Knapsack and Graph Optimization Problems

Chapter 14

What is an Optimization Problem?

- Find the –est something
  - Biggest
  - Smallest
  - Faster
  - Most-est
  - Fewest
- These questions can usually mapped to a classic optimization problem

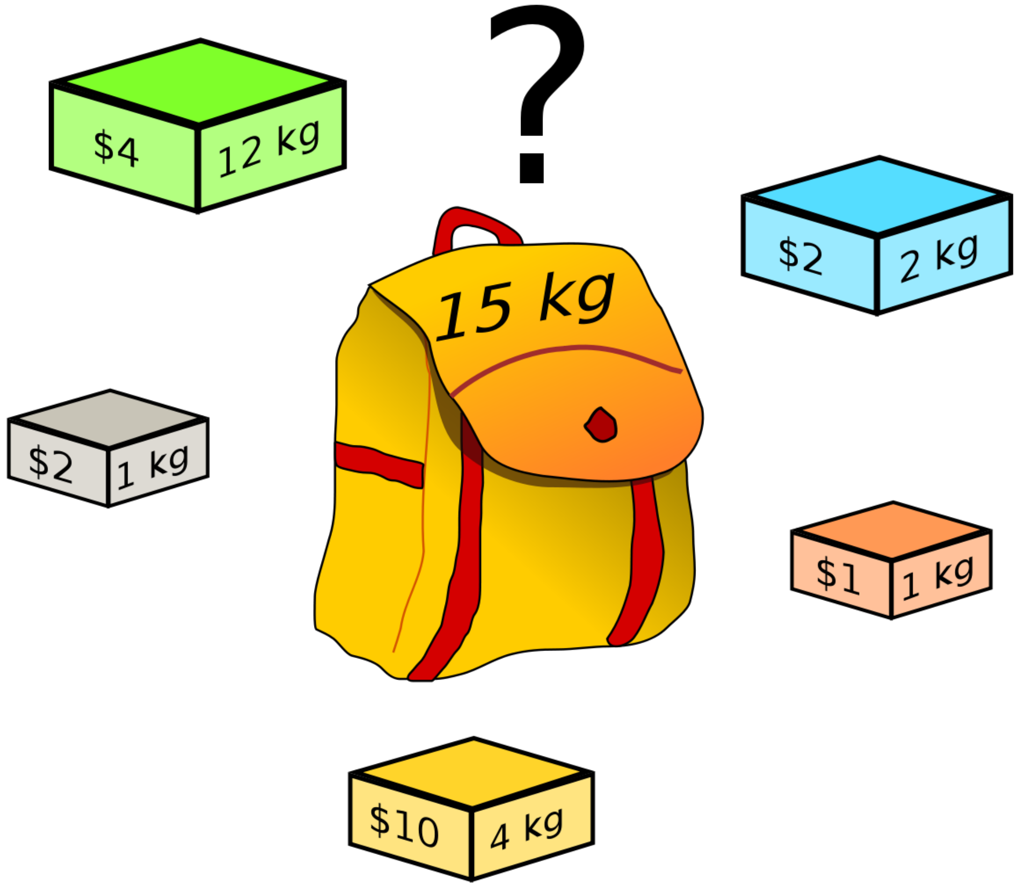
Two parts to an optimization problem

- **Objective Function** is the ‘thing’ you want to optimize (minimize/maximize)
- **Constraints** that must be honored

- Examples:
  - Find the cheapest airfare from Indianapolis to Athens with two or fewer stops and with travel time under 16 hours each way
  - Find the US-based news website with the highest percentage of verifiable international new stories

Knapsack Problems

- How do you store the set of items with the most value within the capacity of a knapsack?
- What is the **Objective Function**?
- What is/are the **Constraints**?



