**A logo with a shield and lions

Description automatically generated**

Short Path by using Algorithms on Python and Implementation of RIP and OSFP using Cisco Packet Tracer

**Lab 2**

**Telecommunication Software**

**Submitted by**

ABDUL HAYEE

[241AME011]

**Submitted to:**

TIANHUA CHEN

**FACULTY OF COMPUTER SCIENCE, INFORMATION TECHNOLOGY AND ENERGY**

INSTITUTE OF PHOTONICS, ELCTRONICS AND ELECTRONIC COMMUNICATIONS

RIGA TECHNICAL UNIVERSITY

[15th December ,2024]

**SUPERVISOR SIGNATURE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CANDIDATE SIGNATURE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Task 1: Python Stack and Bracket Matching

This task focuses on implementing fundamental data structure operations using a Python Stack. The objective is to demonstrate stack manipulation through two practical applications: converting decimal numbers to octal representation and validating bracket matching in complex string patterns.

A screen shot of a computer

Description automatically generated

#### Output

A black background with white text

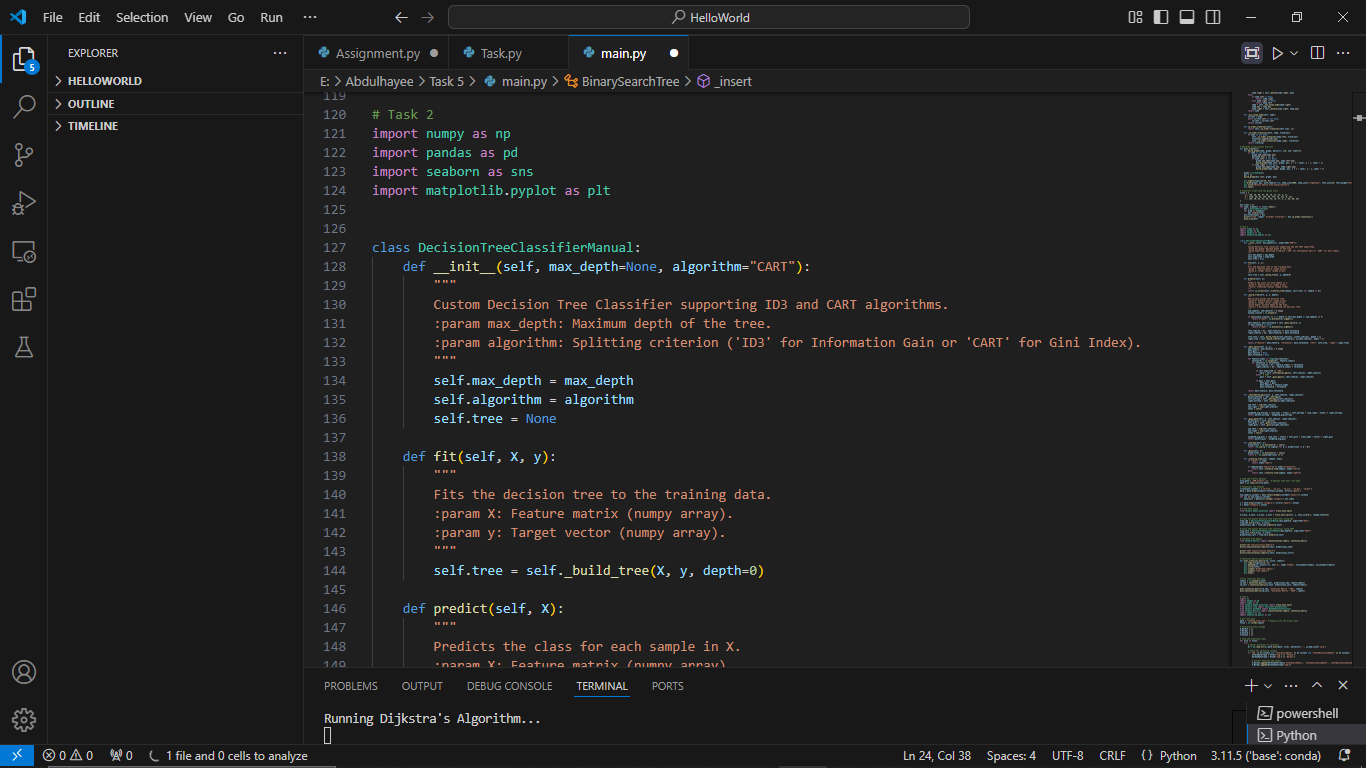
Description automatically generated

In this task, I implemented a robust stack-based solution for decimal to octal conversion. I developed a function that systematically converts decimal numbers by using stack operations, pushing and popping digits during the conversion process. Additionally, I created a bracket matching algorithm that uses a stack to track and validate the proper nesting of different bracket types, including square brackets, curly braces, and parentheses.

