CSC4202 DESIGN AND ANALYSIS OF ALGORITHMS

CSC4202 GROUP PROJECT

GROUP NAME	**NULLCASE**
NAME	**HARDEEPAK KAUR MALL A/P JASWANT SINGH (225220)** **SWETHA A/P SUPPAIYAH (223009)** **AHNISHAA A/P GURULINGGAM (223590)** **VISHRUTI BALAKRISNA NAIDU (224912)**
SUBJECT	**CSC4202(G1)**
TEACHER	**PROF. MADYA DR. NORWATI MUSTAPHA**

i. Illustrate the Problem

Scenario:
Following a devastating earthquake in Sabah, many remote communities are isolated and in urgent need of critical supplies such as food, water, and medical kits. A disaster relief center coordinates the deliveries using trucks, each having a limited weight capacity.

Each item differs in:

- **Weight**
- **Priority value** (based on urgency and usefulness)

Objective:
Select the optimal combination of items to maximize the total priority value of items delivered, **without exceeding** the truck's weight capacity.

Why It Matters:

- Saves lives by ensuring efficient aid distribution
- Reduces wasted transportation effort and time
- Models real-world disaster logistics

ii. Explain the Algorithm Paradigm & Pseudocode

Chosen Paradigm: Dynamic Programming (0/1 Knapsack)

Each item may have limited stock (e.g., 3 medical kits). The algorithm handles this by duplicating the item in the expanded list based on available stock before applying dynamic programming.

Based on the selected region, items relevant to that region's crisis (e.g., Medical in Ranau) have their priority value increased by 2 before optimization.

Justification:

- **Overlapping Subproblems:** Best solutions for larger capacities reuse computations from smaller capacities
- **Optimal Substructure:** Optimal solution for a weight limit `w` and `i` items depends on optimal solutions for smaller `w` and fewer items

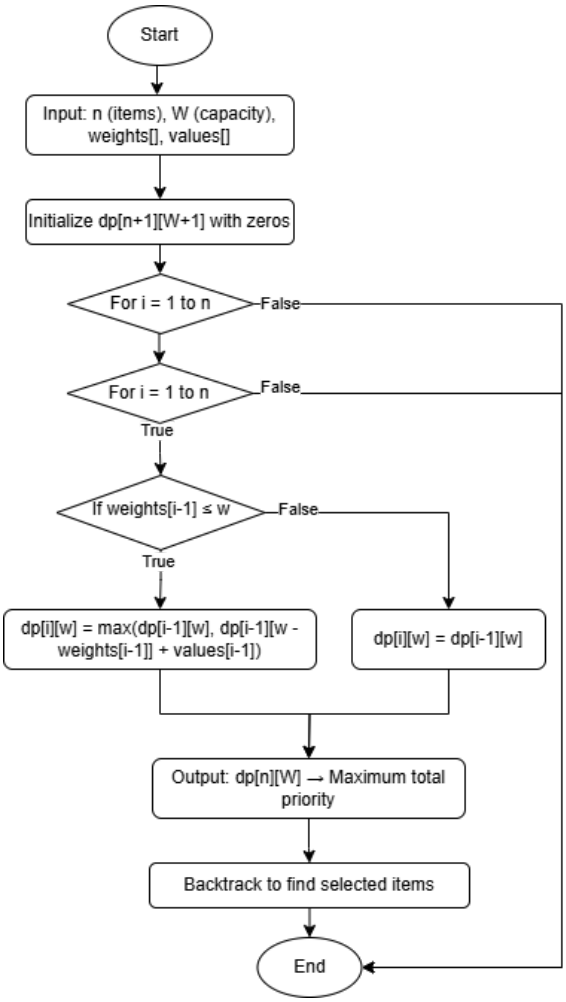
Recurrence Relation:

```
if weight[i] ≤ w:
   dp[i][w] = max(dp[i-1][w], dp[i-1][w - weight[i]] + value[i])
else:
   dp[i][w] = dp[i-1][w]
```

