



Dhirubhai Ambani Institute of Information and Communication Technology

Gandhinagar, Gujarat

Name : Ghori Zeel Jivrajbhai

Student ID : 202201287

Course : IT 314 Software Engineering

Professor : Saurabh Tiwari

Semester : Autumn 2024

Lab : Program Inspection, Debugging and Static Analysis

Knapsack Problem

Given code :

```
//Knapsack

public class Knapsack {

    public static void main(String[] args) {

        int N = Integer.parseInt(args[0]); // number of items

        int W = Integer.parseInt(args[1]); // maximum weight of knapsack

        int[] profit = new int[N+1];

        int[] weight = new int[N+1];

        // generate random instance, items 1..N
        for (int n = 1; n <= N; n++) {

            profit[n] = (int) (Math.random() * 1000);

            weight[n] = (int) (Math.random() * W);

        }

        // opt[n][w] = max profit of packing items 1..n with weight limit w
        // sol[n][w] = does opt solution to pack items 1..n with weight limit w include item n?

        int[][] opt = new int[N+1][W+1];

        boolean[][] sol = new boolean[N+1][W+1];

        for (int n = 1; n <= N; n++) {

            for (int w = 1; w <= W; w++) {

                // don't take item n

                int option1 = opt[n-1][w];

                // take item n

                int option2 = Integer.MIN_VALUE;

                if (weight[n] <= w) option2 = profit[n] + opt[n-1][w-weight[n]];

                // select better of two options

                opt[n][w] = Math.max(option1, option2);

            }

        }

    }

}
```

```

        sol[n][w] = (option2 > option1);
    }
}
// determine which items to take
boolean[] take = new boolean[N+1];
for (int n = N, w = W; n > 0; n--) {
    if (sol[n][w]) { take[n] = true; w = w - weight[n]; }
    else { take[n] = false; }
}
// print results
System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
for (int n = 1; n <= N; n++) {
    System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
}
}
}

```

Input: 6, 2000

Output:

Item	Profit	Weight	Take
1	336	784	false
2	674	1583	false
3	763	392	true
4	544	1136	true
5	14	1258	false
6	738	306	true

Program Inspection for Knapsack

1. How many errors are there in the program? Mention the errors you have identified.

- **Errors Identified: 4**

- **Increment Error:** The line `int option1 = opt[n++][w];` incorrectly increments `n`, causing it to exceed the intended range. It should be `int option1 = opt[n][w];` to access the correct `opt` values without modifying `n`.
- **Profit Indexing Error:** The line `if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[n]];` has a logic error in the profit indexing. It should use `profit[n]` instead of `profit[n-2]`, as we want to include the current item's profit.
- **Weight Check Logic Error:** The condition `if (weight[n] > w)` should be inverted to `if (weight[n] <= w)` to correctly check if the item can be included in the knapsack.
- **Array Initialization:** The arrays for profit and weight should be initialized starting from index 1 and accessed accordingly, but the initialization should ensure no items are accidentally counted if not generated correctly.

2. Which category of program inspection would you find more effective?

- **Effective Category:**

- **Logic Errors:** This category is particularly effective as it deals with the correctness of algorithm flow and ensures that the logic aligns with the problem requirements. In this case, checking conditions and indexing correctly is critical for the expected behavior.

3. Which type of error you are not able to identify using the program inspection?

- **Errors Not Identified:**

- **Performance Issues:** While program inspection can highlight logic and syntax errors, it may not effectively identify performance issues, such as inefficiencies in memory usage or runtime complexity, particularly in larger input sizes.

4. Is the program inspection technique worth applicable?

- **Applicability of Program Inspection:**

- Yes, the program inspection technique is worthwhile. It helps identify structural and logical errors, which are essential for ensuring the program functions as intended. However, it should be complemented with testing, especially for edge cases and performance.

Code Debugging

The screenshot shows an IDE with three tabs: `Armstrong.java`, `GCD_LCM.java`, and `KnapSack.java`. The `KnapSack.java` tab is active, displaying the following code:

```
13 for (int n = 1; n <= N; n++) {
14     profit[n] = (int) (Math.random() * 1000);
15     weight[n] = (int) (Math.random() * W);
16 }
17
18 // opt[n][w] = max profit of packing items 1..n with weight limit w
19 // sol[n][w] = does opt solution to pack items 1..n with weight limit w
20 int[][] opt = new int[N+1][W+1];
21 boolean[][] sol = new boolean[N+1][W+1];
22
23 for (int n = 1; n <= N; n++) {
24     for (int w = 1; w <= W; w++) {
25
26         // don't take item n
27         int option1 = opt[n-1][w];
28
29         // take item n
30         int option2 = Integer.MIN_VALUE;
31         if (weight[n] > w) option2 = profit[n-1] + opt[n-1][w-weight[n]];
32
33         // select better of two options
34         opt[n][w] = Math.max(option1, option2);
35         sol[n][w] = (option2 > option1);
36     }
37 }
38
39 // determine which items to take
```

The debugger window on the right shows the following variables and values:

Name	Value
main() is throwing	ArrayIndexOutOfBoundsException...
args	String[2] (id=26)
N	5
W	10
profit	(id=28)
weight	(id=30)
opt	(id=31)
sol	(id=32)
n	2
w	1
option1	0
option2	-2147483648

The expression window shows the expression `2`.

