FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

DEPARTMENT OF COMMUNICATION TECHNOLOGY AND NETWORK

CSC4202 - 5:

DESIGN AND ALGORITHM ANALYSIS

GROUP PROJECT

REPORT ALLOCATING HOSPITAL RESOURCES DURING A PANDEMIC

LECTURER'S NAME: DR NUR ARZILAWATI BINTI MD YUNUS

NAME	MATRIC NO.
IZZATUL SYAIRAH BINTI IBRAHIM	210196
SITI KHADIJAH BINTI MOHD HAFIZ	211147
NUR ADIBAH BINTI SAMSUL AZMAN	212225

TABLE OF CONTENTS

Github Link: https://github.com/izzatulsyairah/csc4202

1. Scenario

   During the height of a pandemic, hospitals are struggling with a surge of patients requiring critical care. Resources such as ICU beds, ventilators, and specialized medical staff are in short supply. Each patient's condition varies, ranging from mild to severe, and their likelihood of survival with or without intensive treatment varies accordingly.

   The hospital administration has established a triage committee to allocate these scarce resources efficiently. The committee must consider multiple factors:

   a. **Severity of Condition**: Patients are categorized based on the severity of their illness:
      i. Category A: Mild symptoms, low immediate risk of deterioration.
      ii. Category B: Moderate symptoms, significant risk of deterioration without intervention.
      iii. Category C: Severe symptoms, high immediate risk of death without intervention.
   b. **Survival Probability**: Each patient has a probability of survival with and without intensive care:
      i. With ICU care: High, Moderate, or Low survival probability.
      ii. Without ICU care: Very Low, Low, or Moderate survival probability.
      iii. Without ICU care: Very Low, Low, or Moderate survival probability.
   c. **Resource Availability**:
      i. Limited number of ICU beds.
      ii. Limited number of ventilators.
      iii. Limited availability of specialized medical staff.
   d. **Time Sensitivity**: The condition of patients may worsen over time, necessitating timely decisions.

   The goal is to develop an optimal allocation strategy that maximizes the overall survival rate of patients. The committee must decide which patients receive ICU beds and ventilators, taking into account the dynamic nature of the pandemic and the constantly changing availability of resources.

2. **Why is scenario importance ?**

a. **Maximizing Survival Rates**: The primary importance of finding an optimal solution is to maximize the number of lives saved. By efficiently allocating limited resources to those patients who will benefit most, the overall survival rate can be significantly increased.
b. **Ethical Considerations**: In a pandemic, ethical dilemmas arise when deciding who gets access to life-saving treatments. An optimal solution provides a fair and systematic approach to resource allocation, reducing the emotional and moral burden on medical staff and ensuring that decisions are made based on objective criteria rather than ad-hoc judgments.
c. **Resource Utilization**: Hospitals have a limited capacity to handle critically ill patients. Optimal allocation ensures that ICU beds, ventilators, and medical staff are used in the most efficient manner, preventing wastage of resources and ensuring that they are available for the most critical cases.
d. **Dynamic Response to Changing Conditions**: The pandemic situation is fluid, with patient inflow rates and resource availability changing daily. An optimal allocation strategy allows for adaptive decision-making, ensuring that the allocation of resources can be recalibrated as the situations evolves.
e. **Reducing Overload and Burnout**: Efficient resource allocation helps manage the workload of medical staff, preventing burnout and maintaining the quality of care. This is crucial for the long-term sustainability of healthcare services during a prolonged crisis.
f. **Public Trust and Compliance**: Transparent and optimal allocation strategies foster public trust in the healthcare system. When the public perceives that resources are allocated fairly and efficiently, they are more likely to comply with public health measures and cooperate with healthcare providers.

3. Algorithm Suitability Reviews

3.1 Sorting Algorithms

Sorting algorithms are efficient for ordering elements and provide a simple and clear sequence for processing tasks. However, they are not directly applicable for resource allocation problems like the one described in the scenario. Sorting can arrange patients by severity or survival probability, but it does not address the dynamic allocation of resources or the changing conditions of patients over time. Hence, sorting algorithms are not suitable for solving this problem.