Pseudocode:

```
Input:
   n        ← number of items
   W        ← truck capacity
   weights[] ← array of item weights
   values[]  ← array of item priority values

Initialize dp[0..n][0..W] ← 0

For i from 1 to n:
   For w from 1 to W:
      If weights[i-1] ≤ w:
         dp[i][w] ← max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1])
      Else:
         dp[i][w] ← dp[i-1][w]

Return dp[n][W]  // maximum total priority value

// Backtracking
Set w ← W
For i from n down to 1:
   If dp[i][w] ≠ dp[i-1][w]:
      Print "Item", i, "is selected"
      w ← w - weights[i-1]
```

Flowchart:


Start
Input: n (items), W (capacity), weights[], values[]
Initialize dp[n+1][W+1] with zeros
For i = 1 to n
False
For i = 1 to n
False
True
If weights[i-1] ≤ w
False
True
dp[i][w] = max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1])
dp[i][w] = dp[i-1][w]
Output: dp[n][W] → Maximum total priority
Backtrack to find selected items
End

iii. Demonstrate the Program & Describe the Output

Execution Flow:

```
START APP

Initialize UI

// User interaction
Wait for user to select region
Wait for user to enter capacity

On "Optimize" button click:
   If input is invalid:
      Show error message
      Exit process
   EndIf

   Adjust item priorities based on selected region
   Expand items based on available stock

   // Optimization using 0/1 Knapsack (Dynamic Programming)
   result = RunKnapsack(items, capacity)

   selectedItems = BacktrackSelectedItems(result)

   Display selectedItems in outputArea

   If user clicks "Export":
      Save selectedItems to .txt file
   EndIf

END APP
```

Code:

```java
// File: EarthquakeReliefOptimizerFX.java
package Project;
import javafx.application.Application;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
```

