# Course Introduction

1. This lecture will provide all the tools you need to cover 80% of all graph problems.
2. Visualizing things is the key to victory for your data structures and algorithms, especially in solving graph problems.
3. We will go through different animations to understand the algorithm deeply.
4. General Roadmap

* We will go over all the strategies and theories.
* Sketch out a nice meaningful picture.
* Talk about the complexity of the algorithm in the approach video.
* Followed every approach; we are also going to implement the code.

# Graph Basics

## What is a graph?

Graph=nodes + edges

Nodes🡪 Also known as vertexes. Some tiny circles with data (number, letter, etc.) inside of them.

Edges🡪 any connections between nodes

A graph can be described as relations between things.

## Directed Graph vs Undirected Graph

A picture containing chart

Description automatically generatedA picture containing text, wall

Description automatically generated

1. The directed graph has arrowheads along the edges (consider directions), while the undirected graph does not have arrowheads. (NOT consider directions)
2. In the directed graph, we can travel from node A to node C, but we cannot travel from node C to node A. But in the undirected graph, we can travel either from node A to node C or node C to node A.
3. **Neighbor Node** 🡪 Any node that is accessible through an edge. (Ex. In the directed graph, node B and node C are neighbour nodes of node A. **But node A is NOT a neighbour node of node C because it is NOT accessible**).
4. Convert the graph to an adjacency list (usually in a HashMap data structure)

A picture containing timeline

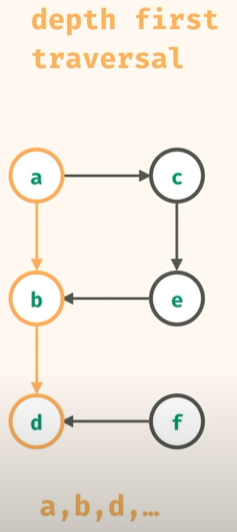
Description automatically generated

* The keys of this adjacency list are every node inside the graph. (A🡪F)
* The values of the adjacency list are going to arrays
* Even the node D is empty, it should still be a key inside the adjacent list. So we can know that node D exists.

# Depth First and Breadth First Traversal（深度优先 vs 广度优先）

## Basic

In the depth-first traversal, we start at node A. The algorithm will first follow the edges as A🡪B🡪D (D is the dead end). Then the algorithm follows the edges as A🡪C🡪E🡪B🡪D

 Timeline

Description automatically generated

In the breath first traversal, the algorithm will visit nodes in the sequence of A🡪B🡪C….

A picture containing chart

Description automatically generatedDiagram

Description automatically generated with medium confidence

## Pseudocode

1. The depth-first traversal use **stack**, and the breadth-first traversal use **queue**.
2. *Review the tutorial video* ***from 12:00 to 17:00*** *to visualize how the depth-first traversal and breadth-first traversal have been implemented.*
3. ```code implementation``` </basic\_dpt\_and\_bft>
4. DFT: use iterative or recursive method || BFT: mostly only use an iterative method
5. We will talk about Big O in the following sections.

# Has Path Problem

## Source

<http://www.structy.net/problems/has-path>

## Description

Text

Description automatically generated

Shape, rectangle

Description automatically generated

Shape, rectangle

Description automatically generated

Shape, rectangle

Description automatically generated

Shape, rectangle

Description automatically generated

Shape, rectangle

Description automatically generated

*Recall: this is an adjacency list of the directed acyclic graph. The acyclic means no circle.*

## Solution

We can either apply the depth-first or the breadth-first method.

```code implementation``` </hasPath>

## Complexity

### Method\_0

n=number of nodes

e=number of edges

Time: O(e)

Space: O(n)

### Mehod\_1

n=number of nodes

n^2=number of edges

The worst case is that each node is connected bi-directionally

A picture containing text, clock

Description automatically generated

Time: O(n^2)

Space: O(n)

# Undirected Path

## Description

Diagram

Description automatically generated

* Every pair represent connections of nodes.

For example, [i,j] represents the connections between node i and node j. We can travel from node i to node j, or from node j to node i.

* The edges graph can also transfer into the adjacent graph

Text

Description automatically generated A picture containing text, clock

Description automatically generated

* For the undirected graph, the very common question is that if the graph has a cycle, we will add one more edge between the node k and node j

A picture containing text, clock, watch

Description automatically generatedA picture containing text, clock, watch

Description automatically generated

* We will talk about if there are paths between the source node and the destination node
* **To prevent an infinite loop, we will mark all nodes that have been visited.**
* Learn more details from the 46:28

## Complexity

n=number of nodes

e=number of edges

Time: O(e)

Space: O(n)

## Undirected Path Problem

### Source:

<https://structy.net/problems/undirected-path>

### Details

* ``` Code Implementation``` </undirected\_path>
* We will convert our edge list to the adjacency list by building a helper function

### Key Takeaways

* Phase 1: convert the edge list to an adjacency list
* Phase 2: Perform traversal with the check of preventing infinite loop (in JS we add the Set() object)