L22: Graphs, DFS

Alex Steiger

CompSci 201: Spring 2024

4/3/2024

Person in CS: Katherine Johnson

- 1918-2020
- NACA -> NASA mathematician
- “Computer” for many historic space missions
  - Glenn’s orbit around Earth, ‘62
  - Apollo 11/lunar landing, ‘69
- Presidential Medal of Freedom ‘15
- Subject of *Hidden Figures*
  - Non-fiction book and film

Logistics, coming up

- Today, Wednesday, April 3
  - APT 8 due
- Next Monday, April 8
  - Project P5: Huffman due
- Next Wednesday, April 10
  - APT Quiz 2 due (will release on Saturday)
  - Covers linked list, sorting, trees
  - No regular APTs this week, just the quiz

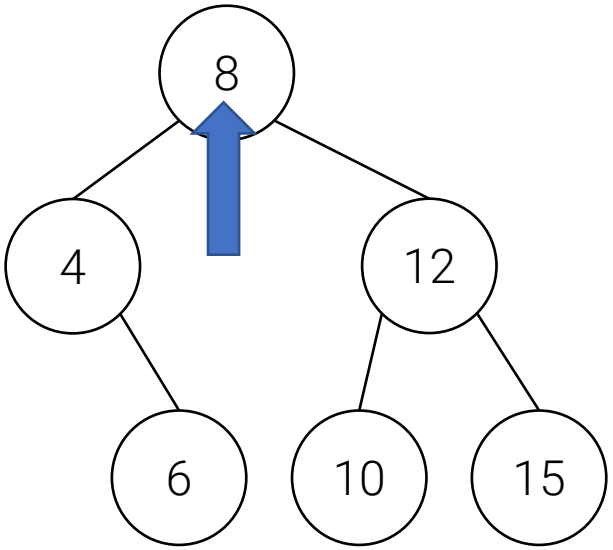
Binary Search Tree Performance

Code at: coursework.cs.duke.edu/cs-201-spring-24/live-coding

Recursive **search**, pictures, pseudocode

```
boolean search(int x, TreeNode t) {
```

- If `t == null`: Return `false`
- If `x == t.info`: Return `true`
- If `x < t.info`: search left
- Else: search right



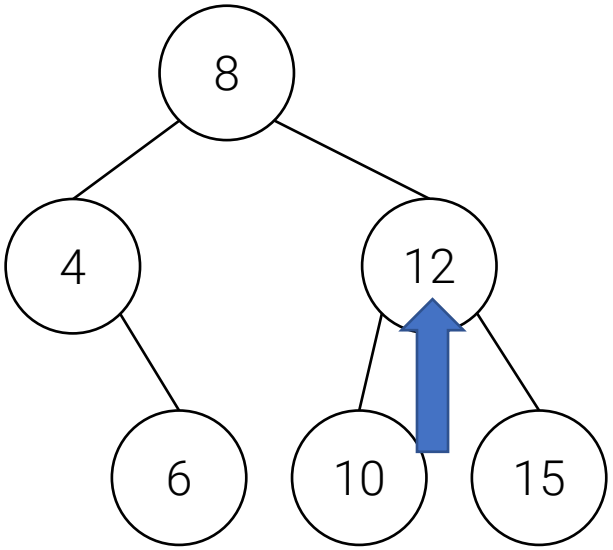
Searching for 10

10 > 8, so search right

Recursive **search**, pictures, pseudocode

```
boolean search(int x, TreeNode t) {
```

- If `t == null`: Return `false`
- If `x == t.info`: Return `true`
- If `x < t.info`: search left
- Else: search right

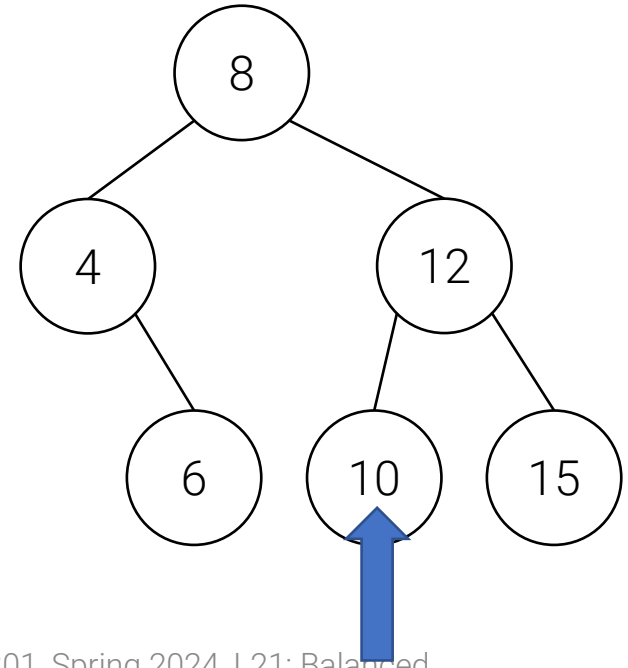


10 < 12 so search left

Recursive **search**, pictures, pseudocode

```
boolean search(int x, TreeNode t) {
```

- If `t == null`: Return `false`
- If `x == t.info`: Return `true`
- If `x < t.info`: search left
- Else: search right



10 == 10 so return true

Recursive **search** code

```
29    private boolean search(int x, TreeNode t) {
30        if (t == null) {
31            return false;
32        }
33        if (t.info == x) {
34            return true;
35        }
36        if (x < t.info) {
37            return search(x, t.left);
38        }
39        return search(x, t.right);
40    }
```