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
public class EarthquakeReliefOptimizerFX extends Application {
   static class Item {
       String name;
       int weight;
       int priority;
       String category;
       int stock;
       Item(String name, int weight, int priority, String category, int
stock) {
           this.name = name;
           this.weight = weight;
           this.priority = priority;
           this.category = category;
           this.stock = stock;
       }
   }
   private final Item[] items = {
       new Item("Rice (5kg)", 5000, 8, "Food", 5),
       new Item("Bottled Water (1.5L x6)", 9000, 10, "Water", 3),
       new Item("Medical Kit", 3000, 10, "Medical", 3),
       new Item("Blanket", 4000, 6, "Shelter", 4),
       new Item("Canned Food (x10)", 6000, 7, "Food", 4),
       new Item("Infant Formula", 2000, 9, "Medical", 2),
       new Item("Hygiene Kit", 3500, 7, "Sanitation", 3),
       new Item("Flashlight + Batteries", 1500, 5, "Utility", 5)
   };
   @Override
   public void start(Stage primaryStage) {
       primaryStage.setTitle("Earthquake Relief Optimizer");
       Label title = new Label("Earthquake Relief Optimizer");
       title.setFont(Font.font("Segoe UI Semibold", 30));
       title.setTextFill(Color.web("#ffffff"));
       // Section: Available Items Title
       Label itemsTitle = new Label("Available Relief Items");
       itemsTitle.setFont(Font.font("Segoe UI", 18));
       itemsTitle.setTextFill(Color.web("#333"));
       // TableView for Items
       TableView<Item> itemTable = new TableView<>();
       itemTable.setPrefHeight(200);
       TableColumn<Item, String> nameCol = new TableColumn<>("Item Name");
       nameCol.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().name));
       nameCol.setPrefWidth(200);
       TableColumn<Item, Integer> weightCol = new TableColumn<>("Weight
(g)");
       weightCol.setCellValueFactory(data -> new
SimpleIntegerProperty(data.getValue().weight).asObject());
       weightCol.setPrefWidth(100);
       TableColumn<Item, Integer> priorityCol = new
TableColumn<>("Priority");
       priorityCol.setCellValueFactory(data -> new
SimpleIntegerProperty(data.getValue().priority).asObject());
       priorityCol.setPrefWidth(80);
       TableColumn<Item, String> categoryCol = new TableColumn<>("Category");
       categoryCol.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().category));
       categoryCol.setPrefWidth(100);
```

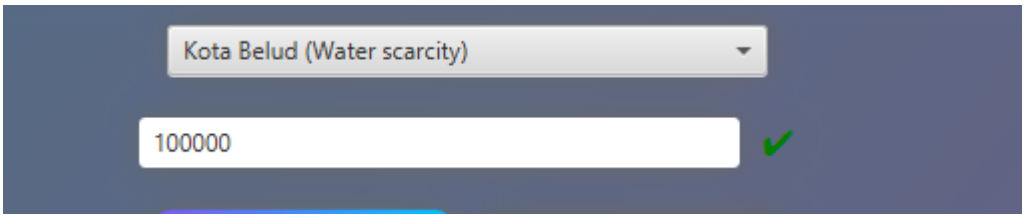
```
        TableColumn<Item, Integer> stockCol = new TableColumn<>("Stock");
        stockCol.setCellValueFactory(data -> new
SimpleIntegerProperty(data.getValue().stock).asObject());
        stockCol.setPrefWidth(60);
        itemTable.getColumns().addAll(nameCol, weightCol, priorityCol,
categoryCol, stockCol);
        itemTable.getItems().addAll(items);
        ComboBox<String> regionComboBox = new ComboBox<>();
        regionComboBox.getItems().addAll(
            "Ranau (Medical emergency)",
            "Kota Belud (Water scarcity)",
            "Kudat (Food shortage)"
        );
        regionComboBox.setPromptText("Select Affected Region");
        regionComboBox.setPrefWidth(300);
        TextField capacityField = new TextField();
        capacityField.setPromptText("Enter truck capacity in grams");
        capacityField.setPrefWidth(300);
        Label statusIcon = new Label("X");
        statusIcon.setFont(Font.font(18));
        statusIcon.setTextFill(Color.RED);
        capacityField.textProperty().addListener((obs, oldVal, newVal) -> {
            if (newVal.matches("\\d+")) {
                capacityField.setStyle("-fx-background-color: white;
-fx-border-radius: 10;");
                statusIcon.setText("✔");
                statusIcon.setTextFill(Color.GREEN);
            } else {
                statusIcon.setText("X");
                statusIcon.setTextFill(Color.RED);
            }
        });
        HBox inputRow = new HBox(10, capacityField, statusIcon);
        inputRow.setAlignment(Pos.CENTER);
        Button optimizeButton = new Button("Optimize");
        optimizeButton.setPrefWidth(150);
        optimizeButton.setStyle("-fx-background-color: linear-gradient(to
right, #7f5af0, #00c3ff); -fx-text-fill: white; -fx-font-size: 14px;
-fx-font-weight: bold; -fx-background-radius: 10;");
        Button exportButton = new Button("Export Result");
        exportButton.setPrefWidth(150);
        exportButton.setStyle("-fx-background-color: white; -fx-border-color:
#00c3ff; -fx-text-fill: #0077cc; -fx-font-weight: bold;
-fx-background-radius: 10;");
        HBox buttonRow = new HBox(15, optimizeButton, exportButton);
        buttonRow.setAlignment(Pos.CENTER);
        Label badge = new Label("Items Selected: 0");
        badge.setTextFill(Color.web("#222"));
        badge.setFont(Font.font("Segoe UI", 14));
        TextArea outputArea = new TextArea();
        outputArea.setEditable(false);
        outputArea.setWrapText(true);
        outputArea.setFont(Font.font("Consolas", 13));
        outputArea.setStyle("-fx-control-inner-background:
rgba(255,255,255,0.96);");
        VBox.setVgrow(outputArea, Priority.ALWAYS);
        VBox layout = new VBox(20, title, itemsTitle, itemTable,
regionComboBox, inputRow, buttonRow, badge, outputArea);
        layout.setPadding(new Insets(30));
        layout.setAlignment(Pos.TOP_CENTER);
```

