Graphs
Announcements()
Change in Schedule

• I'm changing the schedule around a bit...
  • Today: Graphs
  • Tuesday: Shortest Path Algorithms
  • Wednesday: Minimum Spanning Trees
  • Tuesday: Review Session for Midterm II
Final

- Final: Monday, August 12\textsuperscript{th}, 7-10pm
  - Location: Cubberly Auditorium
  - Cumulative (but weighted towards post midterm material)
  - Covers material up through Tuesday
    - SCPD students and students who require special arrangements should email me in the next couple days
- We do the final “early” so we have time to grade it, get it back to you and resolve any grading issues.
}  //Announcements
Data Structures Cheat Sheet

- **Vector, Stack**: dynamic array
- **Queue**: linked list or dynamic array
- **Set, Map**: Binary Search Tree
- **HashSet, HashMap**: Hash Table
- **Lexicon**: Trie
Graphs
A Social Network
PANFLUTE FLOWCHART

**Diagram:**
- **Diamond (Orange):** do you need one?
  - **Branch (YES):** no you don't
  - **Branch (NO):** no panflute
A graph is a mathematical structure for representing relationships.
A **graph** is a mathematical structure for representing relationships.
A **graph** is a mathematical structure for representing relationships.
A graph is a mathematical structure for representing relationships.

A graph consists of a set of nodes connected by edges.
A **graph** is a mathematical structure for representing relationships.

A graph consists of a set of **nodes** connected by **edges**.
A graph is a mathematical structure for representing relationships. A graph consists of a set of nodes connected by edges.
Some graphs are directed.
Some graphs are undirected.
Some graphs are **undirected**.

You can think of them as directed graphs with edges both ways.
Graphs

• “Yo Teach, why are we studying graphs?”

  • We study graphs because a lot of problems can be modeled in terms of graphs

  • Also, there are many off-the-shelf graph algorithms that we apply if we're able to formulate a problem as a graph problem.
Pathfinding

- Each intersection is a node
- Each street connecting intersections is an edge
- Find paths between intersections that minimize distance or travel time
Content-Aware Resizing
Content-Aware Resizing

- Each pixel is a node in a graph
- Each pixel is connected to adjacent pixels
- Find paths from the top of the image to the bottom that minimize the “energy function”
The Wikipedia Graph

- Wikipedia (and the web in general) is a graph!
- Each page is a node.
- There is an edge from one page to another if the first page links to the second.
Social Networks and Epidemics

How can we represent graphs in C++?
Representing Graphs

We can represent a graph as a map from nodes to the list of nodes each node is connected to.
What interesting things can we do with graphs?
Connected Components

- A **connected component** is a subset of the nodes in a graph such that:
  - For every pair of nodes in the subset there exists a path between them
  - No node in the subset is not connected any node not in the subset
Connected Components

1 Connected Component
Connected Components

2 Connected Components
Connected Components

3 Connected Components
Connected Components

6 Connected Components
Connected Components

• Detecting connected components in a graph is important because it can provide useful insights into the structure of graph
  • e.g. How do people in a community separate themselves into separate groups.
• In order to detect connected components we first need to be able to iterate over nodes in a graph.
• We'll come back to connected components later.
Iterating over a Graph

- Given a linked list, there was just one way to traverse the list.
  - Keep going forward.
- In a binary search tree, there are many traversal strategies:
  - An *inorder* traversal that produces all the elements in sorted order.
  - A *postorder* traversal used to delete all the nodes in the BST.
- There are *many* ways to iterate over a graph.
Iterating over a Graph

- All methods of iterating over a graph involve keeping track of 3 sets of nodes:
  - Set of Nodes already visited
  - Set of Nodes to look at next
  - Everything else
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph
Iterating over a Graph

• All methods of iterating over a graph involve keeping track of 3 sets of nodes:
  
  - Set of Nodes already visited
  - Set of Nodes to look at next
  - Everything else

• Methods of iterating over nodes differ in how they choose which node to look at next
Recursive Depth First Search

- To detect connected components we just want to see whether or not some path exists between them (we don't care about finding the “shortest” path).

- One way to detect connectivity would be to just pick an arbitrary direction and keep following it.
  - When you run out of room to go in one direction, just go back and look in a different direction
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search
Recursive Depth-First Search

• To do a depth-first search (DFS) from a node $u$, do the following:
  • If $u$ is already marked, stop.
  • Mark $u$.
  • For each neighbor $v$ of $u$:
    - Recursively run DFS from $v$.
• The backtracking here is similar to the backtracking done in standard recursion.
Iterative Depth-First Search

- DFS is most commonly implemented iteratively using a Stack
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search

A -- B
    
D -- E -- F
      
C

Set

Stack

A

E

B
Depth-first search
Depth-first search

A --- B --- C
  |     |     |
  D --- E --- F

Stack: A, B
Set: A, E

Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search

Stack:
- B
- D
- F

Set:
- A
- C
- E

Diagram:
- A
- B
- C
- D
- E
- F
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Depth-first search
Coding Depth-First Search
DFS and Connected Components

- Detecting connected components becomes relatively straightforward once we have Depth First Search.
- Not going to code it up, but I encourage you to!
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding shortest paths.
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding shortest paths.
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding shortest paths.

A

B

Stack
Problems with DFS

- Useful when trying to explore everything.
- Not good at finding shortest paths.
Breadth-First Search
Breadth-First Search

• Specialization of the general search algorithm where nodes to visit are put into a *queue*.

• Explores nodes one hop away, then two hops away, etc.

• Finds path with fewest edges from start node to all other nodes.
Note: The following animation has been simplified for pedagogic purposes. In reality there would be a set keeping track of visited nodes and redundant adds to the Queue.
Breadth-first search
Breadth-first search

Queue
Breadth-first search
Breadth-first search

Queue
Breadth-first search
Breadth-first search

Queue: B, E

Graph:
- A connected to B
- B connected to E
- C connected to D
- E connected to F
Breadth-first search
Breadth-first search

Queue

E  C
Breadth-first search
Breadth-first search
Breadth-first search
Breadth-first search

Queue

C  D  F
Breadth-first search
Breadth-first search

Queue

D    F
Breadth-first search
Breadth-first search
Breadth-first search
Breadth-first search

Queue
Coding Breadth-First Search
CAT
SAT
RAN
MAN
MAT
CAN
RAT

Diagram shows a network of connections between the words CAT, SAT, RAT, MAN, AND RAN. Cat and sat are connected, as are man and ran. Cat is also connected to rat, and man to rat. Man is also connected to mat, and ran to mat. Cat, mat, and rat form a triangle, while sat and man form another triangle.
CAN → MAN → RAN → CAT → SAT → RAT → CAN
CAT, SAT, RAT, CAN, MAN, RAN, MAT
Next Time

• **Shortest Paths**
  • Dijkstra's Algorithm.
  • A* Search.