UNIT 5
GRAPH MATRICES AND APPLICATIONS

MOTIVATIONAL OVERVIEW

The Problem with Pictorial Graphs

- Graphs were introduced as an abstraction of software structure.
- Whenever a graph is used as a model, sooner or later we trace paths through it- to find a set of covering paths, a set of values that will sensitize paths, the logic function that controls the flow, the processing time of the routine, the equations that define the domain, or whether a state is reachable or not.
- Path is not easy, and it's subject to error. You can miss a link here and there or cover some links twice.
- One solution to this problem is to represent the graph as a matrix and to use matrix operations equivalent to path tracing. These methods are more methodical and mechanical and don't depend on your ability to see a path they are more reliable.

Tool Building

- If you build test tools or want to know how they work, sooner or later you will be implementing or investigating analysis routines based on these methods.
- It is hard to build algorithms over visual graphs so the properties or graph matrices are fundamental to tool building.

The Basic Algorithms

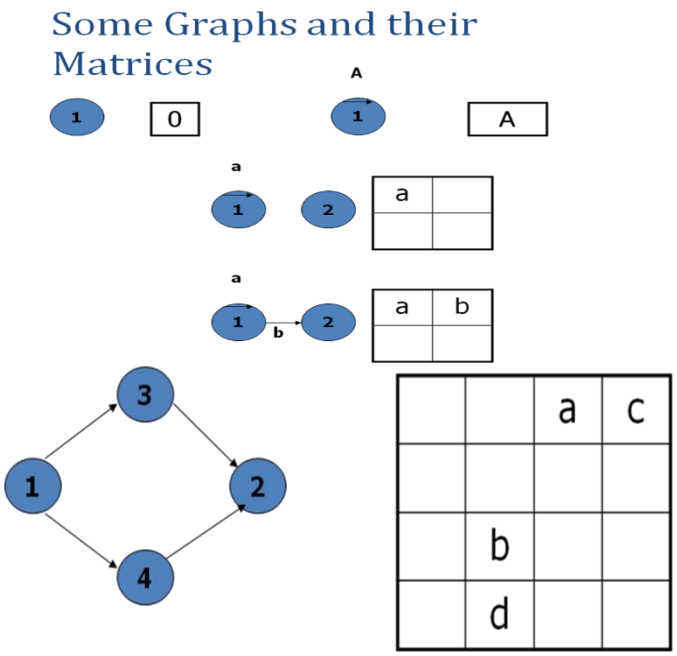
- The basic tool kit consists of:
- Matrix multiplication, which is used to get the path expression from every node to every other node.
- A partitioning algorithm for converting graphs with loops into loop free graphs or equivalence classes.
- A collapsing process which gets the path expression from any node to any other node.

THE MATRIX OF A GRAPH

- A graph matrix is a square array with one row and one column for every node in the graph.
- Each row-column combination corresponds to a relation between the node corresponding to the row and the node corresponding to the column.
- The relation for example, could be as simple as the link name, if there is a link between the nodes.

- Some of the things to be observed:
- The size of the matrix equals the number of nodes.
- There is a place to put every possible direct connection or link between any and any other node.

- The entry at a row and column intersection is the link weight of the link that connects the two nodes in that direction.
- A connection from node i to j does not imply a connection from node j to node i.
- If there are several links between two nodes, then the entry is a sum; the "+" sign denotes parallel links as usual.



A simple weight

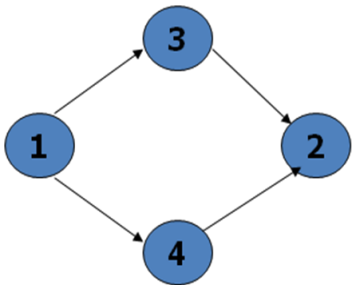
- A simplest weight we can use is to note that there is or isn't a connection. Let "1" mean that there is a connection and "0" mean that there isn't.
- The arithmetic rules are:
- 1+1=1 1*1=1
- 1+0=1 1*0=0
- 0+0=0 0*0=0
- A matrix defined like this is called connection matrix.

Connection matrix

- The connection matrix is obtained by replacing each entry with 1 if there is a link and 0 if there isn't.
- As usual we don't write down 0 entries to reduce the clutter.

		a	c
	b		
	d		

		1	1
	1		
	1		



Connection Matrix-continued

- Each row of a matrix denotes the out links of the node corresponding to that row.
- Each column denotes the in links corresponding to that node.
- A branch is a node with more than one nonzero entry in its row.
- A junction is node with more than one nonzero entry in its column.
- A self loop is an entry along the diagonal.