```
        layout.setStyle("-fx-background-color: rgba(255,255,255,0.15);
-fx-background-radius: 20;");
        StackPane card = new StackPane(layout);
        card.setPadding(new Insets(30));
        card.setMaxWidth(880);
        card.setEffect(new DropShadow(20, Color.gray(0.3)));
        StackPane root = new StackPane(card);
        root.setStyle("-fx-background-color: linear-gradient(to bottom right,
#2b5876, #4e4376);");
        Scene scene = new Scene(root, 1000, 760);
        primaryStage.setScene(scene);
        primaryStage.show();
        optimizeButton.setOnAction(event -> {
            String regionText = regionComboBox.getValue();
            String capacityText = capacityField.getText();
            if (regionText == null || capacityText.isEmpty() ||
!capacityText.matches("\\d+")) {
                outputArea.setText("Please select a valid region and enter a
valid truck capacity.");
                badge.setText("Items Selected: 0");
                return;
            }
            int capacity = Integer.parseInt(capacityText);
            String region = regionText.split(" ")[0];
            String result = optimizeRelief(region, capacity, badge);
            outputArea.setText(result);
        });
        exportButton.setOnAction(event -> {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save Optimization Result");
            fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));
            File file = fileChooser.showSaveDialog(primaryStage);
            if (file != null) {
                try (FileWriter writer = new FileWriter(file)) {
                    writer.write(outputArea.getText());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
    }
    private String optimizeRelief(String region, int capacity, Label badge) {
        List<Item> itemList = new ArrayList<>();
        for (Item item : items) {
            int adjustedPriority = item.priority;
            if (region.equals("Ranau") && item.category.equals("Medical"))
adjustedPriority += 2;
