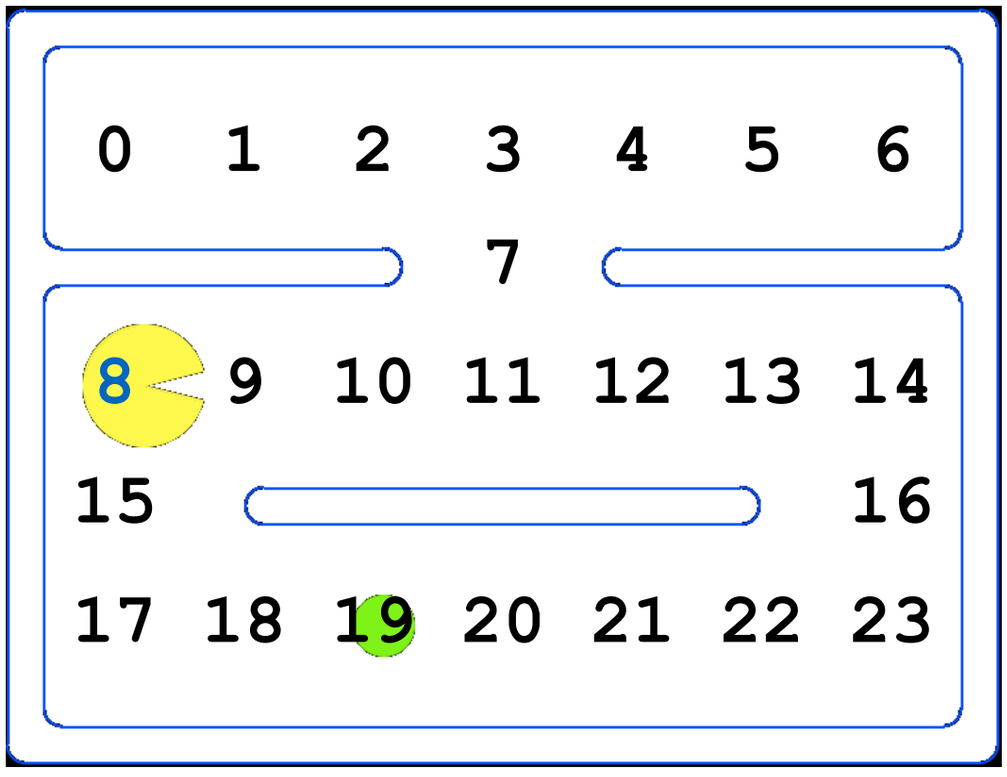
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

-dequeue the current node

-if current == goal, done!

-otherwise, mark current's neighbors as visited and add them to the queue



current:

queue: **8**

VISITED

CAME FROM

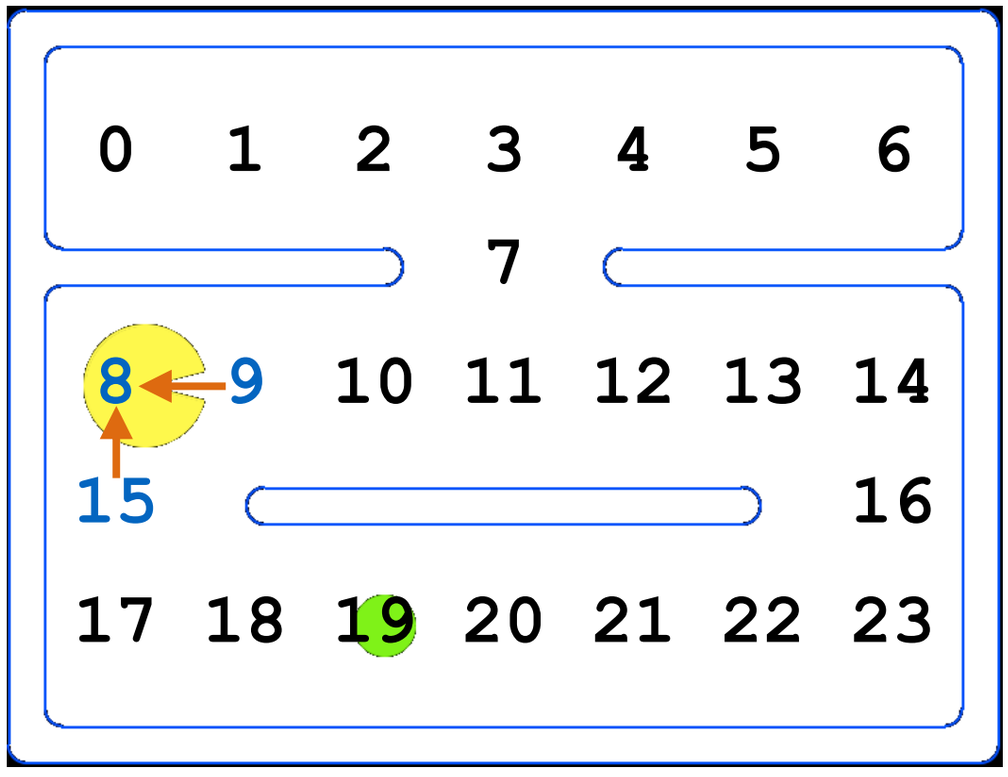
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

-dequeue the current node

-if current == goal, done!

-otherwise, mark current's neighbors as visited and add them to the queue




```
current: 8
  queue: 15  9
```

VISITED

CAME FROM

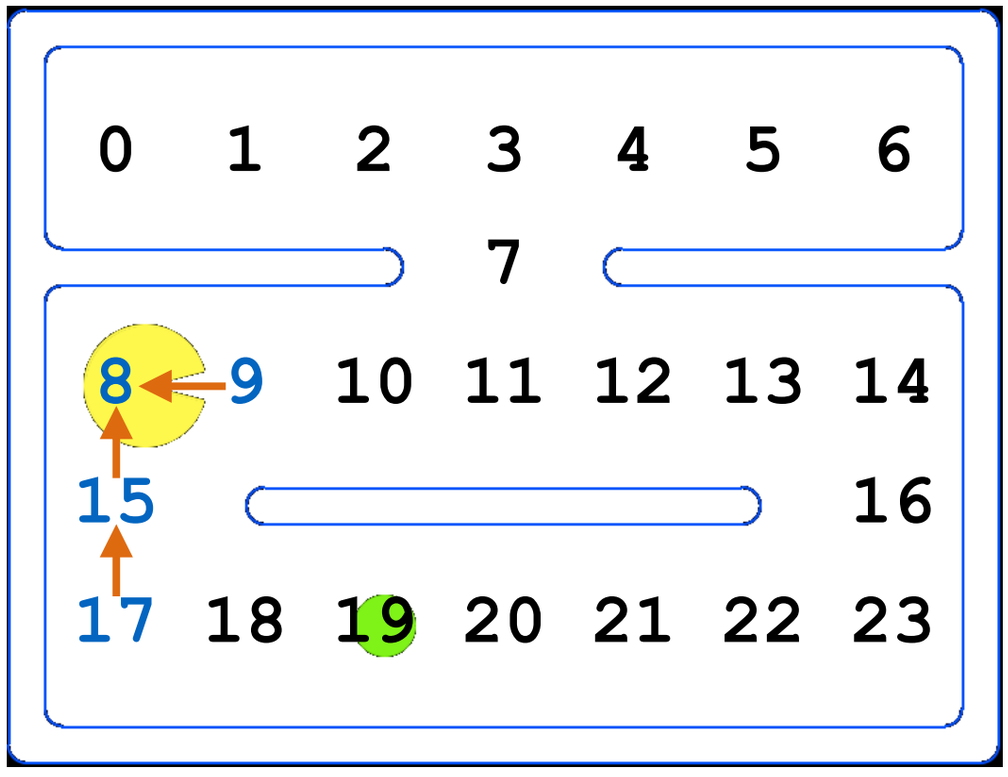
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue




```
current:15
  queue: 9  17
```

VISITED

CAME FROM

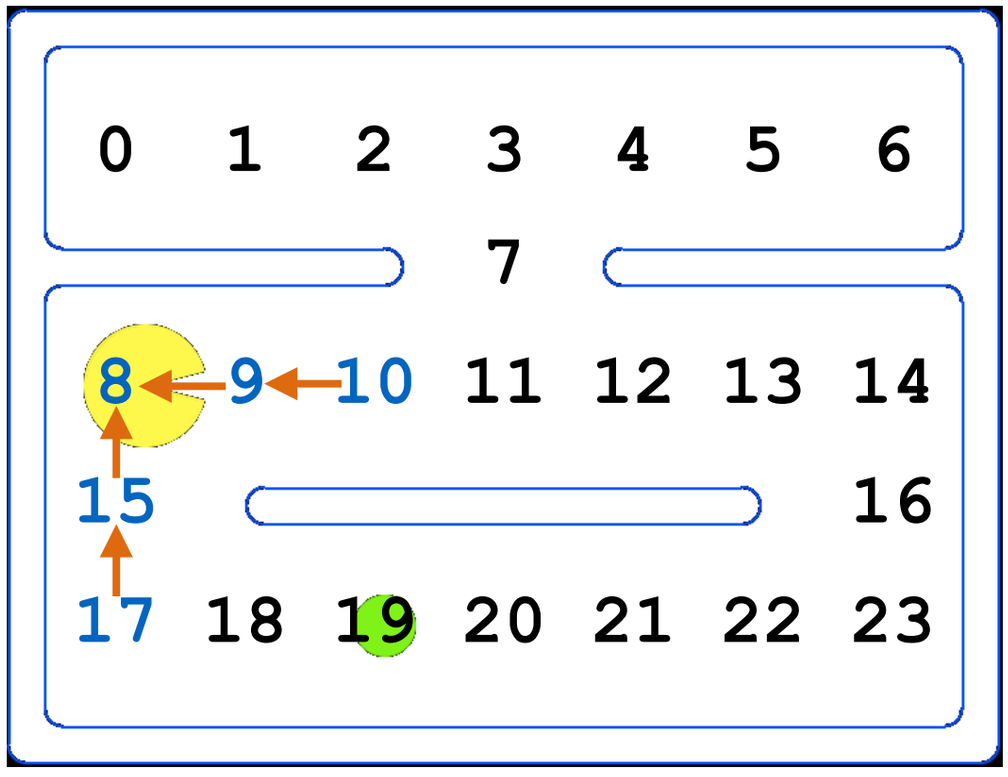
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue




```
current: 9
  queue:17 10
```