Cyclomatic Complexity

- The cyclomatic complexity obtained by subtracting 1 from the total number of entries in each row and ignoring rows with no entries, we obtain the equivalent number of decisions for each row. Adding these values and then adding 1 to the sum yields the graph's cyclomatic complexity.

		1	1	2-1=1
	1			1-1=0
	1			1-1=0

1+1=2 (cyclomatic complexity)

Relations

- A relation is a property that exists between two objects of interest.
- For example,
- “Node a is connected to node b” or aRb where “R” means “is connected to”.
- “a>=b” or aRb where “R” means greater than or equal”.
- A graph consists of set of abstract objects called nodes and a relation R between the nodes.
- If aRb, which is to say that a has the relation R to b, it is denoted by a link from a to b.
- For some relations we can associate properties called as link weights.

Transitive Relations

- A relation is transitive if aRb and bRc implies aRc.
- Most relations used in testing are transitive.
- Examples of transitive relations include: is connected to, is greater than or equal to, is less than or equal to, is a relative of, is faster than, is slower than, takes more time than, is a subset of, includes, shadows, is the boss of.
- Examples of intransitive relations include: is acquainted with, is a friend of, is a neighbor of, is lied to, has a du chain between.

Reflexive Relations

- A relation R is reflexive if, for every a, aRa.
- A reflexive relation is equivalent to a self loop at every node.
- Examples of reflexive relations include: equals, is acquainted with, is a relative of.
- Examples of irreflexive relations include: not equals, is a friend of, is on top of, is under.

Symmetric Relations

- A relation R is symmetric if for every a and b, aRb implies bRa.
- A symmetric relation mean that if there is a link from a to b then there is also a link from b to a.
- A graph whose relations are not symmetric are called directed graph.

- A graph over a symmetric relation is called an undirected graph.
- The matrix of an undirected graph is symmetric ($a_{ij}=a_{ji}$) for all i,j)

Antisymmetric Relations

- A relation R is antisymmetric if for every a and b, if aRb and bRa, then a=b, or they are the same elements.
- Examples of antisymmetric relations: is greater than or equal to, is a subset of, time.
- Examples of nonantisymmetric relations: is connected to, can be reached from, is greater than, is a relative of, is a friend of

Equivalence Relations

- An equivalence relation is a relation that satisfies the reflexive, transitive, and symmetric properties.
- Equality is the most familiar example of an equivalence relation.
- If a set of objects satisfy an equivalence relation, we say that they form an equivalence class over that relation.
- The importance of equivalence classes and relations is that any member of the equivalence class is, with respect to the relation, equivalent to any other member of that class.
- The idea behind partition testing strategies such as domain testing and path testing, is that we can partition the input space into equivalence classes.
- Testing any member of the equivalence class is as effective as testing them all.

Partial Ordering Relations

- A partial ordering relation satisfies the reflexive, transitive, and antisymmetric properties.
- Partial ordered graphs have several important properties: they are loop free, there is at least one maximum element, and there is at least one minimum element.

The Powers of a Matrix

- Each entry in the graph's matrix expresses a relation between the pair of nodes that corresponds to that entry.
- Squaring the matrix yields a new matrix that expresses the relation between each pair of nodes via one intermediate node under the assumption that the relation is transitive.
- The square of the matrix represents all path segments two links long.
- The third power represents all path segments three links long.

Matrix Powers and Products

- Given a matrix whose entries are aij, the square of that matrix is obtained by replacing every entry with
  - n
  - $a_{ij}=\Sigma\ a_{ik}\ a_{kj}$
  - k=1
- more generally, given two matrices A and B with entries aik and bkj, respectively, their product is a new matrix C, whose entries are cij, where:
  - n
    - $C_{ij}=\Sigma\ a_{ik}\ b_{kj}$
    - k=1

CASE 2: Single loop, Non-zero minimum, No excluded values

2. Try one less than the expected minimum. What happens if the loop control variable's value is less than the minimum? What prevents the value from being less than the minimum?
3. The minimum number of iterations.
4. One more than the minimum number of iterations.
5. Once, unless covered by a previous test.
6. Twice, unless covered by a previous test.
7. A typical value.
8. One less than the maximum value.
9. The maximum number of iterations.
10. Attempt one more than the maximum number of iterations.

Partitioning Algorithm

- Consider any graph over a transitive relation. The graph may have loops.
- We would like to partition the graph by grouping nodes in such a way that every loop is contained within one group or another.
- Such a graph is partially ordered.
- There are many used for an algorithm that does that:
- We might want to embed the loops within a subroutine so as to have a resulting graph which is loop free at the top level.
- Many graphs with loops are easy to analyze if you know where to break the loops.
- While you and I can recognize loops, it's much harder to program a tool to do it unless you have a solid algorithm on which to base the tool.