Base case: no more tree to search, did not find x

Base case: found x

If x is less, search in left subtree

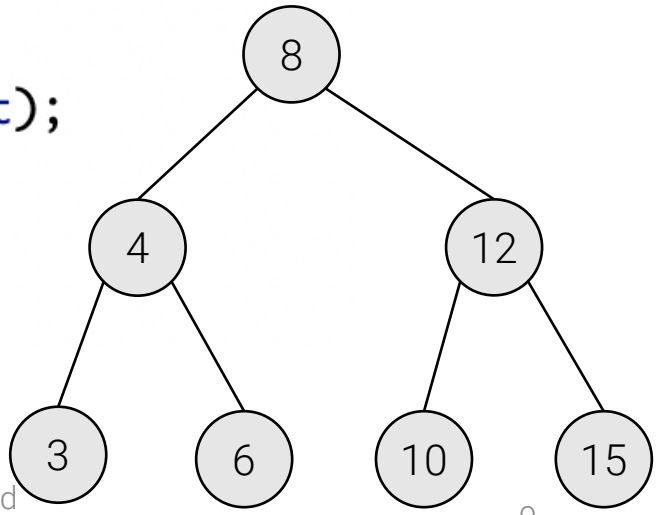
Else search in right subtree

Runtime complexity of BST add/contains on balanced tree

```
29      private boolean search(int x, TreeNode t) {
30          if (t == null) {
31              return false;
32          }
33          if (t.info == x) {
34              return true;
35          }
36          if (x < t.info) {
37              return search(x, t.left);
38          }
39          return search(x, t.right);
40      }
```

Completely balanced tree:

- T(N) = T(N/2) + O(1)
- Solution is **O(log(N))**, same as binary search

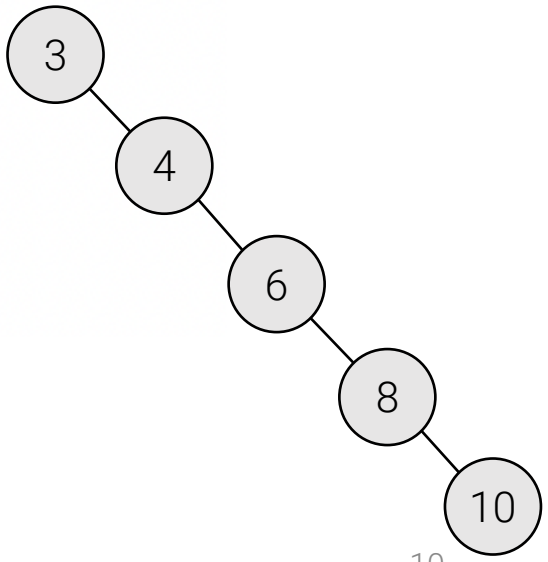


Runtime performance of BST on perfectly **un**balanced tree

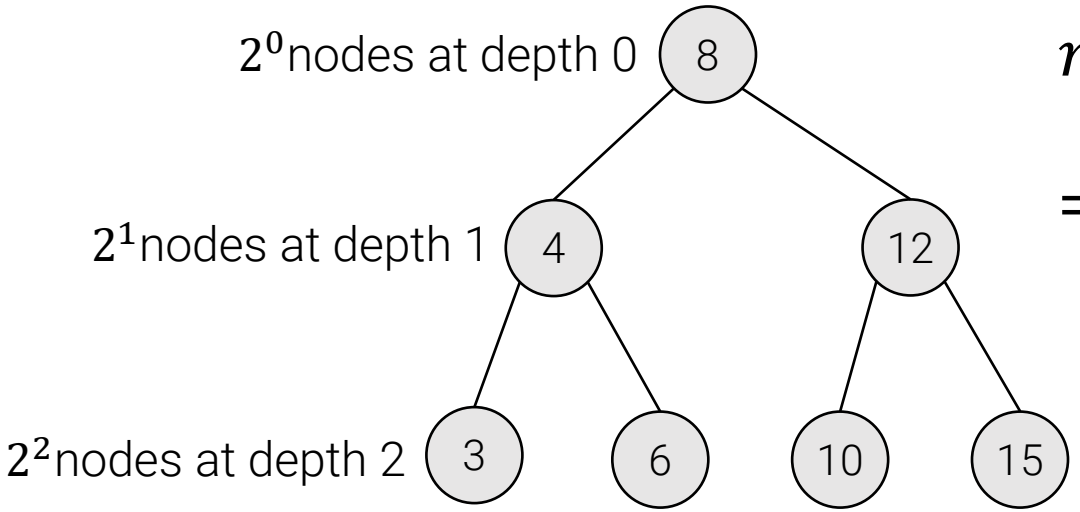
```
29      private boolean search(int x, TreeNode t) {
30          if (t == null) {
31              return false;
32          }
33          if (t.info == x) {
34              return true;
35          }
36          if (x < t.info) {
37              return search(x, t.left);
38          }
39          return search(x, t.right);
40      }
```

Perfectly unbalanced tree:

- T(N) = T(N-1) + O(1)
- Solution is **O(N)**, search in linked list



Another perspective: Balanced BST has height $\mathbf{O(\log(n))}$



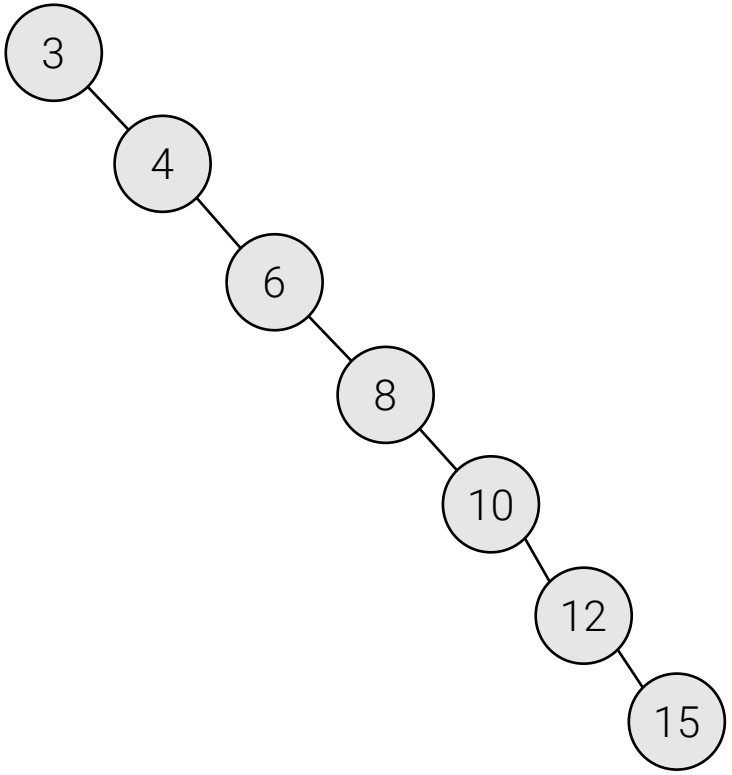
$$n = 2^0 + 2^1 + \cdots + 2^h$$
$$= 2^{h+1} - 1$$
$$\Rightarrow h = \log_2(n+1) - 1$$

