Mutation Based Test Case Prioritization

By

Faiza Farooq

MCS153013

MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
ISLAMABAD
2017

Mutation Based Test Case Prioritization

By

Faiza Farooq

A thesis submitted to the
Department of Computer Science in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
ISLAMABAD
2017

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY ISLAMABAD

CERTIFICATE OF APPROVAL

Mutation Based Test Case Prioritization

By

Faiza Farooq

MCS153013

THESIS EXAMINING COMMITTEE

S No	Examiner	Name	Organization
(a)	External Examiner	Dr. Asim Noor	Comsats, Islamabad
(b)	Internal Examiner	Dr. Nayyer Masood	CUST, Islamabad
(c)	Supervisor	Dr. Aamer Nadeem	CUST, Islamabad

_____

Dr. Aamer Nadeem

Thesis Supervisor

October, 2017

_____

Dr. Nayyer Masood

Head

Department of Computer Science

Dated : October, 2017

_____

Dr. Muhammad Abdul Qadir

Dean

Faculty of Computing

Dated : October, 2017

Certificate

This is to certify that **Miss Faiza Farooq** has incorporated all observations, suggestions and comments made by the external evaluators as well as the internal examiners and thesis supervisor. The title of her Thesis is: **Mutation Based Test Case Prioritization.**

Dr. Aamer Nadeem
(Thesis Supervisor)

I dedicate my dissertation work to my family, teachers and friends. A special feeling of gratitude is for my loving parents for their love, endless support and encouragement.

ACKNOWLEDGMENT

All worship and praise is for ALLAH (S.W.T), the creator of whole worlds. First and leading, I would like to say thanks to Him for providing me the strength, knowledge and blessings to complete this research work. Secondly, special thanks to my respected supervisor Dr. Aamer Nadeem for his assistance, valuable time and guidance. I sincerely thank him for his support, encouragement and advice in the research area. He enabled me to develop an understanding of the subject. He has taught me, both consciously and unconsciously, how good experimental work is carried out. Sir you will always be remembered in my prayers. I would like to thank Hassaan Minhas and Mubashir Kaleem, students of BS(CS), for implementation of mutation testing tool. I would also like to thank all members of CSD research group for their comments and feedback on my research work.

I am highly beholden to my parents, for their assistance, support (moral as well as financial) and encouragement throughout the completion of this Master of Science degree. This all is due to love that they shower on me in every moment of my life. No words can ever be sufficient for the gratitude I have for my parents. I hope I have met my parents' high expectations.

I pray to ALLAH (S.W.T) that may He bestow me with true success in all fields in both worlds and shower His blessed knowledge upon me for the betterment of all Muslims and whole Mankind.

Aameen

Faiza Farooq

Declaration

It is declared that this is an original piece of my own work, except where otherwise acknowledged in text and references. This work has not been submitted in any form for another degree or diploma at any university or other institution for tertiary education and shall not be submitted by me in future for obtaining any degree from this or any other University or Institution.

Faiza Farooq

MCS-153013

Abstract

To verify that the software is error free and works as intended, software testing is carried out. Testing is an important process since it ensures the software reliability and quality. Software commonly undergoes many modifications after its release. After modifying the software, regression testing is done to ensure that no new faults have been introduced in the previously tested software after modification and the software continues to work correctly. Regression testing is an expensive process since the size of test suite might be too large to execute. Thus to reduce the cost of such testing, three types of regression testing techniques are used which include test case selection, test suite minimization and test case prioritization. The focus of our research is on test case prioritization. Software testers prioritize their test cases so that those test cases which are more likely to detect faults earlier are executed first, in case of not having enough resources to execute the test suite in full. Test case prioritization is mainly used to increase rate of fault detection of test suite. A number of white box and black box techniques have been introduced to prioritize the test cases. These techniques are mostly coverage based prioritization techniques and assign the higher priorities to test cases achieving greater code coverage. These techniques may assign lower priorities to those test cases which have higher potential of exposing faults in the system. A better approach is to prioritize test cases based on their potential of fault detection. There are only few fault based prioritization techniques which assign priorities to test cases based on their fault exposing potential but it is difficult to determine the fault exposing potential because there is no way to find out the total numbers of faults in the program and that what faults are detected by particular test case.

In this thesis, we propose an approach which exploits mutation testing in order to assign priorities to test cases. Using mutation testing, we introduce different changes (mutations) in original program thus creating a number of mutated copies of the program called mutants, and priorities are assigned to test cases based on their potential to find these faults, i.e., kill mutants. This approach can significantly increase the rate of fault detection as the priorities are assigned to the test cases based on their ability of fault detection. We have implemented our tool, MuCap, which is used to generate mutants, generates and executes test cases on those mutants and then prioritizes the test cases based on number of mutants killed. We have evaluated and compared

our approach with existing technique using a number of example programs. The result shows that our approach performs better than existing approaches in terms of APFD.

Contents

List of Figures

List of Tables

Chapter 1 : Introduction

Software development process is complex and error prone. Errors can occur at any stage of the development process. These errors must be detected and removed as early as possible to reduce the development cost, as these errors, if propagated to latter stages of development cycle, become harder to remove. Thus, to verify that software is bug free, testing process is carried out (Baresi and Pezz`e, 2006). In testing process, software is executed using test data and observed behavior is compared with expected behavior to verify that the software works as intended. After the development and testing of the system, it is released for the use.