# 

# Task 2: Bellman-Ford Algorithm

The Bellman-Ford algorithm is a crucial graph traversal technique for finding the shortest paths in weighted graphs, particularly those containing negative edge weights. This task involves implementing the algorithm across multiple graph structures to demonstrate its versatility and computational approach.



A diagram of a hexagon with green dots

Description automatically generated

I implemented the Bellman-Ford algorithm to traverse five distinct graph configurations from the course slides. I carefully coded the algorithm to find the shortest paths between neighboring nodes in each graph, ensuring comprehensive path calculation. I also developed visualization capabilities to plot these graphs, including detailed representations of nodes, edges, and their respective weights.

A screen shot of a computer screen

Description automatically generated

Task 3: Dijkstra’s Algorithm

Dijkstra's algorithm is a fundamental shortest path algorithm used in graph theory and network routing. This task requires implementing the algorithm to explore pathfinding in weighted, directed graphs with non-negative edge weights.

A screen shot of a computer program

Description automatically generated

A diagram of a hexagon with red dots

Description automatically generatedI successfully implemented Dijkstra's algorithm to traverse the same five graphs used in the Bellman-Ford task. I developed a robust implementation that efficiently calculates the shortest paths from all nodes to their neighboring nodes. I enhanced the implementation by creating graph plotting functionality that visually represents nodes, edges, and their corresponding weights.

# Task 4: Prim’s Algorithm

Prim's algorithm is a greedy algorithm used to find the minimum spanning tree in a weighted, undirected graph. This task involves applying the algorithm to identify the most cost-effective network configuration.

A diagram of a constellation

Description automatically generatedI implemented Prim's algorithm for the five given graph structures, focusing on traversing all nodes to determine the minimum weight cost. I carefully coded the algorithm to select edges that create a minimum spanning tree, ensuring the most efficient network connectivity. I developed visualization techniques to represent the algorithm's results graphically.



# Task 5 - Prims Algorithm import networkx as nx

import matplotlib.pyplot as plt

G = nx.Graph() G.add\_edge('u', 'v', weight=3)

G.add\_edge('u', 'w', weight=2)

G.add\_edge('v', 'x', weight=1)

G.add\_edge('w', 'x', weight=1)

G.add\_edge('v', 'y', weight=2)

G.add\_edge('x', 'z', weight=4)

G.add\_edge('y', 'z', weight=1)

G.add\_edge('x', 't', weight=5)

G.add\_edge('w', 's', weight=4)

G.add\_edge('s', 't', weight=3)

def prims(graph): minimum\_spanning\_tree = nx.Graph() start\_node = list(graph.nodes())[0]

minimum\_spanning\_tree.add\_node(start\_node)

while len(minimum\_spanning\_tree) < len(graph): edge\_to\_add = None

# Task 5: Q-Learning and Shortest Path

This task explores reinforcement learning techniques in path finding, specifically using Q-learning to determine optimal routes in a graph. The objective is to compare Q-learning results with traditional graph traversal algorithms.