Geometric series formula:

$$\sum_{i=0}^{k-1} ar^i = \frac{a(1-r^k)}{1-r}$$

Results in $O(\log n)$ height in worst-case ⇒ $O(\log n)$ time for add/contains/remove

Another perspective: Unbalanced BST has $O(n)$ height



For example, results from:

```
Sort(values)
For each e in values:
  insert(e)
```

Results in $O(n)$ height in worst-case ⇒
$O(n)$ time for add/contains/remove

Experiment: How much difference does it make empirically to do 100,000 random searches?

Timings in milliseconds
See example code in coursework.cs.duke.edu/cs-201-spring-24/live-coding

N	sorted order DIY binary search tree	random order DIY binary search tree	sorted order java.util.TreeSet
1,000	370	4	8
2,000	715	5	11
4,000	1422	5	14
8,000	2905	8	13
16,000	5991	7	12
32,000	Runtime exception	10	13
64,000	…	8	14
…	…	…	…
1,000,000	…	15	24

Average Case: Random Binary Search Tree has $O(\log(n))$ *expected* height

- Given $x_1, \ldots, x_n$ unique keys
- Let $\sigma(x_1, \ldots, x_n)$ be a uniform random permutation

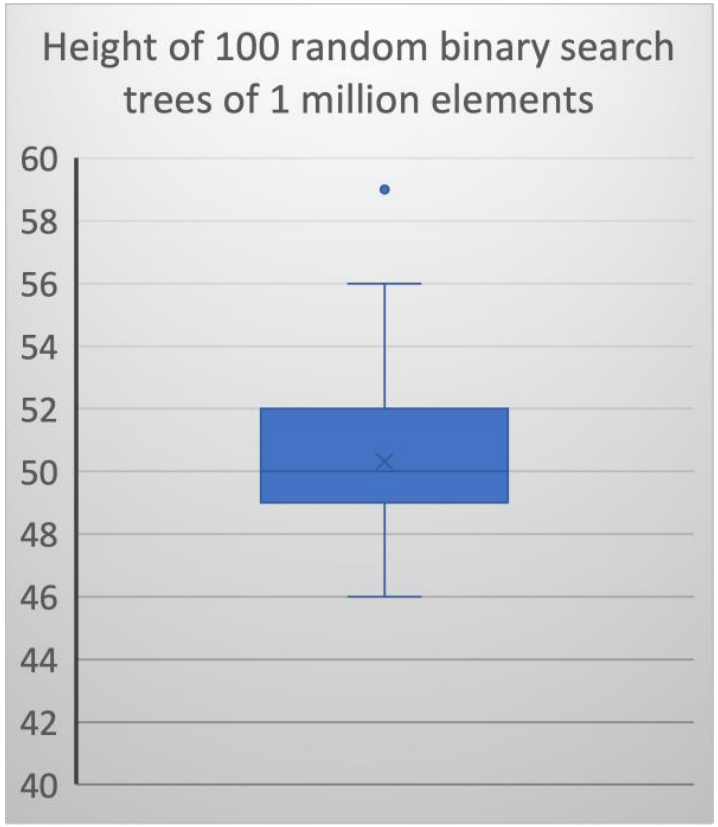
Theorem 12.4 in *Introduction to Algorithms* (restated):
Expected height of n-node BST inserted in order of σ:

$$\log_2\left(\frac{n^3 + 6n^2 + 11n + 6}{24}\right) \rightarrow O(\log(n))$$

Stronger statements about random binary search trees

- At most $h_n \rightarrow_{n \rightarrow \infty} 4.3 \log_2(n)$ with high probability
  - ***Luc Devroye. 1986. A note on the height of binary search trees. J. ACM 33, 3 (July 1986), 489–498. https://doi.org/10.1145/5925.5930***

- Empirical performance. Note that for $n = 1$ million:
  - $2\log_2(n) \approx 40$
  - $3\log_2(n) \approx 60$

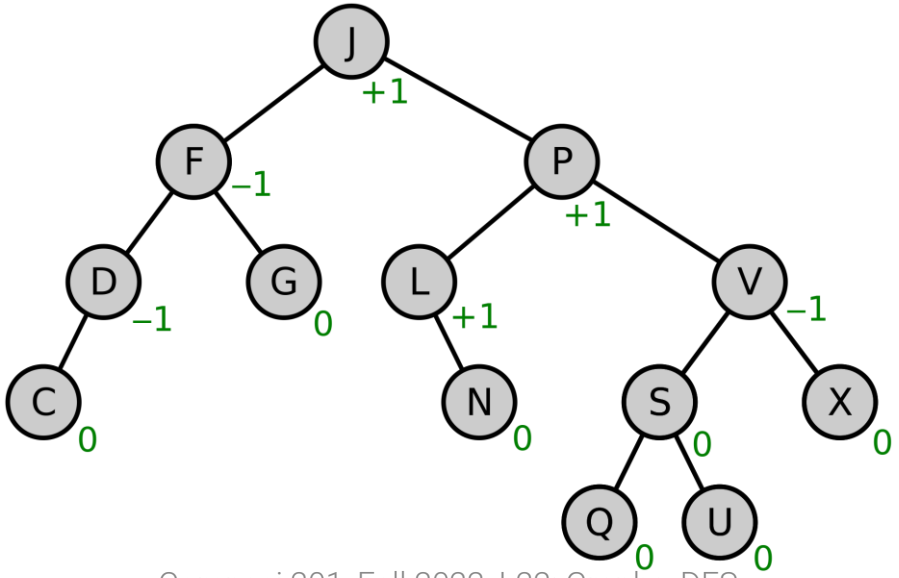


Balanced / Self-balancing BSTs

- AVL trees (zyBook Ch. 22)
- Red-black trees (zyBook Ch. 22, optional)
- 2-3 trees
- B-trees
- Treaps
- …many others, still a current research topic!
  - Imagine an incoming ***stream*** of insert/delete/search ops.
  - How should the tree be implemented ***now*** to minimize total time spent on the ***future*** (unknown) operations?

