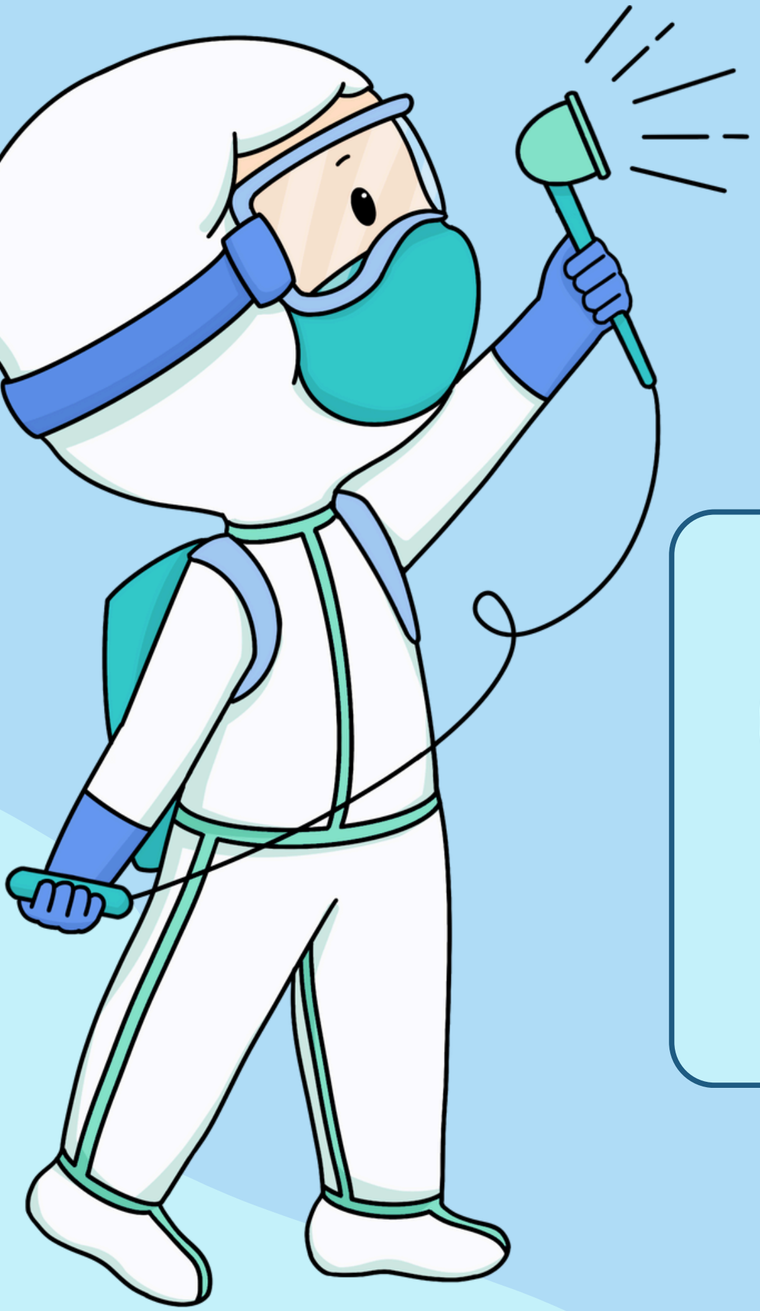
ALLOCATING HOSPITAL RESOURCES DURING A PANDEMIC