1.1 Regression Testing

Software may be modified after the release for enhancing its performance, removing the bugs or adding or deleting a module. Whenever software is modified, it is retested to ensure that new errors have not been introduced in the code which was previously tested and it continues to work correctly. Such testing is called regression testing. It provides the confidence that new addition to the software does not change the behavior of the existing part of the software (Elbaum, 2002). Regression testing is carried out between two versions of the software; original and the modified version. It might begin in the development phase after identification and correction of errors by reusing the existing test cases to retest the system. The process of regression testing is shown in Figure 1.1.

Regression testing is mainly a part of maintenance phase where the system is updated, fine-tuned or corrected. Some maintenance requires the change in the specifications of the software and is known as progressive maintenance. Other does not require the specification to be changed but just identification and correction of errors and is known as corrective maintenance. Based on the type of modification/maintenance, Leung and White (1989) define two types of regression testing; one is called progressive regression testing and other is known as corrective regression testing.

- Progressive regression testing is performed whenever specification is changed by adding new features.
- Corrective regression testing is performed when software specification remains unchanged i.e. correction of errors.

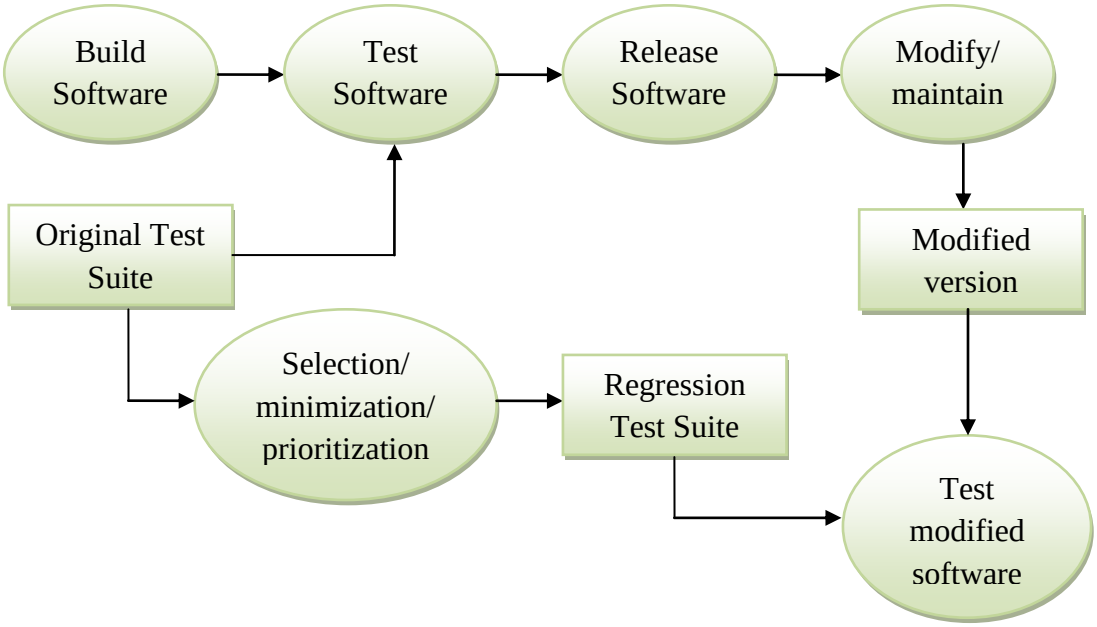

Figure 1.1: Regression Testing

Leung and White (1989) also categorize the test cases for regression testing into five classes where the first three classes including Retestable, Reusable and Obsolete contain the test cases which are already part of the test suite and the other two classes including New-Structural and New-Specification consist of test cases that need to be designed to perform regression testing of the modified program.

- Reusable test cases test the parts of the programs which are unmodified.
- Retestable test cases are those which test the changed parts of the system.
- Obsolete test cases include those that can no longer be used because i) their input/output relation with the system is no more accurate because of the modification, ii) they no longer test, what they were supposed to test, iii) they are

structural test cases that do not provide any structural coverage because of the modifications.

- New-Structural test cases are used to test the modified program code; they are designed to increase structural coverage.
- New-Specification test cases are used to test code generated against modified specification.

The typical approach for the regression testing is the "retest all" approach requiring the software to be executed against every test case in existing test suite. However, it is an expensive process and as the software evolves, the size of test suite tends to increase, making it even more complex and expensive to execute every test case (Burnstein et al., 2003). Thus to minimize the cost of regression testing, different approaches are used including test suite minimization, test case selection and test case prioritization as shown in Figure 1.2.

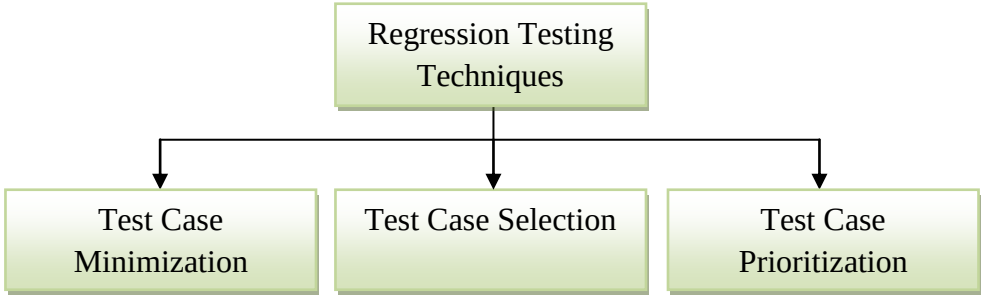

Figure 1.2: Regression Testing Techniques

To limit the cost of regression testing without reducing the overall effectiveness, software testers choose a subset of test suites using these techniques.

