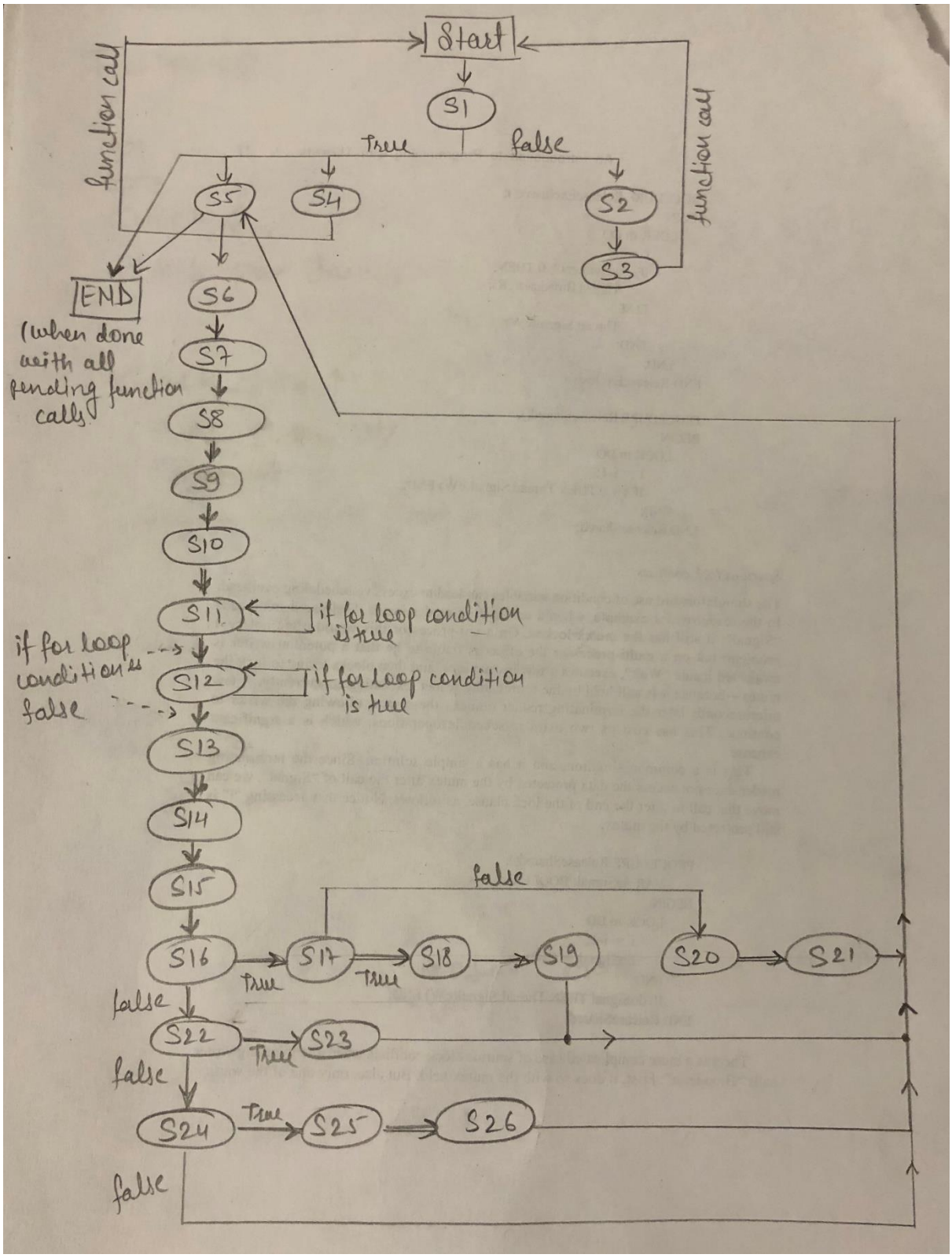
HOMEWORK#2

Name: Akshata Deo
SJSU ID: 012565761

Question #1: Branch Software Testing

```
public static void mergeSort(int[] array, int left, int right) {
        if (right <= left) return; (S1)
        int mid = (left+right)/2; (S2)
        mergeSort(array, left, mid); (S3)
        mergeSort(array, mid+1, right); (S4)
        merge(array, left, mid, right); (S5)
}
void merge(int[] array, int left, int mid, int right) { (S6)
        int lengthLeft = mid - left + 1; (S7)
        int lengthRight = right - mid; (S8)
        int leftArray[] = new int [lengthLeft]; (S9)
        int rightArray[] = new int [lengthRight]; (S10)
        for (int i = 0; i < lengthLeft; i++) leftArray[i] = array[left+i]; (S11)
        for (int i = 0; i < lengthRight; i++) rightArray[i] = array[mid+i+1]; (S12)
        int leftIndex = 0; (S13)
        int rightIndex = 0; (S14)
        for (int i = left; i < right + 1; i++) { (S15)
                if (leftIndex < lengthLeft && rightIndex < lengthRight) { (S16)
                        if (leftArray[leftIndex] < rightArray[rightIndex]) { (S17)
                                array[i] = leftArray[leftIndex]; (S18)
                                leftIndex++; (S19)
                        }
                        else { (S20)
                                array[i] = rightArray[rightIndex]; rightIndex++; (S21)
