

Lab 4

Blind Search

Oct 5, 2023

Department of Computing
The Hong Kong Polytechnic University

Today's Arrangement

- Graph Basics
- Introduction to necessary packages
 - NetworkX
 - Matplotlib
- Blind search algorithms
 - BFS and DFS
 - Dijkstra's Algorithm
- Four Exercises
- Assignment 1

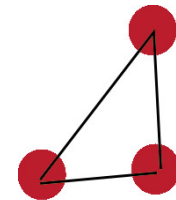


Graph Basics

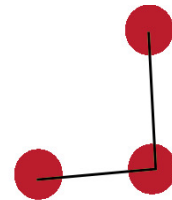


Graph Basics

- Basic elements
 - Vertices (or nodes)
 - Edges (or arcs, links)
 - Labels and weights
- Types of Graphs
 - Directed / Undirected
 - Weighted / Unweighted
 - Cyclic / Acyclic graphs
 - ...
- Representation of Graphs
 - Adjacency Matrix / Adjacency List / ...
- More about graph basics
 - <https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/>



Cyclic

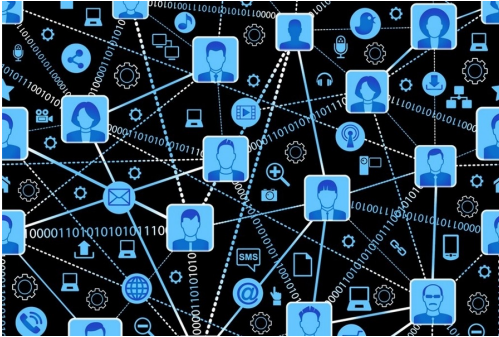


Acyclic

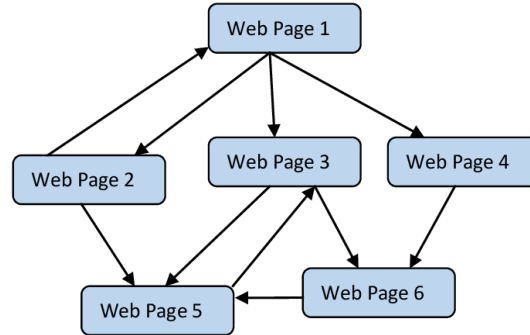
Introduction to necessary packages

NetworkX

- Networks are everywhere



Social networks



Webpage links



Traffic flows

- NetworkX is a useful tool to analyze those networks

NetworkX

- NetworkX: A Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
 - Data structures for graphs, digraphs, and multigraphs
 - Many standard graph algorithms
 - Network structure and analysis measures
 - Nodes can be "anything" (text, images, etc.)
 - Edges can hold arbitrary data (weights, time-series, etc.)
- Official document:
<https://networkx.org/documentation/stable/index.html>




NetworkX

- Set up environment
 - Make sure NetworkX has been installed (If not, run the command: `pip install networkx`)
 - Start a Jupyter Notebook and import the package NetworkX:
`import networkx as nx`
- Build a graph
 - Note: Different classes exist for directed and undirected graphs
`g = nx.Graph() # create undirected graph`


NetworkX

- Add and remove nodes

Add one node

`g.add_node(1)` 

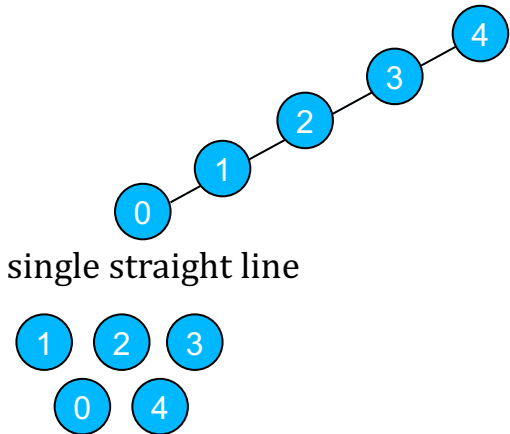
Add a list of nodes

`g.add_nodes_from([2, 3])` 

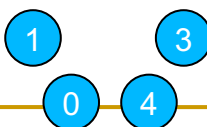
Add from another graph

`h = nx.path_graph(5)` # In a path graph, all vertices and edges lie on a single straight line

`g.add_nodes_from(h)` # Add all the nodes from graph 'h' to graph 'g'.



remove any node of the graph

`g.remove_node(2)` 

NetworkX

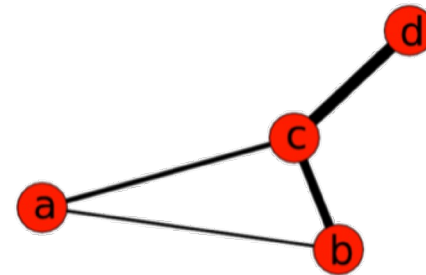
- A node can be any hashable object (a string, a function, a file, etc.)

```
import math
g = nx.Graph()
g.add_node('string')
g.add_node(math.cos) # cosine function
f = open('temp.txt', 'w') # file handle
g.add_node(f)
print(g.nodes())
```