CSC4202 - 5



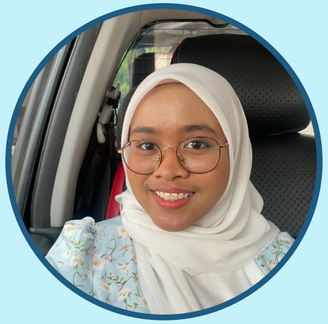
Our team

SITI KHADIJAH

211147

IZZATUL SYAIRAH

210196



NUR ADIBAH

212225

SCENARIO

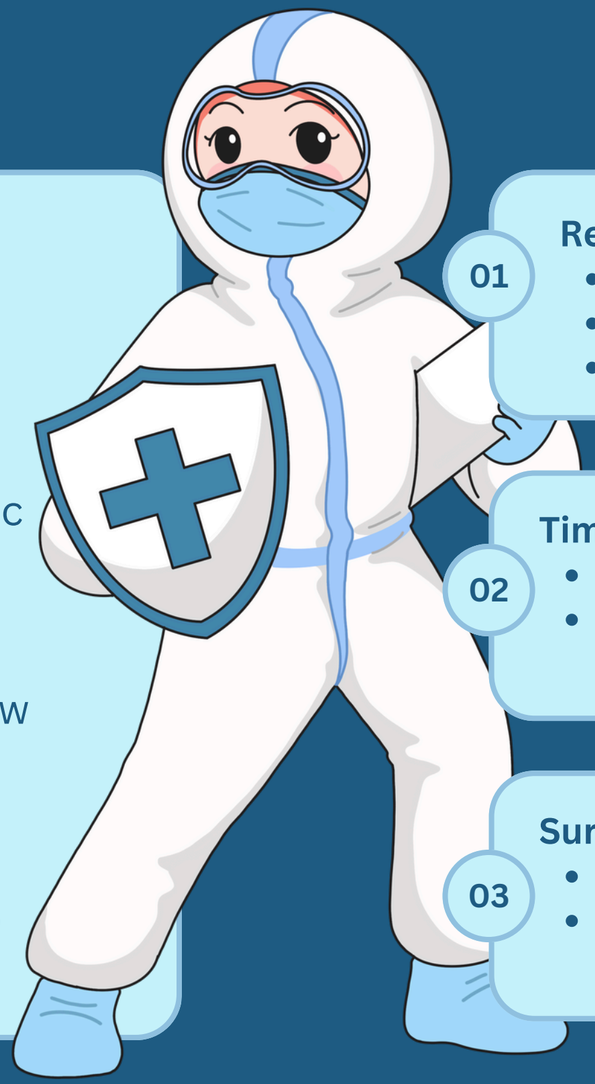
During the height of a pandemic, hospitals are struggling with a surge of patients requiring critical care. Resources such as ICU beds, ventilators, and specialized medical staff are in short supply. Each patient’s condition varies, ranging from mild to severe, and their likelihood of survival with or without intensive treatment varies accordingly.
The hospital administration has established a triage committee to allocate these scarce resources efficiently.

Objective:

- Maximize overall survival rate.
- Optimal allocation strategy:
  - Balance needs with resources.
  - Adapt to dynamic pandemic conditions.

Patient Categorization:

- Category A: Mild symptoms, low risk.
- Category B: Moderate symptoms, significant risk.
- Category C: Severe symptoms, high risk.



01

Resource Availability:

- ICU Beds: Limited.
- Ventilators: Limited.
- Specialized Staff: Limited.

02

Time Sensitivity:

- Conditions worsen over time.
- Timely decisions essential.

03

Survival Probability:

- With ICU care: High, Moderate, Low.
- Without ICU care: Very Low, Low, Moderate.

WHY FINDING AN OPTIMAL SOLUTION FOR THIS SCENARIO IS IMPORTANT

Maximizing Survival Rates:

- Objective: Save the most lives.
- Efficient Allocation: Prioritize patients who will benefit the most, significantly increasing overall survival rates.

Ethical Considerations:

- Systematic Approach: Provide a fair and objective method for resource allocation.
- Reduce Burden: Lessens the emotional and moral strain on medical staff, ensuring decisions are based on clear criteria.

Resource Utilization:

- Prevent Wastage: Ensure ICU beds, ventilators, and medical staff are used efficiently.
- Critical Cases: Resources are available for the most critical patients.

Reducing Overload and Burnout:

- Manage Workload: Helps prevent burnout by managing the workload of medical staff.
- Sustainable Care: Maintains quality of care during prolonged crises.

Algorithm Design to solve problem (DP)

State Representation

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

Time (t): day or time step.

Resources (r): available ICU beds and ventilators.

DP Table: dp[t][beds][vents] stores the maximum achievable benefit at time t with a certain number of ICU beds and ventilators available.

Recurrence Relation

calculates the maximum benefit of allocating resources to patients by considering both the immediate and future benefits.

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.

Optimization Function

DP Table Examination: At the final time step, the algorithm examines the DP table to identify the maximum achievable benefit.