3.2 Divide and Conquer (DAC) Algorithms

Divide and Conquer (DAC) algorithms break down complex problems into simpler, independent subproblems, solve each subproblem separately, and then combine their solutions. This approach works well for problems where subproblems are independent and can be solved recursively. However, in the context of resource allocation, subproblems are not independent because the allocation of resources to one patient affects the availability of resources for others. Furthermore, DAC does not efficiently handle the dynamic nature of patient conditions and resource availability, making it less suitable for this problem.

3.3 Dynamic Programming (DP) Algorithms

Dynamic Programming (DP) is ideal for problems with overlapping subproblems and an optimal substructure, both of which are characteristics of the resource allocation problem. DP efficiently stores solutions to subproblems to avoid redundant calculations and can handle dynamic changes in patient conditions and resource availability. The challenge with DP lies in defining states and recurrence relations, but once established, it provides an optimal and systematic approach to maximize patient survival rates. DP is well-suited for this problem because it can adapt to changes over time and consider the probabilistic nature of patient survival.

3.4 Greedy Algorithms

Greedy algorithms make locally optimal choices at each step with the hope of finding a global optimum. They are simple and often faster than DP due to less computational overhead. However, greedy algorithms are not always optimal for complex problems like dynamic resource allocation. In this context, making a locally optimal choice

(e.g., giving resources to the patient with the highest immediate survival probability) may lead to suboptimal overall outcomes because it does not account for future needs and changing conditions. Thus, greedy algorithms are less effective for this problem.

3.5 Graph Algorithms

Graph algorithms are suitable for problems that can be modeled as networks, such as flow networks. They efficiently handle connectivity and pathfinding, and can represent different states and transitions. However, modeling the dynamic and probabilistic nature of resource allocation and patient conditions as a graph can be complex. While they offer flexibility in state representation, graph algorithms may not capture the temporal dynamics and probabilistic outcomes as effectively as DP. Therefore, they are not the best fit for this specific problem.

4. Design The Algorithm To Solve The Problem

Dynamic Programming (DP) is chosen for this problem due to its effectiveness in handling multi-faceted and time-dependent optimization problems. DP systematically breaks down the problem into simpler subproblems, solving each subproblem only once and storing its solution, which makes it efficient in terms of both time and space.

Dynamic Programming (DP) effectively addresses the complexity of resource allocation during a pandemic by handling multiple constraints such as limited ICU beds, ventilators, and varying patient conditions. Its ability to optimize resource use over time allows for real-time adjustments based on evolving patient needs and resource availability, ensuring timely and efficient responses. By exploring all possible allocation scenarios, DP provides an optimal solution that maximizes overall survival rates, making it highly effective in managing scarce medical resources during critical situations.

4.1 Algorithm Design

4.1.1 State Representation

The algorithm uses a DP table to represent the state of resource allocation at any given time. Each state includes:

Time (t): Represents the day or time step.
Resources (r): Represents the available ICU beds, ventilators, and medical staff.
DP Table: dp[t][beds][vents] stores the maximum achievable benefit at time t with a certain number of ICU beds and ventilators available.

4.1.2 Recurrence Relation

The core of the DP approach is the recurrence relation, which calculates the maximum benefit of allocating resources to patients by considering both the immediate and future benefits.The recurrence relation captures the essence of dynamic resource allocation by balancing immediate benefits with future states. Each state (t, beds, vents) is updated by considering the maximum benefit achievable by allocating resources to any patient.

Dp [t] [beds] [vents] = max (Benefit (t,beds,vents,allocation) + dp [t−1] [remainingBeds] [remainingVents])

Benefit Function: Computes the survival benefit of allocating resources to a patient.

Transition: Updates the DP table by considering the immediate allocation and the state resulting from this allocation.

4.1.3 Optimization Function

To extract the optimal benefit, the algorithm considers the maximum value from the DP table at the final time step. The DP approach iteratively updates the states and finally extracts the maximum achievable survival benefit by considering all possible resource states at the final time step. This ensures that the solution is optimized not only for immediate gains but also for long-term benefits over multiple time steps.