- Test suite minimization is used to eliminate test cases which are redundant based on some coverage criteria from the test suite leaving behind the minimal hitting set (Garey and Johnson, 1979). For each test requirement r_i in the testing process, a subset of test cases T_i is created such that each test case in T_i meets requirement r_i. These subsets might overlap. In test suite minimization, T' is found such that it contains test cases which cover all subsets. If all subsets are covered then all requirements are met. Test suite minimization is an NP-Complete problem and

different heuristics have been proposed to solve the problem of minimal hitting set (Chen et al., 1996; Harrold et al., 1993; Horgan et al., 1992; Offutt et al., 1995).

- Test case selection selects the most relevant test cases which are valid and traverse the required parts of the program but is an expensive process since it is a modification aware process requiring the identification of modified parts of the program and selection of the test cases relevant to execution of those identified modified parts of the software (Rothermel and Harrold, 1994).
- The test case prioritization, in contrast, does not eliminate any test cases, rather tends to find an ordered list of test cases which provides the maximum benefits to the software testers. The software testers assign priority to every test case in test suite to ensure that the test cases having higher priorities are run earlier in the testing process (Srivastava et al., 2008).

1.2 Test Case Prioritization

It produces an ordered list of test cases by assigning them priorities based on some criteria so that the tester is provided with the maximum benefit. One testing benefit is increased *rate of fault detection* which is the measure of how early faults can be detected by the test suite (Elbaum, 2002). If maximum number of faults can be detected by minimum number of test cases and those test cases are among the top test cases of priority list then the rate of fault detection increases. Improved rate of fault detection provides the feedback on the system in earlier phases. A priority list can be used to decide when to stop testing in case of having not enough resources to execute all tests. Prioritization will also make sure that if, in any case, testing is halted prematurely, then most important tests, with higher priorities, will have been executed (Yoo and Harman, 2012). Test case prioritization was first introduced by Wong et al. (1997) and was applied to the test cases selected by the RTS techniques. Harrold et al. (1999) and Rothermel et al.(1999) then extended the concept and proposed it in more general context.

Test case prioritization does not select test cases and rather allows every test case to be executed. However, Rothermel discussed that if elimination of test cases is acceptable then in some cases, test case prioritization can be used with test suite minimization and

test case selection, where it is applied to the selected subset of the test suite (Elbaum et al., 2001).

Test case prioritization is formally defined as:

Definition: The Test Case Prioritization Problem:
Given: A test suite 'T' initially designed for original program P; the set of permutations of T, PT; and f: PT → ℝ , a function from PT to the real numbers.
Problem: To find $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$

Finding an optimal ordering of test cases is an NP hard problem and there is no deterministic solution for it (Li et al., 2007).To solve the NP hard problem of prioritization, many techniques and algorithms have been proposed in the literature which are mainly heuristics and give sub-optimal results (Fazlalizadeh et al., 2009). Different algorithms are available for the prioritization including:

- Greedy Algorithm which selects the test cases based on the total number of entities covered. Higher the number of entities covered, higher the priority (Elbaum, 2002).
- Additional Greedy Algorithm is a variation of greedy algorithm which selects the test case based on the total number of entities covered which were not covered by previously selected test case (Elbaum, 2002).
- 2-Optimal Algorithm, which is a variation of K-Optimal algorithm, is form of greedy algorithm used for prioritizing test cases. Like the previously described algorithm, it iteratively chooses test case(s) which achieve maximum coverage of statements not covered by the previously selected test cases. Unlike additional greedy, in each round it adds two best test cases to the prioritized test suite which achieve maximum additional coverage instead of just one test case. When all the entities have been covered, it prioritizes the remaining redundant test cases by resetting the coverage (Lin, 1965).
- Genetic Algorithm (Kaur and Goyal, 2011) in which desired population of chromosomes is used to search optimal solution. The new set of population can be used to replace initial population. Test suites with random orders represent the

chromosomes in population and fitness function of chromosomes is evaluated based on total code coverage.

- Genetic Algorithm & Simulated Annealing Algorithms is a technique which combines Genetic Algorithm (GA) and Simulated Annealing (SA) and is termed as GASA. This Technique takes the advantage of both algorithms to make regression testing more efficient. GASA uses the quick processing property of GA and efficient solution property of SA. Solution is randomly generated by GA and is further refined by SA (Maheswari and Mala, 2015).

Each of the algorithm takes the test suite and coverage information and produces the prioritized list. The algorithm that we are using for our research is Additional Greedy Algorithm which gives the first priority to test case that achieves maximum coverage vector and then assigns the second priority to the test case that covers the maximum number of uncovered entities and so on until all the uncovered entities have been covered.

1.3 Prioritization Criteria

Different criteria are used for prioritization of test case. These prioritization criterions are used for assigning the *award values* to the test cases. The test case having the maximum award value is given the highest priority. Some prioritization techniques use the coverage factor of elements of the source code including statements, branches, spanning entities or functions for assigning the award values (Rothermel et al., 2001). Others use the information about specification of the system including the interaction between the events, requirements priority etc, to prioritize test cases in test suite (Henard et al., 2016). The prioritized lists created using these criterions are then evaluated. The basic parameter for the evaluation of prioritization techniques is rate of fault detection i.e. Average Percentage of Faults Detected (APFD) which is the measure of how early faults are detected in the testing process by the test suite (Rothermel et al., 1999). APFD is calculated using following formula:

$$APFD = 1 - \frac{TF_1 + \cdots \ldots . TF_m}{nm} + \frac{1}{2n}$$

Where T is the test suite with n test cases and F is the set of m faults identifies by T. For ordering T', TF_i represents the order of the first test case that exposes the *ith* fault F_i.