                        }
                }
                else if (leftIndex < lengthLeft) { (S22)
                        array[i] = leftArray[leftIndex]; leftIndex++; (S23)
                }
                else if (rightIndex < lengthRight) { (S24)
                        array[i] = rightArray[rightIndex]; (S25)
                        rightIndex++; (S26)
                }
        }
}
```

(a). Program flow graph based on the source code of MergeSort() function:



(b). Branch table for branch testing:

Predicate Nodes	Conditions	Possible Outcomes

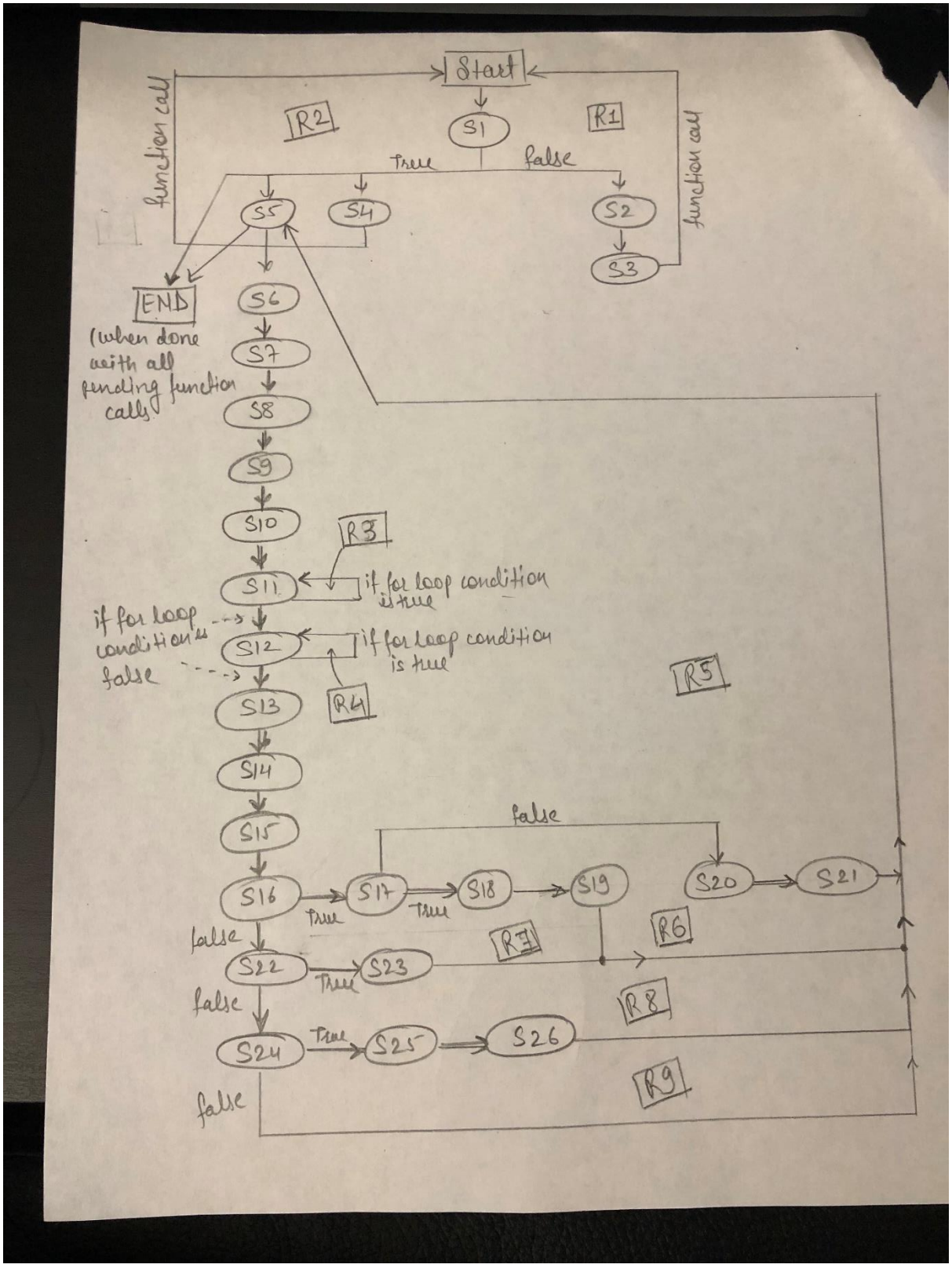
S1	right <= left	• True: If there is no pending function call, then END the program. If there is pending function call, return the value to the respective function (S4 / S5) • False: S2 -> S3 -> START
S5	Pending function call	• True: S6 -> S7 -> S8 -> S9 -> S10 -> S11 • False: END the program
S11	i < lengthLeft	• True: S11 • False: S12
S12	i < lengthRight	• True: S12 • False: S13 -> S14 -> S15 -> S16
S16	(leftIndex < lengthLeft) && (rightIndex < lengthRight)	• True: S17 • False: S22
S17	leftArray[leftIndex] < rightArray[rightIndex]	• True: S18 -> S19 -> S5 • False: S20 -> S21 -> S5
S22	leftIndex < lengthLeft	• True: S23 -> S5 • False: S24
S24	rightIndex < lengthRight	• True: S25 -> S26 -> S5 • False: S5