            if (region.equals("Kota") && item.category.equals("Water"))
adjustedPriority += 2;
            if (region.equals("Kudat") && item.category.equals("Food"))
adjustedPriority += 2;
            itemList.add(new Item(item.name, item.weight, adjustedPriority,
item.category, item.stock));
        }
        List<Item> expanded = new ArrayList<>();
        for (Item item : itemList) {
            for (int i = 0; i < item.stock; i++) {
                expanded.add(new Item(item.name, item.weight, item.priority,
item.category, 1));
            }
```

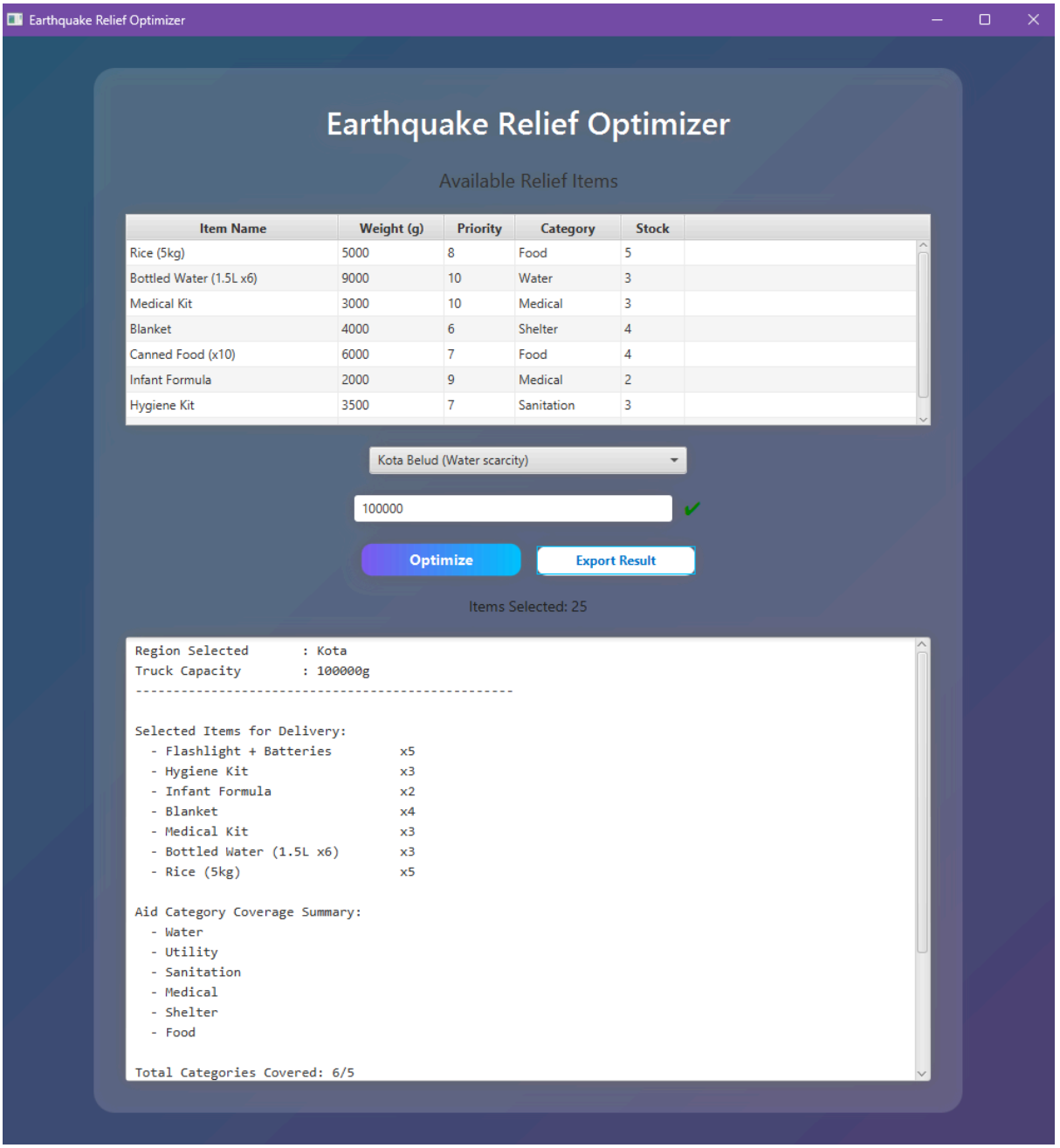
```
        }
        int n = expanded.size();
        int[] weights = new int[n];
        int[] values = new int[n];
        for (int i = 0; i < n; i++) {
            weights[i] = expanded.get(i).weight;
            values[i] = expanded.get(i).priority;
        }
        int[][] dp = new int[n + 1][capacity + 1];
        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= capacity; w++) {
                if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i
- 1]] + values[i - 1]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        Map<String, Integer> selectedItems = new LinkedHashMap<>();
        Map<String, String> itemCategories = new HashMap<>();
        int w = capacity;
        for (int i = n; i > 0 && w > 0; i--) {
            if (dp[i][w] != dp[i - 1][w]) {
                Item item = expanded.get(i - 1);
                selectedItems.put(item.name,
selectedItems.getOrDefault(item.name, 0) + 1);
                itemCategories.put(item.name, item.category);
                w -= item.weight;
            }
        }
        int totalCount =
selectedItems.values().stream().mapToInt(Integer::intValue).sum();
        badge.setText("Items Selected: " + totalCount);
        StringBuilder result = new StringBuilder();
        result.append("Region Selected       : ").append(region).append("\n");
        result.append("Truck Capacity        :
").append(capacity).append("g\n");