1.4 Problem Statement

A lot of research work has been done in this area, many white box and black box techniques have been developed to prioritize the test cases for increasing the rate of fault detection. Most of those techniques are coverage based and only consider if a branch, statement or function has been covered by the test case. In coverage based techniques, it is assumed that the coverage will maximize the rate of fault detection but this relation between coverage and fault detection rate does not always hold. The test case with the better ability to find the faults may be given the low priority. There are only few existing test case prioritization techniques which prioritize test cases based on their ability to expose faults but the fault exposing potential of the test case cannot be calculated accurately.

1.5 Research Questions

In this research work, we will use mutation testing as criteria to generate a priority list by using the mutants of a given program. However, the following questions must be taken into account:

***RQ. 1*:** What are the existing and most commonly used test case prioritization techniques?

To answer this research question, a literature survey is conducted through which we identify the existing and most commonly used techniques.

***RQ. 2*:** What are the gaps in the existing prioritization techniques?

Through study of existing techniques in literature in detail, we identify such gaps.

***RQ. 3*:** How well does the mutation based prioritization technique compare with the well studied white box and black box prioritization techniques with respect to fault detection rate?

A number of experiments are performed on different subject programs and their respective test suites and APFDs of each test suite are calculated for proposed and existing techniques to perform comparison.

Our research will be focused to answer these research questions with reference to the prioritization algorithm.

1.6 Research Methodology

1. First of all, we have done literature review to identify the most relevant and most commonly used white box and black box prioritization techniques. After studying various prioritization techniques, we have reached the conclusion that these techniques are coverage based. There are only a few fault based prioritization techniques.
2. To overcome the gap in existing techniques, we have proposed a new approach that will assign priority to each test case based on its ability to expose maximum number of faults, i.e. killing the maximum number of mutants.
3. The implementation of our approach will be performed in following steps:
   i. In the first phase, we collect all data including subject programs. After collecting the data, we create mutants of each program by applying different mutation operators.
   ii. After having created the mutants, in next phase, we collect test suite of each subject program, then mutation testing is performed; each mutant is selected and executed against tests in test suite until it shows a different behavior from original program or it has been executed against all tests. We compare the original program's output with the output of each mutant, if the mutant has different output against a selected test case, the mutant will said to be killed by that test case. We will note the number of mutants killed by each test case.
   iii. In next phase, we assign priority to each test based on number of mutants killed. Greater the number of killed mutants by a test case, higher will be its priority in the priority list.

iv. We then generate another prioritized list for same program and its corresponding set of test cases by using some existing coverage technique, since white box prioritization techniques are more commonly used, so we use white box technique.

4. After creating both prioritized lists (white box coverage based and mutation based), we perform a comparison between both techniques. The rate of fault detection will be used as the main parameter for the comparison.

1.7 Research Contributions

Research contribution in this thesis is to propose a technique which prioritizes the test cases based on their ability to kill mutants generated using mutation testing. In mutation testing, a slight change is introduced in the original program by using any mutation operator, thus creating a mutant of a program. Each mutant represents a faulty version of the program. The test case that exposes the maximum number of those faults, i.e., killing the maximum number of mutants is said to be most efficient test case and will be given the highest priority. We have also evaluated existing technique on our test data and compared it with our approach.

1.8 Thesis Organization

Rest of the thesis is organized as, in Chapter two existing test case prioritization techniques are discussed, mainly test cases can be prioritized using either white box prioritization techniques or using black box prioritization techniques. Both types of techniques are discussed in this chapter. Third chapter is about proposed solution, fourth chapter is on implementation details, fifth chapter is about results and discussion and sixth chapter is on conclusions which we have made from this research and future work on how we can extend this.

Chapter 2 : Literature Review

The prioritization techniques fall into two categories: White box and Black box prioritization.

- In white box prioritization techniques code of the software is used to define criteria for prioritizing the test cases. These techniques are mainly coverage based techniques.
- Black box prioritization techniques have no structural information; rather use the specification of the software to prioritize test cases, for example, requirements priorities, events in event driven systems, etc, are used for prioritization.

This chapter contains an overview of techniques which are already proposed by different researchers to prioritize the test cases.

2.1 White Box Prioritization Techniques

Such prioritization techniques are based upon the source code of the software. These techniques are only differentiated by the elements of source code they use for defining the prioritization criterion. Different white box techniques are defined below.

2.1.1 Prioritization techniques based on coverage

Elbaum et al. (2001) introduced two types of white box prioritization strategies:

a) Coverage-based techniques that prioritize test cases based on their coverage of elements of source code. These coverage based techniques are further divided into two categories; a) Total coverage based prioritization which assigns priorities to test case based on the total number of functions, branches or statements covered. b) Additional coverage based prioritization, which assigns priorities to test cases based on the additional entities, i.e., functions, branches or statements covered. The test case that covers maximum number of uncovered entities is given highest priority.

b) Fault based techniques that prioritize test cases based on their fault exposing potential. For calculating the fault exposing potential of test cases, they used mutation testing. For each statement, number of mutants is created and mutation score of each test case for each statement is calculated and the test case that has the maximum accumulated mutation score is given the highest priority. The test cases are ordered in the descending order of their mutation score.

2.1.2 History based prioritization

History-based prioritization was introduced by Kim and Porter (2002) which is based on the history of test cases in order to prioritize them. In this technique, information from previous execution cycles was used as criteria for selection of subset of test suite that must be executed for a modified program. RTS technique was applied to test suite T that produced T' in the 1st step. After that in 2nd step, every test in T' was assigned selection probability. In 3rd step, probabilities assigned in previous step were used to select a test case and run it. The final step includes the repetition of 3rd step until the testing time is finished.

The selection probability was assigned to each test case based on its prior performance. They used test histories based on execution history, fault detection rate and program entities covered.

