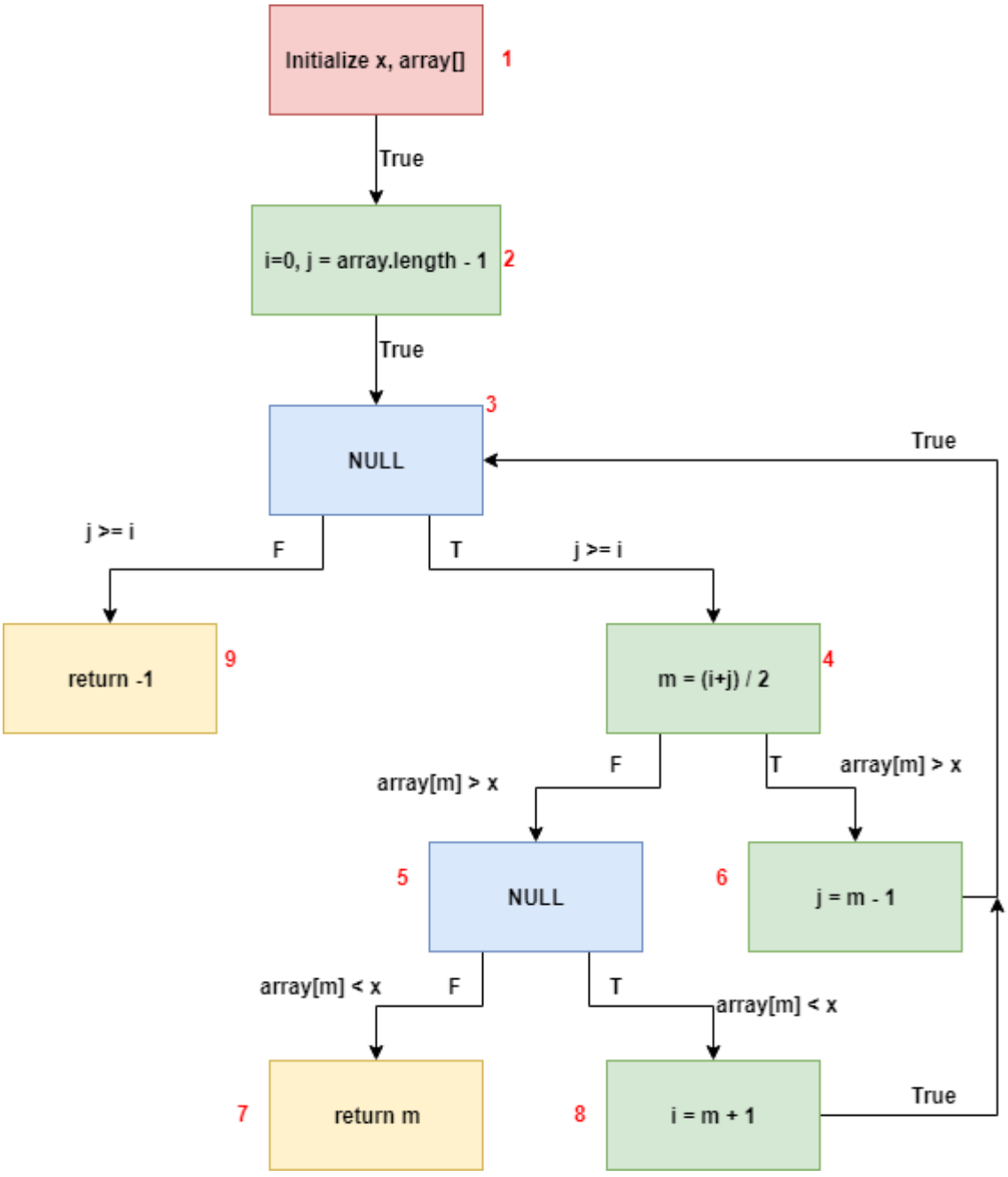
