

College of Computer Studies

**Algorithm**

**(CC0007)**

**<Machine Problem 3>**

*Submitted by:*

ATIENZA, KRYSCH CHALIA

ISNAIN, RAHMAT ZAINA

ESTEBAN, SETH JOHANN

|  |
| --- |
| *Submitted to:*  **<DR. HADJI J. TEJUCO>**  Professor  June 22, 2021 |

Table of Contents

[Introduction 3](#_Toc75296352)

[PART 1: TRAVELLING SALESMAN 3](#_Toc75296353)

[Algorithm 3](#_Toc75296354)

[Screen Output and User’s manual 3](#_Toc75296355)

[Test Cases 20](#_Toc75296356)

[Source Code 21](#_Toc75296357)

[PART 2: KNAPSACK PROBLEM 39](#_Toc75296358)

[Algorithm 39](#_Toc75296359)

[Screen Output and User’s manual 40](#_Toc75296360)

[Test Cases 42](#_Toc75296361)

[Source Code 42](#_Toc75296362)

[Conclusion 47](#_Toc75296363)

[References 47](#_Toc75296364)

# Introduction

We were asked to create a program that simulates the knapsack problem where it should be written in the java language. It should have a GUI interface where the user can input values to be computed by the program like the capacity, weights, and values. It should also have a clear button to remove values inputted and a solve button to compute inputted values. We also added an error message for invalid inputs

# 

# KNAPSACK PROBLEM

# Algorithm

Diagram

Description automatically generated

# Screen Output and User’s manual

**Graphical user interface, application

Description automatically generated**

**Figure 1. User Interface**

Figure 1:

1. Capacity input field
2. Weights input field
3. Values input field
4. Button to clear all fields
5. Button to solve given input
6. Area where table will be shown
7. Area where max capacity our knapsack can carry and the chosen items will be shown.

Table

Description automatically generated

**Figure 2. Sample input and output**

# Test Cases

Positive Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Test Data | Expected Result | Actual Result | Pass/Fail |
| 1 | The user is inputting values in Figure 2 to the program | 1. Run the application 2. Input values to designated field | input values in figure 2 | The user should see the output as shown in figure 2 | The program shows the expected value | Pass |

Negative Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Test Data | Expected Result | Actual Result | Pass/Fail |
| 1 | The user is inputting an alphabet to the program | 1. Run the application 2. input A to a capacity | Input ‘A’ to capacity | The user should see an error message. | The program showman error message | Fail |

# Source Code

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

**public** **class** knapsack

{

**static** JTextArea tableArea;

**static** JTextArea answerArea;

**public** **static** **void** knapsackGUI() {

JFrame frame = **new** JFrame("Knapsack Problem");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(640, 680);

frame.getContentPane().setBackground(**new** Color(187, 230, 228));

//capacity label

JLabel capacityLabel = **new** JLabel("Enter capacity: ");

frame.getContentPane().add(capacityLabel);

capacityLabel.setBounds(20, 0, 400, 100);

//capacity input text field

JTextField capacityInput = **new** JTextField();

capacityInput.setBounds(130, 40, 60, 25);

frame.getContentPane().add(capacityInput);

//weight label

JLabel weightLabel = **new** JLabel("Enter weights: ");

frame.getContentPane().add(weightLabel);

weightLabel.setBounds(20, 30, 400, 100);

//weight input text field

JTextField weightInput = **new** JTextField();

weightInput.setBounds(130, 70, 200, 25);

frame.getContentPane().add(weightInput);

//value label

JLabel valueLabel = **new** JLabel("Enter values: ");

frame.getContentPane().add(valueLabel);

valueLabel.setBounds(20, 60, 400, 100);

//value input text field

JTextField valueInput = **new** JTextField();

valueInput.setBounds(130, 100, 200, 25);

frame.getContentPane().add(valueInput);

//clear button

JButton clearButton = **new** JButton("CLEAR");

clearButton.setBounds(385, 70, 90, 45);

frame.getContentPane().add(clearButton);

//clear ACTION to set the text field into clear

clearButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

capacityInput.setText(" ");

weightInput.setText(" ");

valueInput.setText(" ");

tableArea.setText(" ");

answerArea.setText(" ");

}

});

//solve button

JButton solveButton = **new** JButton("SOLVE");

solveButton.setBounds(490, 70, 90, 45);

frame.getContentPane().add(solveButton);

//solve button ACTION

solveButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

**try** {

**int** capacity = Integer.parseInt(capacityInput.getText());

String arr1 = weightInput.getText();

String arr2 = valueInput.getText();

String arr1\_arr[] = arr1.split(" ");

String arr2\_arr[] = arr2.split(" ");

**int** weight[] = **new** **int**[arr2\_arr.length];

**int** value[] = **new** **int**[arr1\_arr.length];

**int** numOfItems = value.length;

**for** (**int** i = 0; i < arr1\_arr.length; i++) {

weight[i] = Integer.parseInt(arr1\_arr[i]);