State Iteration: Throughout the process, the DP approach updates and tracks different resource states and their corresponding benefits.

Final Extraction: The maximum survival benefit is extracted by evaluating all possible resource states in the DP table at the final time step.

Problem Illustration

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.

Problem Illustration

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
Allocate to C	0	0	Patient C	15

Initial State: dp[0][2][1] = 0
Update A: dp[1][1][1] = max(dp[1][1][1], 5 + dp[0][2][1]) = 5
Update B: dp[1][1][0] = max(dp[1][1][0], 10 + dp[0][2][1]) = 10
Update C: dp[1][0][0] = max(dp[1][0][0], 15 + dp[0][2][1]) = 15

Problem Illustration

Day 2 (Initial state from Day 1)

- 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 A	1	1	Allocate to A	5
Allocate to B	0	0	Allocate to B	15 (5+10)
Previous State B	1	0	Allocate to B	10
Allocate to A	0	0	Allocate to A	15((10+5)

Update A : dp[2][0][0] = max(dp[2][0][0], 10 + dp[1][1][1]) = 15
Update B : dp[2][0][0] = max(dp[2][0][0], 5 + dp[1][1][0]) = 15

Problem Illustration

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

maximum benefit is dp[3][0][0] = 15

Problem Illustration

Extract maximum benefits

- 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.

pseudocode

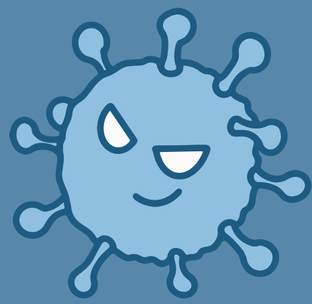
```
Algorithm AllocateResources {
  Input: maxTime, initialResources, patients, BenefitFunction
  Output: MaximumBenefit

  // 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
}
```

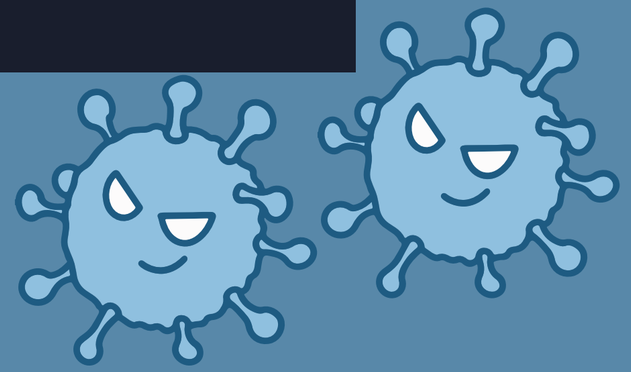
```
import java.util.ArrayList;
import java.util.List;
class Patient {
    int id;
    int severity;
    int neededBeds;
    int neededVents;
    float survivalWithICU;
    float survivalWithoutICU;
    public Patient(int id, int severity, int neededBeds, int
neededVents, float survivalWithICU, float survivalWithoutICU) {
        this.id = id;
        this.severity = severity;
        this.neededBeds = neededBeds;
        this.neededVents = neededVents;
        this.survivalWithICU = survivalWithICU;
        this.survivalWithoutICU = survivalWithoutICU;
    }
    public int getNeededBeds() {
        return neededBeds;
    }
    public int getNeededVents() {
        return neededVents;
    }
    public float getSurvivalWithICU() {
        return survivalWithICU;
    }
    public float getSurvivalWithoutICU() {
        return survivalWithoutICU;
    }}

public class ResourceAllocator {
 public static float allocateResources(int maxTime, int beds, int
vents, List<Patient> patients) {
 float[][][] dp = new float[maxTime + 1][beds + 1][vents + 1];
        for (int t = 1; t <= maxTime; t++) {
            for (int b = 0; b <= beds; b++) {
                for (int v = 0; v <= vents; v++) {
                    for (Patient p : patients) {
                        int neededBeds = p.getNeededBeds();
                        int neededVents = p.getNeededVents();
                        if (b >= neededBeds && v >= neededVents) {
                            float immediateBenefit = benefitFunction(p);
                            int remainingBeds = b - neededBeds;
                            int remainingVents = v - neededVents;
                            dp[t][b][v] = Math.max(dp[t][b][v], immediateBenefit + dp[t - 1]
[remainingBeds][remainingVents]);
                        } }
                }
            }
        }
        float maximumBenefit = 0;
        for (int b = 0; b <= beds; b++) {
            for (int v = 0; v <= vents; v++) {
                maximumBenefit = Math.max(maximumBenefit, dp[maxTime][b][v]);
            }
        }
        return maximumBenefit;
    }

public static float benefitFunction(Patient patient) {
 return patient.getSurvivalWithICU() - patient.getSurvivalWithoutICU();
 }
 public static void main(String[] args) {
 int maxTime = 3; // Specifies the maximum time duration considered for resource
allocation decisions. In this case, it's set to 3 time units.
 int beds = 2;// Represents the total number of available ICU beds.
 int vents = 1; // Indicates the total number of available ventilators.
 List<Patient> patients = new ArrayList<>();
 patients.add(new Patient(1, 1, 1, 0, 0.9f, 0.5f));
 patients.add(new Patient(2, 2, 1, 1, 0.8f, 0.3f));
 patients.add(new Patient(3, 3, 2, 1, 0.7f, 0.2f));
 float result = allocateResources(maxTime, beds, vents, patients);
 System.out.println("Maximum Benefit: " + result);
 }
}
```