NetworkX

- Add edges

```
g = nx.Graph()
g.add_edge('a', 'b', weight = 0.1)
g.add_edge('b', 'c', weight = 1.5)
g.add_edge('a', 'c', weight = 1.0)
g.add_edge('c', 'd', weight = 2.2)
```



NetworkX

- Accessing nodes and edges

```
# Show all the nodes
print("Nodes:", g.nodes())

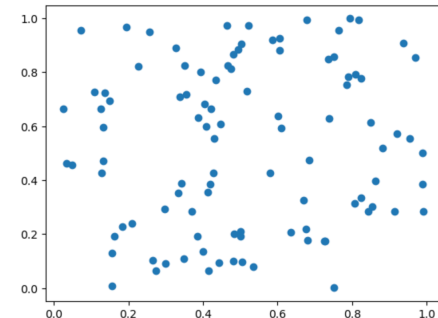
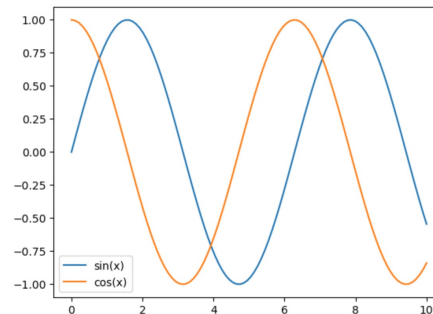
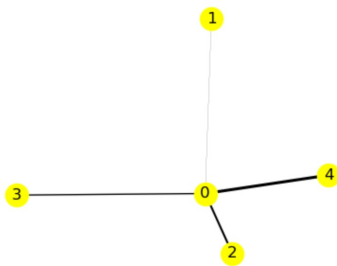
# Show all the edges
print("Edges:", g.edges())

# Number of edges and nodes
print("Number of nodes:", g.number_of_nodes())
print("Number of edges:", g.number_of_edges())

# Neighbors and the degree of the node 1
print("Neighbors of node 1:", list(g.neighbors(1)))
print("Degree of node 1:", g.degree(1))
```

Matplotlib

- Matplotlib
 - A powerful data visualization library for the Python programming language.
 - can create a wide variety of plots, including line plots, scatter plots, bar plots, pie charts, and histograms.



Blind search algorithms

Blind Search Algorithms

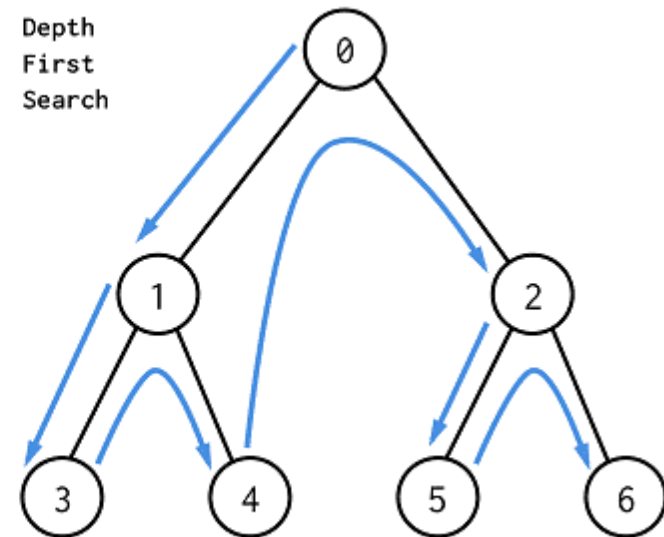
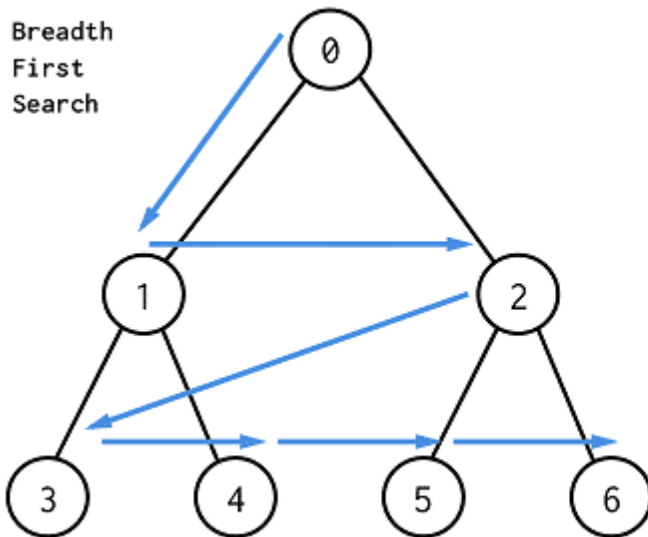
- Blind Search (Uninformed Search)
 - No information about the search space, other than to distinguish the goal state from all the others.
 - A brute-force way of searching for a solution.
- Common blind search algorithms
 - Breadth First Search (tree/graph traversal)
 - Depth First Search (tree/graph traversal)
 - Dijkstra's Algorithm (shortest path search for weighted graphs)

BFS and DFS

- Breadth First Search
 - Starts from the root node (selected or arbitrary node in the case of a graph) and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
- Depth First Search
 - Starts from the root node (selected or arbitrary node in the case of a graph) and explores as far as possible along each branch before backtracking.

