Nallapuneni Vamsi Krishna

AP20110010735

Vamsikrishna_nallapuneni@srmap.edu.in

PROBLEM – 1

Approach Explanation

the approach and focus on the best and most efficient method to identify the 3 poisonous bottles among 1000 bottles.

1. Numbering Bottles: We have 1000 wine bottles, each with a unique number from 1 to 1000.

2. Binary Representation: We represent each bottle number in binary form. Since 1000 can be represented in binary with 10 digits (bits), we use 10 bits for each bottle.

3. Assigning Codes to Prisoners:

- We'll have 10 prisoners, each assigned to a specific bit position in the binary representation of the bottle numbers.
- Prisoner 1 checks the first bit (the rightmost bit) of every bottle.
- Prisoner 2 checks the second bit from the right of every bottle.
- And so on, with each prisoner checking a different bit position.

4. Binary Search Iterations:

- Each prisoner only drinks from bottles where their assigned bit position is set to '1' in the binary representation.
- For example, if a prisoner is assigned to the third bit position, they'll only drink from bottles where the third bit from the right is '1' (like 010, 011, 110, etc.).
- This way, each prisoner tests a specific set of bottles based on their assigned position in the binary representation.

5. Observing Deaths:

- The next day, we note down which prisoners have died.

- Since each prisoner only drank from bottles where their assigned bit was '1', we can determine which bits are '1' in the binary representation of the poisoned bottle numbers.

6. Identifying Poisonous Bottles:

- By combining the information from all the dead prisoners, we can determine which bottles have '1' in the positions where prisoners died.

- These bottles are the poisonous ones.

Advantages of This Approach:

1. Minimal Resource Usage: We only need 10 prisoners, regardless of how many bottles there are (as long as it's less than 1024).

2. Efficient Testing: Each prisoner tests a specific subset of bottles, reducing the overall testing time.

3. Simple and Scalable:This method is easy to understand and can be scaled up or down depending on the number of bottles.

Conclusion:

This approach is the best and most efficient because it optimally uses binary representation and assigns prisoners to test specific bits, minimizing the number of prisoners needed and ensuring a systematic and quick identification of the poisonous bottles.

PROBLEM -2

Approach Explanation:

1. Setup: We start by randomly creating details for 20 links and 20 traffic types. Each link has a latency (how long data takes to travel) and a bandwidth (how much data can be transmitted). Each traffic type has a maximum jitter limit (how much latency variation it can tolerate).

2. Calculate Jitter: We define a function to calculate the jitter for a given set of links. Jitter is just the difference between the longest and shortest latency among the links in a set. This helps us understand how much the data's travel time can vary within that set of links.

3. Find Maximum Bandwidth: We create a function that looks at all possible combinations of links for each traffic type. This means we check every possible way data could flow through different links for each type of traffic.

4. Check Jitter Limits: For each combination of links, we calculate the jitter using the method we defined earlier. Then we compare this jitter with the maximum jitter limit set for that particular type of traffic. If the calculated jitter is less than or equal to the limit, it means the data can flow smoothly without exceeding the allowed jitter.

5. Optimize Bandwidth Allocation:While checking combinations and jitter limits, we also keep track of the maximum total bandwidth that can be allocated for each traffic type. This is important because we want to allocate as much bandwidth as possible without violating the jitter constraints.

6. Output Results: Finally, we print out the maximum bandwidth allocation that meets the jitter requirements for each type of traffic. This tells us how much data can be transmitted efficiently for each traffic type considering the network's latency characteristics and the jitter limits.

Code :-

```
import random

links = [(random.randint(100, 500), random.randint(10, 100)) for _ in range(20)]

traffic_jitters = [random.randint(50, 300) for _ in range(20)]

def calculate_jitter(link_set):
    latencies = [link[0] for link in link_set]
    return max(latencies) - min(latencies)


def find_max_bandwidth(links, traffic_jitters):
    max_bandwidths = []
```

```
    for jitter_limit in traffic_jitters:
        max_bandwidth = 0

        link_combinations = []
        for r in range(1, len(links) + 1):
            link_combinations.extend(itertools.combinations(links, r))

        for link_set in link_combinations:
            jitter = calculate_jitter(link_set)
            if jitter <= jitter_limit:
                total_bandwidth = sum(link[1] for link in link_set)
                max_bandwidth = max(max_bandwidth, total_bandwidth)

        max_bandwidths.append(max_bandwidth)

    return max_bandwidths

max_bandwidths = find_max_bandwidth(links, traffic_jitters)
for i, max_bandwidth in enumerate(max_bandwidths):
    print(f"Traffic Type {i+1}: Max Bandwidth = {max_bandwidth}")
```

In this code:

- We first generate random latencies and bandwidths for the links, and random jitter limits for the traffic types.

- We define a function `calculate_jitter` that calculates the jitter for a given set of links.

- The `find_max_bandwidth` function finds the maximum bandwidth allocation for each traffic type while respecting the jitter limits.

- Finally, we print out the results showing the maximum bandwidth that can be allocated for each traffic type.