Maximum Benefit = max (dp [maxTime] [beds] [vents])

4.2 Data type

Entiti	Attribute	Data type
Patient	Id Severity neededBeds neededVents SurvivalWithICU survivalWithoutICU	Int Enum Int Int Float float
ResourceState	Beds vents	Int int
Time	-	int
Benefits	-	float
Dp Table	Structure: dp[time][beds][vents]	float

4.3 Objective Function

The **objective function** aims to maximize the overall survival benefit of patients over the given time period. This benefit is defined as the difference between survival probabilities with and without ICU care for the allocated patients.

Objective Function:

Maximize $\sum$ t=1 maxTime $\sum p \in$ patients Benefit(p,t)

Where:

Benefit(p,t) = survivalWithICU(p) − survivalWithoutICU(p)

4.4 Constraints

4.4.1 Resource Constraints

- **ICU Beds**: The number of ICU beds allocated to patients cannot exceed the available beds at any time.
- **Ventilators**: The number of ventilators allocated to patients cannot exceed the available ventilators at any time.

Formulation:

$\sum p \in patientsneededBeds(p) \leq beds(t)$

$\sum p \in patientsneededVents(p) \leq vents(t)$

4.4.2 Time Constraints

- Allocation decisions are made dynamically at each time step.
- Patients' conditions and resource availability may change over time, requiring continuous updates.

Formulation:

$time \in \{1,2,\ldots,maxTime\}$

4.5 Requirement and Constraint

Objective Requirements

- Maximize total survival benefit by optimal allocation of ICU beds and ventilators.
- Adjust allocations dynamically based on changing patient conditions and resource availability.

Space Complexity

- The DP table requires space proportional to the product of the number of time steps, ICU beds, and ventilators.
- **Space Complexity**: O(maxTime×beds×vents)O(\text{maxTime} \times \text{beds} \times \text{vents})O(maxTime×beds×vents)

Time Complexity

- For each time step, iterate over all resource states and patients to update the DP table.
- **Time Complexity**: O(maxTime×beds×vents×number of patients)

Value Constraints

- Survival probabilities should be in the range [0, 1].
- The sum of resource requirements at any time step should not exceed available resources.

4.6 Illustration Problem

Scenario Parameters

- Initial Resources: 2 ICU beds, 1 ventilator.
- Patients:
  - Patient A: Needs 1 ICU bed, 0 ventilators, Benefit: 5.
  - Patient B: Needs 1 ICU bed, 1 ventilator, Benefit: 10.
  - Patient C: Needs 2 ICU beds, 1 ventilator, Benefit: 15.
- Time Steps: 3 days.

The DP table is initialized with zeros, representing no benefit accumulated at the start. The table's dimensions are defined by the number of time steps, available ICU beds, and ventilators.

Day 1

On the first day, resources are allocated to patients by evaluating both immediate and potential future benefits. The DP table is updated by comparing current allocation benefits with previously calculated values for each state of ICU beds and ventilators.

DAY 1	AVAILABLE BEDS	AVAILABLE VENTS	POSSIBLE ALLOCATION	BENEFITS
Initial State	2	1	-	0
Allocate to A	1	1	Patient A	5
Allocate to B	1	0	Patient B	10

Day 2

On the second day, the process is repeated, considering updated states from the previous day. Each allocation is evaluated for its impact on both immediate and future benefits.

DAY 2	AVAILABLE BEDS	AVAILABLE VENTS	POSSIBLE ALLOCATION	BENEFITS
Previous State	1	1	Allocate to A	5
Allocate to B	0	0	Allocate to B	15 (5+10)

Day 3

On the final day, the algorithm determines the maximum achievable benefit by evaluating all possible resource allocations and extracting the optimal one.

DAY 3	AVAILABLE BEDS	AVAILABLE VENTS	POSSIBLE ALLOCATION	BENEFITS
Previous State	0	0	-	15

Extract Maximum Benefit

To find the maximum benefit, inspect the DP table at dp[maxTime][beds][vents]. In this example, after 3 days, the maximum benefit is 15, achieved by optimally allocating resources to patients based on the conditions at each time step.

Summary

- Day 1: Allocating to Patient B provides a benefit of 10.
- Day 2: Allocating to Patient A next maximizes the cumulative benefit to 15.
- Day 3: No more resources are available, and the final benefit remains 15.