2.1.3 Additional Spanning Entities Coverage based Prioritization

Marre´ and Bertolino (2003) introduced a concept of spanning set of entities to be used as criterion to prioritize the test cases. They used the subsumption relationship between entities to find the spanning set, i.e., given two entities E and E', if every complete path that covers E also covers E', then E subsumes E', but there exist some entities that are unconstrained, i.e., an entity E is called unconstrained if it is not subsumed by any other entity E' without being itself subsumed by E, the smallest subset of such unconstrained entities is called a spanning set with a property that any subset of test cases that covers this subset of entities covers every entity in the program. Their technique was divided further into two approaches: Additional Spanning Statements and Additional Spanning

Branches. The test case that covers the maximum number of uncovered spanning statements or branches is given the highest priority.

2.2 Black Box Prioritization Techniques

Black box prioritization techniques use the specification of the system to define the prioritization criterion and have no knowledge of structure of the software. Different black box techniques are defined below.

2.2.1 Interaction Coverage Based Prioritization

Managing test suite for event driven systems is difficult as the number of event combinations and sequences grow exponentially with the number of events. Bryce and Memon (2007) proposed a testing technique which extends the t-way software interaction over sequences of events. Their proposed algorithm greedily selects a test case that covers the maximum number of previously uncovered t-tuples of event interactions between unique windows. If more than one test case covers same number of event interactions, the tie is broken randomly.

2.2.2 Requirements clustering based prioritization

Software under test may have many requirements which are of high priority. All requirements are not equally important. Arafeen and Do (2013) introduced a new approach which uses requirements information to prioritize the test cases. This approach uses a text mining approach for extraction of useful words from the text. Based on those words, relevant requirements are clustered and test cases are prioritized in these requirement clusters. Within a cluster, test cases are prioritized using code complexity. After that, clusters are prioritized using code modification information and customer assigned requirements priority. Finally, after having prioritized test cases and clusters, test cases are selected from clusters by visiting each cluster using different selection methods.

2.2.3 Prioritization of Requirements for Testing (PORT)

Value driven techniques improve the user based software quality by utilizing testing effort on the requirements which are of highest priority to the customer. Srikanth et al. (2005) proposed a value based technique named as "*prioritization of requirements for testing*", which contains four factors values: (1) customer-assigned priority of requirements; (2) developer-perceived implementation complexity; (3) requirements volatility; and (4) fault proneness.

Firstly, PORT calculates Prioritization Factor Value (PFV) of all requirements. Prioritization factor value "is the summation of the product of factor value and the assigned factor weight for each of the factors". The importance of testing a requirement is obtained from Prioritization factor value of that requirement. Value of PFV for a requirement is necessary for calculation of WP (weighted priority) of its associated test case. Weighted priority is "the PFV contribution of the requirement(s) the test case maps". Test cases are arranged on the basis of weighted priority value in such a way that test case with high weighted priority value is assigned highest priority and so on.

2.2.4 History based test case prioritization

Source code of a program is not available in black box environment. Only limited information is present to prioritize the test cases. Qu et al. (2007) introduced a black box technique for prioritizing the test cases. This technique uses the run-time and test history information. History information is used to initialize the test suite. Test case relation matrix R, which depicts the fault detection relationship of test cases, is formed by using available information. Then a test case is drawn from test suite and it is run. Remaining test cases are ordered using test case relation matrix and run time information.

2.2.5 A Hierarchical System Test Case Prioritization Technique

Many approaches based on requirements have been introduced to prioritize test cases. However, along with requirements, other factors including implementation and test case complexity can also contribute in test case prioritization. Kumar et al. (2013) introduced a hierarchical test case prioritization approach based on requirements. In this approach, the prioritization process is performed at three levels. First of all, each requirement is

assigned a priority based on 12 different factors including customer assigned priority, developer assigned priority, requirement volatility, fault proneness, expected fault, implementation complexity, execution frequency, traceability, show stopper requirement, penalty, cost and time. Customer, developer, analyst and tester assign values to those requirement factors. The higher the value of these factors, higher the priority of that requirement. After getting the prioritized list of requirements, a mapping between each requirement and its corresponding modules is performed. If a requirement has more than one corresponding modules then the modules are prioritized based on cyclomatic complexity and non dc paths. Module with higher cyclomatic complexity and non dc paths is given the highest priority. The last level of prioritization process includes the test case prioritization. After getting the prioritized list of modules, the mapping between a module and its corresponding test cases is performed. The test cases corresponding to modules are then prioritized based on 4 factors including test impact, test case complexity, requirement coverage and dependency. The resulting test suite is the prioritized test suite.

2.3 Comparison

Each above technique produces an ordered list of test cases based on their respective criteria. All of the prioritized techniques are then evaluated using some faulty programs. Such faulty programs are usually developed through mutation rather than hand-seeded faults since the mutation faults are the representative of real faults. Hand seeded faults can be problematic for validity of results (Do et al., 2005).

Based on different programs, the above described techniques yielded the following APFD (Henard et al., 2016).