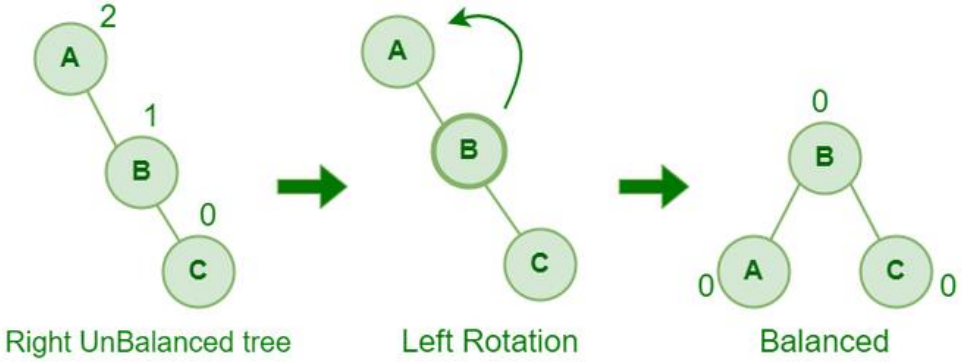
A Glimpse at AVL Trees

- How to “balance” a tree?
  - Minimize ***height*** of the tree (~all subtrees)
    - height: length of longest node-to-leaf path
- Measure ***balance factor (BF)*** of each node
  - Difference of right subtree height and left subtree height
- AVL trees ensure BF is -1,0,1 for all nodes

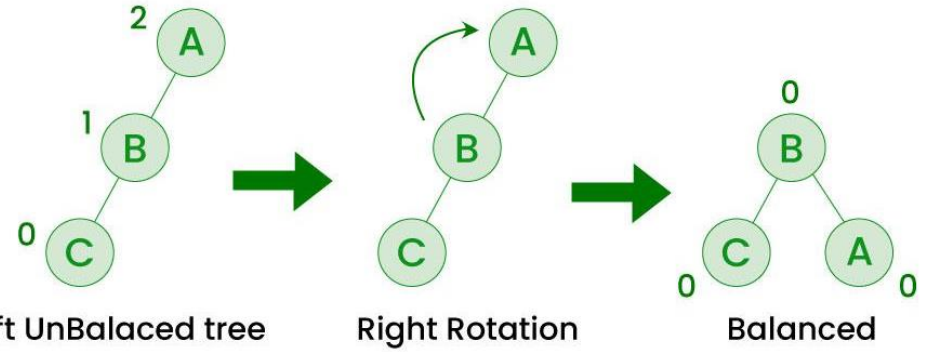


A Glimpse at Balancing

- How to improve balance factor? *Tree rotations!*
  - (Note: In the figures below, the ***absolute values*** of balance factors are shown.)
- *Left Rotation*



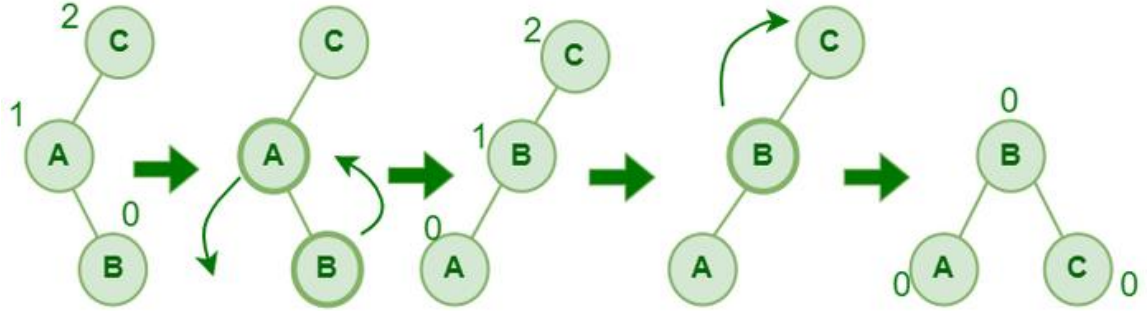
- *Right Rotation*



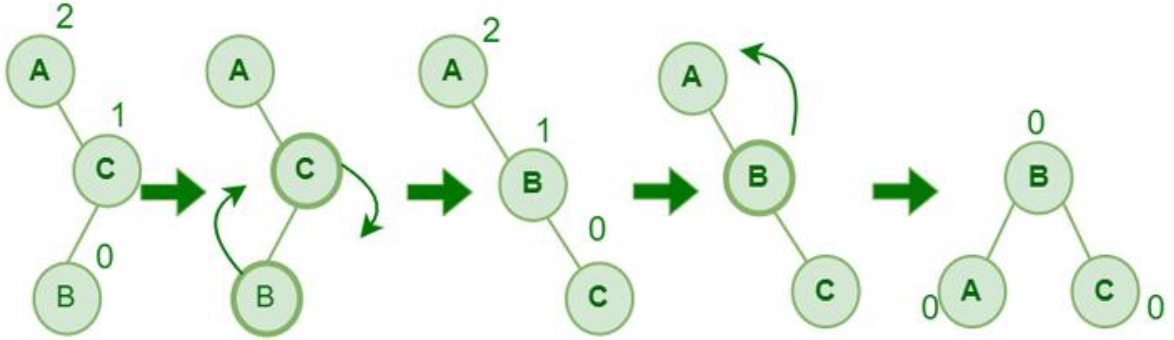
Another Glimpse

- How to improve balance factor? *Tree rotations!*
  - (Note: In the figures below, the ***absolute values*** of balance factors are shown.)