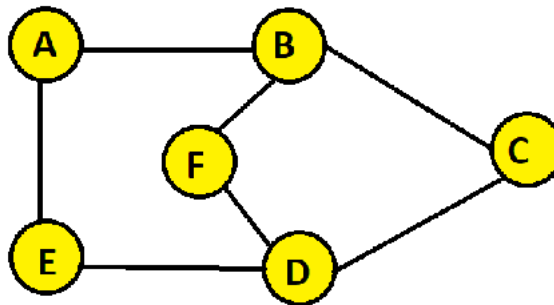
BFS and DFS

- BFS and DFS for a tree



BFS and DFS

- BFS and DFS for a graph
 - Unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, a boolean visited array is used to mark the visited vertices.



DFS: C->B->A->E->D->F

BFS: C->B->D->A->F->E

Dijkstra's Algorithm

- A simple modification to breadth-first search
 - with a priority queue instead of a normal first-in-first-out queue
- An algorithm for finding the shortest paths between nodes in a weighted graph
 - You can find the shortest path from a node (called the "source node") to all other nodes in the graph.
- Check this video for understanding:

<https://www.youtube.com/watch?v=pVfj6mxhdMw>

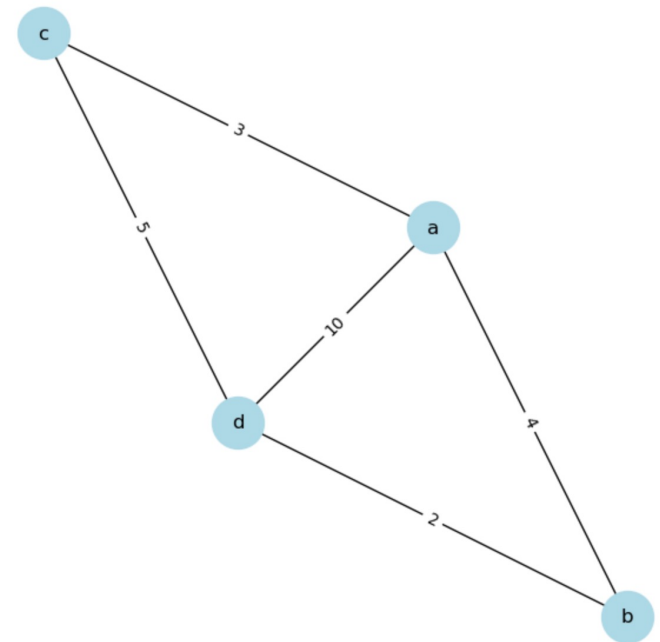


Exercise 1



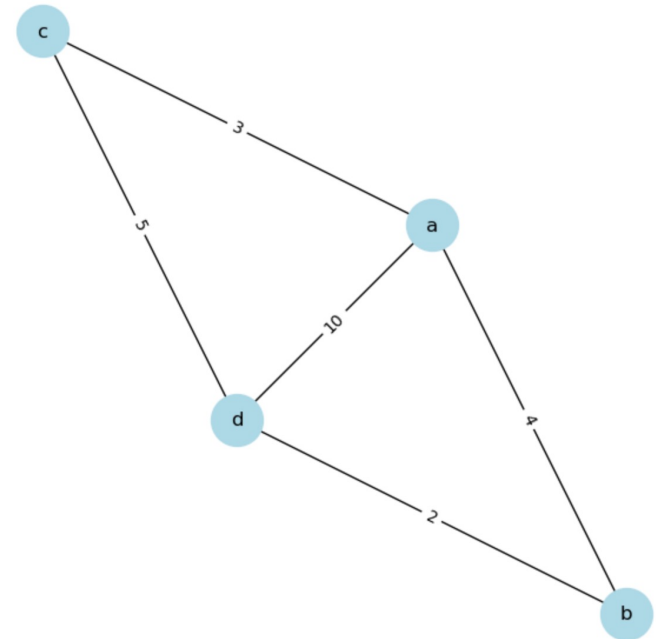
Exercise 1: Drawing graphs

- Use proper prompt to ask ChatGPT to help drawing the graph using the packages networkx and matplotlib.
- Requirements:
 1. Add all nodes correctly.
 2. Add edges with their weights and label the weights correctly.
 3. Specify the style (color, font, etc.) and positions of nodes.
 4. Understand each used function (such as the attributes it takes, the output it returns and so on) with the help of ChatGPT.



Exercise 1: Drawing graphs

- Example prompt:
 - Write a Python script that uses the packages `networkx` and `matplotlib` to create a graph.
 - The list of nodes: ['a', 'b', 'c', 'd'].
 - Label the names of the nodes and fix the positions of the nodes by the coordinates: {'a': [2, 2], 'b': [3,0], 'c': [0, 3], 'd': [1, 1]}.
 - Color all the nodes light blue.
 - The list of edges with their weights: [('a', 'b', 4), ('a', 'c', 3), ('a', 'd', 10), ('b', 'd', 2), ('c', 'd', 5)].
 - Label the weights on the edges.