Node Reduction Algorithm (General)

- The matrix powers usually tell us more than we want to know about most graphs.
- In the context of testing, we usually interested in establishing a relation between two nodes- typically the entry and exit nodes.
- In a debugging context it is unlikely that we would want to know the path expression between every node and every other node.
- The advantage of matrix reduction method is that it is more methodical than the graphical method called as node by node removal algorithm.

1. Select a node for removal; replace the node by equivalent links that bypass that node and add those links to the links they parallel.
2. Combine the parallel terms and simplify as you can.
3. Observe loop terms and adjust the out links of every node that had a self loop to account for the effect of the loop.
4. The result is a matrix whose size has been reduced by 1. Continue until only the two nodes of interest exist.

BUILDING TOOLS

Overview:

- We draw graphs or display them on screens as visual objects; we prove theorems and develop graph algorithms by using matrices;
- and when we want to process graphs in a computer, because we're building tools, we represent them as linked lists.
- We use linked lists because graph matrices are usually very sparse; that is, the rows and columns are mostly empty.

Matrix Representation Software

Node Degree and Graph Density:

- The **out-degree** of a node is the number of outlinks it has.
- The **in-degree** of a node is the number of inlinks it has.
- The **degree** of a node is the sum of the out-degree and in-degree.
- The average degree of a node (the mean over all nodes) for a typical graph defined over software is between 3 and 4.
- The degree of a simple branch is 3, as is the degree of a simple junction.
- The degree of a loop, if wholly contained in one statement, is only 4.
- A mean node degree of 5 or 6 say, would be a very busy flowgraph indeed.

What's Wrong with Arrays?

- **1.** *Space*—Space grows as n^2 for the matrix representation, but for a linked list only as *kn*, where *k* is a small number such as 3 or 4.
- **2.** *Weights*—Most weights are complicated and can have several components. That would require an additional weight matrix for each such weight.
- **3.** *Variable-Length Weights*—If the weights are regular expressions, say, or algebraic expressions (which is what we need for a timing analyzer), then we need a two-dimensional string array, most of whose entries would be null.
- **4.** *Processing Time*—Even though operations over null entries are fast, it still takes time to access such entries and discard them. The matrix representation forces us to spend a lot of time processing combinations of entries that we know will yield null results.

Lecture Notes on Gaps

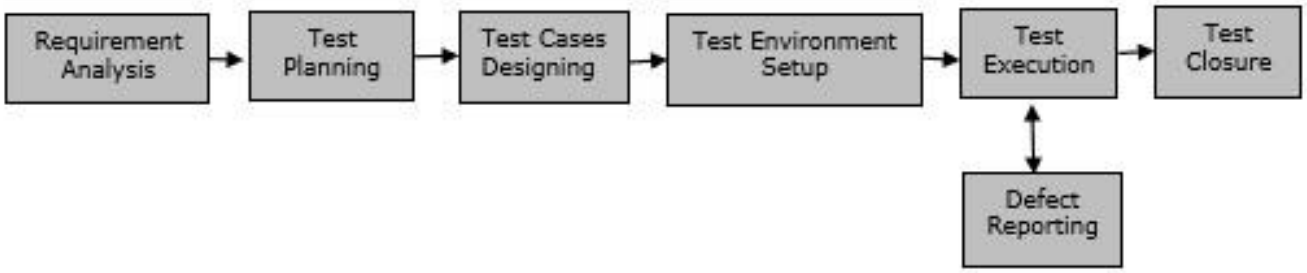
SOFTWARE TESTING LIFE CYCLE:

STLC stands for Software Testing Life Cycle. STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product.

- STLC is an integral part of Software Development Life Cycle (SDLC). But, STLC deals only with the testing phases.
- STLC starts as soon as requirements are defined or SRD (Software Requirement Document) is shared by stakeholders.
- STLC provides a step-by-step process to ensure quality software.
- In the early stage of STLC, while the software or the product is developing, the tester can analyze and define the scope of testing, entry and exit criteria and also the Test Cases. It helps to reduce the test cycle time along with better quality.
- As soon as the development phase is over, the testers are ready with test cases and start with execution. This helps to find bugs in the initial phase.

STLC Phases

STLC has the following different phases but it is not mandatory to follow all phases. Phases are dependent on the nature of the software or the product, time and resources allocated for the testing and the model of SDLC that is to be followed.