- *Left-Right Rotation*



- *Right-Left Rotation*



Asymptotic Runtimes

- Rotations are used to guarantee O(log N) height

  ⇒ O(log N) insertions, deletions, search

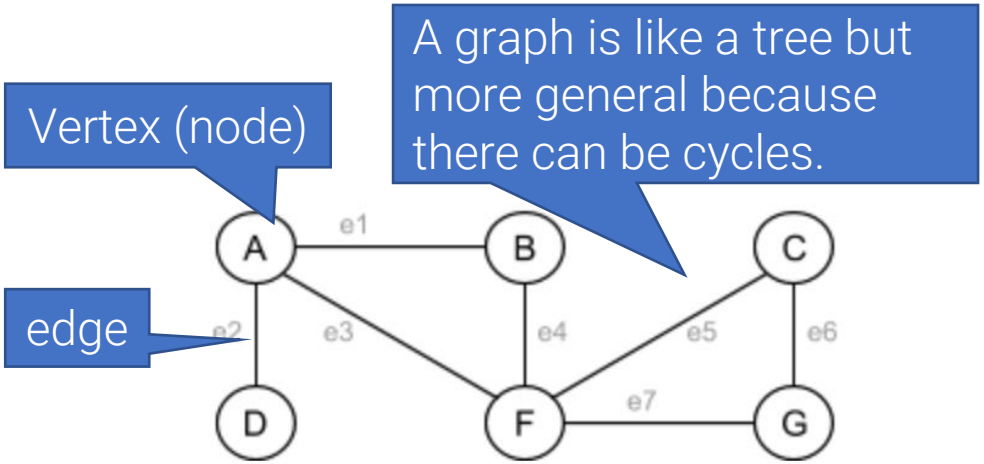
- java.util.TreeMap/TreeSet use red-black trees
  - Same runtime as above

Graphs

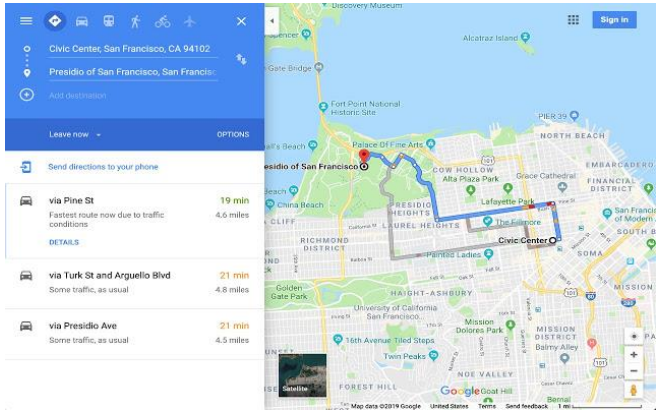
What is a graph?

A ***graph*** is a data structure for representing connections among items, and consists of ***vertices*** connected by ***edges***:

- A ***vertex*** (or node) represents an item in the graph.
- An ***edge*** represents a connection between two vertices in a graph.



Zybook chapter 23



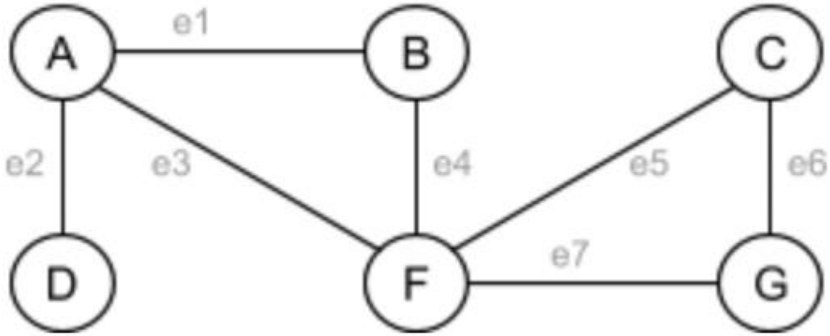
Maps/directions software:

- Vertices ~= intersections
- Edges ~= roads

Undirected versus directed graphs

Undirected Graph

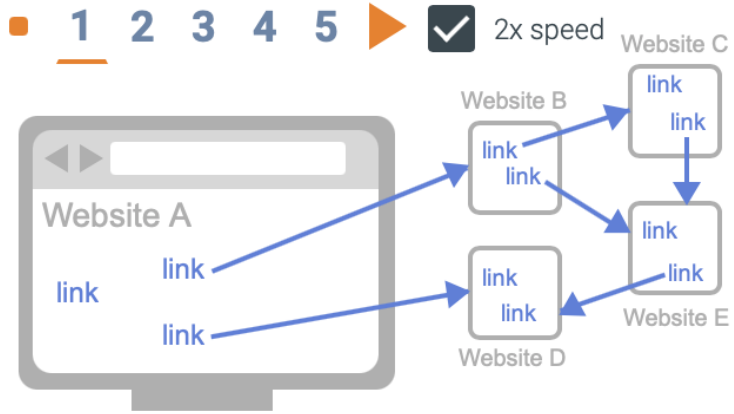
Edges go both ways



Facebook network, most road networks, are undirected

Directed Graph

Edges go one way only

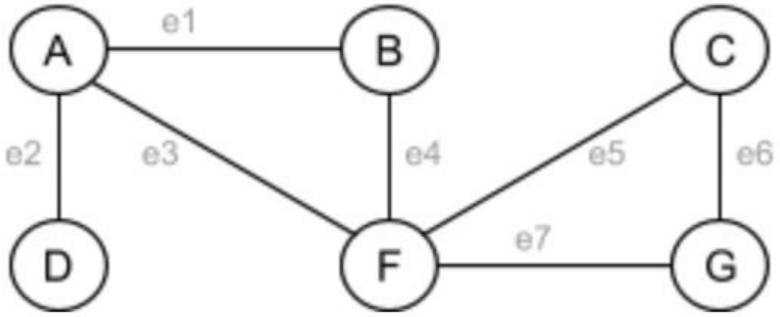


Worldwide web is a directed graph of webpages (nodes) and links (directed edges)

Zybook chapter 23

Simple Graphs and Graph Sizes

- In a ***simple*** graph, there is at most one (undirected) edge between nodes (or 2 directed).