Exercise 1: Drawing graphs

- Resulting code:

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a graph
G = nx.Graph()

# Define the list of nodes
nodes = ['a', 'b', 'c', 'd']

# Add nodes to the graph with fixed positions
fixed_positions = {'a': [2, 2], 'b': [3, 0], 'c': [0, 3], 'd': [1, 1]}
for node, pos in fixed_positions.items():
    G.add_node(node, pos=pos)

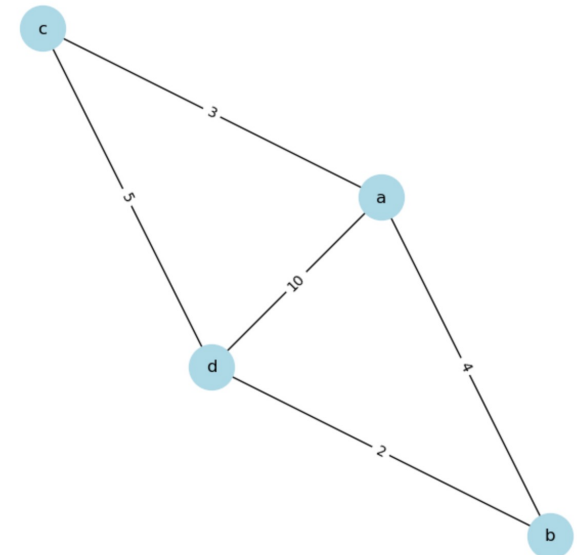
# Define the list of edges with their weights
edges = [('a', 'b', 4), ('a', 'c', 3), ('a', 'd', 10), ('b', 'd', 2), ('c', 'd', 5)]

# Add edges to the graph with weights
for edge in edges:
    G.add_edge(edge[0], edge[1], weight=edge[2])

# Draw the graph
plt.figure(figsize=(6, 6))
pos = nx.get_node_attributes(G, 'pos')
nx.draw(G, pos, with_labels=True, node_size=1000, node_color='lightblue', font_size=12)

# Label the weights on the edges
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels, font_size=10)

# Show the graph
plt.axis('off')
plt.show()
```



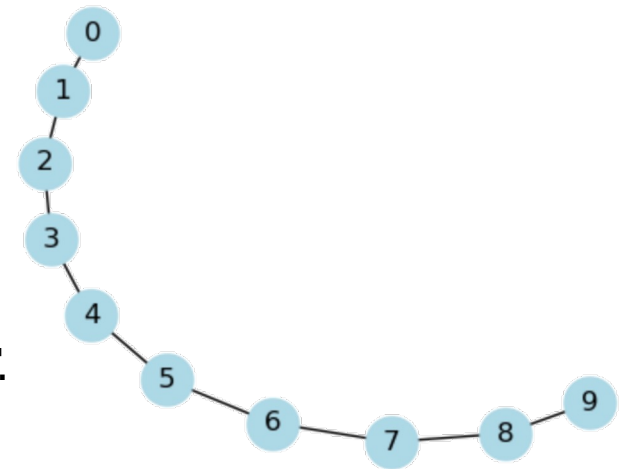


Exercise 2



Exercise 2: BFS and DFS

- Use proper prompt to ask ChatGPT to do the following tasks.
- Tasks:
 1. Draw the path graph.
 2. Do breadth-first-search starting at the node 4.
 3. Do depth-first-search starting at the node 4.
 4. Understand the code with the help of ChatGPT.

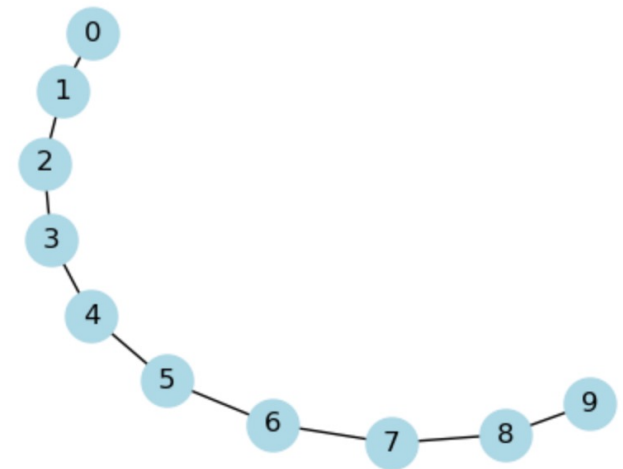


Exercise 2: BFS and DFS

- Example prompt:

Write a Python script that uses the packages `networkx` and `matplotlib` to do the following tasks.

1. Draw a path graph consisting of 10 nodes (from node 0 to node 9).
2. Then, do the breadth-first-search and depth-first-search starting at the node 4. Print the orders of the nodes visited by BFS and DFS.



Exercise 2: BFS and DFS

- Resulting code and output:

```
import networkx as nx
import matplotlib.pyplot as plt

# Function to perform breadth-first search
def bfs(graph, start):
    visited = set() # Set to keep track of visited nodes
    queue = [start] # Queue to store nodes to be visited

    while queue:
        node = queue.pop(0)
        if node not in visited:
            print(node, end=' ')
            visited.add(node)
            queue.extend(graph[node])

# Function to perform depth-first search
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set() # Set to keep track of visited nodes

    visited.add(start)
    print(start, end=' ')

