Student name: Quan Hong Doan

Student ID: 986956

Assignment: Lab 1

Week: 01

Due: Oct 20, 2019

**Knapsack problem**

**Problem 1**:

**Answer**:

/\*\*

\* This is class for the Knapsack Optimization problem

\* by using Dynamic Programming approach.

\* The algo is based on https://en.wikipedia.org/wiki/Knapsack\_problem

\*/

public class KnapsackOptimizationDP {

public static void main(String[] args) {

// Testing data

int val[] = new int[] { 60, 100, 120 };

int wt[] = new int[] { 10, 20, 30 };

int W = 50;

int n = val.length;

int result = runKnapsackDP(W, wt, val, n);

System.out.println("Knapsack optimization: " + result);

}

public static int runKnapsackDP(int W, int wt[], int val[], int n) {

int i, w;

int M[][] = new int[n + 1][W + 1];

// Build matrix M[][] in bottom-up manner

for (int i = 0; i <= n; i++) {

for (int j = 0; j <= W; w++) {

if (i == 0 || j == 0) {

M[i][j] = 0;

}

else if (wt[i - 1]<= j) {

M[i][j] = max(val[i - 1] + M[i - 1][j - wt[i - 1]], M[i - 1][j]);

}

else {

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

}

}

}

return M[n][W];

}

// Return a maximum integer

public static int max(int a, int b) {

return (a > b) ? a : b;

}

}

**Problem 2**:

**Answer**:

1. With the Greedy strategies, the result is not always correct. The reason is when we run the Greedy, we also have a bias of the approach by using order of the attributes, like using weight or value to start the order. In other words, when we apply the ordering, that means we also give up some other options, which can be combined and not followed the order.

Giving another sample:

Knapsack capacity W = 25 and

Item A B C D

Value 12 9 9 5

Weight 24 10 10 7

The optimal answer is: B and C with weight = 10 + 10 = 20, and value = 9 + 9 = 18

If we use order descending for weight, the value is only 12, not optimal.

If weight order ascending: the value is only 14 while weight is 17.

1. Similarly, this approach fails in the same way as option 1 with the weight approach. We cannot use this to reach the optimization value.
2. This approach is using ratio, which is the feasible of the three approaches in Greedy strategies, where we can divide the entity into fraction. However, for 0-1 knapsack we have to go DP, as we cannot take or not take the item.

From the given example of answer 1, the greedy fractional is to take the largest ratio from the items.

Item A B C D

Value 12 9 9 5

Weight 24 10 10 7

Ratio (V/W) 0.5 0.9 0.9 0.71

So the feasible is B and C, which is the same as the optimal value.

**Problem 3**:

**Answer**:

1. As in the answer of the Problem 1, we are using a matrix (2D array) to store the calculated values.

i = M.length -1

j = M[0].length -1

currentValue = M[i][j]

While we don't check the first row as it is providing supplementary, we compare the current value to the value from the previous row in the same column. If they are same, this means, we didn't use the corresponding item, hence we go to (i - 1) row. If they are different, meaning the item was used, we update both row and column.

In other words, we don’t use the nth item.

1. It’s true. With the above code from the Answer of Problem 1, and the explanation of the Answer 1 of Problem 3, we can see the S’ is a subset of , which we have to do in our loop. Each round of j is a subset of .