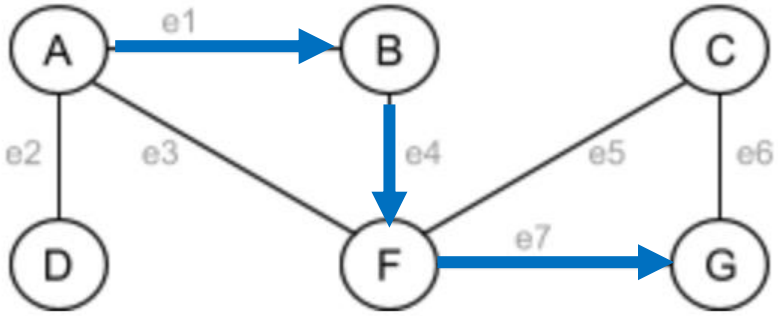


- Usually parameterize the size of the graph as:
  - N (or |V|) = number of vertices/nodes
  - M (or |E|) = number of edges
    - M <= N^2 for a **simple** graph

Zybook chapter 23

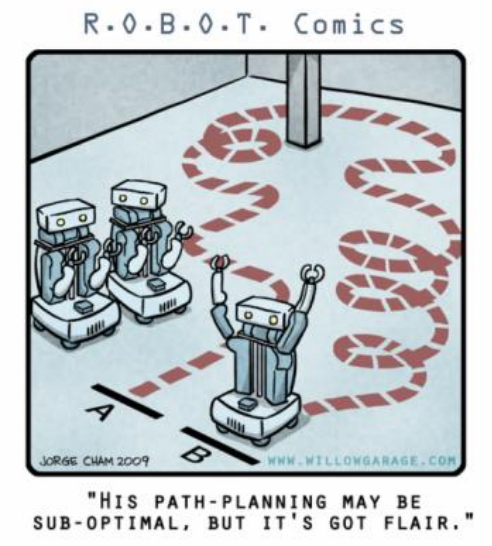
Paths in Graphs

- A simple **path** is a sequence of unique vertices where subsequent nodes are connected by edges
  - (Also commonly defined as a sequence of edges with unique vertices)



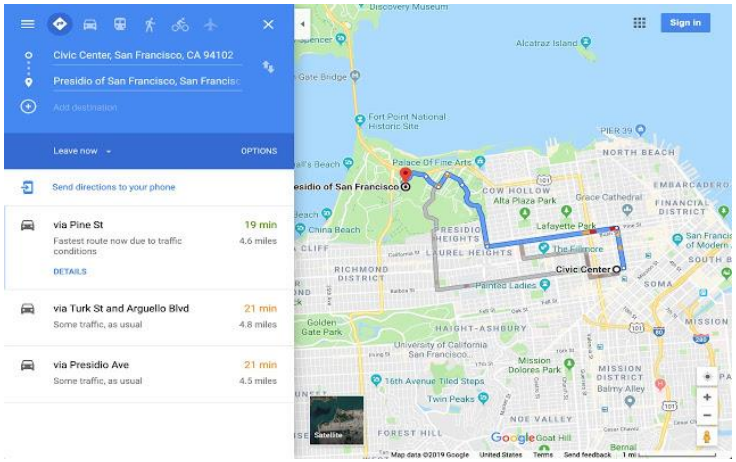
- Example in bold blue: [A, B, F, G].
  - (or [e1, e4, e7])

Pathfinding or Graph Search



Is there a way to get from point A to point B?

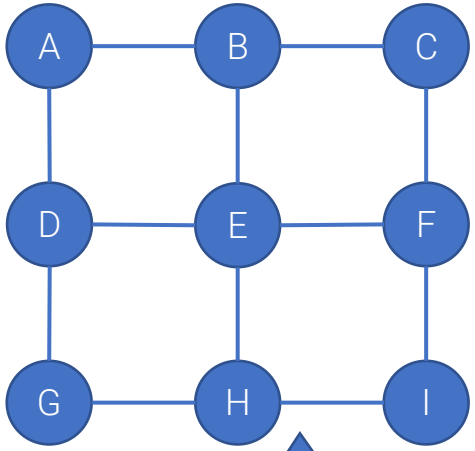
- Maps/directions
- Video games
- Robot motion planning
- Etc.



Recursive Depth-first search (DFS) in Grid Graphs

Two-dimensional grid is simple graph with implicit structure

	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I



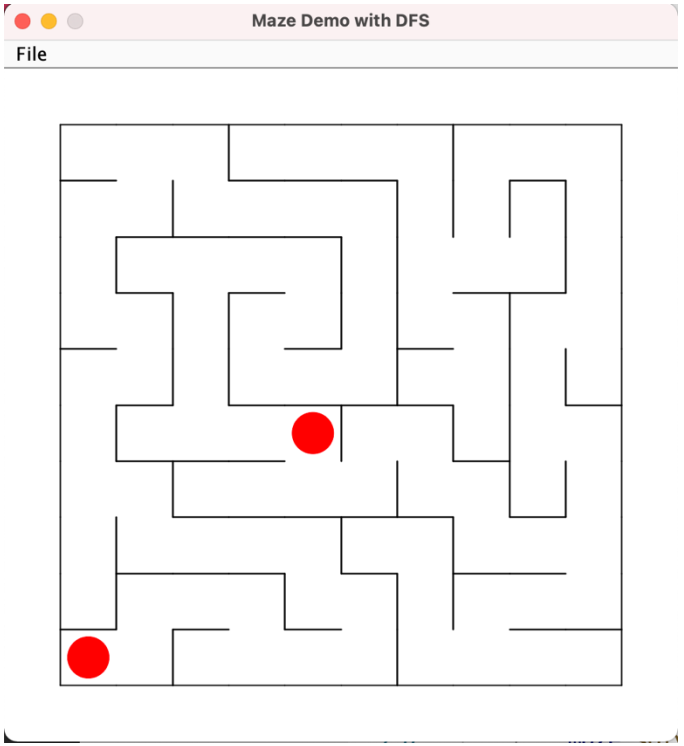
Represent as 2d array, e.g., char[ ][ ]
Two nodes adjacent if indices +/- 1

If not all of these edges are present, can represent a maze.