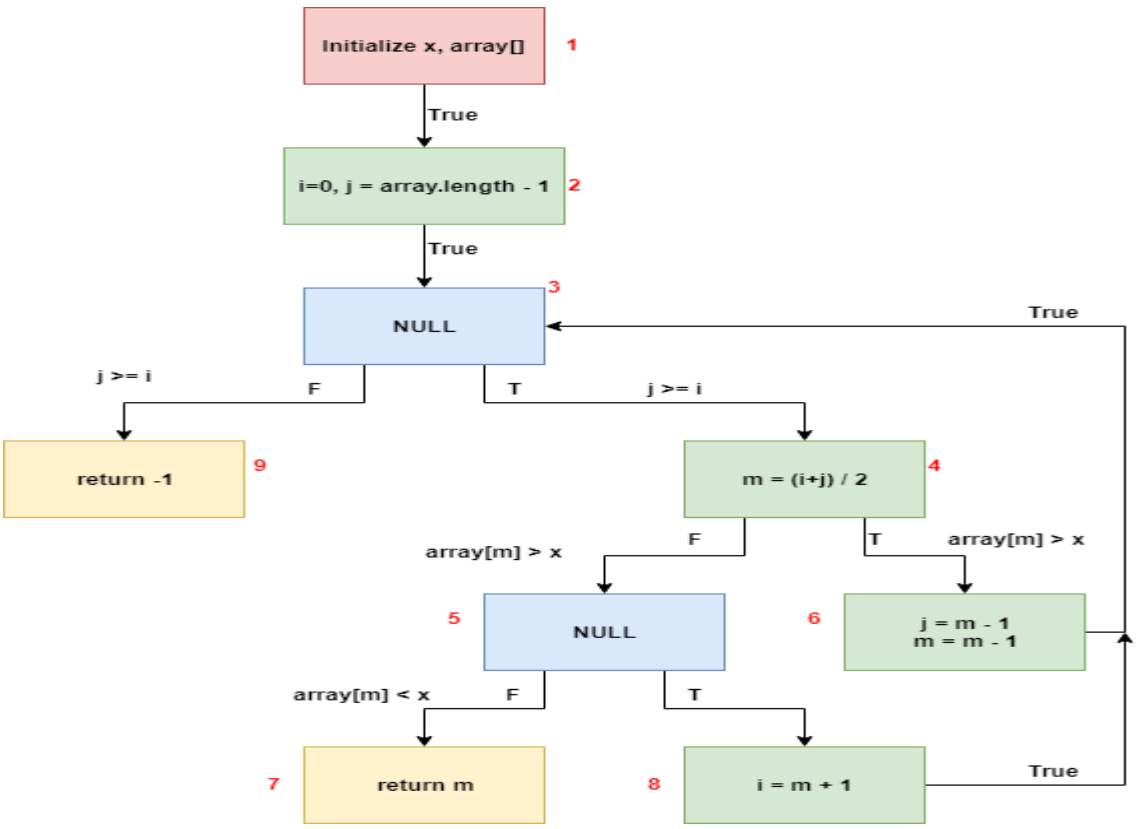
CENG 437 HOMEWORK3