(c). Test cases for all predicate nodes:

Test Case ID	Input	Expected Output
mergeSort_1	right <= left	Program will exit on S1 only
mergeSort_2	If array is already sorted like {1,4,7,9,12,67}	Return sorted array as it is without any changes like {1,4,7,9,12,67}
mergeSort_3	If array is unsorted like {1,3,7,4,0,5}	Return a sorted array like {0,1,3,4,5,7}

Question #2: Basis Path Testing

(a).



Following are the three different ways to compute its Cyclomatic number:

1. M(G) = Number of regions in G
   M(G) = 9

2. M(G) = |E| - |N| + 2

Where,
|E| = Number of edges = 34
|N| = Number of nodes = 27
M(G) = 34 – 27 + 2 = 9

3. M(G) = |P| + 1
   Where, |P| = Number of predicate nodes
   M(G) = 8+1 = 9

(b). Graph Matrix

	Start	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	End	
Start		1																											
S1			1		1	1																						1	02:01
S2,S3,S4	1			1																									01:01
S5							1																					1	02:01
S6,S7,S8								1	1	1																			01:01
S9,S10											1	1																	01:01
S11												1	1																02:01
S12													1	1															02:01
S13,S14															1	1													01:01
S15																	1												01:01
S16																		1					1						02:01
S17																			1		1								02:01
S18,S19						1														1									01:01
S20,S21						1																1							01:01
S22																								1	1				02:01
S23						1																							01:01
S24						1																				1			02:01
S25,S26						1																					1		01:01
End																													01:01
	Number of Predicate Nodes																												9

(c). Basis Path Testing:

Path_ID	Conditions Applied	Full Path
P1	Multiple Conditions	Start -> S1 -> S2 -> S3 -> Start -> S1 -> S4 -> Start -> S1 -> End
P2	(leftIndex < lengthLeft) && (rightIndex < lengthRight) = TRUE leftArray[leftIndex] < rightArray[rightIndex] = TRUE	Start -> S1 -> S5 -> S6 -> S7 -> S8 -> S9 -> S10 -> S11 -> S12 -> S13 -> S14 -> S15 -> S16 -> S17 -> S18 -> S19 -> S5 -> End
P3	(leftIndex < lengthLeft) && (rightIndex < lengthRight) = TRUE leftArray[leftIndex] < rightArray[rightIndex] = FALSE	Start -> S1 -> S5 -> S6 -> S7 -> S8 -> S9 -> S10 -> S11 -> S12 -> S13 -> S14 -> S15 -> S16 -> S17 -> S20 -> S21 -> S5 -> End
P4	(leftIndex < lengthLeft) && (rightIndex < lengthRight) = FALSE leftIndex < lengthLeft = TRUE	Start -> S1 -> S5 -> S6 -> S7 -> S8 -> S9 -> S10 -> S11 -> S12 -> S13 -> S14 -> S15 -> S16 -> S22 -> S23 -> S5 -> End
P5	(leftIndex < lengthLeft) && (rightIndex < lengthRight) = FALSE leftIndex < lengthLeft = FALSE rightIndex < lengthRight = TRUE	Start -> S1 -> S5 -> S6 -> S7 -> S8 -> S9 -> S10 -> S11 -> S12 -> S13 -> S14 -> S15 -> S16 -> S22 -> S24 -> S25 -> S26 -> S5 -> End

P6	(leftIndex < lengthLeft) && (rightIndex < lengthRight) = FALSE leftIndex < lengthLeft = FALSE rightIndex < lengthRight = FALSE	Start -> S1 -> S5 -> S6 -> S7 -> S8 -> S9 -> S10 -> S11 -> S12 -> S13 -> S14 -> S15 -> S16 -> S22 -> S24 -> S5 -> End

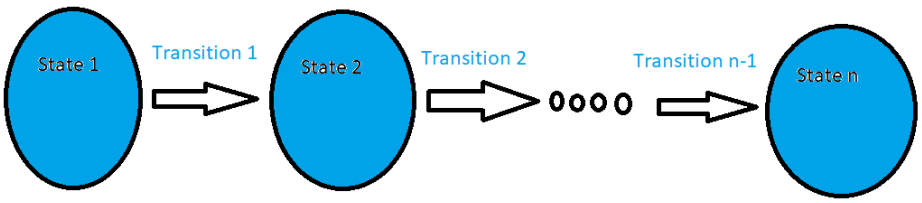
(d). Following are the basis set of test cases:

Test Case ID	Input	Expected Output
mergeSort_1	right <= left	Program will exit on S1 only
mergeSort_2	If array is already sorted like {1,4,7,9,12,67}	Return sorted array as it is without any changes like {1,4,7,9,12,67}
mergeSort_3	If array is unsorted like {1,3,7,4,0,5}	Return a sorted array like {0,1,3,4,5,7}

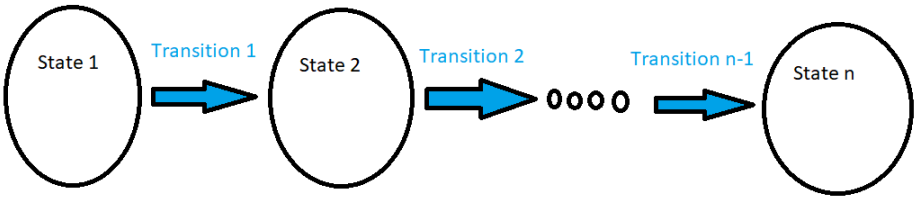
Question #3: State-Based Software Testing

(a). Following are the two state-based test criteria:

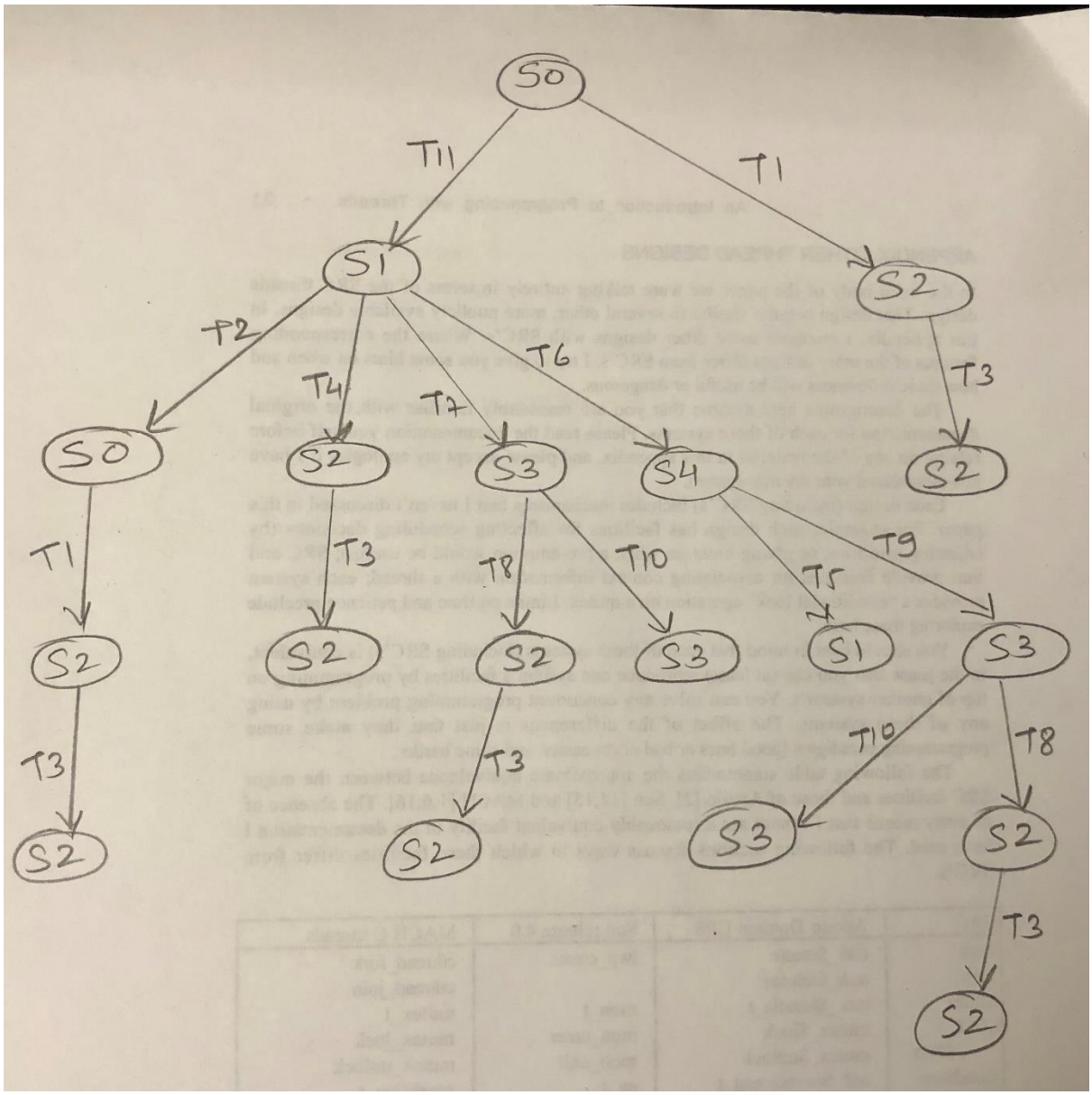
1. **All-state test coverage criteria:** Each state of the system is tested at least once during the testing process. It is the technique to check the output variables of a program. All variables defined when executing a test scope are considered by state coverage. However, all-state test coverage criteria is not the most efficient method for full coverage of the system, still, famous amongst testers.



2. **All-transition test coverage criteria:** All transitions test coverage criteria specify that each transition must be tested at least once. For transitions testing, state of the transition must be in acceptable form. There is no specific condition to be in the acceptable state. This coverage criterion is one of the most commonly used and highly recommended by the testers.



(b). Following is the state-based tree diagram:



(c). Following are the list of all state paths from S0 to S3:

- S0 -> T11 -> S1 -> T7 -> S3 -> T10 -> S3
- S0 -> T11 -> S1 -> T6 -> S4 -> T9 -> S3 -> T10 -> S3

(d). Following are the list of state test set which achieves all-transition- node coverage:

LEVEL-1 • S0 -> T11 -> S1 • S0 -> T1 -> S2
LEVEL-2 • S0 -> T11 -> S1 -> T2 -> S0 • S0 -> T11 -> S1 -> T4 -> S2 • S0 -> T11 -> S1 -> T7 -> S3 • S0 -> T11 -> S1 -> T6 -> S4 • S0 -> T1 -> S2 -> T3 -> S2
LEVEL-3 • S0 -> T11 -> S1 -> T2 -> S0 -> T1 -> S2 • S0 -> T11 -> S1 -> T4 -> S2 -> T3 -> S2 • S0 -> T11 -> S1 -> T7 -> S3 -> T8 -> S2 • S0 -> T11 -> S1 -> T7 -> S3 -> T10 -> S3 • S0 -> T11 -> S1 -> T6 -> S4 -> T5 -> S1 • S0 -> T11 -> S1 -> T6 -> S4 -> T9 -> S3
LEVEL-4 • S0 -> T11 -> S1 -> T2 -> S0 -> T1 -> S2 -> T3 -> S2 • S0 -> T11 -> S1 -> T7 -> S3 -> T8 -> S2 -> T3 -> S2 • S0 -> T11 -> S1 -> T6 -> S4 -> T9 -> S3 -> T10 -> S3 • S0 -> T11 -> S1 -> T6 -> S4 -> T9 -> S3 -> T8 -> S2
LEVEL-5 • S0 -> T11 -> S1 -> T6 -> S4 -> T9 -> S3 -> T8 -> S2 -> T3 -> S2

Question #4: Big Data Quality Validation Tools

(a).

[1] F. S. Gharehchopogh and Z. A. Khalifelu, "Analysis and evaluation of unstructured data: text mining versus natural language processing," *2011 5th International Conference on Application of Information and Communication Technologies (AICT)*, Baku, 2011, pp. 1-4.

[2] Y. Song, D. Zhou, J. Huang, I. G. Councill, H. Zha and C. L. Giles, "Boosting the Feature Space: Text Classification for Unstructured Data on the Web," *Sixth International Conference on Data Mining (ICDM'06)*, Hong Kong, 2006, pp. 1064-1069.

[3] S. Kaisler, F. Armour, J. A. Espinosa and W. Money, "Big Data: Issues and Challenges Moving Forward," *2013 46th Hawaii International Conference on System Sciences*, Wailea, Maui, HI, 2013, pp. 995-1004.

[4] L. Soibelman, J. Wu, C. Caldas, I. Brilakis, and K. Y. Lin, "Management and analysis of unstructured construction data types." *Advanced Engineering Informatics*, 2008, *22*(1), pp.15-27.

[5] H. Li, Z. Peng, X. Feng, and H. Ma, "Leakage prevention method for unstructured data based on classification." *In International Conference on Applications and Techniques in Information Security*, Heidelberg, November 2015, pp. 337-343.

(b).

1. TestingWhiz: - It is an automated Big Data testing solution, which automates the validation of structured and unstructured data sets, schemas, approaches and inherent processes residing at different sources in the application. It supports languages such as 'Hive', 'Map-reduce' 'Sqoop' and 'Pig.'

2. QureySurge: - It is an automated Data Testing solution for Hadoop data lakes & NoSQL data stores.

3. Diftong: - A tool for validating big data workflows. This tool compares two tabular databases resulting from different versions of a workflow to detect and prevent potential unwanted alterations.

4. Apache Pig: - Apache Pig is a platform for analysing large data sets, coupled with infrastructure for evaluating these programs.

5. RightData: - Commercial big data testing tool with a scalable data testing engine which allows the user to the easily create test scenarios between disparate systems and allows to automate the whole testing process and ensure the data ingested into Big Data platform, transformed using map reduce logic remains intact compared to its source.

(c).

Tool name	**Maker**	**Major Features**	**Technology**	**Focus Area**	**Platform**	**Limitations**
TestingWhiz	Cygnet Infotech	Automated, Real-time information, Better data quality	Hadoop Teradata MongoDB	Post ETL data validation	Web, mobile and Cloud	Multiple layers
QuerySurge	RTTS	Leverage Analytics, High coverage	Hadoop NoSQL	Data Testing solution for Hadoop data lakes & NoSQL data store	Cloud	Quality Less portability
Diftlong	Uppsala University, Sweden	Comparison based analysis	Hadoop No SQL	Databases	Web	Performance optimization is required
Apache Pig	Apache Foundation	Extensibility	Hadoop	parallelization	All	Maintenance, Support

RightData Big Data Testing	RightData Inc.	data staging validation, map reduce validation and output validations	Hadoop	Data Staging, Map Reduce	Cloud	Prioritizing correlations, limited transferability