VISITED

CAME FROM

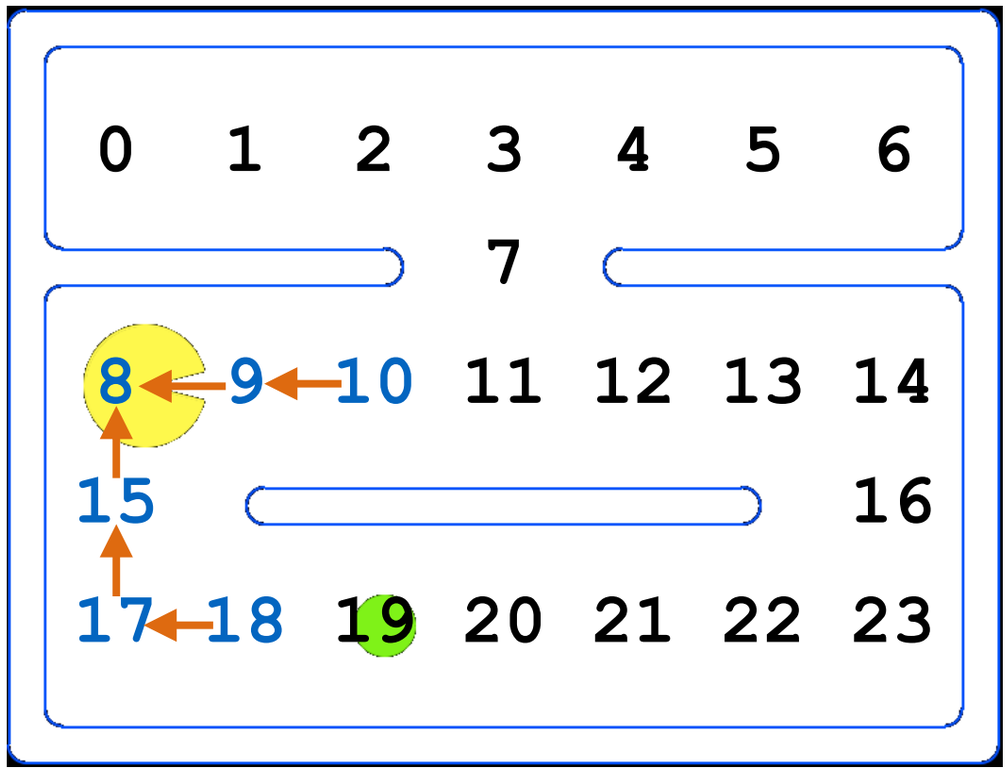
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

-dequeue the current node

-if current == goal, done!

-otherwise, mark current's neighbors as visited and add them to the queue



current: 17

queue: 10 18

VISITED

CAME FROM

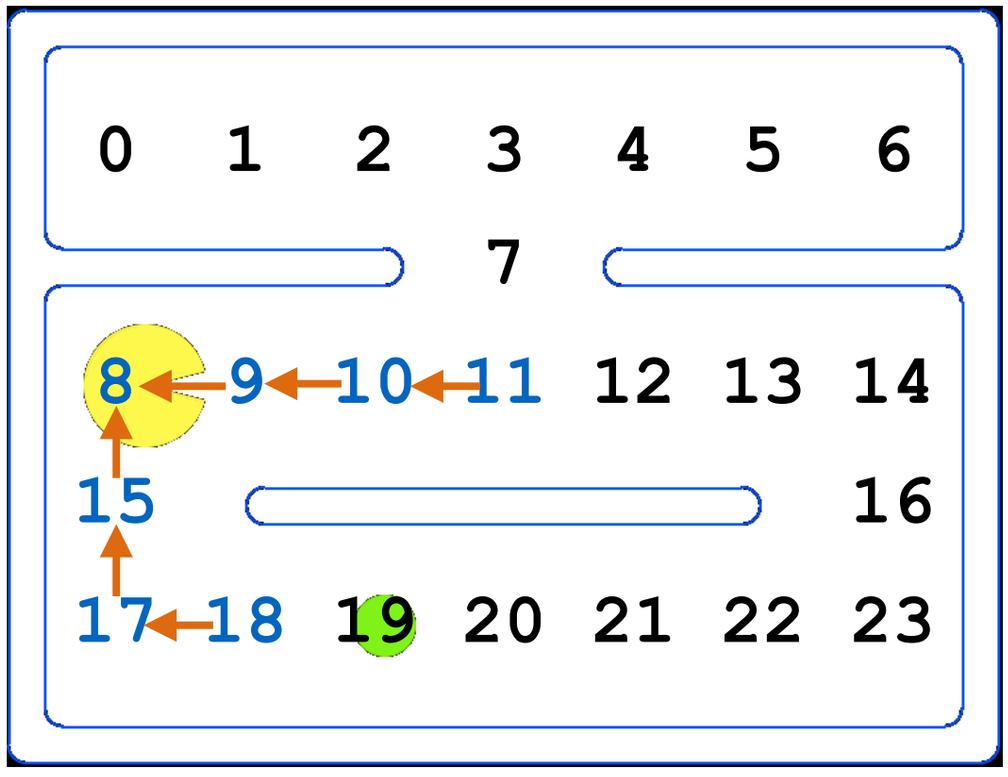
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue



current: 10

queue: 18 11

VISITED

CAME FROM

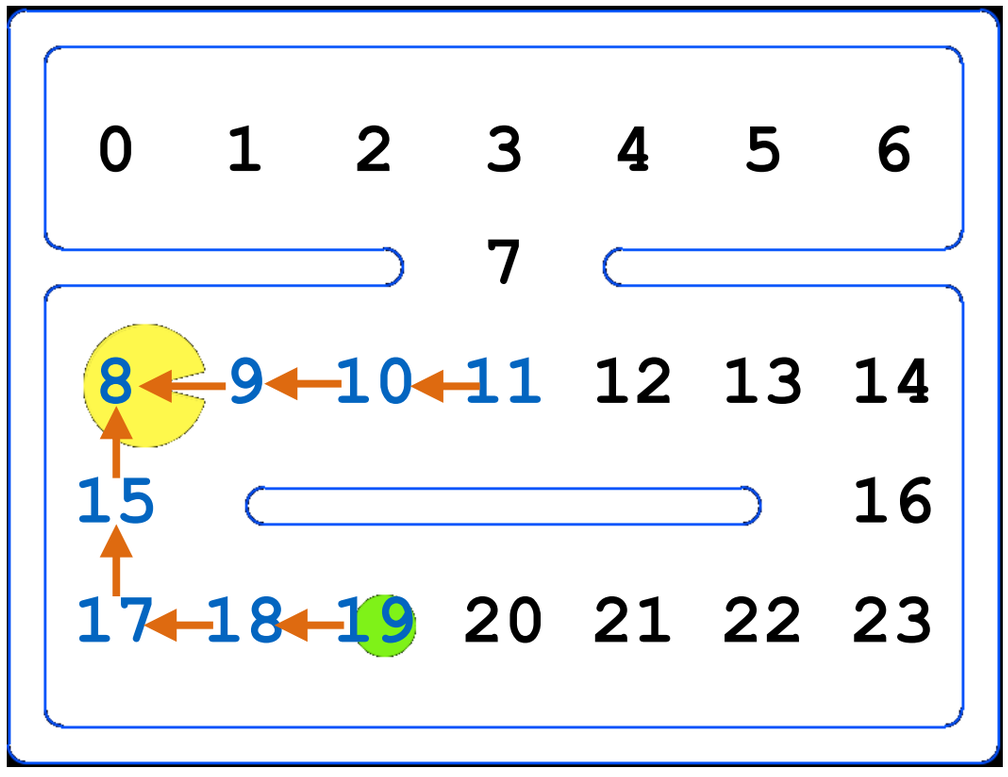
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue



current: 18

queue: 11 19

VISITED

CAME FROM

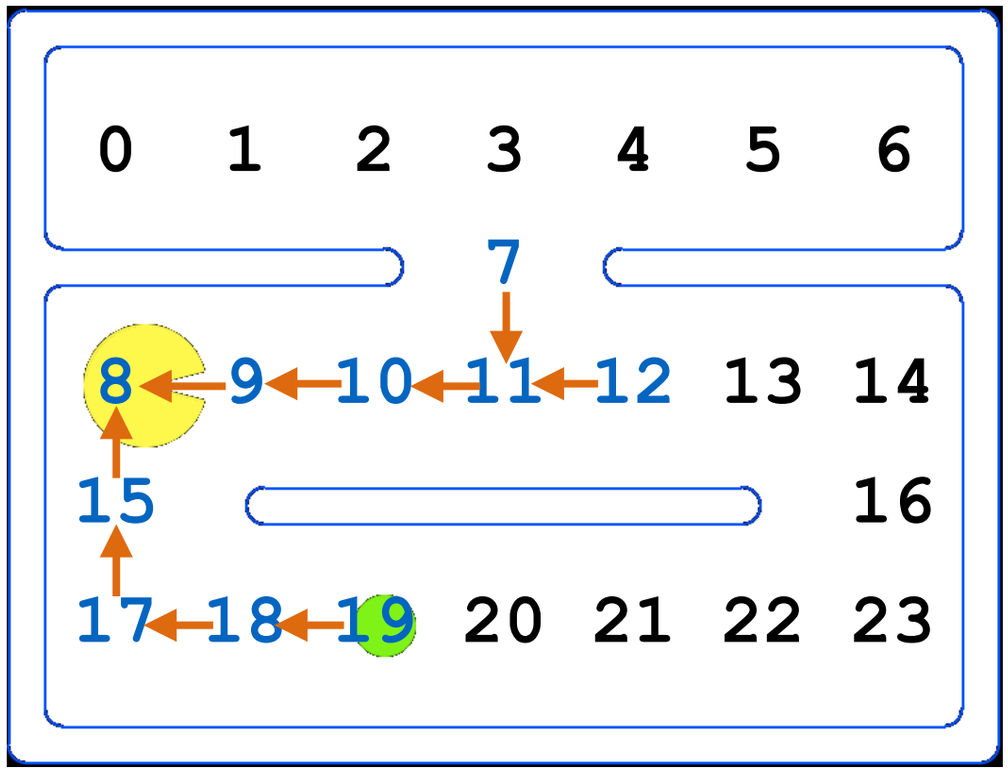
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue



current: 11

queue: 19 7 12

VISITED

CAME FROM

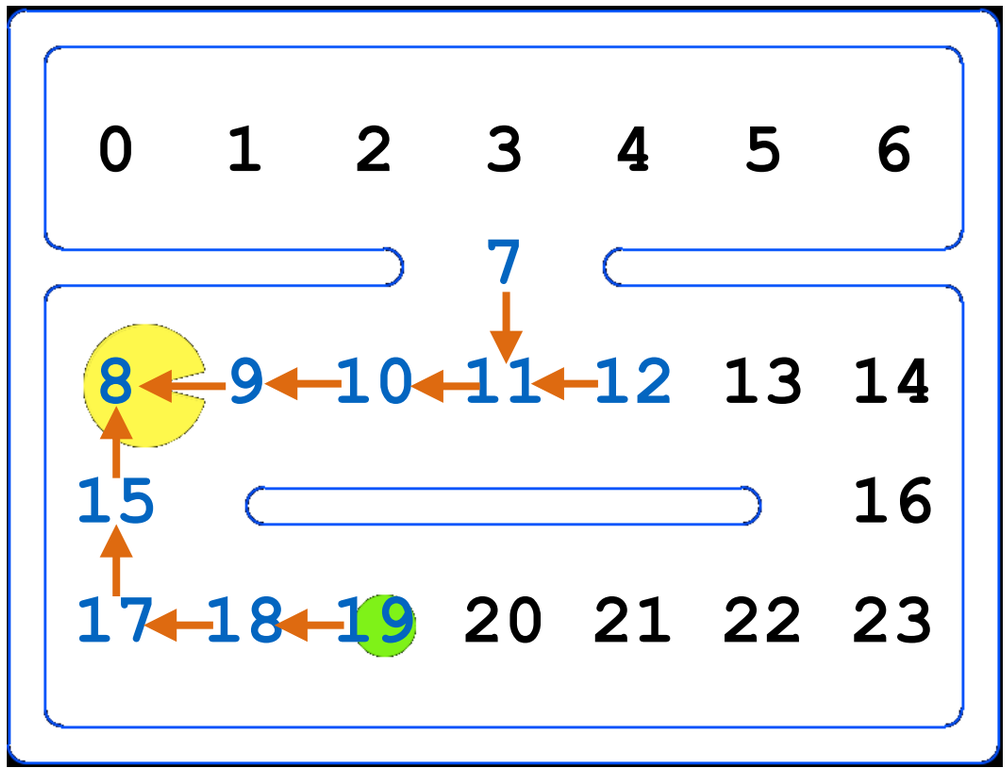
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue




```
current:19 GOAL!
  queue: 7  12
```

VISITED

CAME FROM

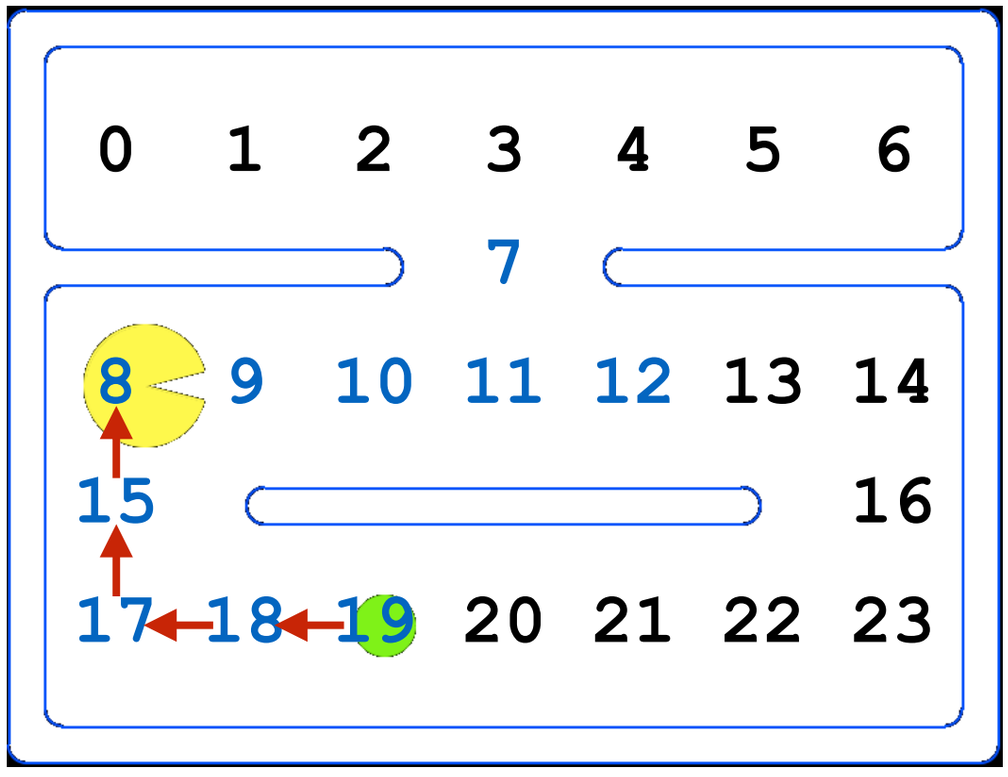
-put the starting node in the queue and mark it as visited

-while the queue is not empty:

  -dequeue the current node

  -if current == goal, done!

  -otherwise, mark current's neighbors as visited and add them to the queue




```
current:19 GOAL!
  queue: 7  12
```

VISITED

CAME FROM

-reconstruct the path…

-shortest pathfinding problem

-the problem:

-input: a simple text file describing the maze

-includes wall locations, start point, and end point

-output: a similar text file with the shortest path from start to end indicated in the maze

-represent all possible moves with a graph, then do a breadth first search

INPUT

```
5 10
XXXXXXXXXX
X S      X
X        X
X    G   X
XXXXXXXXXX
```

OUTPUT

```
5 10
XXXXXXXXXX
X S....  X
X     .  X
X    G   X
XXXXXXXXXX
```

X WALL SEGMENT

S STARTING POINT

G GOAL

AN OPEN SPACE

. SOLUTION PATH INDICATOR

INPUT

```
5 10
XXXXXXXXXX
X S      X
XXXXXXXX X
X G      X
XXXXXXXXXX
```

OUTPUT

```
5 10
XXXXXXXXXX
X S.....X
XXXXXXXX.X
X G.....X
XXXXXXXXXX
```

X WALL SEGMENT

S STARTING POINT

G GOAL

AN OPEN SPACE

. SOLUTION PATH INDICATOR

INPUT

```
10 19
XXXXXXXXXXXXXXXXXXXX
X        S          X
X                   X
X                   X
XX X        X       X
X X XXX        X    X
X G    XX           X
X                   X
X                   X
XXXXXXXXXXXXXXXXXXXX
```

OUTPUT

```
10 19
XXXXXXXXXXXXXXXXXXXX
X        S.         X
X         .         X
X         .         X
XX X      . X       X
X X XXX   .    X    X
X G....XX.          X
X      ....         X
X                   X
XXXXXXXXXXXXXXXXXXXX
```

X WALL SEGMENT

S STARTING POINT

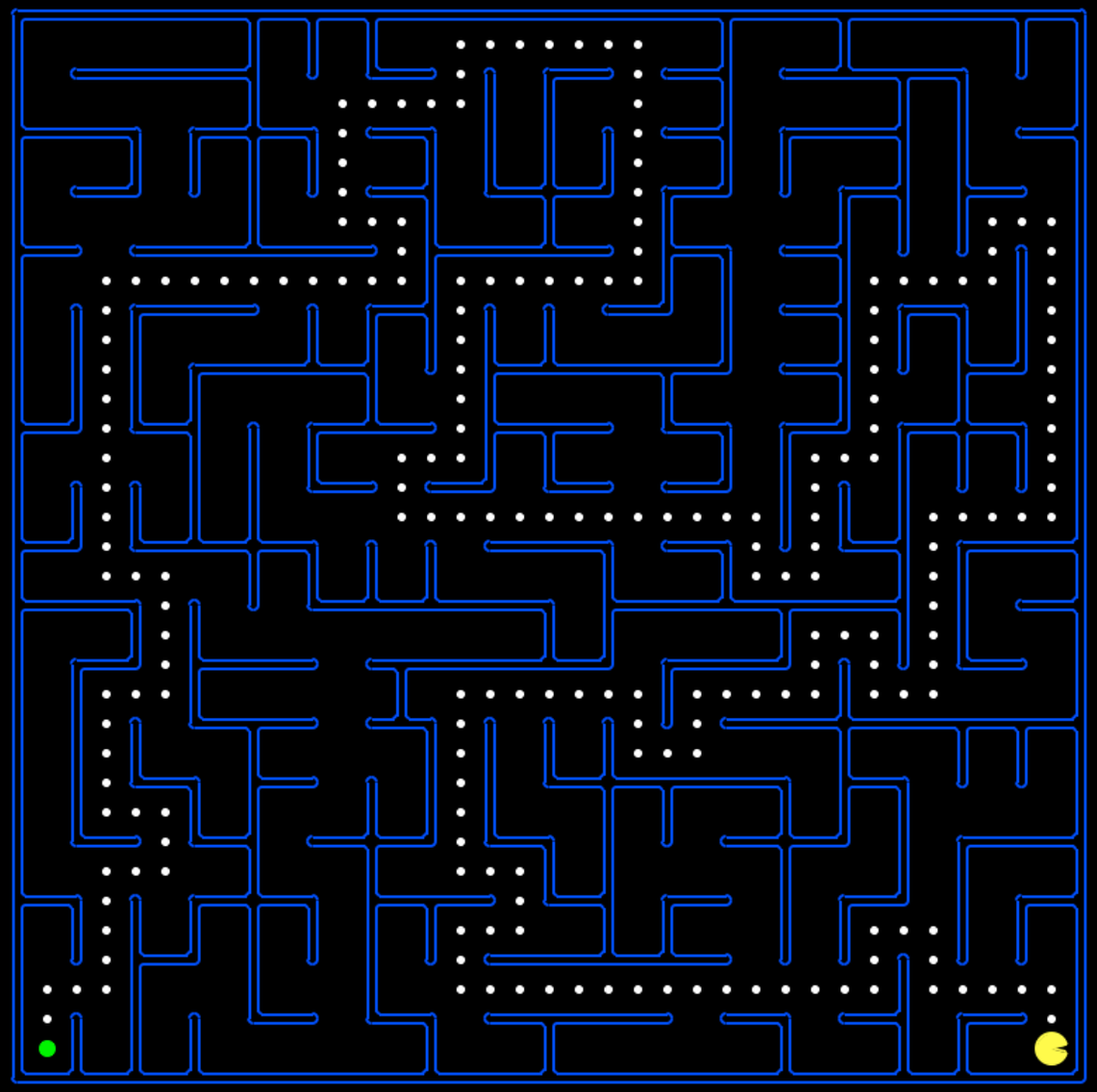
G GOAL

AN OPEN SPACE

. SOLUTION PATH INDICATOR

CAN ANY NODE HAVE AN EDGE TO ANY OTHER NODE?