Table 2.1: Comparison of White box prioritization techniques

No.	Technique	Information required	Tool Support	APFD
1	Total Function	Function Coverage	Yes	69%
2	Additional Function	Uncovered Functions	Yes	87%
3	Total Statement	Statement Coverage	Yes	67%
4	Additional Statement	Uncovered Statements	Yes	89%
5	Total Branches	Branch Coverage	Yes	67%

6	Additional Branches	Uncovered Branches	Yes	90%
7	Total FEP	Fault Exposing Probability	Yes	88%
8	Additional FEP	Confidence Factor	Yes	89%
9	Additional Spanning Statements	Spanning statements and their coverage	No	89%
10	Additional Spanning Branches	Spanning branches and their coverage	No	90%
11	History Based	Exec. History, APFD and program entities covered	Yes	N/A

Table 2.2: Comparison of Black box prioritization techniques

No.	Technique	Information required	Tool Support	APFD
1	T-wise	Event Sequences	No	87%
2	Requirements Clustering based	Customer assigned Req. priority, code complexity, modification information	Yes	N/A
3	PORT	PFV & WP	Yes	47%
4	History Based	Relation matrix and run time information	No	N/A
5	HSTCP	12 Factors value, cyclomatic complexity, non dc paths, test impact, test case complexity, requirement coverage and dependency	Yes	67%

Out of all white box prioritization techniques, Additional Branch Coverage and Additional Spanning Branches have yielded the highest rate of fault detection and out of 5 black box prioritization techniques, t-wise yielded the greatest APFD.

2.4 Gaps Analysis

Besides APFD, there are certain parameters which can be used to compare black box and white box prioritization techniques, such as, black box prioritization techniques are not code coverage based which means they do not need code for prioritizing the test cases

which is a benefit since in the component based software development process, components from a third party can also be adopted and used without having the source code of that particular component. White box prioritization techniques need code coverage information to assign the priorities to test cases and cannot be used when the source code is unavailable. Among the above mentioned techniques, the Total coverage based techniques have some drawbacks as compared to Additional coverage based techniques. When using Total coverage, the test cases are prioritized based upon the total number of entities covered. There can be more than one test case that covers the same entities. So there is repetition of the entities covered which is not desirable. Additional coverage eliminates this drawback by assigning the priorities based upon the coverage of uncovered entities. Total coverage also makes it possible to skip any entity, for example, if a test case only covers one entity which is not covered by any other test case and that test case is given the lowest priority, then in case of premature halting of testing process, there is a possibility that the test case with lowest priority is never executed and that entity covered by the test case is skipped.

All of these techniques are coverage based techniques, white box prioritization techniques use code coverage and black box prioritization techniques use specification coverage information. These techniques are not fault based, only fault based techniques are FEP total and FEP additional but those prioritize test cases based on their ability to expose faults but the fault exposing potential of the test case cannot be calculated accurately.

Chapter 3 : Proposed Approach

From the literature survey, we have concluded that most of the existing prioritization techniques use coverage information of the test cases to prioritize them. White box prioritization techniques use the code coverage information and black box prioritization techniques use the specification. These techniques cannot be used when the coverage information is not available. There are only few techniques which are fault based prioritization techniques including total FEP and additional FEP, but these fault based techniques use probability of fault detection of test cases which cannot be calculated accurately. We have proposed an approach that uses the ability of test cases to detect faults, i.e., mutation testing, in order to prioritize them.

Because of the use of fault detection ability of test cases rather than coverage, it is likely that mutation based prioritization will yield better prioritization in terms of APFD as compared to existing prioritization techniques.

3.1 Mutation Testing

Mutation testing is a fault based technique which is used to measure the effectiveness of a test suite (Ma and Offutt, 2006). In mutation testing, we take the original program and introduce a slight change using mutation operators in the original code thus creating a mutant which represents a faulty version of the program. Some of the mutation operators are listed below:

Table 3.1: Mutation Operators (Offutt et al., 1993)

Mutation Operators	Description
AAR	Array reference for array reference replacement
ABS	Absolute value insertion
ACR	Array reference for constant replacement
AOR	Arithmetic operator replacement
ASR	Array reference for scalar variable replacement
CAR	Constant for array reference replacement
CNR	Comparable array name replacement

CRP	Constant replacement
CSR	Constant for scalar variable replacement
DER	DO statement alteration
DSA	DATA statement alteration
GLR	GOTO label replacement
LCR	Logical connector replacement
ROR	Relational operator replacement
RSR	RETURN statement replacement
SAN	Statement analysis
SAR	Scalar variable for array reference replacement
SCR	Scalar for constant replacement
SDL	Statement deletion
SRC	Source constant replacement
SVR	Scalar variable replacement
UOI	Unary operator insertion

In mutation testing, mutants are generated by applying mutation operators resulting in mutated copies of the original program. After the generation of mutants, in next step, these mutants are tried to be killed using the test data. If the mutant gives different output than the original program against the same test case, then the mutant is said to be killed by that test case. Each mutant is executed against each test case and it is monitored that whether it gives different output from the original program. An example of mutant is shown below:

Table 3.2: Mutant example

Original Program	**Mutated Program**

i f (a > b)
{
cout <<”a is greater than b”<<endl ;
}
	i f (a < b)
	{
	cout <<”a is greater than b”<<endl ;
	}

A mutant that is not killed by any test in test suite remains “alive”. Alive mutant can be killed by enhancing the test suite and adding more test cases which can kill those alive mutants. However, a mutant is considered equivalent to its parent program if there exists no test that causes original and mutant program to generate different outputs. Equivalent mutants always give same output as the original program and are not considered while calculating the fault detection ability of test suite.

3.2 Mutation Based Prioritization

In mutation based prioritization, we will prioritize the test cases based on the number of mutants killed by test cases. The test case that kills the maximum number of mutants will be given the highest priority and the test case that kills the maximum number of unkilled mutants will be given second priority and so on. We have used the additional approach to prioritize the test cases because of the drawbacks of the total approach. The only information that is needed in this approach is the mapping between the test cases and mutants killed. When the system under test i.e. SUT, is first tested, all the information can be gathered and we use the assumption that we have the mapping between the test cases and mutants so we will use this information as an input to our prioritization algorithm which in return will generate a priority list based on mutation. Proposed solution context diagram is shown in Figure 3.1.

