Graph Overview

- Graph Definitions
  - Directed graph (digraph)
  - Undirected graph
  - Directed acyclic graph (dag)
  - Paths & cycles
- Graph Properties
  - Graph coloring
  - Planarity
  - Bipartite graphs
- Graph Implementations
  - Adjacency matrix
  - Adjacency lists
- Graph Searching
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
- Graph Algorithms
  - Single-source shortest paths (Dijkstra’s Algorithm)
  - Minimum spanning tree (MST)
    - Prim’s Algorithm
    - Kruskal’s Algorithm

Depth-First Search

- Follow edges depth-first starting from an arbitrary vertex s, using a Stack to remember where you came from
- When you encounter a vertex previously visited, or there are no outgoing edges, retreat and try another path
- Eventually visit all vertices reachable from s
- If there are still unvisited vertices, repeat

Easy to see this takes O(m) time
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

Depth-First Search

Depth-First Search

Depth-First Search
Depth-First Search

**DFS Notes**

- Same as BFS, except we use a Stack instead of a Queue to determine which edge to explore next.
- Can also implement DFS recursively:
  - The Stack is represented implicitly via the Stack-Frames created by the recursive calls.

```rust
// Initially, vertices are unmarked
for all vertices v {
    if (v is marked) continue;
    recursiveDFS(v);
}

recursiveDFS(s) {
    Mark s:
    for (each v adj to s) {
        if (v is marked) continue;
        recursiveDFS(v);
    }
}
```
GUI Motivation

- Interacting with a program
  - Program Driven
    - Statements execute in sequential, predetermined order
    - Typically use keyboard or file I/O
  - Event Driven
    - Program waits for user input to activate certain statements
    - Typically use a GUI (Graphical User Interface)
- Design...Which to pick?
  - Program driven by another program?
  - Program used at command line?
  - Program interacts often with user?
  - Program used in window environment?
- How does Java do GUIs?

Java Foundation Classes

- Java Foundation Classes
  - Classes for building GUIs
    - Swing
    - Applet
    - Accessibility API
    - Java 2D API
    - Drag-and-drop Support
    - Internationalization
  - Our main focus: Swing
    - Building blocks of GUIs
    - windows & components
    - User interactions
    - Built upon something called the AWT (Abstract Window Toolkit)
  - What are the other things....?

Other Aspects of the JFC

- Pluggable look-and-feel Support
  - Controls look-and-feel for particular windowing environment
  - E.g., Windows, Motif
- Accessibility API
  - Supports assistive technologies such as screen readers and Braille
- Java 2D
  - Drawing
    - Includes rectangles, lines, circles, images, ....
  - Drag-and-drop:
    - Support for drag and drop between Java application and a native application
  - Internationalization
    - Support for other languages

Brief Example

```java
import java.awt.
import java.awt.event.
public class Intro extends JFrame {
    private int count;
    private JButton b = new JButton("Push Me!");
    private JLabel label = new JLabel(generateLabel());
    public static void main(String[] args) {
        JFrame f = new Intro();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200,100);
        f.setVisible(true);
    }
    public Intro() {
       .setLayout(new FlowLayout(FlowLayout.LEFT));
        add(b);
        add(label);
        b.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                count++;
                label.setText(generateLabel());
            }
        });
    }
    private String generateLabel() {
        return "Count: " + count;
    }
}
```

GUI Statics vs. GUI Dynamics

- Statics: what’s drawn on the screen
  - Components
    - E.g., buttons, labels, lists, sliders
  - Containers: components that contain other components
    - E.g., frames, panels, dialog boxes
  - Layout managers: control placement and sizing of components
- Dynamics: user interactions
  - Events
    - E.g., button-press, mouse-click, key-press
  - Listeners: an object that responds to an event
  - Helper classes
    - E.g., Graphics, Color, Font, FontMetrics, Dimension

Overview for Statics

- Determine which components you want
- Choose a top-level container in which to put the components
- Choose a layout manager to determine how components are arranged
- Place the components
AWT vs. Swing

- **AWT**
  - Initial GUI toolkit for Java
  - Provided a "Java" look and feel
  - Basic API: java.awt.*

- **Swing**
  - More recent (Java 1.2) GUI toolkit
  - Added functionality (new components)
  - Supports look and feel for various platforms (Windows, Motif, Mac)
  - Basic API: javax.swing.*

- Did Swing replace AWT?
  - Not quite: both use the AWT event model

Components

- Components = what you see
  - Visual part of an interface
  - Represents something with position and size
  - Can be painted on screen and receive events
  - Buttons, labels, lists, sliders, etc.