HOW DO WE REPRESENT WALLS?



-for this specific problem, we can store the graph as a 2D array

```
Node nodes[][];
nodes = new Node[5][10;
```

-`Node` class doesn't need a list of edges

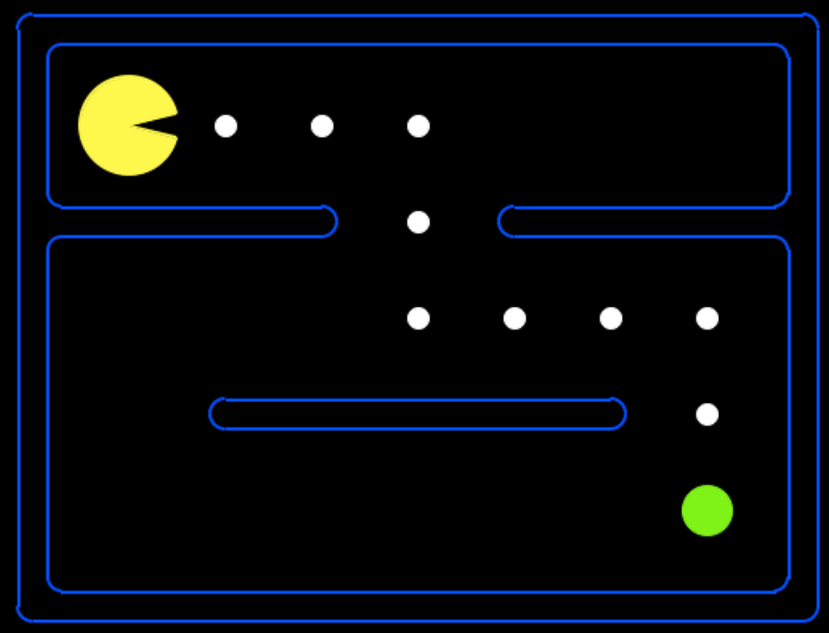
-neighbors are implied (up, down, left right)

-ie. for node `N`, the up neighbor would be:

`nodes[N.row-1][N.col]`

-walls are `null`

-ie. no neighbor if `null`



-while reading the input:

-for every character that makes up the maze `[i][j]`

-if it is a wall

```
nodes[i][j] = null
```

-else

```
nodes[i][j] = new Node(…)
```

-make sure to hand the start and goal nodes

rules

-the path cannot go through or on top of walls

-the path must be connected (no skips or jumps)

-diagonally-adjacent spaces are not connected
    -only up, down, left, right

-if no path exists, the output file will have no dots

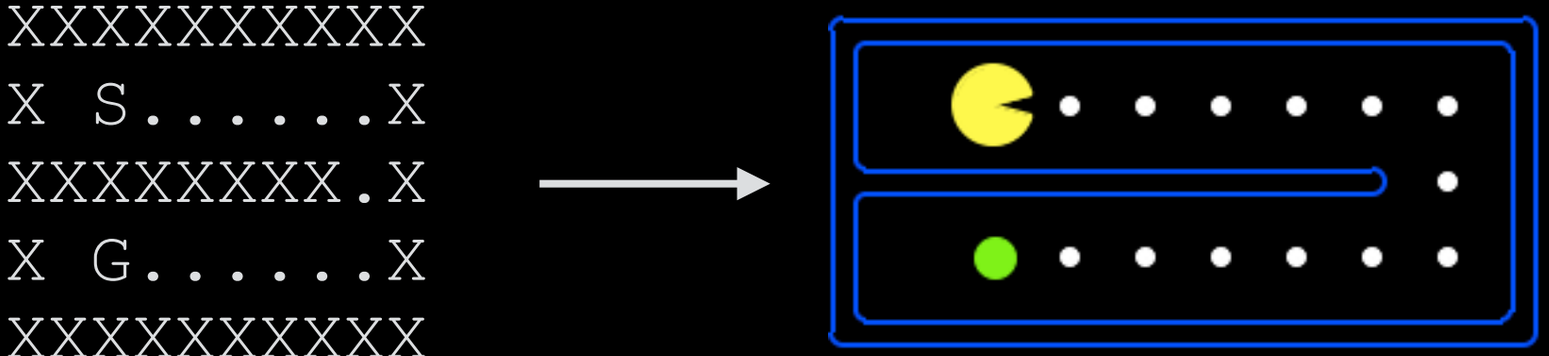
-if multiple shortest paths exist, any of them are valid

-must produce output in exact format specified

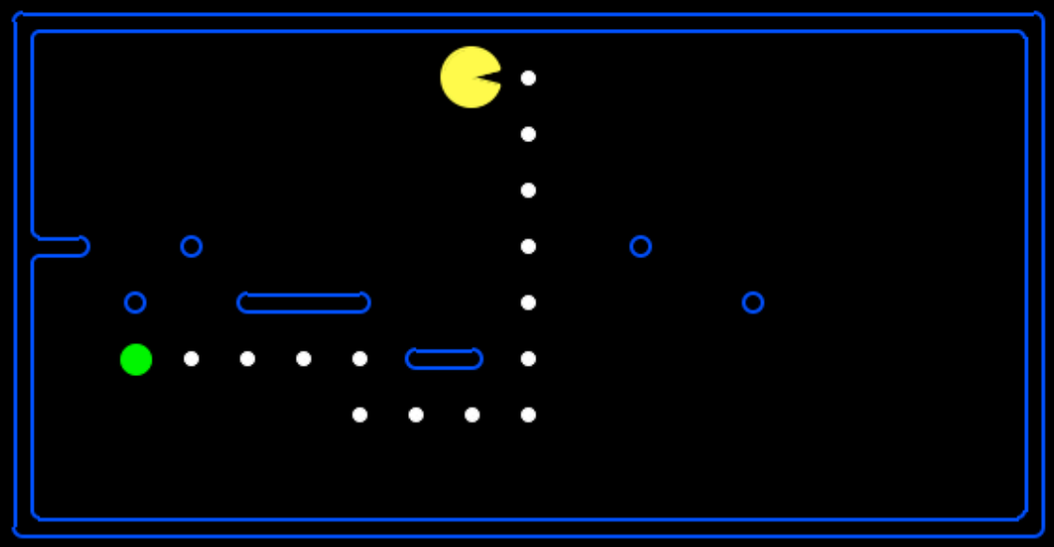
-all you have to do is read in a file and produce a new file

-BUT, we are providing a program to read in your solution and display it as a pacman game board

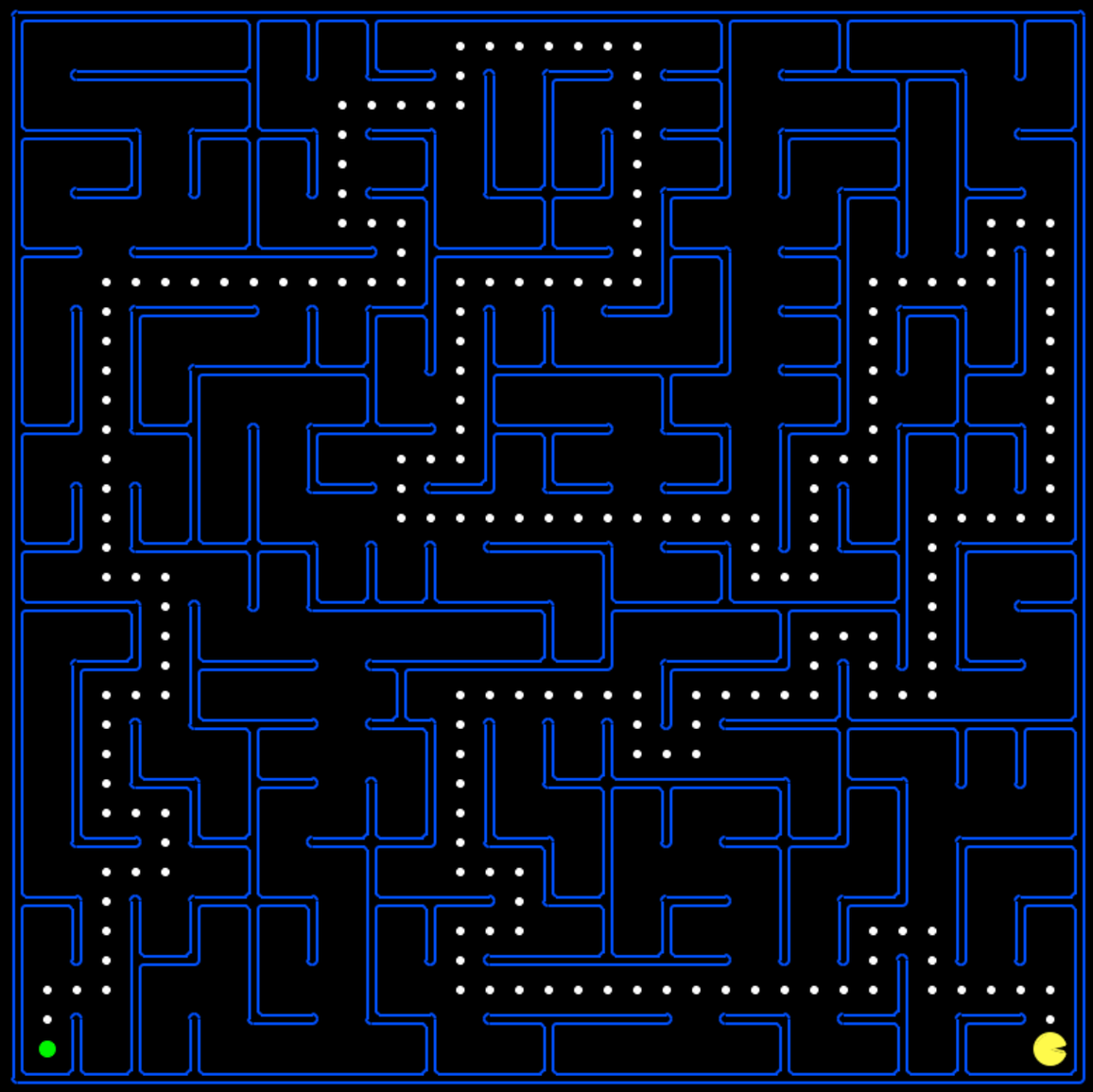
```
5 10
XXXXXXXXXX
X S.....X
XXXXXXXX.X
X G.....X
XXXXXXXXXX
```