There are 6 major phases of STLC −

- **Requirement Analysis** − When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).
- **Test Planning** − Test Team plans the strategy and approach.
- **Test Case Designing** − Develop the test cases based on scope and criteria's.
- **Test Environment Setup** − When integrated environment is ready to validate the product.
- **Test Execution** − Real-time validation of product and finding bugs.
- **Test Closure** − Once testing is completed, matrix, reports, results are documented.

TEST CASE DESIGN STRATEGIES FOR PROGRAMMERS AND INDEPENDENT TESTERS:

There are many different test case design techniques used to test the functionality and various features of your software. Designing good test cases ensure that every aspect of your software gets tested so that you can find and fix any issues.

A basic example of test case design:

Title: Login to the website or app
Description: User should be able to successfully log in to their account on the website/app
Preconditions: User must already be registered and use their correct login details
Assumptions: They are using a supported device or browser to log in
Test Steps:

1. Open website or app
2. Enter the username and password in the appropriate fields
3. Click “login”

Expected Result: The user should log in successfully.

What are the types of test case design techniques?

The main purpose of test case design techniques is to test the functionalities and features of the software with the help of effective test cases. The test case design techniques are broadly classified into three major categories.

1. Specification-Based techniques
2. Structure-Based techniques
3. Experience-Based techniques

1. Specification-Based or Black-Box techniques

This technique leverages the external description of the software such as technical specifications, design, and client’s requirements to design test cases. The technique enables testers to develop test cases that provide full test coverage. The Specification-based or black box test case design techniques are divided further into 5 categories. These categories are as follows:

Boundary Value Analysis (BVA)

This technique is applied to explore errors at the boundary of the input domain. BVA catches any input errors that might interrupt with the proper functioning of the program.

Equivalence Partitioning (EP)

In Equivalence Partitioning, the test input data is partitioned into a number of classes having an equivalent number of data. The test cases are then designed for each class or partition. This helps to reduce the number of test cases.

Decision Table Testing

In this technique, test cases are designed on the basis of the decision tables that are formulated using different combinations of inputs and their corresponding outputs based on various conditions and scenarios adhering to different business rules.

State Transition Diagrams

In this technique, the software under test is perceived as a system having a finite number of states of different types. The transition from one state to another is guided by a set of rules. The rules define the response to different inputs. This technique can be implemented on the systems which have certain workflows within them.

Use Case Testing

A use case is a description of a particular use of the software by a user. In this technique, the test cases are designed to execute different business scenarios and end-user functionalities. Use case testing helps to identify test cases that cover the entire system.

2. Structure-Based or White-Box techniques

The structure-based or white-box technique design test cases based on the internal structure of the software. This technique exhaustively tests the developed code. Developers who have complete information of the software code, its internal structure, and design help to design the test cases. This technique is further divided into five categories.

Statement Testing & Coverage

This technique involves execution of all the executable statements in the source code at least once. The percentage of the executable statements is calculated as per the given requirement. This is the least preferred metric for checking test coverage.

Decision Testing Coverage

This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed. This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

Condition Testing

Condition testing also is known as Predicate coverage testing, each Boolean expression is predicted as TRUE or FALSE. All the testing outcomes are at least tested once. This type of testing involves 100% coverage of the code. The test cases are designed as such that the condition outcomes are easily executed.

Multiple Condition Testing

The purpose of Multiple condition testing is to test the different combination of conditions to get 100% coverage. To ensure complete coverage, two or more test scripts are required which requires more efforts.

All Path Testing

In this technique, the source code of a program is leveraged to find every executable path. This helps to determine all the faults within a particular code.

3. Experience-Based techniques

These techniques are highly dependent on tester's experience to understand the most important areas of the software. The outcomes of these techniques are based on the skills, knowledge, and expertise of the people involved. The types of experience-based techniques are as follows:

Error Guessing

In this technique, the testers anticipate errors based on their experience, availability of data and their knowledge of product failure. Error guessing is dependent on the skills, intuition, and experience of the testers.

Exploratory Testing

This technique is used to test the application without any formal documentation. There is minimum time available for testing and maximum for test execution. In exploratory testing, the test design and test execution are performed concurrently.

Test Case Software for managing Test Cases

A test case software can help in writing better test cases and managing them. It also enables you to report bugs from any failed step. A tool provides robust reports generated through built-in filters which also gives you actionable insights.

ReQtest is a test case software preferred by Test Managers in 20+ countries across the globe.

Conclusion

The successful application of test case design techniques will render test cases that ensure the success of software testing.