Listed below steps are involved to achieve our goal:

- **Mutants Generation**
- **Mapping of test cases to mutants**
- **Applying prioritization algorithms to generate priority list**

3.2.1 Mutants Generation

In our approach we have used ROR operator to generate mutants of the original program. ROR operator sees which relational operator is used in a condition and replaces it with other relational operators. ROR operators are listed in the table below:

Table 3.3: Relational operators

Relational Operators
>
<
< =
> =
= =
! =

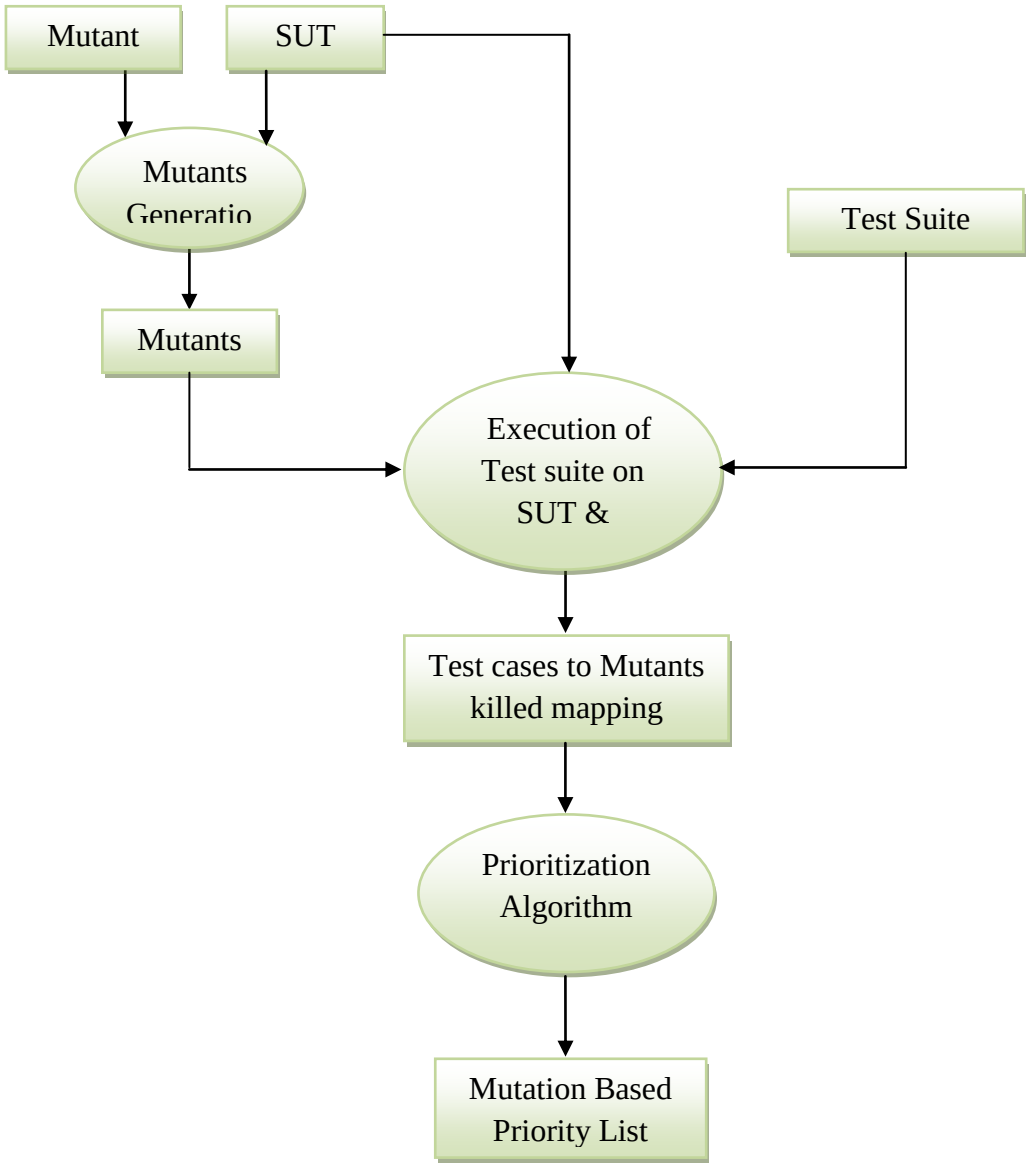

Figure 3.1: An illustration of proposed solution

ROR operator generates 7 mutants against one condition in the original program having relational operator including any 5 from the above mentioned and 2 others replacing relational operator with "*true"* and *"false"* values.

For example, consider the following conditions in the original program:

```
if (marks > 80 && attendance >= 30)
            {
             grade = "A";
```

```
            }
else
          if (marks > 80 && attendance<30)
            {
            grade = "B";
            }
```

There are two decisions in the original program with 2 conditions per decision. Against 4 conditions, 28 mutants will be generated as follows:

Table 3.4: Mutated conditions of the program

Original Condition	Mutated Condition
1. marks > 80	1. marks > = 80
	2. marks < 80
	3. marks < = 80
	4. marks = = 80
	5. marks != 80
	6. true
	7. false
2. attendance > = 30	8. attendance > 30
	9. attendance < 30
	10. attendance < = 30
	11. attendance = = 30
	12. attendance != 30
	13. true
	14. false
3. marks > 80	15. marks > = 80
	16. marks < 80
	17. marks < = 80
	18. marks = = 80
	19. marks != 80
	20. true
	21. false
4. attendance < 30	22. attendance > 30
	23. attendance > = 30
	24. attendance < = 30
	25. attendance = = 30
	26. attendance != 30
	27. true
	28. false

There will be 28 mutant programs of the original program having one of these mutated conditions each.