The DP algorithm ensures resources are allocated optimally over time, balancing immediate and future benefits, and adjusting to changes in patient needs and resource availability. This approach maximizes overall survival benefit in a dynamically changing environment.

4.7 Algorithm Paradigm

Optimal Substructure: The problem can be divided into smaller subproblems that can be solved independently and combined to solve the larger problem. For our scenario, the optimal allocation of resources for the entire period can be derived from the optimal allocations of preceding time steps.

Overlapping Subproblems: The subproblems share sub sub problems, meaning the same states (combinations of available resources) recur over different time steps. DP avoids redundant calculations by storing the results of these subproblems.

State Representation: In our problem, a state can be represented by the current time, the number of available ICU beds, and the number of available ventilators. The DP table records the maximum benefit (patient survival probability) achievable for each state.

Recurrence Relation: The core of the DP approach involves finding a recurrence relation that defines the optimal solution of a problem in terms of the optimal solutions of its subproblems. For our scenario, it calculates the maximum benefit by considering both the immediate and future benefits of resource allocation to different patients.

4.8 Pseudocode

```
Algorithm AllocateResources {
   Input: maxTime, initialResources, patients, BenefitFunction
   Output: MaximumBenefit

   // Initialize DP table
   Define dp as a 3-dimensional array [maxTime + 1][beds + 1][vents + 1] all set to
0

   // Iterate over each time step
   For t from 1 to maxTime {
      For each state of ICU beds b from 0 to beds {
         For each state of ventilators v from 0 to vents {
            For each patient p in patients {
               Define neededBeds = p.getNeededBeds()
               Define neededVents = p.getNeededVents()
               If b >= neededBeds and v >= neededVents {
                  Define immediateBenefit = BenefitFunction(p, t)
                  Define remainingBeds = b - neededBeds
                  Define remainingVents = v - neededVents
                  dp[t][b][v] = max(dp[t][b][v], immediateBenefit + dp[t-
1][remainingBeds][remainingVents])
               }
            }
         }
      }
   }

   // Extract maximum benefit
   Define MaximumBenefit = 0
   For each state of ICU beds b from 0 to beds {
      For each state of ventilators v from 0 to vents {
         MaximumBenefit = max(MaximumBenefit, dp[maxTime][b][v])
      }
   }

   Return MaximumBenefit
}
```

```
Function BenefitFunction(patient, time) {
    Define survivalWithICU = patient.getSurvivalWithICU()
    Define survivalWithoutICU = patient.getSurvivalWithoutICU()
    Define benefit = survivalWithICU - survivalWithoutICU
    Return benefit
}
```

5. Algorithm Specification

The chosen dynamic programming (DP) approach is particularly suitable for handling complex, multifaceted problems such as resource allocation during a pandemic. DP efficiently manages time-dependent optimization by breaking down the problem into simpler subproblems, solving each subproblem once, and storing the solutions. This method ensures both time and space efficiency, crucial during critical situations when timely and efficient resource allocation can save lives.

5.1 Input and Output

1. **maxTime**: The total number of time steps or days over which the resource allocation decisions will be made.
2. **resources**: The initial quantities of ICU beds and ventilators available.
3. **patients**: A list of patients, each characterized by several attributes: ID, severity of condition, required ICU beds and ventilators, survival probability with ICU care, and survival probability without ICU care.
4. **BenefitFunction**: A function to calculate the immediate benefit derived from allocating resources to a particular patient.

Output:

The algorithm aims to compute MaximumBenefit, which represents the highest achievable overall survival benefit from the optimal allocation of available resources.

The algorithm uses a DP table (dp) to represent the state of resource allocation at any given time. Each state is defined by:

- t: The current time step.
- beds: The number of available ICU beds.
- vents: The number of available ventilators.
  The DP table entry dp[t][beds][vents] stores the maximum achievable benefit at time step t given beds ICU beds and vents ventilators.