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

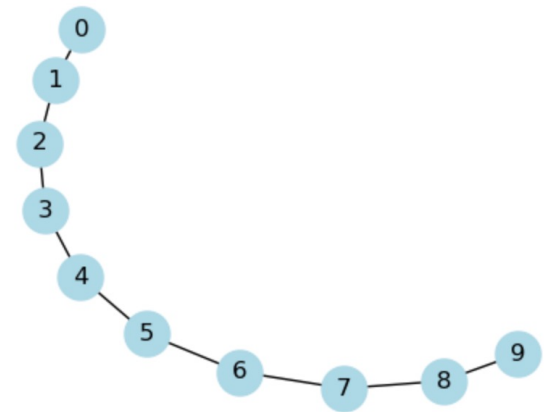
# Creating a path graph with 10 nodes
graph = nx.path_graph(10)

# Drawing the graph
pos = nx.spring_layout(graph) # Layout algorithm for positioning the nodes
nx.draw(graph, pos, with_labels=True, node_color='lightblue', node_size=500)
plt.title('Path Graph')
plt.show()

# Performing BFS and printing the order of visited nodes
print('BFS Order:', end=' ')
bfs_order = bfs(graph, 4)
print()

# Performing DFS and printing the order of visited nodes
print('DFS Order:', end=' ')
dfs_order = dfs(graph, 4)
```

Path Graph



BFS Order: 4 3 5 2 6 1 7 0 8 9

DFS Order: 4 3 2 1 0 5 6 7 8 9



Exercise 3

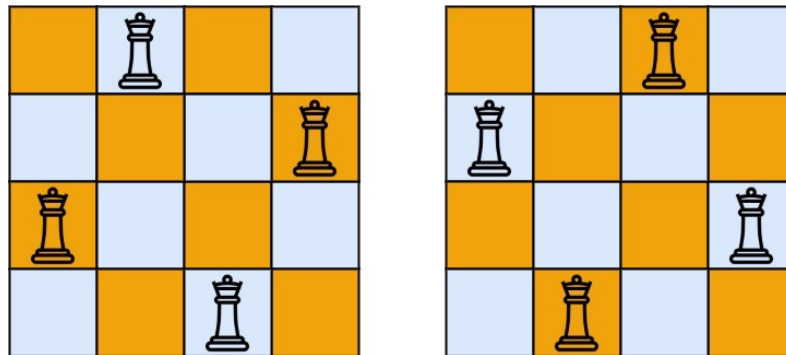


Exercise 3: 4-queens' problem

4 – Queens' problem is to place 4 queens on a 4 x 4 chessboard in such a manner that no queens attack each other by being in the same row, column, or diagonal.

Requirements:

- Use DFS to solve this problem.
- Draw the chessboard of each step in the searching.
- Understand the code with the help of ChatGPT.



Exercise 3: 4-queens' problem

- Example prompt:

Write a Python script to solve the 4-queens' problem using DFS. Follow the instructions to solve this problem.

1. Determine the position of the queen in the four columns in turn.
2. Represent the chessboard with a 2-dimension array. For each chessboard, label the positions where there are queens with "Q".
3. Show each step of the search process, including the intermediate states.

Exercise 3: 4-queens' problem

- Resulting code and output:

```
def is_valid(board, row, col):
    # Check if there is any queen in the same row
    for i in range(col):
        if board[row][i] == 'Q':
            return False

    # Check upper diagonal on left side
    i = row
    j = col
    while i >= 0 and j >= 0:
        if board[i][j] == 'Q':
            return False
        i -= 1
        j -= 1

    # Check lower diagonal on left side
    i = row
    j = col
    while i < 4 and j >= 0:
        if board[i][j] == 'Q':
            return False
        i += 1
        j -= 1

    return True
```

```
def solve(board, col):
    if col >= 4:
        return True

    for i in range(4):
        if is_valid(board, i, col):
            board[i][col] = 'Q'
            print_board(board)

            if solve(board, col + 1):
                return True

            board[i][col] = '-' # Backtrack

    return False
```

```
def print_board(board):
    for row in board:
        print(' '.join(row))
    print()
```