1-a) DFS for binarySearch1():



1-b) DFS for binarySearch2():



2. Assuming that the input array[ ] has at least one element in it, find an infeasible path in the data flow graph for the binarySearch1() function.

1-2-3(F)-9 is the infeasible path, because if array[] has at least one element, initially, result of the j>=i condition always gives True.
Suppose we have 1 element inside input array[]:
After the definitions in node 2: i = 0 and j = 0,
Then in edge (3,9) gives an unsolvable predicate 0 >= 0 = False, and path becomes infeasible.

3. Identify a data flow anomaly in the code binarySearch2().

There is Type1(dd) anomaly in **binarySearch2()** code.

Because, if the code executes the if block, we define variable m (m = m-1) and when if block is executed, program passes the “else if” and “else” code blocks and then again executes the other definition statement of m (m = (i+j)/2). The definition statement(m = m-1) becomes redundant, and this is the Define and Define again anomaly.

NOTE:
I will use the tables(Table1, Table2) for the questions 4 and 5:

- **Table1: def() and c-use() sets of Nodes in DFS for BinarySearch1**

Nodes i	def(i)	c-use(i)
1	x, array	{}
2	i, j	array
3	{}	{}
4	m	j, i
5	{}	{}
6	j	m
7	{}	m
8	i	m
9	{}	{}

- **Table2: Predicates and p-use() Set of Edges in DFS for BinarySearch1**

Edges(i, j)	predicate(i, j)	p-use(i, j)
(1,2)	True	{}
(2,3)	True	{}
(3,4)	j > i	j, i
(4,5)	array[m] > x	array, m, x
(4,6)	array[m] > x	array, m, x
(5,7)	array[m] > x	array, m, x
(5,8)	array[m] > x	array, m, x
(6,3)	True	{}
(8,3)	True	{}
(3,9)	j >= i	j, i

4. By referring to the data flow graph obtained from binarySearch1(), find a set of complete paths satisfying the all-defs selection criterion with respect to variable "m".

For variable "m":

- It has **global definition** in just node 4. (from Table1 column def(i))
- There is **global c-use** of variable "m" in node 6, node 7 and node 8. (from Table1 column c-use(i))
- There is **p-use** of variable "m" in edges (4,5), (4,6), (5,7), (5,8). (from Table2 column p-use(i, j))
- There exist **def-clear paths**:
  = 4-5-7
  = 4-5-8
  = 4-6

➔ **Finally, I have chosen the complete paths:**

- **1 – 2 – 3 –** 4 – 5 – 7 which includes the def-clear path 4-5-7 to satisfy the all-defs criterion.
- **1 – 2 – 3 –** 4 – 5 – 8 **– 3 – 9** which includes the def-clear path 4-5-8 to satisfy the all-defs criterion.
- **1 – 2 – 3 –** 4 – 6 **– 3 – 9** which includes the def-clear path 4-6 to satisfy the all-defs criterion.

5. By referring to the data flow graph obtained from binarySearch1(), find a set of complete paths satisfying the all-defs selection criterion with respect to variable "j".

For variable "j":

- It has **global definition** in just node 2 and node 6. (from Table1 column def(i))
- There is **global c-use** of variable "j" in node 4. (from Table1 column c-use(i))
- There is **p-use** of variable "j" in edges (3,4), (3,9). (from Table2 column p-use(i, j))
- There exist **def-clear paths**:
  = **2-3-4**
  = **2-3-9**
  = **6-3-4**
  = **6-3-9**

➔ **Finally, I choose the complete paths:**

- **1 – 2 – 3 – 4 – 5 – 7** which includes the def-clear path 2-3-5 to satisfy the all-defs criterion.
- **1 – 2 – 3 – 9** which includes the def-clear path 2-3-9 to satisfy the all-defs criterion.
- **1 – 2 – 3 – 4 – 6 – 3 – 4 – 5 - 7** which includes the def-clear path 6-3-9 to satisfy the all-defs criterion.
- **1 – 2 – 3 – 4 – 6 – 3 – 9** which includes the def-clear path 6-3-9 to satisfy the all-defs criterion.

NOTE: I will use the tables(Table3, Table4) for the question 6:

- **Table3: Def() and c-use() Sets of Nodes in DFS for BinarySearch2**

Nodes i	def(i)	c-use(i)
1	x, array	{}
2	i, j	array
3	{}	{}
4	m	j, i
5	{}	{}
6	j,m	m
7	{}	m
8	i	m
9	{}	{}

- **Table4: Predicates and p-use() Set of Edges in DFS for BinarySearch2**

Edges(i, j)	predicate(i, j)	p-use(i, j)
(1,2)	True	{}
(2,3)	True	{}
(3,4)	j > i	j, i
(4,5)	array[m] > x	array, m, x
(4,6)	array[m] > x	array, m, x
(5,7)	array[m] > x	array, m, x
(5,8)	array[m] > x	array, m, x
(6,3)	True	{}
(8,3)	True	{}
(3,9)	j >= i	j, i

6. By referring to the data flow graph obtained from binarySearch2(), find a set of complete paths satisfying the all-defs selection criterion with respect to variable "m".