Greedy algorithms

- Chose the “best” item first
- Continue to the next
- Until limit is reached

- Not guaranteed to be **optimal**

Target list

	Value	Weight	Value/Weight
Clock	175	10	17.5
Painting	90	9	10
Radio	20	4	5
Vase	50	2	25
Book	10	1	10
Computer	200	20	10

What would the class look like

What is the complexity of our algorithm?

```
def greedy(items, maxWeight, objectiveFunction):
    itemsCopy = sorted(items, key=objectiveFunction,\
                       reverse=True)
    knapsack = []
    totalValue = 0
    weightRemaining = maxWeight
    for i in range(len(itemsCopy)):
        if itemsCopy[i].getWeight() <= weightRemaining:
            knapsack.append(itemsCopy[i])
            totalValue += itemsCopy[i].getValue()
            weightRemaining -= itemsCopy[i].getWeight()
    return knapsack, totalValue, weightRemaining
```

- O(n log(n))

If greed is not good – then what?

- The **optimal** solution is a ‘classic’ **0/1 knapsack problem**
  - Each item is represented by a pair <*value, weight*>
  - A knapsack can hold no more than *w* weight
  - A vector *I* of length n contains all available items
  - A vector *V* of length *n* contains 1’s and 0’s to indicate whether an item is taken (1) or left behind (0)
  - Find *V* that maximizes

$$\sum_{i=0}^{n-1} V[i] * I[i].value$$

Such that

$$\sum_{i=0}^{n-1} V[i] * I[i].weight \le w$$

How are we going to do that?

- Generate all possible sets of items
  - Sound familiar?
  - Power set
- Remove all sets not meeting constraint
  - Total weight > w
- Select the remaining set with the highest value

- $O(n2^n)$
- Exhaustive enumeration

Greedy vs Exhaustive Enumeration

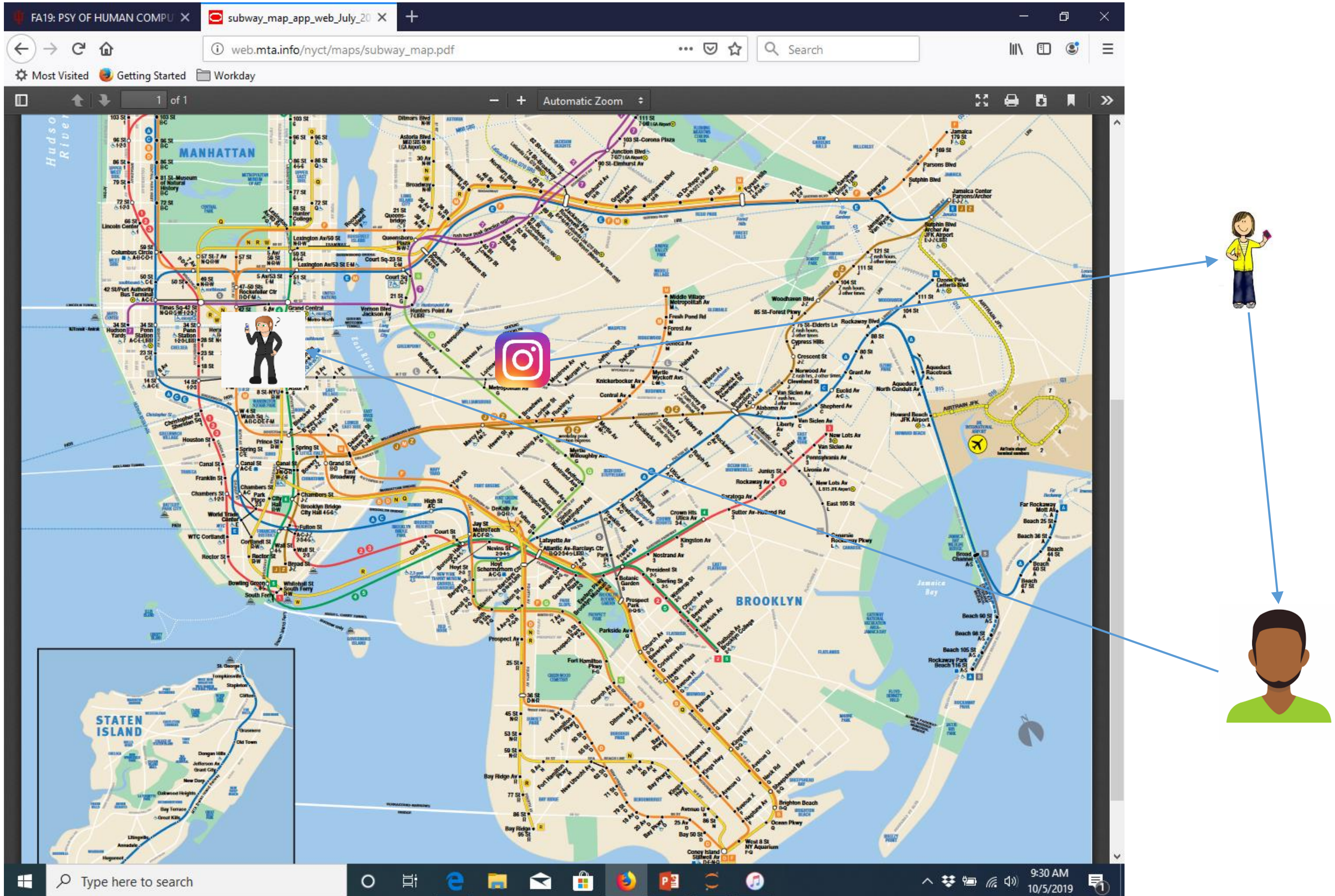
- Greedy algorithms are **locally optimal** for a evaluation function
  - Not necessarily **globally optimal**
- Exhaustive enumeration algorithms are **globally optimal**
  - Not necessarily (almost by definition) not efficient
- Greedy algorithms are often good enough and are frequently used