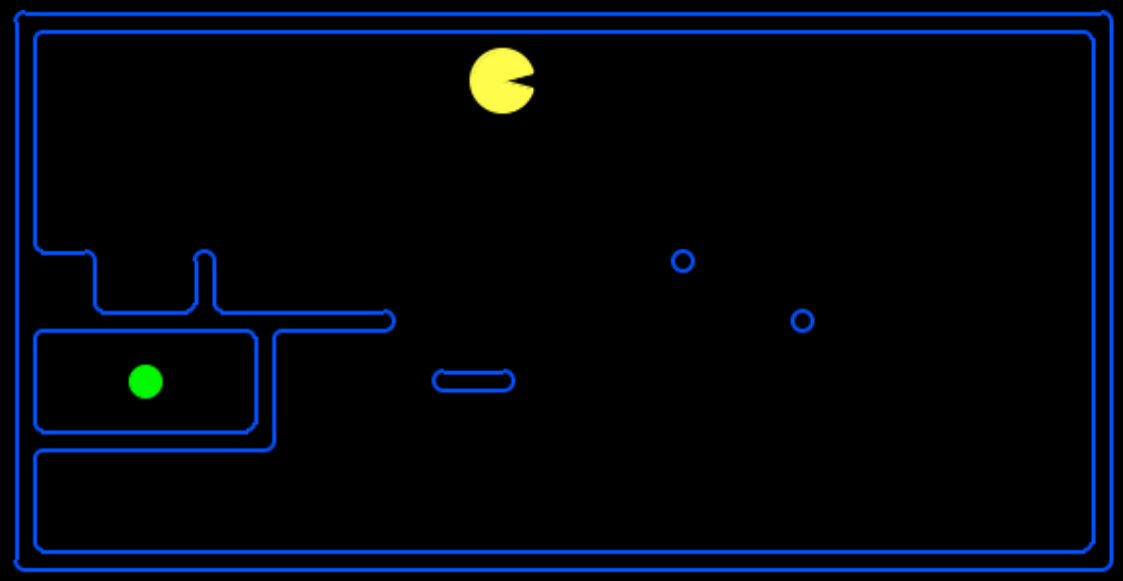
```
XXXXXXXXXXXXXXXXXXXX
X         S.       X
X          .       X
X          .       X
XX  X      . X     X
X  X  XXX  .   X   X
X G....XX.         X
X      ....        X
X                  X
XXXXXXXXXXXXXXXXXXXX
```



```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X       X X X  .......  X   X     X X
X XXXXXXX X XXX.X XXX.XXX XXXXXXX X X
X       X  .....X X  .  X     X X   X
XXXXX XXXXX.XXX X X X.XXX XXXXX X XXX
X   X X X X X.  X X X X.  X X   X X   X
X XXX X X X.XXX XXXXX.XXX X XXX XXX X
X       X  ...X   X  .X     X X X...X
XXX XXXXXXXXX.XXXXXXX.XXX XXX X X.X.X
X  ...........X.......X X   X.....X.X
X X.XXXXX X XXX.X X XXX X XXX.XXX X.X
X X.X     X X X.X X     X   X.X X X.X
X X.X XXXXXXX X.XXXXXXXXX XXX.X XXX.X
X X.X X     X  .X     X     X.  X  .X
XXX.XXX X XXXXX.XXXXX XXX XXX.XXXXX.X
X   .  X X X  ...X X     X X...X X X.X
X X.X X X XXX.XXX XXX XXX X.X X X X.X
X X.X X X    ............X.X X.....X
XXX.XXXXXXX X X XXXXX XXX.X.XXX.XXXXX
X  ...  X X X X     X   X...  X.X   X
XXXXX.X X XXXXXXXXX XXXXXXXXXXX.X XXX
X   X.X           X X     X...X.X   X
X XXX.XXXXX XXXXXXXXX XXXXX.X.X.XXX X
X X...X      X .......X.....X...    X
X X.X XXXXX XXX.X X X.X.XXXXXXXXXXXXX
X X.X   X     X.X X X...    X   X X X
X X.XXX XXX X X.X XXXXXXXXX XXX X X X
X X...X X   X X.X   X X   X X X     X
X XXX.XXX XXXXX.XXX X X XXXXX X XXXXX
X  ...  X   X  ...X X     X   X X   X
XXX.X XXXXX XXXXX.XXX XXX X XXX X XXX
X X.X X X X X X...  X X   X X...X X X
X X.XXX X X X X.XXXXXXXXX X X.X.X X X
X...X   X   X  ..............X.....X
X.X X X XXX XXX XXXXXXX XXX XXX XXX.X
XGX X X       X   X       X   X X  SX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```



```
XXXXXXXXXXXXXXXXXXXX
X         S          X
X                    X
X                    X
XX  X          X     X
XXXXXXX          X   X
X  G  X   XX         X
XXXXX                X
X                    X
XXXXXXXXXXXXXXXXXXXX
```



NO SOLUTION

file output

```
try
{
  PrintWriter output = new PrintWriter(
    new FileWriter("example.txt"));

  output.print("G");
  output.print("X");
  output.println();

  ouput.close();
}
```

FILE WILL CONTAIN "GX" AND A NEWLINE

reading numbers

```
int height, width;

String[] dimensions =
  input.readLine().split(" ");

try
{
  height = Integer.parseInt(dimensions[0]);
  width = Integer.parseInt(dimensions[1]);
}
```

weighted graphs

-sometimes it makes sense to associate a cost with traversing an edge

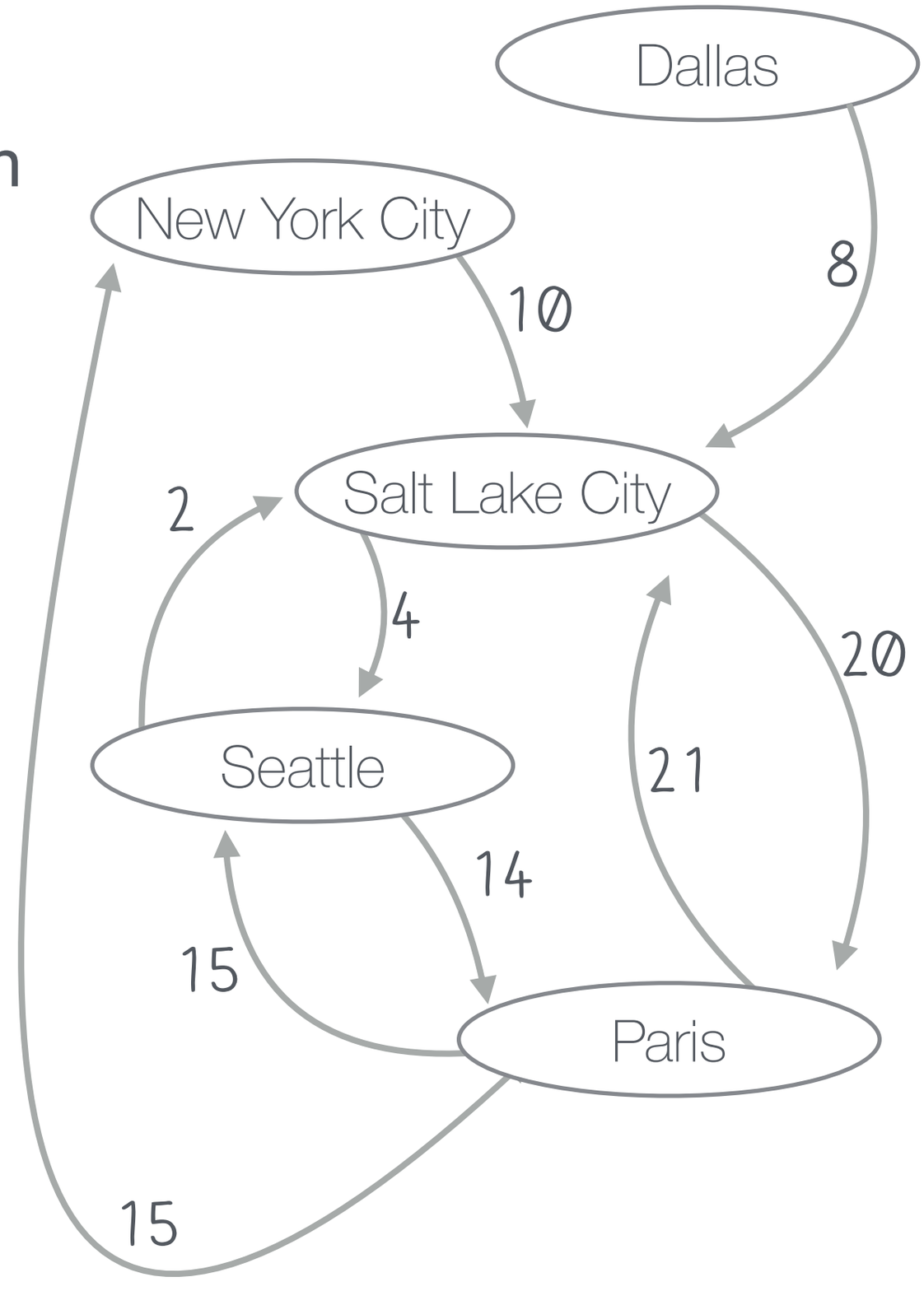
-we can add **weight** to each edge

-this is just a number!

-a higher weight indicates a more costly step

-**weighted path length** is the sum of all edge weights on a path

-this is ***NOT*** the same as path length!

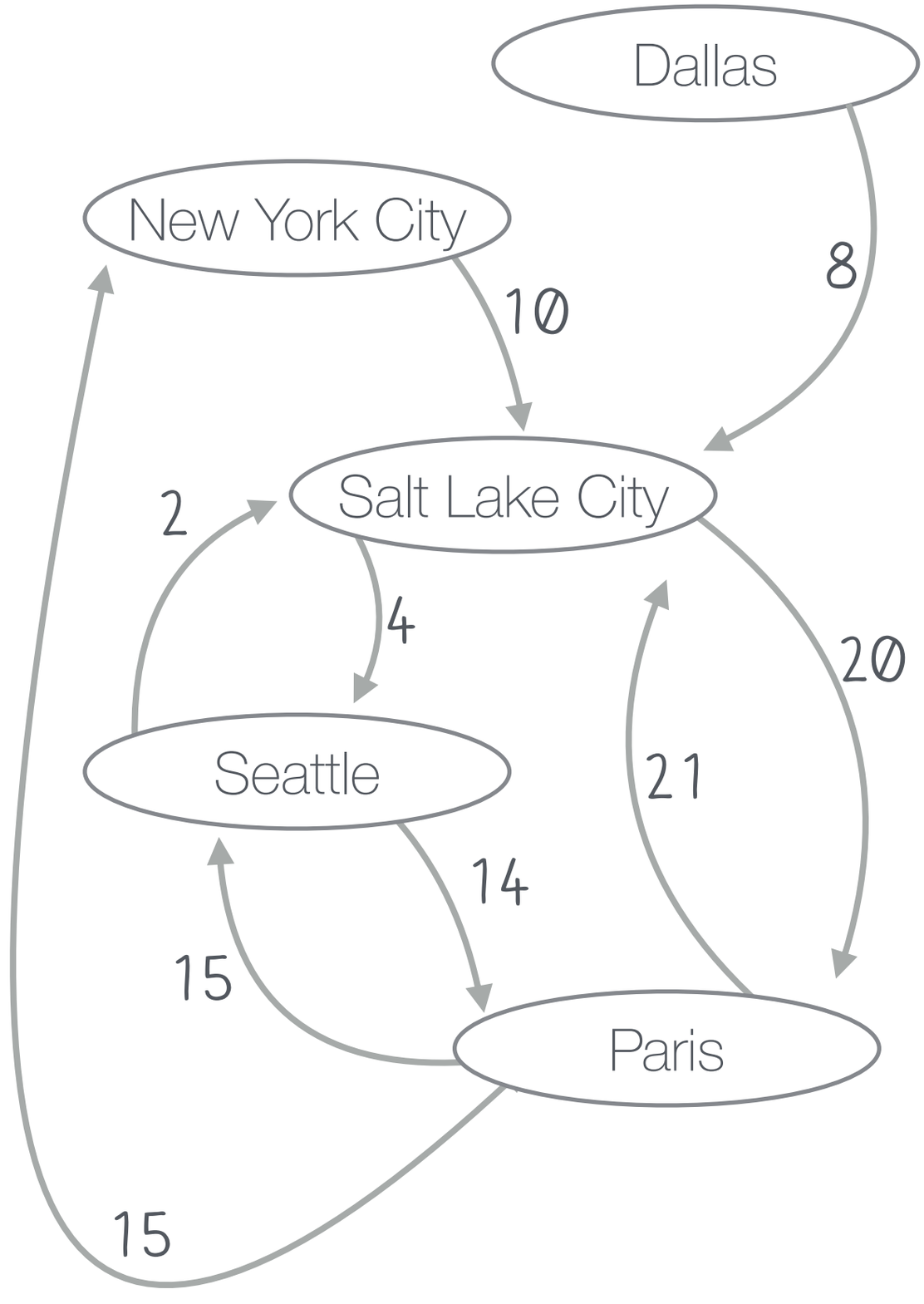


-what is the ***shortest*** path from SLC to Paris?

-what is the ***cheapest*** path from SLC to Paris?

-cheapest is not always the shortest!

-will regular BFS find the cheapest path?