```
board = [['-' for _ in range(4)] for _ in range(4)]
solve(board, 0)
```

```
Q - - -
- - - -
- - - -
- - - -

Q - - -
- - - -
- Q - -
- - - -

Q - - -
- - - -
- - - -
- Q - -

Q - - -
- - Q -
- - - -
- Q - -

- - - -
Q - - -
- - - -
- - - -

- - - -
Q - - -
- - - -
- Q - -

- - Q -
Q - - -
- - - Q
- Q - -
```



Exercise 4



Exercise 4: Shortest Path

Formulate the real-world problem to a shortest path problem and solve it.

- You have just purchased a new car for \$20,000. The cost of maintaining this car during a year depends on its age from the year you purchased it:

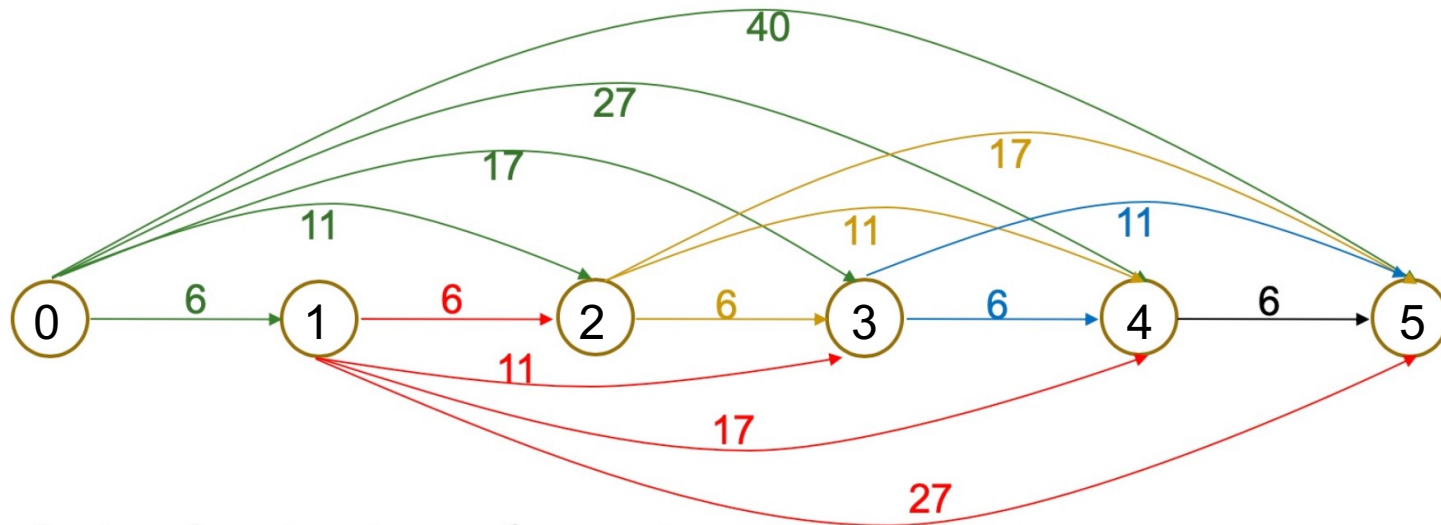
Age of car (years)	0	1	2	3	4
Annual maintenance cost (\$)	1000	2000	3000	6000	10000

- To avoid the high maintenance costs associated with an older car, you may trade in your car and purchase a new car. The price you receive on a trade-in depends on the age of the car at the time of the trade-in:

Age of car (years)	1	2	3	4	5
Trade-in price (\$)	15000	12000	9000	5000	2000

Exercise 4: Shortest Path

Formulation of the problem:



- node i = beginning of year i
- edge (i, j) = purchase new car at the beginning of year i and keep it until year j

Exercise 4: Shortest Path

Example prompts (4 prompts in total):

- The first prompt:
 - You have just buy a new car for \$20,000.
 - The cost of maintaining this car increases with the years. In the first year from the year you buy it, the maintaining cost is \$1000. In the second year from the year you buy it, the maintaining cost is \$2000. Then, in the third, fourth, and fifth year from the year you buy it, the maintaining cost is \$3000, \$6000 and \$10000 respectively.
 - To avoid the high maintenance costs, you may trade in the old car and purchase a new car at any year. If you sell the car after maintaining the car for 1, 2, 3, 4, and 5 years respectively, you can get \$15000, \$12000, \$9000, \$5000, and \$2000 back respectively.
 - Now calculate the net cost incurred from buying this car at year 0 and selling this car at year 1, 2, 3, 4 and 5 respectively.

Exercise 4: Shortest Path

- The first response:

To calculate the net cost incurred, we need to subtract the amount received from selling the car from the initial purchase cost and the maintenance costs. Let's calculate the net cost for each scenario:

1. Selling at year 1:

Net cost = Initial purchase cost - Amount received from selling + Maintenance cost for year 1

Net cost = \$20,000 - \$15,000 + \$1,000

Net cost = \$6,000

2. Selling at year 2:

Net cost = Initial purchase cost - Amount received from selling + Maintenance cost for year 1 + Maintenance cost for year 2

Net cost = \$20,000 - \$12,000 + \$1,000 + \$2,000

Net cost = \$11,000

3. Selling at year 3:

Net cost = Initial purchase cost - Amount received from selling + Maintenance cost for year 1 + Maintenance cost for year 2 + Maintenance cost for year 3

Net cost = \$20,000 - \$9,000 + \$1,000 + \$2,000 + \$3,000

Net cost = \$17,000

4. Selling at year 4:

Net cost = Initial purchase cost - Amount received from selling + Maintenance cost for year 1 + Maintenance cost for year 2 + Maintenance cost for year 3 + Maintenance cost for year 4