A maze is a grid graph

```
17  public class MazeDemo {
18      private int mySize;                  // dimension of maze
19      private boolean[][] north;           // is there a wall to north of cell i, j
20      private boolean[][] east;
21      private boolean[][] south;
22      private boolean[][] west;
```

- Example: 10 x 10 grid
- Edge = no wall, no edge = wall.
- Look for a path from start (lower left) to middle.

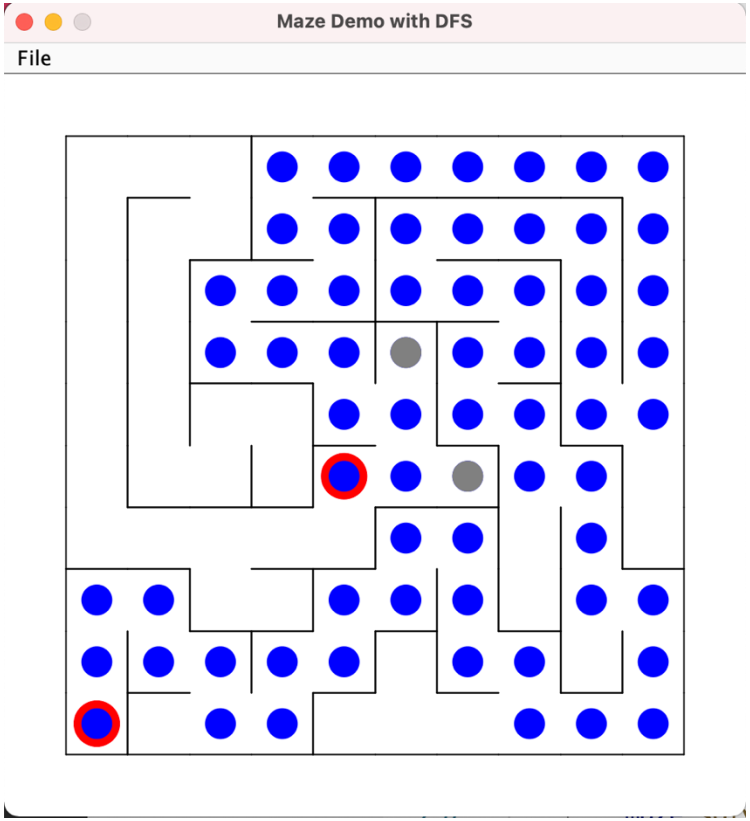


Depth First Search for Solving Maze

Always explore (recurse on) a new (unvisited) adjacent vertex if possible.

If nothing new (unvisited) vertex to explore:

- *backtrack* to the most recent vertex adjacent to an unvisited vertex, and then continue.
- if no such vertex, maze is unsolvable.



How is DFS Graph Traversal like Recursive Tree Traversal?

Tree traversals assumed only two adjacent nodes (children) and no cycles

```
49    public void inOrder(TreeNode root) {
50        if (root != null) {
51            inOrder(root.left);
52            System.out.println(root.info);
53            inOrder(root.right);
54        }
55    }
```

- Just try recursing on every adjacent vertex?
- Unlike in a tree, there are cycles: How do we avoid infinite recursion?

Base Cases and Visited Set

```
23      private boolean[][] visited;
```

Need to keep track to avoid infinite recursion

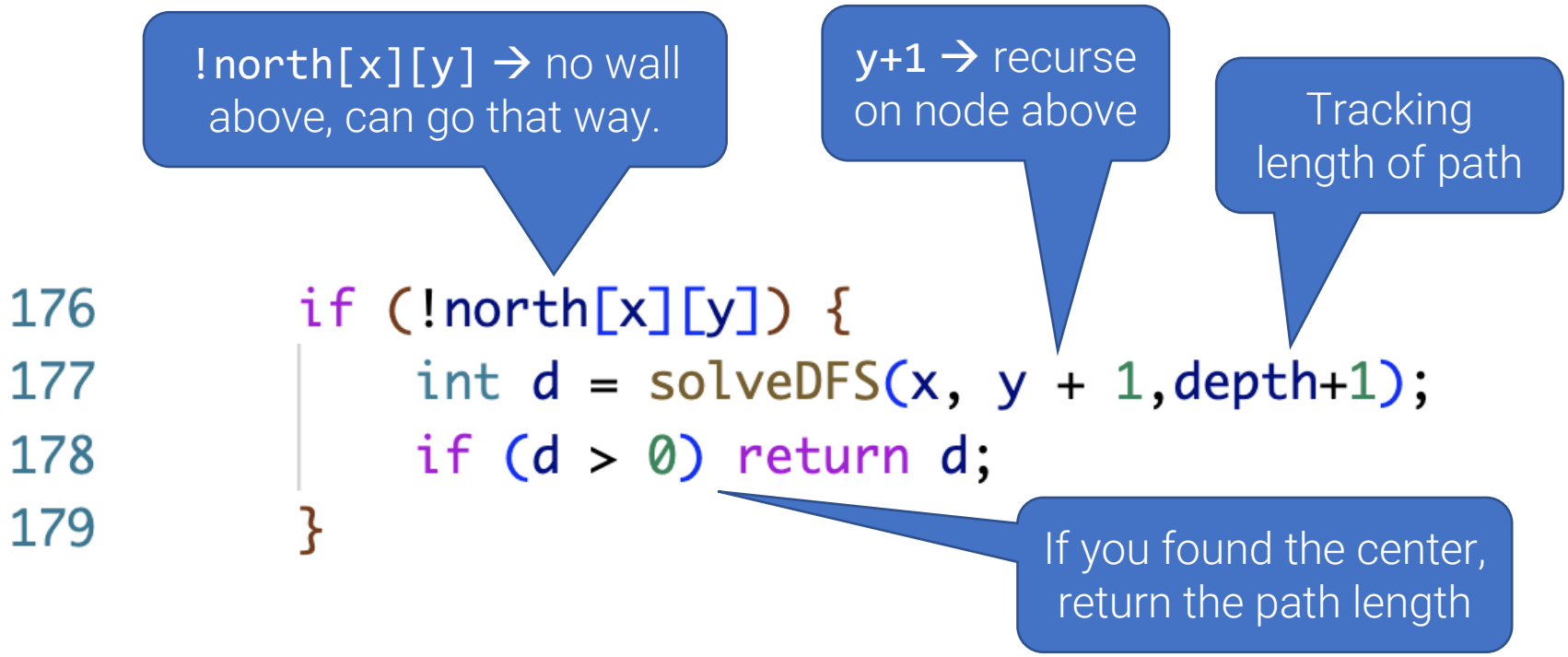
```
160   private int solveDFS(int x, int y, int depth) {
161       if (x == 0 || y == 0 || x == mySize +1 || y == mySize +1) return 0;
162       if (visited[x][y]) return 0;
163
164       visited[x][y] = true;
```

- Line 161: Base case: Searching off the grid
- Line 162: Base case: Already explored here

```
171       // reached middle which is goal of maze
172       if (x == mySize /2 && y == mySize /2) {
173           return depth;
174       }
```

- Line 172: Base case: Found the middle!