5.2 Recurrence Relation

The core of the DP approach is its recurrence relation, which calculates the maximum benefit by considering both immediate and future benefits of resource allocation:

$$dp[t][beds][vents] = max\{Benefit(t,beds,vents,allocation)+$$

$$dp[t-1][remainingBeds][remainingVents]\}$$

Equation Components:

1. **Dp[t][beds][vents]** represents the maximum achievable benefit at time step t with beds ICU beds and vents ventilators available.
2. **Benefit(t, beds, vents, allocation)** is the immediate benefit obtained from allocating a certain number of ICU beds and ventilators to patients at time t. The benefit is typically calculated as the difference in survival probabilities with and without ICU care for the allocated patients. The function takes into account the current time step and the resources being allocated (i.e., the number of beds and ventilators). It quantifies the immediate improvement in patient survival due to the resource allocation.
3. **dp[t-1][remainingBeds][remainingVents]** represents the maximum benefit that can be achieved in the previous time step (t-1) given the remaining resources after the current allocation. 'remainingBeds' and 'remainingVents' are the number of ICU beds and ventilators left after making the current allocation to patients.
4. **max {...}**: The max function ensures that the algorithm selects the allocation that provides the highest total benefit, combining both the immediate benefit from the current allocation and the maximum possible benefit from future time steps (as captured in the DP table from the previous step).

To extract the optimal benefit, the algorithm evaluates the maximum value from the DP table at the final time step:

$$MaximumBenefit = max\{dp[maxTime][beds][vents]\}$$

This step ensures the solution is optimized not just for immediate gains but also for long-term benefits across multiple time steps.

6. Program Java Language

The goal of this code is to find the maximum overall survival benefit by optimally allocating ICU beds and ventilators to patients over a specified number of time steps during a pandemic scenario. The survival benefit is defined as the difference in survival probabilities between receiving ICU care and not receiving ICU care.

a. Dynamic

```
import java.util.ArrayList;
import java.util.List;

class Patient {
    int id;
    String category;
    int neededBeds;
    int neededVents;
    float survivalWithICU;
    float survivalWithoutICU;

    // Constructor to initialize patient attributes
    public Patient(int id, String category, int neededBeds, int neededVents, float
survivalWithICU, float survivalWithoutICU) {
        this.id = id;
        this.category = category;
        this.neededBeds = neededBeds;
        this.neededVents = neededVents;
        this.survivalWithICU = survivalWithICU;
        this.survivalWithoutICU = survivalWithoutICU;
    }
}

public class ResourceAllocationDP {

    // Function to allocate resources and maximize survival gain
    public static float allocateResources(int maxTime, int initialBeds, int
initialVents, List<Patient> patients) {
        // Initialize DP table
        float[][][] dp = new float[maxTime + 1][initialBeds + 1][initialVents + 1];

        // Iterate over each time step
        for (int t = 1; t <= maxTime; t++) {
            for (int beds = 0; beds <= initialBeds; beds++) {
```

```
            for (int vents = 0; vents <= initialVents; vents++) {
                // Iterate over each patient
                for (Patient patient : patients) {
                    int neededBeds = patient.neededBeds;
                    int neededVents = patient.neededVents;
                    // Calculate the survival gain of giving ICU care to this patient
                    float survivalGain = patient.survivalWithICU -
patient.survivalWithoutICU;

                    // Check if current resources are sufficient to allocate to this
patient
                    if (beds >= neededBeds && vents >= neededVents) {
                        // Calculate remaining resources after allocating to this patient
                        int remainingBeds = beds - neededBeds;
                        int remainingVents = vents - neededVents;

                        // Update DP table to maximize survival gain
                        dp[t][beds][vents] = Math.max(dp[t][beds][vents],
                                survivalGain + dp[t - 1][remainingBeds][remainingVents]);
                    }
                }
            }
        }

        // Extract maximum survival gain at the final time step
        float maxSurvivalGain = 0;
        for (int beds = 0; beds <= initialBeds; beds++) {
            for (int vents = 0; vents <= initialVents; vents++) {
                maxSurvivalGain = Math.max(maxSurvivalGain,
dp[maxTime][beds][vents]);
            }
        }

        // Return the maximum survival gain achievable
        return maxSurvivalGain;
    }

    // Example usage of the allocateResources function
    public static void main(String[] args) {
        // Example parameters
        int maxTime = 3;
        int initialBeds = 2;
        int initialVents = 1;
```

```
        // Creating example patients
        List<Patient> patients = new ArrayList<>();
        patients.add(new Patient(1, "A", 1, 0, 0.9f, 0.1f));  // Example patient with id
1, category A, needs 1 bed, 0 vents, survival probabilities
        patients.add(new Patient(2, "B", 1, 1, 0.8f, 0.2f));  // Example patient with id
2, category B, needs 1 bed, 1 vent, survival probabilities
        patients.add(new Patient(3, "C", 2, 1, 0.7f, 0.3f));  // Example patient with id
3, category C, needs 2 beds, 1 vent, survival probabilities

        // Compute maximum survival gain
        float maxSurvivalGain = allocateResources(maxTime, initialBeds,
initialVents, patients);

        // Output the maximum survival gain obtained
        System.out.println("Best Possible Survival : " + maxSurvivalGain);
    }
}
```