```
class Node{
  E data;
  List<Edge> neighbors;
}

class Edge{
  Node otherEnd;
  double weight;
}
```

-make a new `Edge` class, which contains the reference and the weight

-instead of nodes having direct references to neighbors

dijkstra's algorithm

-Dijkstra's algorithm finds the *cheapest* path

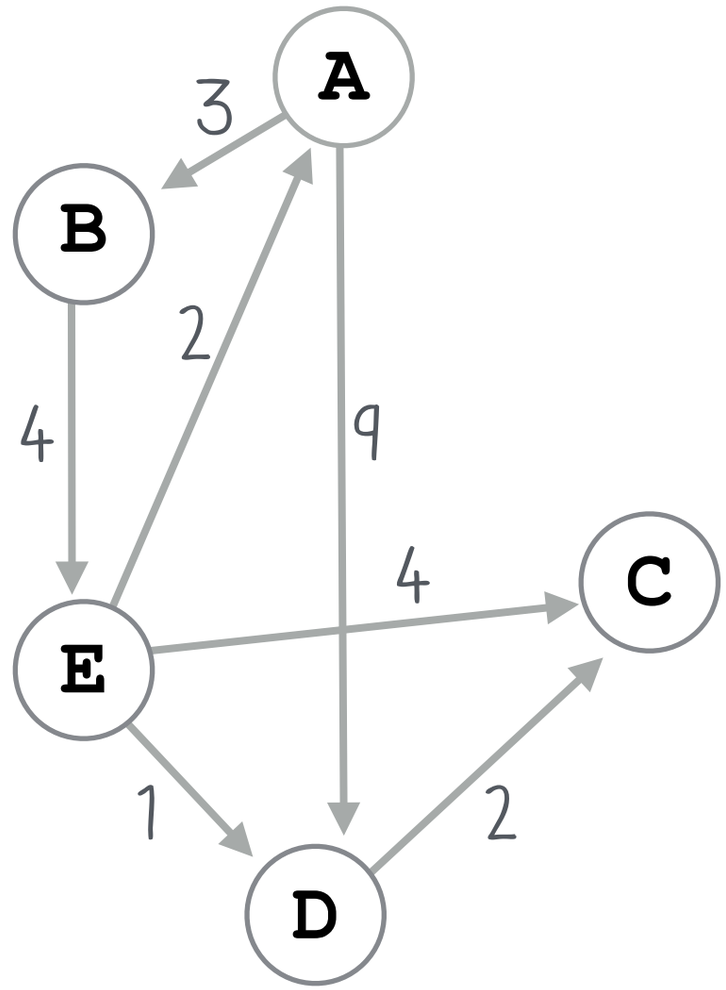
-keep track of the total path cost from start node to the current node

-cost of path to next node is total cost so far plus weight of edge to next node

-instead of traversing nodes in the order they were encountered, traverse in order of cheapest total cost first

WE WANT TO FIND A PATH FROM **A** TO **C**

THIS TIME WE USE A PRIORITY QUEUE.
MARK NODES **AFTER** REMOVAL FROM
THE QUEUE.



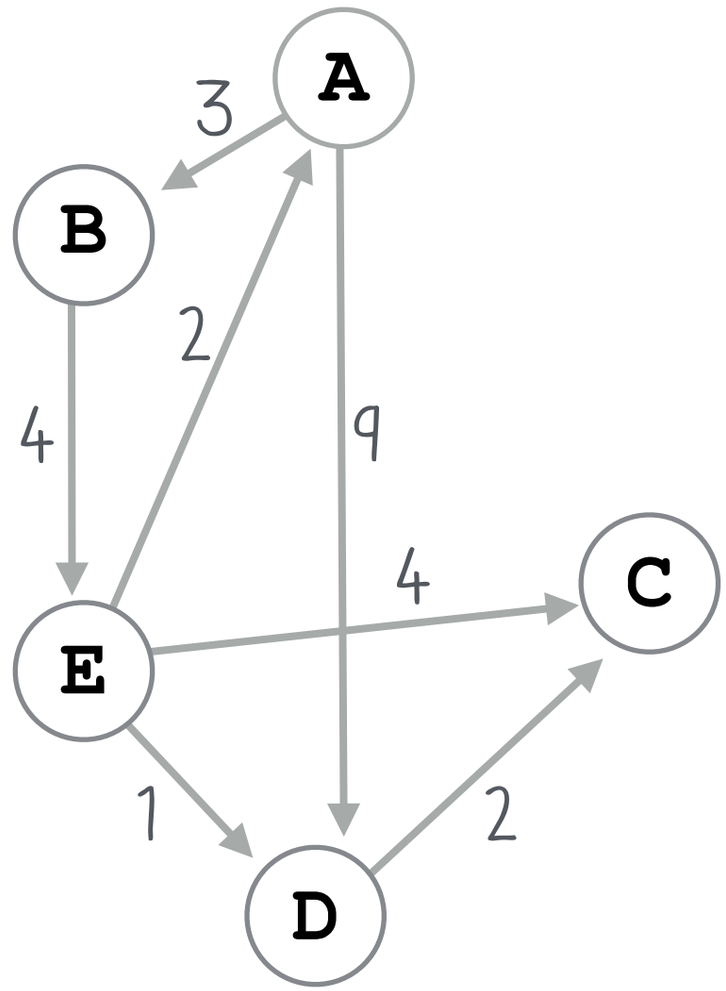
priority queue:

VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

```
A.costSoFar = 0
```



priority queue: | **A(0)** | | |

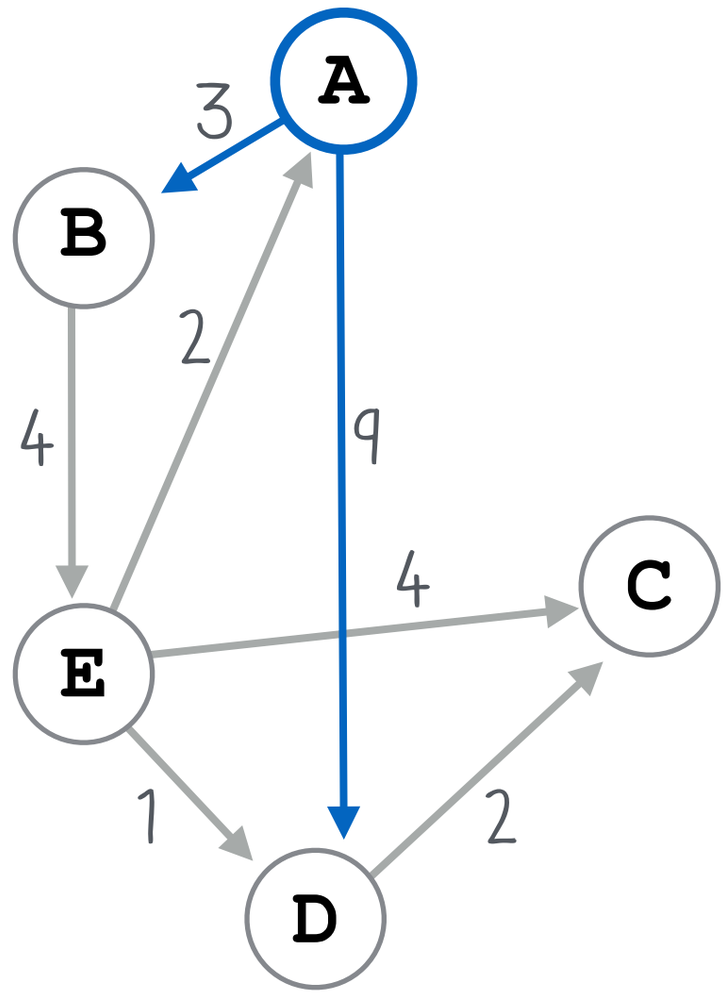
VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

DEQUEUE **A(0)**, AND ENQUEUE **A**'S NEIGHBORS WITH **A**'S COST-SO-FAR PLUS THE EDGE WEIGHT