Some more questions

- Assuming the cost of flying between cities A and C is equal to the cost of flying from A to B and then B to C …
  - What is the smallest number of stops between two cities
  - What is the least expensive airfare between two cities
  - What is the least expensive airfare between two cities involving no more the *n* stops
  - What is the least expensive way to visit a collection of cities
    - Traveling sales rep problem

- These are classic graph optimization problems

What is a graph?



What is a graph? (wordy)

- A **graph** is a collection of **nodes** (or **vertices**)
- **Nodes** are connected by **arcs** (or **edges**)
- If **edges** are uni-directional we call this a **directed graph** (or **digraph**)
  - **Source** or **parent** nodes
  - **Destination** or **child** nodes
- A **weighted graph** gives a weight to each **edge**
  - Price and travel time in our airline example
- We can create a digraph by converting A↔B, to A→B and B→A

Graph examples

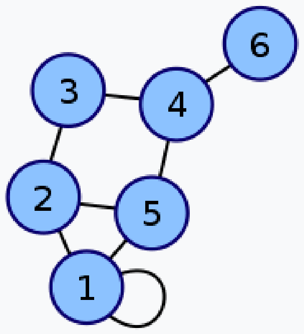
- Map routes
- WWW – pages linked to pages
- Gene expression
- Protein interaction
- Phase transitions
- Disease trajectories

- Let’s look at an example

Adjacency matirx

- Undirected graph
  - Boolean-ish (self-refenced nodes count as true)

Labeled graph



Adjacency matrix

$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Coordinates are 1–6.