7. Algorithm's Correctness Analysis With Time Complexity

7.1 Correctness Analysis

7.1.1 Initialization

The algorithm initializes the DP table such that

$$dp[0][beds][vents] = 0$$

for all combinations of beds and ventilators. This correctly represents the initial state where no resources have been allocated, and thus no benefit has been accrued. Initialization ensures that the base case is covered, which is essential for building up the solution iteratively.

7.1.2 Recurrence Relation

$$dp[t][beds][vents] = max\{Benefit(t,beds,vents,allocation) + dp[t-1][remainingBeds][remainingVents]\}$$

The recurrence relation is derived from the principle of optimality. It ensures that the decision at each time step maximizes the overall benefit by considering both immediate and future gains.
By iterating over all possible resource allocations for each patient and updating the DP table accordingly, the algorithm systematically explores all potential paths to find the optimal solution.
The recurrence relation correctly propagates the benefits of each allocation decision through time, ensuring that the solution is optimal for the entire period rather than just for individual time steps.

7.1.3 Optimization Function

$$MaximumBenefit = max\{dp[maxTime][beds][vents]\}$$

The optimization function extracts the maximum benefit achievable at the final time step across all resource states. This step ensures that the algorithm captures the best possible outcome considering all time steps and all resource configurations.

7.2 Time Complexity Analysis

The time complexity of the algorithm depends on the number of time steps, the number of resources (ICU beds and ventilators), and the number of patients. Let's break down the complexity step-by-step:

- State Space
    - The DP table has dimensions maxTime, maxBeds, and maxVents. This means the state space is proportional to $O(maxTime \times maxBeds \times maxVents)$.
- Transition Complexity
    - For each state (t,beds,vents), the algorithm iterates over all patients to compute the benefit of allocating resources. For each patient, it considers all feasible allocations of ICU beds and ventilators.

    - Let P be the number of patients. For each state, the computation involves checking the benefit for each patient, leading to a transition complexity of $O(P)$.
- Total Complexity
    - Considering all states and transitions, the overall time complexity is $O(maxTime \times maxBeds \times maxVents \times P)$.

7.2.1 Best Case Complexity

In the best case, if the number of patients is minimal or the resource constraints are very relaxed, the complexity is reduced significantly. The best case still involves iterating through the entire DP table, resulting in:

$$O(maxTime \times maxBeds \times maxVents)$$

7.2.1 Average Case Complexity

In a typical scenario where the resources and patients are moderate in number, the complexity remains as stated above:

$$O(maxTime \times maxBeds \times maxVents \times P)$$

7.2.3 Worst Case Complexity

In the worst case, where the number of patients and resources is maximal, the complexity remains at its highest bound:

$$O(maxTime \times maxBeds \times maxVents \times P)$$

7.3 Space Complexity

The DP table stores values for each combination of time steps, beds, and ventilators. Thus, the space complexity is:

$$O(maxTime \times maxBeds \times maxVents)$$

This is because each entry dp[t][beds][vents] needs to store a floating-point value representing the maximum benefit.

In conclusion, the dynamic programming approach effectively addresses the challenges of resource allocation during a pandemic. By optimizing the use of limited resources, the algorithm ensures the highest possible survival rates, upholds ethical standards, and

maintains the sustainability of healthcare services. This systematic and adaptive method fosters public trust and enhances the overall resilience of the healthcare system in the face of a crisis.