For variable "m":

- It has global definition in just node 4 and node 6. (from Table3 column def(i))
- There is global c-use of variable "m" in node 6, node 7 and node 8. (from Table3 column c-use(i))
- There is p-use of variable "m" in edges (4,5), (4,6), (5,7), (5,8). (from Table4 column p-use(i))
- There exist def-clear paths:
  = 4-5-7
  = 4-5-8
  = 4-6

➔ **Finally, I have chosen the complete paths:**

- **1 – 2 – 3 –** 4 – 5 – 7 which includes the def-clear path 4-5-7 to satisfy the all-defs criterion.
- **1 – 2 – 3 –** 4 – 5 – 8 **– 3 – 9** which includes the def-clear path 4-5-8 to satisfy the all-defs criterion.
- **1 – 2 – 3 –** 4 – 6 **– 3 – 9** which includes the def-clear path 4-6 to satisfy the all-defs criterion.
- **1 – 2 – 3 –** 4 – 5 – 8 **– 3 – 4 – 5 – 7** which includes the def-clear path 4-5-8 to satisfy the all-defs criterion.

7. The question asks why each three-operation sequences such as:

- **rd(referenced and defined),**
- **rr(referenced and referenced again)**
- **ud(undefined and defined)**

are not considered to be program anomaly.

A program anomaly is an abnormal way of doing something in the program. For example ur is considered as program anomaly because using a variable without initialize it is an abnormal situation. However other sequences such as **ud** and **rr** don't cause an abnormal situation in a program. For example, consider the following code snipped:

```
1-)   int total = 0;
2-)   int counter;
3-)   counter = 0;
4-)   while(counter > 2)
5-)   {
6-)      total = total + counter;
7-)      counter = counter + 1;
8-)   }
```

In the code above, counter is declared in statement 2, and then initialized in statement 3. There are **ud** sequence here and this is very normal situation in a program. We deal with such a sequence in every program normally and absolutely that cannot be a manifestation of potential program error.

Again in statement 6, variable counter is referenced, and in statement 7 counter is referenced again. There are **rr** sequence here and this is again very normal situation in a program, and we use that kind of sequence almost in every while loop and again that cannot be a manifestation of potential program error.

Briefly, as I explained with that example code snipped, other sequences are normal situations in a program and they are not considered to be program anomaly.

Additionally, question ask for us to give a small Java code sinpped example to each program anomaly:

- **dd anomaly(Defined and Then Defined Again):**

```
public int getPrice()
{
    int price = 100;
    int workingHour = getWorkingHourFromUser(); //gets working hour as input from user.

    if(workingHour < 15)
    {
        price = workingHour * 100;
    }
    else if(workingHour < 25)
    {
        price = workingHour * 150;
    }
    else
    {
        price = workingHour * 176;
    }

 }
```

Here price is defined as 100 initially and then according to workingHour, price is defined again inside if-else if and else blocks. That is dd anomaly and the statement(int price = 100) is redundant according to that anomaly.

- **du anomaly(Defined but not Referenced):**

```
public int getPrice()
{
    int price;
    int workingHour = getWorkingHourFromUser(); //gets working hour as input from user.
    EmployeeType empType = EmployeeType.HOURLY_EMPLOYEE;

    if(workingHour < 15)
    {
        price = workingHour * 100;
    }
    else if(workingHour < 25)
    {
        price = workingHour * 150;
    }
    else
    {
        price = workingHour * 176;
    }

 }
```

Here empType is defined as HOURLY_EMPLOYEE and after that statement, empType is not referenced in any other subsequent computation. So there are du anomaly here.

- **ur anomaly(Undefined but Referenced):**

```
public int getPrice()
{
    int price;
    int workingHour;

    if(workingHour < 15)
    {
        price = workingHour * 100;
    }
    else if(workingHour < 25)
    {
        price = workingHour * 150;
    }
    else
    {
        price = workingHour * 176;
    }

 }
```

Here workingHour is not initialized but it is referenced in if- else if and else blocks. Here there are ur anomaly. We are trying to define variable price with an uninitalized variable workingHour and this is an program anomaly.

Hasan YENİADA

220201024