Output

Output

```
java -cp /tmp/1gc8YrOAGC/ResourceAllocator
Maximum Benefit: 0.9


=== Code Execution Successful ===
```





Correctness Analysis

Initialization

- **dp[0][beds][vents] = 0** for all combinations of beds and ventilators.
- Represents the initial state where no resources are allocated, hence no benefit is accrued.

Optimization Function

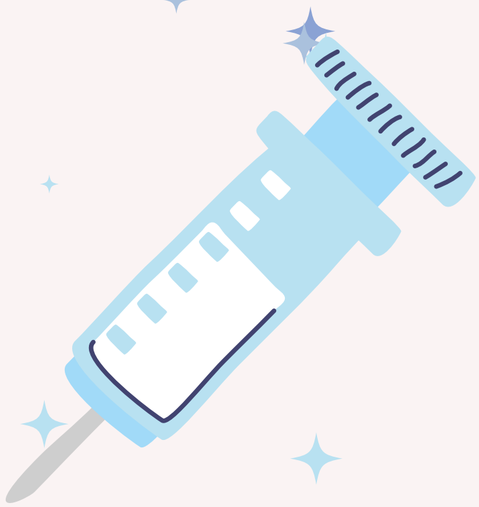
MaximumBenefit = max{dp[maxTime][beds][vents]}

- Captures the best possible outcome considering all time steps and resource configurations.

Recurrence Relation

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

- Ensures each decision maximizes overall benefit by considering both immediate and future gains.
- Iterates over all possible resource allocations for each patient, updating the DP table to find the optimal solution.
- Correctly propagates the benefits of each allocation decision through time.



Time Complexity Analysis

Best Case Complexity

Conditions: Minimal number of patients or very relaxed resource constraints.

Best Case Complexity:
O(maxTime×maxBeds×maxVents)

Worst Case Complexity

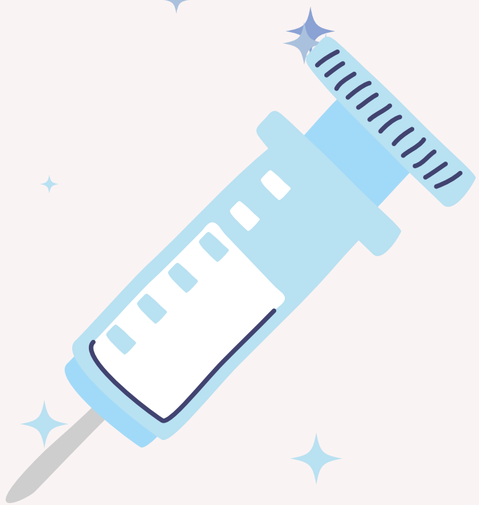
Conditions: Maximal number of patients and resources.

Worst Case Complexity:
O(maxTime×maxBeds×maxVents×P)

Average Case Complexity

Typical Scenario: Moderate number of resources and patients.

Average Case Complexity:
O(maxTime×maxBeds×maxVents×P)



Thank you for your attention