Recursive case



3 more symmetric cases for other 3 directions

Runtime complexity for Recursive DFS maze/grid

- Suppose the grid has N = width x height nodes.

- Each node will be recursed on <= 4 times:
  - Has 4 neighbors (adjacent vertices) that could recurse on it,
  - Keep track of visited so we don't recurse from the same neighbor twice (i.e., visit any vertex at most once)

- Each call takes O(1) non-recursive time.
- Overall runtime complexity: O(N)

L22-WOTO1-GridDFS-Sp24

Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

1

NetID *

solutions

2

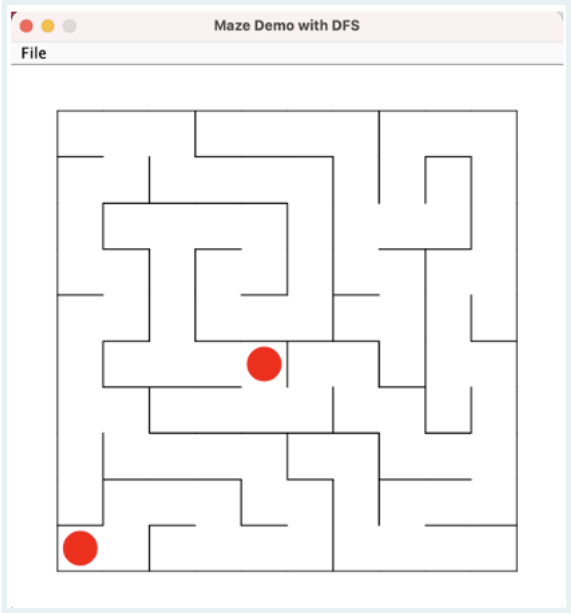
Suppose you are studying an ecosystem and want to understand energy flow by looking at predation between species (in which one organism, animal or plant, kills and consumes another). The graph that would allow you to study whether one species eventually (through some chain of predation) gains energy from another is... *

◯ Undirected, simple

- ◯ Undirected, not simple
- ◉ Directed, simple
- ◯ Directed, not simple

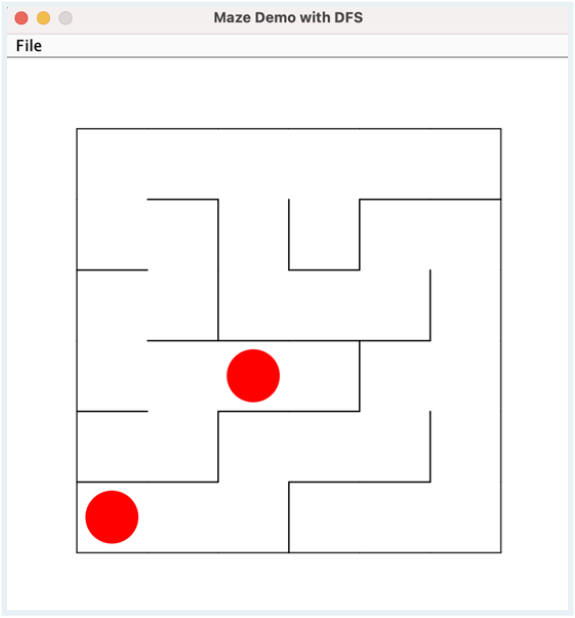
3

In a *grid graph* (such as could represent a maze, at right) with **N total nodes**, there are at most how many edges? *



- ◯ N
- ◉ 4N
- ◯ N^2
- ◯ 4N^2

4



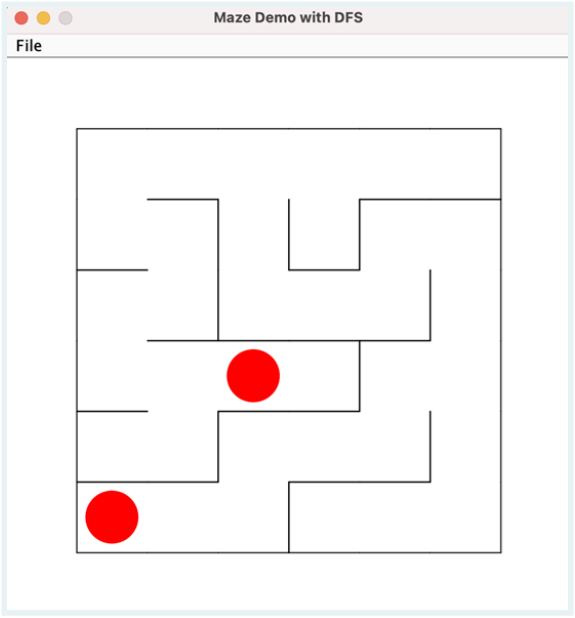
The order of recursive calls for our recurse DFS for the maze is:

1. Up
2. Right
3. Down
4. Left

Will the resulting DFS visualization searching for the middle node ever visit / color in the **bottom right** node? *

- ○ Yes
- ◉ No
- ○ Maybe, it depends

5



The order of recursive calls for our recurse DFS for the maze is:

1. Up
2. Right
3. Down
4. Left

Will the resulting DFS visualization searching for the middle node ever visit / color in the **upper right** corner node? *

- (●) Yes
- ( ) No
- ( ) Maybe, it depends