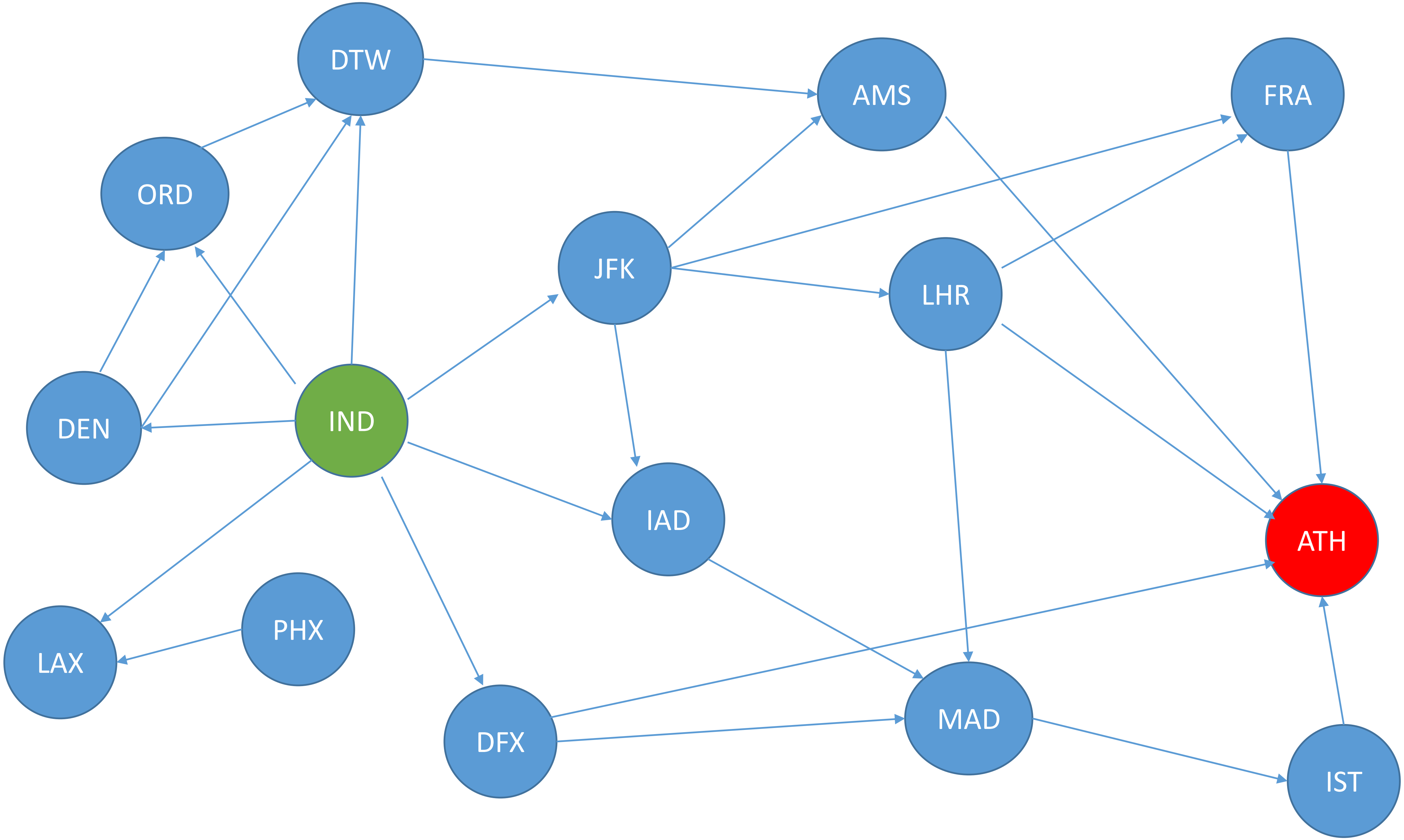
- Directed graph
  - Edge count is based on direction
  - Matrix might not be symmetric
- Weighted graph

Adjacency list

- List of nodes
- List of edges
  - We implemented this as a dictionary with source node as the key

Some classic graph theory problems

- **Shortest path**
  - For some pair of nodes **n1** and **n2**, find the shortest sequence of edges such that:
    - The source node of the first edge is **n1**
    - The destination node of the last edge is **n2**
    - Edges e_n and e_{n+1} are in sequence, if the destination node of e_n is the source node of e_{n+1}