}

**for** (**int** j = 0; j < arr2\_arr.length; j++) {

value[j] = Integer.parseInt(arr2\_arr[j]);

}

knapsack.knapsackDP(weight, value, capacity, numOfItems);

}

**catch** (Exception catcher) {

JOptionPane.showMessageDialog(null, "Oops. Something's not right. Please check your input.");

}

}

});

//table Area

tableArea = **new** JTextArea();

tableArea.setBounds(20, 150 , 600, 250);

tableArea.setBackground(**new** Color(240, 246, 246));

tableArea.setEditable(**false**);

tableArea.setFont(**new** Font("Lorem Ipsum", Font.PLAIN, 15));

//scrollbar for table area

JScrollPane tableScroller = **new** JScrollPane(tableArea);

tableScroller.setBounds(20, 150 , 600, 250);

tableScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_ALWAYS);

tableScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS);

frame.getContentPane().add(tableScroller);

//Answer Area. will show the chosen items and max value

answerArea = **new** JTextArea();

answerArea.setBounds(20, 420 , 600, 200);

answerArea.setBackground(**new** Color(0xF9F9F9));

answerArea.setEditable(**false**);

answerArea.setFont(**new** Font("Lorem Ipsum", Font.BOLD, 16));

frame.add(answerArea);

//scrollbar for answer area

JScrollPane answerScroller = **new** JScrollPane(answerArea);

answerScroller.setBounds(20, 420 , 600, 200);

answerScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_ALWAYS);

answerScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS);

frame.getContentPane().add(answerScroller);

frame.getContentPane().add(**new** JLabel());

frame.setLocationRelativeTo(null);

frame.setVisible(**true**);

}

**public** **static** **void** knapsackDP(**int** weight[], **int** value[], **int** capacity, **int** numOfItems){

//create table and

**int** table[][] = **new** **int**[numOfItems + 1][capacity + 1];

//set default values to 0

**for** (**int** i=0; i<=numOfItems; i++)

**for** (**int** j=0; j<=capacity; j++) {

table[i][j] = 0;

}

tableArea.append("\n");

//build the table in bottom-up manner.

**for** (**int** i = 1; i <= numOfItems; i++) {

tableArea.append("Item #" + i + " |");

**for** (**int** j = 0; j <= capacity; j++) {

table[i][j] = table[i - 1][j];

**if** ((j >= weight[i-1]) && (table[i][j] < table[i - 1][j - weight[i - 1]] + value[i - 1]))

{

table[i][j] = table[i - 1][j - weight[i - 1]] + value[i - 1];

}

//show each row being performed in gui

tableArea.append(String.format("%9d", table[i][j]));

}

//print news line for column

tableArea.append("\n");

}

answerArea.append("\n\nMax value knapsack can carry : "+ table[numOfItems][capacity]);

answerArea.append("\n\nItems Chosen : ");

**while** (numOfItems != 0) {

**if** (table[numOfItems][capacity] != table[numOfItems - 1][capacity]) {

answerArea.append("\n\tItem # " + numOfItems + " : Weight = " + weight[numOfItems - 1] + " and Value = " + value[numOfItems - 1]);

capacity = capacity - weight[numOfItems-1];

}

numOfItems--;

}

}

**public** **static** **void** main(String args[]) { knapsackGUI();}

}

# Conclusion

Machine Problem 3 significantly helps us in terms of programming complex algorithms like the TSP and Knapsack Problem. It is something that we can use in the future by referencing what we did in this problem. Integrating these algorithms with a GUI is also significant because it gives us a really robust program compared to just having it function in an IDE. Overall we understand these problems more and have gained knowledge from it.

# References

*GridBagLayout (Java Platform SE 6)*. (2015, November 19). Docs.Oracle.Com. <https://docs.oracle.com/javase/6/docs/api/java/awt/GridBagLayout.html>

thoaionline. (2009, September 6). *Scrollable JPanel*. Stack Overflow. <https://stackoverflow.com/questions/1385737/scrollable-jpanel>

Emax. (2014, September 28). *Java GridBagLayout not working*. Stack Overflow. <https://stackoverflow.com/questions/26088675/java-gridbaglayout-not-working>

Bart Admiraal. (2016, January 21). Scrollbar JTextarea not working. Stack Overflow. <https://stackoverflow.com/questions/34932983/scrollbar-jtextarea-not-working>

A. (2021, May 21). *Travelling Salesman Problem using Branch and Bound*. Techie Delight. <https://www.techiedelight.com/travelling-salesman-problem-using-branch-and-bound/>

Abdul Bari. (2018, February 20). *4.5 0/1 Knapsack - Two Methods - Dynamic Programming* [Video]. YouTube. https://www.youtube.com/watch?v=nLmhmB6NzcM&t=1192s

Rungta, K. (2021, May 28). *Knapsack Problem: Solve using Dynamic Programming Example*. Guru99. <https://www.guru99.com/knapsack-problem-dynamic-programming.html>

*Is there a way to half tab?* (2012, January 23). Stack Overflow. https://stackoverflow.com/questions/8966930/is-there-a-way-to-half-tab