The screenshot shows the same IDE with the `KnapSack.java` tab active, displaying the full code for the Knapsack problem:

```
1 package DebugKnapSack;
2
3 public class KnapSack {
4
5     public static void main(String[] args) {
6         int N = Integer.parseInt(args[0]); // number of items
7         int W = Integer.parseInt(args[1]); // maximum weight of knapsack
8
9         int[] profit = new int[N+1];
10        int[] weight = new int[N+1];
11
12        // generate random instance, items 1..N
13        for (int n = 1; n <= N; n++) {
14            profit[n] = (int) (Math.random() * 1000);
15            weight[n] = (int) (Math.random() * W);
16        }
17
18        // opt[n][w] = max profit of packing items 1..n with weight limit w
19        // sol[n][w] = does opt solution to pack items 1..n with weight limit w
20        int[][] opt = new int[N+1][W+1];
21        boolean[][] sol = new boolean[N+1][W+1];
22
23        for (int n = 1; n <= N; n++) {
24            for (int w = 1; w <= W; w++) {
25
26                // don't take item n
27                int option1 = opt[n-1][w];
28
29                // take item n
30                int option2 = Integer.MIN_VALUE;
31                if (weight[n] > w) option2 = profit[n-1] + opt[n-1][w-weight[n]];
32
33                // select better of two options
34                opt[n][w] = Math.max(option1, option2);
35                sol[n][w] = (option2 > option1);
36            }
37        }
38
39        // determine which items to take
40    }
41 }
```

Errors Identified

1. Option1 Calculation:

- **Original Line:** `int option1 = opt[n++][w];`
- **Correction:** Change to `int option1 = opt[n-1][w];` (This prevents an out-of-bounds error by using the current value of `n` without incrementing it.)

2. Option2 Calculation:

- **Original Line:** `option2 = profit[n-2] + opt[n-1][w-weight[n]];`
- **Correction:** Change to `option2 = profit[n] + opt[n-1][w-weight[n]];` (This correctly references the profit of the current item.)

3. Weight Update Logic:

- **Original Logic:** `if (sol[n][w]) and w = w - weight[n];`
- **Correction:** Ensure that the condition checks if the item is taken correctly, and the weight update logic should work without causing out-of-bounds errors.

Breakpoints Needed

You can set breakpoints at the following locations for effective debugging:

- **Line 20:** To check how `option1` is assigned.
- **Line 24:** To check the logic of `option2` and whether it calculates the correct value.
- **Line 32:** To check if the items are being selected correctly.

Steps to Fix the Errors

1. **Correct the logic for option1** by changing it to `int option1 = opt[n-1][w];`.
2. **Update the logic for option2** by changing it to `option2 = profit[n] + opt[n-1][w - weight[n]];`.
3. **Ensure correct weight update logic** when determining which items to take.

Fixed Code

```
// Knapsack

public class Knapsack {

    public static void main(String[] args) {

        int N = Integer.parseInt(args[0]); // number of items

        int W = Integer.parseInt(args[1]); // maximum weight

        int[] profit = new int[N + 1];

        int[] weight = new int[N + 1];
```

```

// Generate random instance, items 1..N
for (int n = 1; n <= N; n++) {
    profit[n] = (int) (Math.random() * 1000);
    weight[n] = (int) (Math.random() * W);
}

// opt[n][w] = max profit of packing items 1..n with weight limit w
int[][] opt = new int[N + 1][W + 1];
boolean[][] sol = new boolean[N + 1][W + 1];

for (int n = 1; n <= N; n++) {
    for (int w = 1; w <= W; w++) {
        // Don't take item n
        int option1 = opt[n - 1][w]; // Correct: don't increment

        // Take item n
        int option2 = Integer.MIN_VALUE;
        if (weight[n] <= w) { // Fixed condition: weight[n] should be less or equal to w
            option2 = profit[n] + opt[n - 1][w - weight[n]]; // Fixed: profit[n], not profit[n-2]
        }

        // Select better of two options
        opt[n][w] = Math.max(option1, option2);
        sol[n][w] = (option2 > option1);
    }
}

// Determine which items to take
boolean[] take = new boolean[N + 1];
for (int n = N, w = W; n > 0; n--) {

```

```

        if (sol[n][w]) {
            take[n] = true;

            w = w - weight[n]; // Decrease weight
        } else {
            take[n] = false;
        }
    }
}

// Print results
System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
for (int n = 1; n <= N; n++) {
    System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
}
}
}

```

Input and Output

- **Input:** 6,2000
- **Output:** The output will vary due to the random generation of profits and weights, but it will print the item number, profit, weight, and whether it was taken or not.

Static Analysis Tools

Choose a static analysis tool (in Java, Python, C, C++) in any programming language of your interest and identify the defects. You can also choose your own code fragment from GitHub (more than 2000 LOC) in any programming language to perform static analysis. Submit your results in the .xls or .jpg format only.

Link of the excel file(.xls):

https://docs.google.com/spreadsheets/d/1ETFxQdw1_RNXufBPwhlt7OumxA7lJCjh/edit?usp=sharing&ouid=117105520491645507929&rtpof=true&sd=true