DTW
AMS
FRA
ORD
JFK
LHR
DEN
IND
IAD
ATH
PHX
LAX
DFX
MAD
IST

Some more GT problems

- **Shortest weighted path**
    - Like shortest path except we minimize some function that gives the weight of each edge.
- **Maximum clique**
    - A set of nodes such that there is an edge between each pair of nodes in the set
- **Minimum cut**
    - Given two sets of nodes, what is the minimum number of edges that must be broken to separate the two sets

Solving shortest path

- Social networks
  - Unidirectional – Twitter, Instagram – user follows others
  - Bidirectional – Facebook, LinkdIn – ‘friendship’, ‘connection’

- Six Degrees of Separation
  - Me -> rented a house from -> Sam Roecca
  - Sam Roecca -> wrote on the original -> Flintstones cartoon
  - Flintstones cartoon-> inspired live action -> Flintstones movie
  - Flintstones movie -> featured -> Elizabeth Taylor
  - Elizabeth Taylor -> AIDS activist with -> Sir Elton John
  - Sir Elton – is a knight like -> Mick Jagger

Shortest path solution

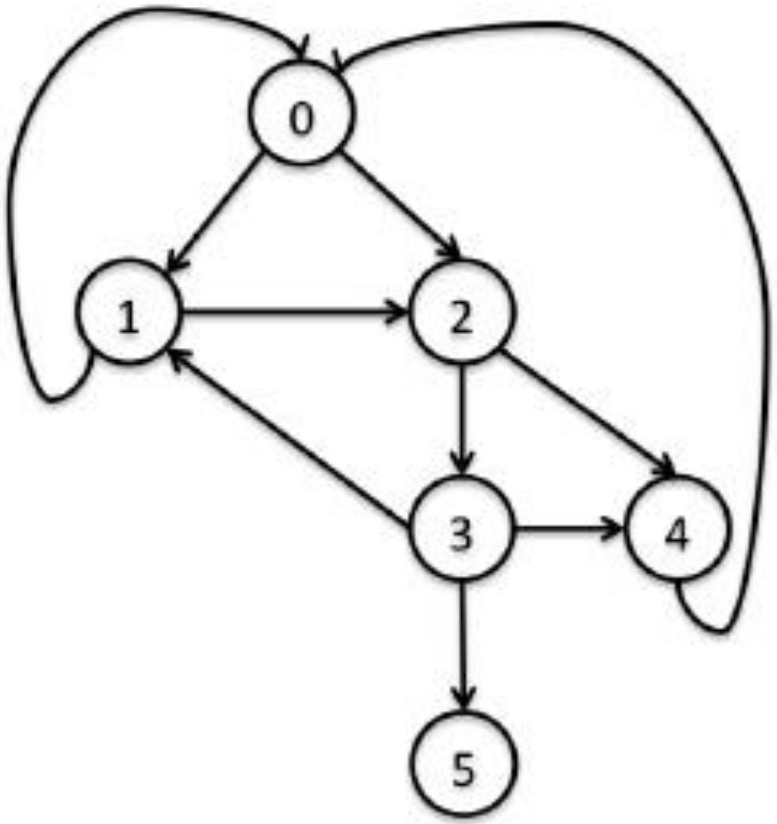
- Let G be the graph representing relationships
- For G, find the shortest sequence of nodes such that
- If n_i and n_{i+1} are consecutive nodes in the graph thee is an edge connecting n_i and n_{i+1}

- **Depth First (DFS)**
- **Breadth First (BFS)**

Depth First Search

- Choose one child node of the original **source** node
- Choose a child node of that destination
- Continue until either **destination** is reached
- … or node has no children
- **Backtrack** to the previous node until all children have been explored
- Continue until all children of the original **source** have been explored

Let’s explore



Breadth First Search

- Visit all children of the **start** node
- If none is the end
  - Visit all children of the children
  - Etc, etc, etc

- Usually implemented iteratively
- First path found will be the shortest path