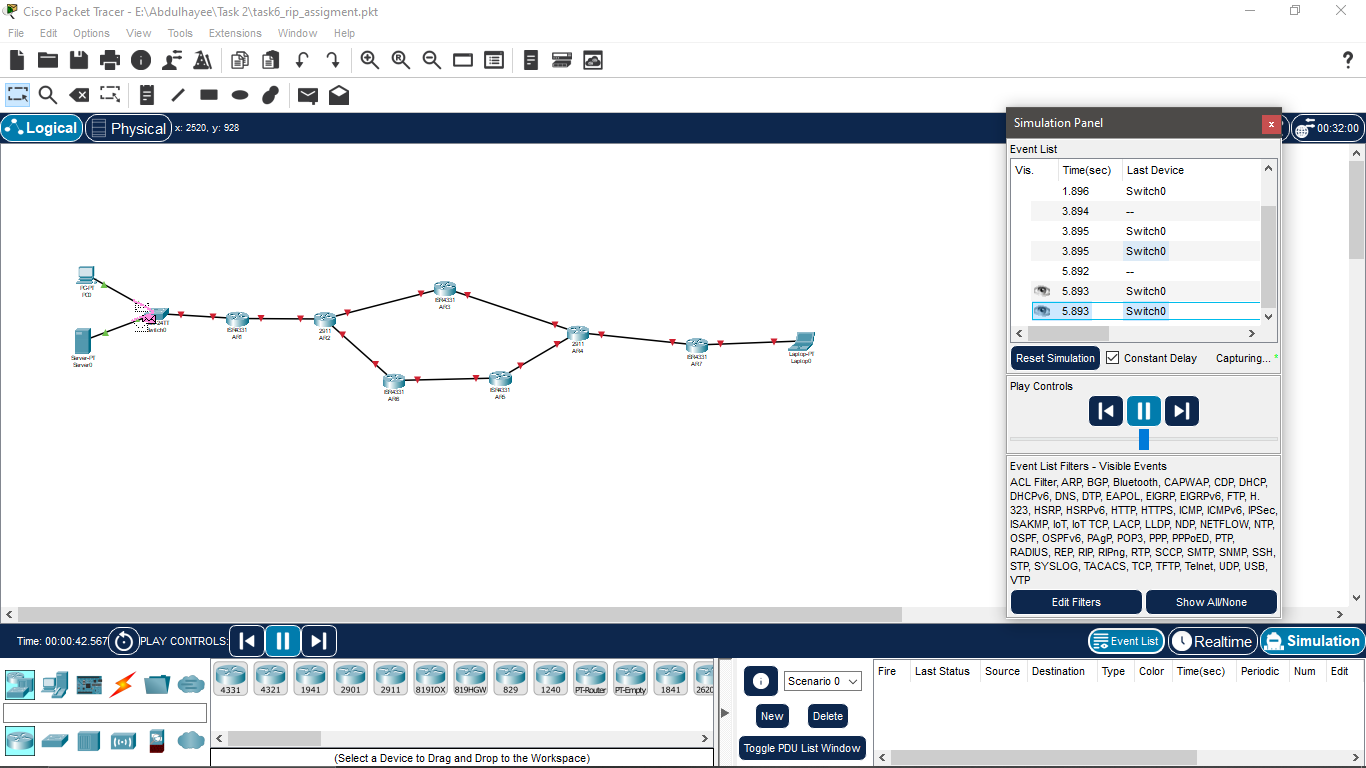
**A screen shot of a computer program

Description automatically generated**

# RIP Network Topology Configuration

Routing Information Protocol (RIP) is a fundamental distance-vector routing protocol designed to enable efficient communication between network devices. In this comprehensive network topology configuration, the primary objective was to create a robust and interconnected network infrastructure utilizing multiple routers, switches, and end devices. The task required meticulous planning of IP addressing, careful device configuration, and strategic implementation of routing protocols to ensure seamless data transmission across different network segments.

The implementation process involved a systematic approach to network design and configuration. I began by strategically placing routers (AR1 to AR7), switches, PCs, laptops, and servers, ensuring comprehensive network coverage. Each device was assigned a unique IP address with consistent subnet masks, creating a well-structured network environment. The critical phase involved configuring RIP version 2 on routers, advertising network addresses, and establishing connectivity between different network segments. By disabling auto-summary and carefully defining network boundaries, I ensured optimal routing performance and minimal redundancy.



**Add Devices** Routers: Switch:

PC and Laptop: Server:

Assign IP Addresses

|  |  |  |  |
| --- | --- | --- | --- |
| Device | Interface | IP Addresses | Subnet Mask |
| PC0 | NIC | 53.1.23.12 | 255.255.255.0 |
| Server0 | NIC | 53.1.23.5 | 255.255.255.0 |
| AR1 | Gig0/0/0 | 123.1.1.1 | 255.255.255.0 |
| AR1 | Gig0/0/1 | 53.1.23.22 | 255.255.255.0 |
| AR2 | Gig0/0/0 | 56.1.1.1 | 255.255.255.0 |
| AR2 | Gig0/0/1 | 123.1.1.2 | 255.255.255.0 |
| AR2 | Loopback0 | 192.168.1.0 | 255.255.255.0 |
| AR3 | Gig0/0/0 | 36.1.1.1 | 255.255.255.0 |
| AR3 | Gig0/0/1 | 123.1.1.3 | 255.255.255.0 |
| AR3 | Loopback0 | 192.168.1.128 | 255.255.255.0 |
| AR4 | Gig0/0/0 | 34.1.1.1 | 255.255.255.0 |
| AR4 | Gig0/0/1 | 32.1.1.3 | 255.255.255.0 |
| AR4 | Loopback0 | 192.168.3.128 | 255.255.255.0 |
| AR5 | Gig0/0/0 | 34.1.1.2 | 255.255.255.0 |
| AR5 | Gig0/0/1 | 41.5.1.4 | 255.255.255.0 |
| AR6 | Loopback0 | 192.168.2.168 | 255.255.255.0 |
| AR7 | Gig0/0/0 | 32.1.1.2 | 255.255.255.0 |
| AR7 | Gig0/0/1 | 78.5.2.55 | 255.255.255.0 |
| Laptop0 | NIC | 78.5.2.8 | 255.255.255.0 |

A diagram of a diagram

Description automatically generated

**Configure RIP on Routers**

Enter the configuration mode of each router and enable RIP version 2 with the following commands:

enablezz

configure terminal router rip

version 2

no auto-summary

network <network\_address> # For all connected networks exit

**Example for AR1:**

network 123.1.1.0

network 53.1.23.0

**Configure Loopback Interfaces:**

For AR2, AR3, AR4, AR5, AR6 routers, configure loopback interfaces:

interface loopback0

ip address <loopback\_ip> 255.255.255.0

**Verify Configuration:**

Use the following commands to check RIP and routing tables on each router

show ip route # Verify routing table

show ip protocols # Verify RIP configuration

**Test Connectivity:**

After configuration, ensure that all devices (PCs, Laptops, and Servers) can ping each other.

Advanced RIP Timers Optimization:

Set custom RIP timers to optimize convergence times:

**Save Configuration:**

Once you have configured the network, save the configuration using the command **copy running-config startup-config** on each router.

A screenshot of a computer

Description automatically generated

**Step-by-Step Configuration:**

**Devices to Add:**

Routers: Place all the 2911 routers as shown in the image**.**

**Switch: Add 2960 switch.**

PC, Server, Laptop: Add these end devices.

Connections: Use the appropriate cables (straight-through and cross-over cables). IP Addresses and Subnets:

Follow the IP addressing scheme shown in the image and configure each router's interfaces:

**Router1:**

GE/0/0: 192.1.1.2/24

OSPF configuration for Area 111

**Router2:**

GE/0/0: 192.1.1.4/24

OSPF configuration for Area 111

**Router3:**

**GE/0/0: 192.1.2.1/24**

**OSPF configuration for Area 111 OSPF Configuration:**

The commands to configure OSPF as shown in the image would be something like: Final Steps:

Navigate to File > Save, select a descriptive filename like "MultiArea\_OSPF\_Topology" or "RIP\_Network\_Configuration", and choose an appropriate storage location. The .pkt file will preserve your entire network design, allowing future reference and potential modifications.

With this final step, your network topology is securely documented and ready for further analysis or presentation.

# Task 8: INET Framework Simulation

In this task, I used some OSPF examples to observe how each protocol operates in terms of convergence time, control overhead, and efficiency in routing. OSPF generally outperforms RIP in large and dynamic networks due to its faster convergence and more efficient handling of routing updates.

A screenshot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

**Conclusion:**

In this laboratory exercise, I implemented several fundamental telecommunications and networking tasks. Starting with stack-based programming, I worked through basic implementation and operations to understand data structure principles. For the algorithmic portion, I applied three important graph algorithms: Bellman-Ford for detecting negative cycles and finding shortest paths, Dijkstra's algorithm for efficient shortest path calculations, and Prim's algorithm for determining minimum spanning trees in networks.

In the networking component, I focused on OSPF routing protocol implementation using OMNeT++ with the INET Framework. This involved creating network topologies with multiple nodes and connections, setting up IP addressing schemes, and configuring basic OSPF routing parameters. Through the INET Framework simulations, I was able to observe and analyze network behavior, routing table updates, and basic protocol operations. These simulations helped me understand the practical aspects of network routing and protocol implementation in a controlled environment.