```
B.costSoFar = A.costSoFar + 3
D.costSoFar = A.costSoFar + 9
B.cameFrom = A
D.cameFrom = A
```



priority queue: | **B(3)** | **D(9)** |

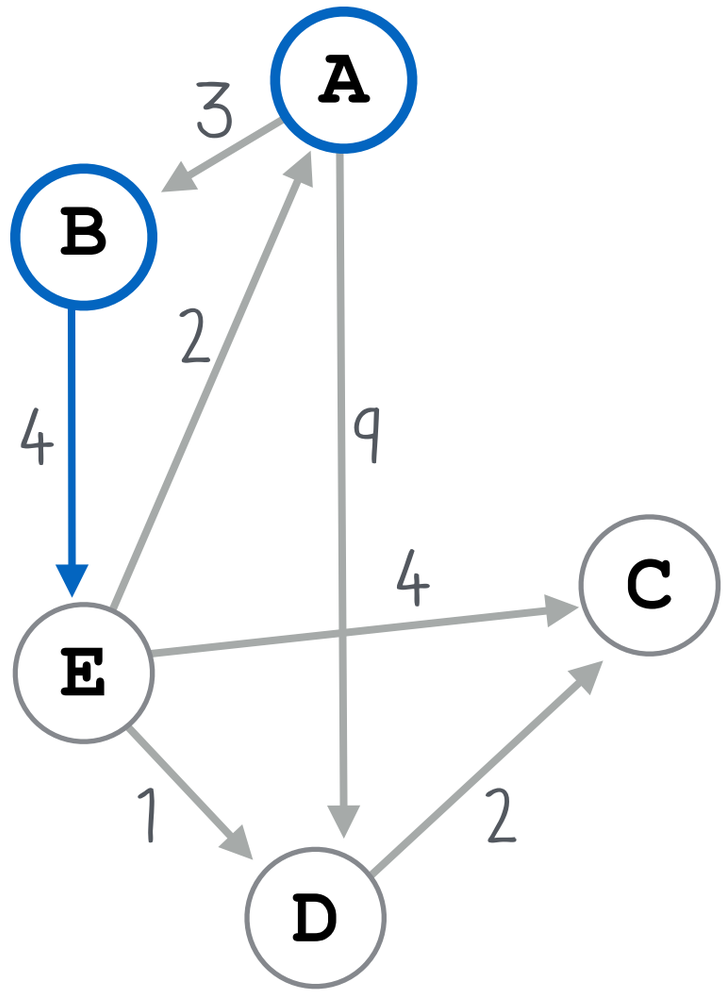
VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

DEQUEUE **B(3)**, AND ENQUEUE **B**'S NEIGHBORS WITH **B**'S COST-SO-FAR PLUS THE EDGE WEIGHT

```
E.costSoFar = B.costSoFar + 4
E.cameFrom = B
```



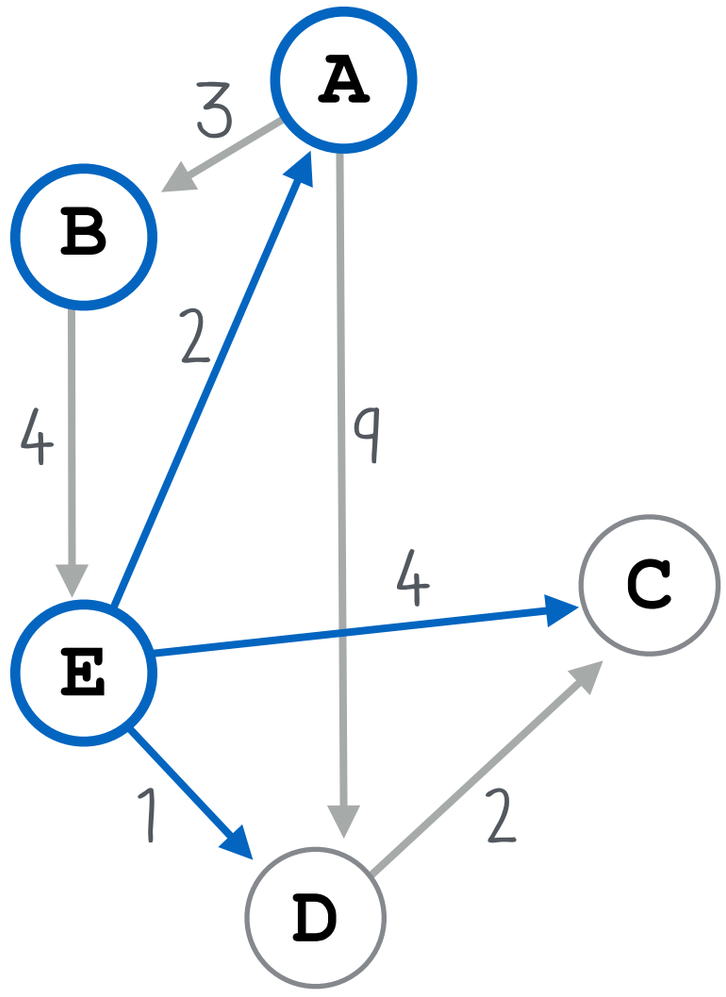
priority queue: | **E(7)** | **D(9)** | |

VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

DEQUEUE **E (7)** , AND ENQUEUE **E**'S NEIGHBORS WITH **E**'S COST-SO-FAR PLUS THE EDGE WEIGHT




```
// A visited, so skip
// shorter path to D found!
C.costSoFar = E.costSoFar + 4
D.costSoFar = E.costSoFar + 1
D.cameFrom = E
```

priority queue: | D(8) | C(11) | |

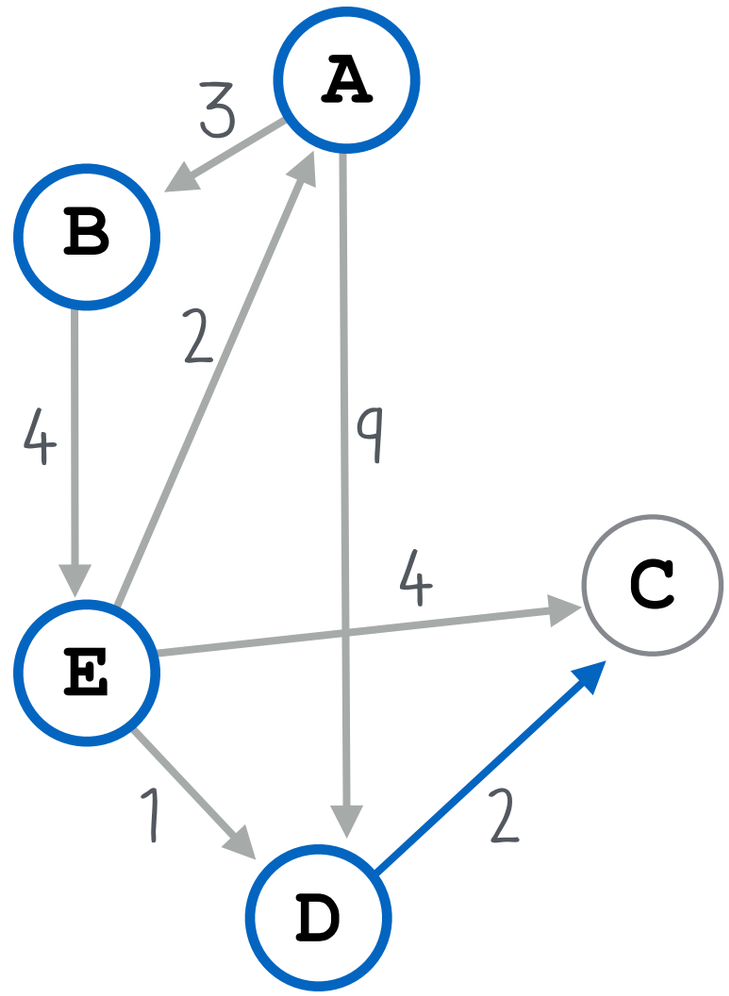
VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

DEQUEUE **D (8)**, AND ENQUEUE **D**'S NEIGHBORS WITH **D**'S COST-SO-FAR PLUS THE EDGE WEIGHT