Net cost = \$20,000 - \$5,000 + \$1,000 + \$2,000 + \$3,000 + \$6,000

Net cost = \$27,000

5. Selling at year 5:

Net cost = Initial purchase cost - Amount received from selling + Maintenance cost for year 1 + Maintenance cost for year 2 + Maintenance cost for year 3 + Maintenance cost for year 4 + Maintenance cost for year 5

Net cost = \$20,000 - \$2,000 + \$1,000 + \$2,000 + \$3,000 + \$6,000 + \$10,000

Net cost = \$40,000

Exercise 4: Shortest Path

- The second prompt:

Create a simple table to save the net costs. Record the start year, end year, the maintenance time (in years) and the net cost of each scenario.

- The second response:

Sure! Here is a table to summarize the net costs for each scenario:

Start Year	End Year	Maintenance Time	Net Cost
0	1	1	\$6,000
0	2	2	\$11,000
0	3	3	\$17,000
0	4	4	\$27,000
0	5	5	\$40,000



Please note that these net costs are calculated based on the given information.

Exercise 4: Shortest Path

- The third prompt:

Now expand the table by adding more cases. Firstly, add all the records where the start year being 1, 2, 3, and 4 (the end year should be larger than the start year and up to 5). Secondly, calculate the maintenance time for each record. Thirdly, the records with the same maintenance time should have the same Net Cost. Use this rule to infer all the Net Costs of the new cases.

Exercise 4: Shortest Path

- The third response:

Sure! I have expanded the table based on your instructions:

Start Year	End Year	Maintenance Time	Net Cost
0	1	1	\$6,000
0	2	2	\$11,000
0	3	3	\$17,000
0	4	4	\$27,000
0	5	5	\$40,000
1	2	1	-
1	3	2	-
1	4	3	-
1	5	4	-
2	3	1	-
2	4	2	-
2	5	3	-
3	4	1	-
3	5	2	-
4	5	1	-

Now let's calculate the maintenance time for each record:

Start Year	End Year	Maintenance Time (years)	Net Cost
0	1	1	\$6,000
0	2	2	\$11,000
0	3	3	\$17,000
0	4	4	\$27,000
0	5	5	\$40,000
1	2	1	-
1	3	2	-
1	4	3	-
1	5	4	-
2	3	1	-
2	4	2	-
2	5	3	-
3	4	1	-
3	5	2	-
4	5	1	-

Now, we can see that each maintenance time (from 1 to 5) corresponds to a specific net cost. So, we can infer the net costs for the new cases based on the net costs we already know:

Start Year	End Year	Maintenance Time (years)	Net Cost
0	1	1	\$6,000
0	2	2	\$11,000
0	3	3	\$17,000
0	4	4	\$27,000
0	5	5	\$40,000
1	2	1	\$6,000
1	3	2	\$11,000
1	4	3	\$17,000
1	5	4	\$27,000
2	3	1	\$6,000
2	4	2	\$11,000
2	5	3	\$17,000
3	4	1	\$6,000
3	5	2	\$11,000
4	5	1	\$6,000

Now, we have calculated the net costs for all the new cases based on the given information.

Exercise 4: Shortest Path

- The fourth prompt:

Now create a graph using `networkx` where the nodes represent the years from year 0 to year 5, and the edges represent the corresponding net cost. Then give the shortest path and the total cost from node 0 to node 5.

Exercise 4: Shortest Path

- The fourth response (the resulting code):

To create a graph using networkx, we'll first import the necessary libraries and create the graph object. Then we'll add the nodes representing the years and the edges representing the net cost. Finally, we'll find the shortest path and calculate the total cost from node 0 to node 5. Here's the code:

```
```python
import networkx as nx
import matplotlib.pyplot as plt

Create the graph object
G = nx.Graph()

Add nodes representing the years
years = [0, 1, 2, 3, 4, 5]
G.add_nodes_from(years)

Add edges representing the net cost
G.add_edge(0, 1, net_cost=6000)
G.add_edge(0, 2, net_cost=11000)
G.add_edge(0, 3, net_cost=17000)
G.add_edge(0, 4, net_cost=27000)
G.add_edge(0, 5, net_cost=40000)
G.add_edge(1, 2, net_cost=6000)
G.add_edge(1, 3, net_cost=11000)
G.add_edge(1, 4, net_cost=17000)
G.add_edge(1, 5, net_cost=27000)
G.add_edge(2, 3, net_cost=6000)
G.add_edge(2, 4, net_cost=11000)
G.add_edge(2, 5, net_cost=17000)
G.add_edge(3, 4, net_cost=6000)
G.add_edge(3, 5, net_cost=11000)
G.add_edge(4, 5, net_cost=6000)
```

```
Draw the graph
nx.draw(G, with_labels=True, font_weight='bold')
plt.show()

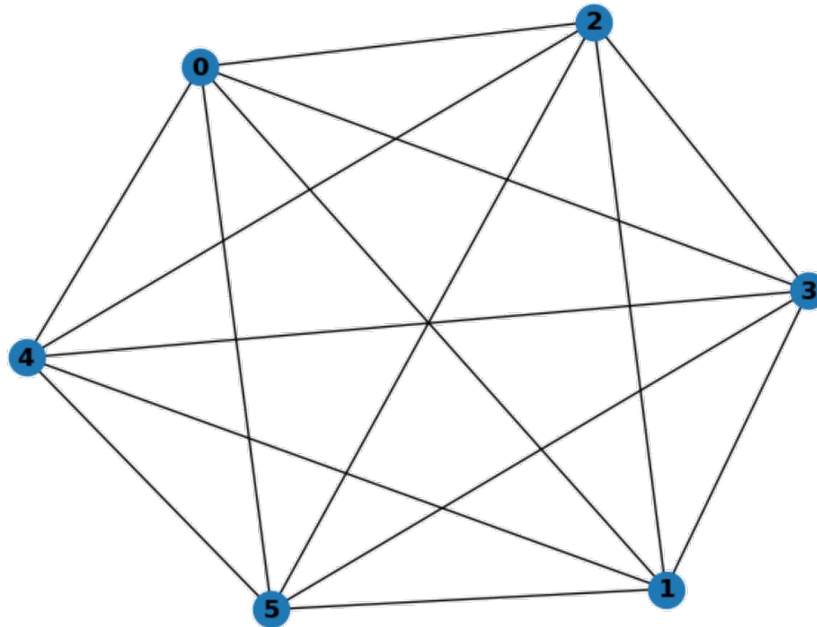
Find the shortest path and calculate the total cost
shortest_path = nx.shortest_path(G, source=0, target=5, weight='net_cost')
total_cost = nx.shortest_path_length(G, source=0, target=5, weight='net_cost')

print("Shortest path from node 0 to node 5:", shortest_path)
print("Total cost from node 0 to node 5:", total_cost)
```
```

This code will create the graph and display it using networkx and matplotlib. It will also print the shortest path from node 0 to node 5 and the total cost along that path.

Exercise 4: Shortest Path

- Output of the code:



Shortest path from node 0 to node 5: [0, 2, 5]
Total cost from node 0 to node 5: 28000

Assignment 1

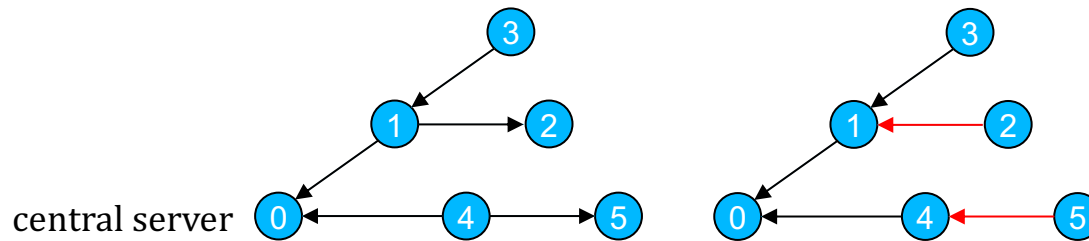
- Group work
 - Submit a .ipynb file via Blackboard
-

Assignment 1

- The Information Center has set up a network to allow data uploads between servers. This network can be represented by a set of permitted directed transmissions, where a permitted directed transmission from server a_i to server b_i can be formulated as $[a_i, b_i]$. It's given that $0 \leq a_i, b_i < n, a_i \neq b_i$. There are n ($n \geq 2$) servers in the network with server 0 being the central server. The number of directed transmissions is less than n .
- The Information Center plans to update this network. The objective is to ensure the central server (0) can receive data uploads from every other server in the network. Your task is to reorient some of these permitted transmissions so that the central server can receive data from other servers. Return the minimum number of transmission changes required.
- If it's not feasible to enable the central server to receive data from other servers by adjusting the directions of existing permitted transmissions, then return -1.

Assignment 1

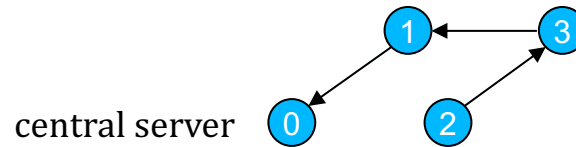
- Example 1:



- Demo input: transmissions = $[[1, 0], [3, 1], [1, 2], [4, 0], [4, 5]]$, num_servers = 6
- Expected output: 2
- Transmission changes are demonstrated in red lines to ensure the central server can receive data from all the other servers.

Assignment 1

- Example 2:



- Demo input: transmissions = $[[1, 0], [3, 1], [2, 3]]$, num_servers = 4
- Expected output: 0
- No transmission route changes are needed, the central server can already receive data from all other servers.

Assignment 1

- Please use the following Python template for submission.
- Your program will be tested on other test cases.

```
import networkx as nx
from typing import List

def solve(transmissions: List[List[int]], num_servers: int) -> int:
    """Your solution to the problem goes in this function.
    Args:
    transmissions (List[List[int]]): The permitted transmission
        between servers,
        e.g., [[1, 0], [3, 1], [1, 2], [4,0], [4,5]]
    num_servers (int): The number of servers in the network,
        e.g., 6
    Returns:
    int: the minimum number of transmission changes required,
        e.g., 2
    """
    return -2
```

```
# test case 1
transmissions = [[1, 0], [3, 1], [1, 2], [4,0], [4,5]]
num_servers = 6
answer = 2
result = solve(transmissions, num_servers)
assert result == answer, f"Test case 1: expected {answer}, got {result}"
print('Passed test case 1...')

# test case 2
transmissions = [[1, 0], [3, 1], [2, 3]]
num_servers = 4
answer = 0
result = solve(transmissions, num_servers)
assert result == answer, f"Test case 2: expected {answer}, got {result}"
print('Passed test case 2...')
```