result.append("--------------------------------------------------\n\n");
        result.append("Selected Items for Delivery:\n");
        if (selectedItems.isEmpty()) {
            result.append("  - No items selected within given capacity.\n");
        } else {
            for (String itemName : selectedItems.keySet()) {
                result.append(String.format("  - %-30s x%d\n", itemName,
selectedItems.get(itemName)));
            }
        }
        result.append("\nAid Category Coverage Summary:\n");
        Set<String> coveredCategories = new HashSet<>();
        for (String itemName : selectedItems.keySet()) {
            coveredCategories.add(itemCategories.get(itemName));
        }
        for (String category : coveredCategories) {
            result.append("  - ").append(category).append("\n");
        }
        result.append(String.format("\nTotal Categories Covered: %d/5\n",
coveredCategories.size()));
        result.append("\nImpact Estimate for ").append(region).append(":\n");
```

```
        result.append(coveredCategories.contains("Food") ? "  - Basic food
needs partially covered\n" : "  - Food shortage remains unaddressed\n");
        result.append(coveredCategories.contains("Water") ? "  - Water supply
ensured\n" : "  - No water aid: hydration risk\n");
        result.append(coveredCategories.contains("Medical") ? "  - Medical
support included\n" : "  - No medical supplies: risk to injured\n");
        result.append(coveredCategories.contains("Sanitation") ? "  - Hygiene
kits included\n" : "  - Hygiene kits missing: risk of disease\n");
        result.append(coveredCategories.contains("Shelter") ? "  - Blankets
provided for shelter\n" : "  - Blankets missing: exposure risk\n");
        return result.toString();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Sample Input:



Output:


Earthquake Relief Optimizer
Earthquake Relief Optimizer
Available Relief Items
Item Name | Weight (g) | Priority | Category | Stock
Rice (5kg) | 5000 | 8 | Food | 5
Bottled Water (1.5L x6) | 9000 | 10 | Water | 3
Medical Kit | 3000 | 10 | Medical | 3
Blanket | 4000 | 6 | Shelter | 4
Canned Food (x10) | 6000 | 7 | Food | 4
Infant Formula | 2000 | 9 | Medical | 2
Hygiene Kit | 3500 | 7 | Sanitation | 3
Kota Belud (Water scarcity)
100000
Optimize
Export Result
Items Selected: 25
Region Selected : Kota
Truck Capacity : 100000g

Selected Items for Delivery:
- Flashlight + Batteries x5
- Hygiene Kit x3
- Infant Formula x2
- Blanket x4
- Medical Kit x3
- Bottled Water (1.5L x6) x3
- Rice (5kg) x5
Aid Category Coverage Summary:
- Water
- Utility
- Sanitation
- Medical
- Shelter
- Food
Total Categories Covered: 6/5

```
Region Selected       : Kota
Truck Capacity        : 100000g
--------------------------------------------------

Selected Items for Delivery:
  - Flashlight + Batteries         x5
  - Hygiene Kit                    x3
  - Infant Formula                 x2
  - Blanket                        x4
  - Medical Kit                    x3
  - Bottled Water (1.5L x6)        x3
  - Rice (5kg)                     x5

Aid Category Coverage Summary:
  - Water
  - Utility
  - Sanitation
  - Medical
  - Shelter
  - Food

Total Categories Covered: 6/5
```

```
Impact Estimate for Kota:
  - Basic food needs partially covered
  - Water supply ensured
  - Medical support included
  - Hygiene kits included
  - Blankets provided for shelter
```

iv. Describe the Algorithm Analysis

Correctness:

- The algorithm always returns the optimal priority value due to full state space evaluation.

Time Complexity:

- **Best / Average / Worst Case:** $\Theta(nW)$
  (where n is number of items and W is truck capacity)

Space Complexity:

- $\Theta(nW)$ using a 2D DP table
- Can be optimized to $\Theta(W)$ using a rolling array

Strengths:

- Guaranteed optimal solution
- Can handle diverse constraints

Consideration:

- Might be memory intensive for large W