- Examples (see next slide)

Components Examples

```java
import javax.swing.*;
import java.awt.*;

public class ComponentExamples extends JFrame {
    public static void main(String[] args) {
        ComponentExamples f = new ComponentExamples();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }

    public ComponentExamples() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        add(new JButton("Button"));
        add(new JLabel("Label"));
        add(new JComboBox(new String[] { "A", "B", "C" } ));
        add(new JCheckBox("JCheckBox"));
        add(new JSlider(0,100));
        add(new JColorChooser());
    }
}
```

Containers

- A container is a component that
  - Can hold other components and
  - Has a layout manager

- Heavyweight vs. lightweight
  - A heavyweight component interacts directly with the host system
  - AWindow, JFrame, and JDialog are heavyweight
  - Except for these top-level containers, Swing components are almost all lightweight
  - JPanel is lightweight

- There are three basic top-level containers
  - JWindow: top-level window with no border
  - JFrame: top-level window with border and (optional) menu bar
  - JDialog: used for dialog windows

- The other important container
  - JPanel: used mostly to organize objects within other containers

Creating a Window

```java
import java.awt.*;

public class Basic1 {
    public static void main(String[] args) {
        // Create window:
        JFrame f = new JFrame("Basic Test!");
        // Set 500x500 pixels^2:
        f.setSize(500,500);
        // Show the window:
        f.setVisible(true);
        // Quit Java after closing the window:
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Creating a Window using an Initialization Block

```java
import java.awt.*;

public class Basic2 {
    public static void main(String[] args) {
        // Create window:
        new B2GUI();
    }
}

class B2GUI {
    { 
        JFrame f = new JFrame("Basic Test2!");
        f.setSize(500,500);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```
Creating a Window using a Constructor

```java
import javax.swing.*
public class Basic3 extends JFrame {
    public static void main(String[] args) {
        new Basic3();
    }
    public Basic3() {
        // Title window:
        setTitle("Basic Test!");
        // Set 500x500 pixels^2:
        setSize(500, 500);
        // Show the window:
        setVisible(true);
        // Quit Java after closing the window:
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Layout Managers

- **A layout manager controls placement and sizing of components in a container**
  - If you do not specify a layout manager, the container will use a default:
    - JFrame default = BorderLayout
    - JPanel default = FlowLayout
- **Five common layout managers:** BorderLayout, BoxLayout, FlowLayout, GridLayout, GridBagLayout

Some Example Layout Managers

- **FlowLayout**
  - Components placed from left to right in order added
  - When a row is filled, a new row is started
  - Lines can be centered, left-justified or right-justified (see FlowLayout constructor)
  - See also BorderLayout
- **GridLayout**
  - Components are placed in grid pattern (think array)
  - Rows, columns defined by GridLayout constructor
  - Grid is filling left-to-right, then top-to-bottom
- **BorderLayout**
  - Divides window into 5 areas: North, South, East, West, Center
  - Adding components:
    - FlowLayout and GridLayout use container.add(component)
    - BorderLayout uses container.add(component, index)
    - Where index is one of:
      - BorderLayout.North
      - BorderLayout.South
      - BorderLayout.East
      - BorderLayout.West
      - BorderLayout.Center

More Layout Managers

- **CardLayout**
  - Tabbed index card look from Windows
- **GridBagLayout**
  - Most versatile, but complicated
- **Custom**
  - Can define your own layout manager
  - Best to try Java's layout managers first...
- **Null**
  - Implies no layout manager
  - Programmer must specify absolute locations
  - Provides great control, but can be dangerous to application because of platform dependency

FlowLayout Example

```java
import javax.swing.*;
import java.awt.*;
public class Statics1 {
    public static void main(String[] args) {
        new S1GUI();
    }
}
class S1GUI {
    private JFrame f;
    private Container c;
    public S1GUI() {
        f = new JFrame("Statics1");
        f.setSize(500, 500);
        f.setLayout(new FlowLayout(FlowLayout.LEFT));
        for (int b = 1; b < 9; b++)
            f.add(new JButton("Button "+b));
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Code Examples

- **Basic1.java**
  - Create a window
- **Basic2.java**
  - Create a window using an initialization block
- **Basic3.java**
  - Create a window using a constructor
- **Calculator.java**
  - Shows use of JOptionPane to produce standard dialogs
- **ComponentExamples.java**
  - Sample components
- **Intro.java**
  - Button & counter
- **Statics1.java**
  - FlowLayout example
- **Statics2.java**
  - GridLayout example