```
// shorter path to C found!
C.costSoFar = D.costSoFar + 2
C.cameFrom = D
```



priority queue: | C(10) | | |

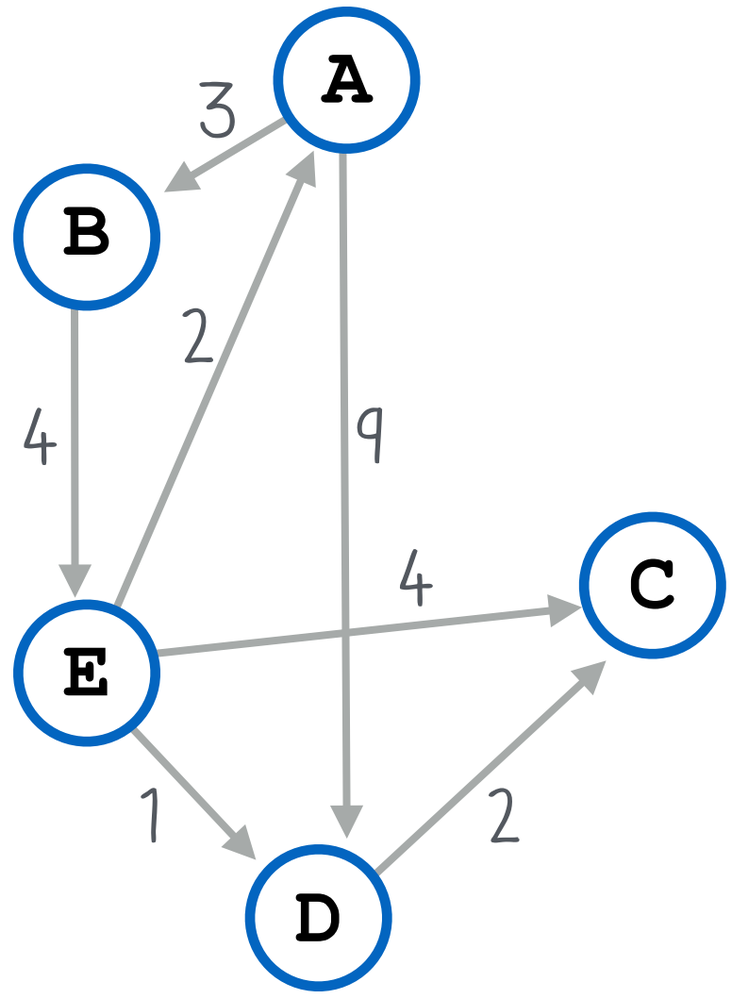
VISITED

UNVISITED

WE WANT TO FIND A PATH FROM **A** TO **C**

DEQUEUE **C(10)**. WE FOUND OUR GOAL! FINAL COST IS 10. RECONSTRUCT PATH.

A — B — E — D — C



priority queue:

VISITED

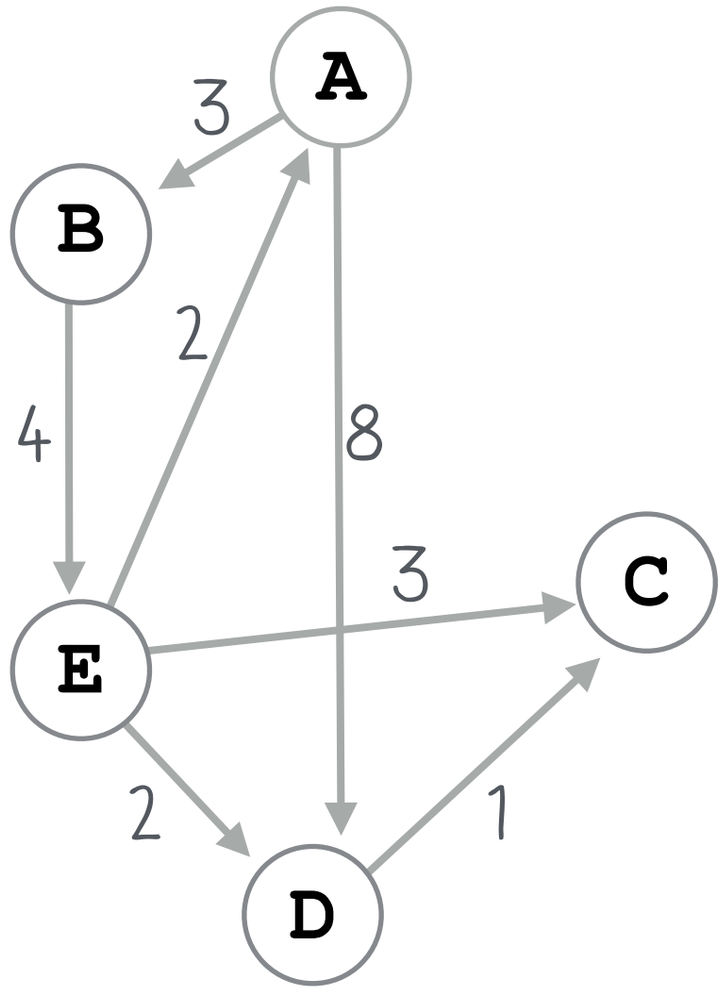
UNVISITED

```
Dijkstra(Node start, Node goal)
{
  initialize all nodes' cost to infinity

  PQ.enqueue(start)
  while(!PQ.empty())
  {
    curr = PQ.dequeue()
    if(curr == goal) {return} \\done!
    curr.visited = true
    foreach unvisited neighbor n of curr:
    {
      if(n.cost > curr.cost + edgeweight
      {
        PQ.enqueue(n) || update n's position in PQ
        n.cameFrom = curr
        n.cost = curr.cost + edgeweight
      }
    }
  }
}
```

WHAT PATH WILL DIJKSTRA'S FIND FROM **A** TO **C**?

A) **A B E C**
B) **A D C**
C) **A B E D C**



midterm review

-midterm 2 will cover all material since the beginning of the semester

since the last midterm…

-linked structures
  -memory allocation (`new`)
  -arrays vs linked structures
  -random access of arrays vs linked structures

-linked lists
  -implementation
  -performance of various operations

since the last midterm…

-stacks
  -implementation
    -array & linked list versions
  -performance

-queues
  -implementation
    -array & linked list versions
  -performance

since the last midterm…

-trees
    -implementation
    -performance
    -traversals (DFT)
        -pre-order, in-order, post-order

-binary trees

-BSTs
    -importance of a balanced BST
    -what causes balance / unbalance
    -insertion, deletion, search

since the last midterm…

-graphs
  -use of graphs
  -DFS
  -BFS
  -Dijkstra's algorithm

-hash tables
  -we will cover these on Tuesday after spring break
  -questions on HT will not be as involved

example questions

-draw a BST that is the result of adding the following items in the order given: Q F E G Y D W

  -what is the result of removing ___?

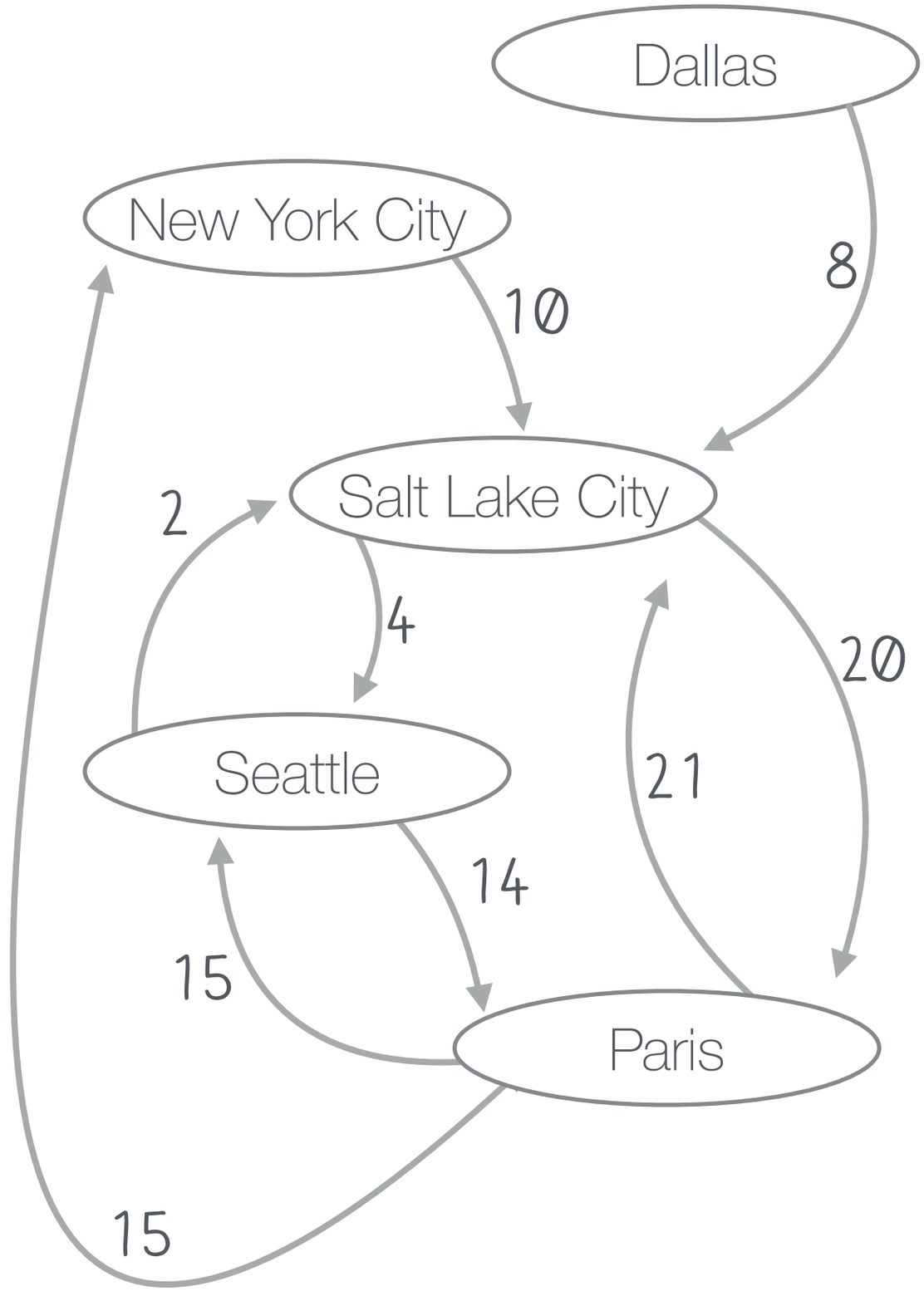
  -…or a similar question for a linked list, stack, queue

-you may be asked to modify a diagram, or draw one from scratch, then be asked questions about the diagram

  -could be of any data structure

-what order will items be used with ___ traversal order on this tree?

-consider a graph (like this one)
  -is it a tree?
  -is it acyclic?
  -is it weighted
  -what path would BFS find from SLC to NYC?
  -what path would Dijkstra's find from SLC to Paris?
-what order are nodes removed from the queue with Dijkstra's? BFS?
-explain a situation in which DFS might visit fewer nodes than BFS when search for a path



- given the following BST node definition, write a method that will find ___
  - ie. maybe the “successor of the node”

```
public class BSTNode<E>
{
  E data;
  BFSNode<E> left;
  BFSNode<E> right;

  public BSTNode<E> successor(BFSNode<E> n)
  {
    // fill in
  }
}
```

-suppose you are writing a program that does ___.

what data structure(s) would you use and why?

-you must defend your structure of choice!

I will not be giving out specific sample problems this time!

You should come up with your own sample problems based on the examples given here.

The lab after spring break will be a test review. Have your sample problems and questions ready for the TAs.

next time…

-reading
    -chapter 20 in book

-homework
    -assignment 8 due tonight
    -assignment 9 due Monday, March 30th