3.2.2 Mapping of test cases to mutants

After creating these mutants, in next step, test data is used to kill these mutants. We collect all the information about mapping of the test cases to mutants. In this phase, we record which test case kills which mutant by executing the test cases on SUT and mutants.

For example, consider the following test cases for the original program:

Table 3.5: Test cases for the program

Test case#	marks	attendance
1	0	0
2	0	1
3	0	15
4	0	29
5	0	30
6	80	0
7	80	30
8	81	0
9	81	30
10	81	31

The above mutants will be executed one by one against each test case and if against any test case they produce different output than the original program then the test case will be said to have killed the mutant. For example, consider the following test case and mutant:

Test case		**Mutated Condition** marks < 80	**Original Condition** marks > 80
Attendance	**Marks**	**Mutant Output**	**Original Output**
0	0	B	null

Since the output of the original program is different than mutant program, it is said to be killed by the test case. Similarly all the mutants will be executed one by one against each test case and their output will be compared with the output of the original program generated against that particular test case. When all the mutants have been executed

against each test case, a mapping between test cases and mutants will be obtained. For the above example, the following mapping exists:

Table 3.6: Mapping of the test case to mutants

Test cases	Mutants																									
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	25	28	
1																x	x	x	x							
2																x	x	x	x							
3																x	x	x	x							
4																x	x	x	x							
5		x	x		x	x																				
6															x		x	x		x						
7	x		x	x		x																				
8									x	x		x	x				x	x	x			x	x	x	x	x
9		x	x	x			x	x	x			x			x											
10		x	x	x			x		x	x	x				x											

The above data will then be used to assign priorities to the test cases.

3.2.3 Applying prioritization algorithm

After collecting the above data, this data will be used as an input to the greedy prioritization algorithm which is mutation based prioritization algorithm generating mutation based priority list. The algorithm is given below in Figure 3.2.

The algorithm, in first step, will find a test case that kills the maximum number of mutants. In next step, that test case will be added to priority list and it will be removed from the test suite and it will not be further considered, also the entitiesCov data will be updated to indicate only the uncovered entities. In third step, a loop will start and it will continue executing until all the entities have been covered and the test suite has no unprioritized test case. In the loop, residual coverage for all the test cases will be calculated which indicates the number of entities that will remain uncovered after the execution of a particular test case. Test case having the minimum residual coverage will be added to the priority list, if two or more such test cases exist, then the test case is selected randomly and it will be removed from the test suite for further consideration, entities coverage information will also be updated to indicate the uncovered entities. The process is repeated until all entities have been covered by atleast one test case or the test suite has no unprioritized test case. If all the entities have been covered and there are still

unprioritized test cases left in the test suite then they will be appended to the priority list randomly as they are the redundant test cases.

File Edit Format View Help

```
Procedure for prioritizing regression tests:

Input:
1.      T': Set of regression tests for P'
2.      EntitiesCov: Set of entities in P covered by tests in T'.
3.      cov: Set of entities covered by executing P against t.
4.      X': A temporary set of regression tests used for calculations.

Output:
1.      PrT: A sequence of tests.

Procedure: PrTest

Step 1: X'=T'. Find t in X' such that cov(t) ≥ cov(u) for all u in X', u
≠ t.

Step 2: Set PrT=< t >, X'= X' \ {t}, entitiesCov= entitiesCov \ cov (t).

Step 3: Repeat while X' ≠ ∅ AND entitiesCov ≠ ∅.
   Step 3.1: Compute residual coverage for each test t in T'   which
   indicated the total number of mutants which will be left unkilled
   after executing test case t.
            resCov (t) = entitiesCov \ [cov(t) ∩ entitiesCov]

   Step 3.2: Find t in X' such that resCov (t) ≤ resCov (u) for all u in
   X', u ≠ t, if two or more such tests exist, select randomly.

   Step 3.3: Set PrT= append (PrT, t), X'=X' \ {t} and entitiesCov=
   entitiesCov \ cov (t)

Step 4: Append to PrT any remaining tests in X'.  All remaining tests
have same residual coverage. All tests are tied. Random selection is one
way to break the tie.
```

Figure 3.2: Greedy Algorithm for Prioritization

To further elaborate the algorithm, consider the information given as an input to the mutation based prioritization algorithm:

T' = {t1, t2, t3, t4, t5, t6, t7, t8, t9, t10}

EntitiesCov = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,25,28}

Table 3.7: Mapping of the test case to mutants

Test case(t)	Mutants killed	cov(t)
1	16,17,19,20	4
2	16,17,19,20	4
3	16,17,19,20	4
4	16,17,19,20	4
5	2,3,5,6	4
6	15,17,18,20	4
7	1,3,4,6	4
8	9,10,12,13,16,17,18,21,22,23,25,28	12
9	2,3,4,7,8,9,12,14	8
10	2,3,4,7,9,10,11,14	8

Highest priority will be given to *t8* because it kills the maximum number of mutants and then will eliminate all the mutants killed by *t8* to updates the coverage information of unprioritized test cases to indicate their coverage of mutants that have not been yet killed and repeats the process until all mutants have been killed by atleast one test case.

In the first step, t8 will be selected and in step 2, data will be updated like this:

PrT = {t8}

T' = {t1, t2, t3, t4, t5, t6, t7, t9, t10}

EntitiesCov = {1,2,3,4,5,6,7,8,11,14,15,19,20}

Then in step 3 the residual coverage will be calculated for each test case until entitiesCov && T' don't get empty:

The residual coverage for all the test cases will be calculated as follow in step 3.1:

resCov(t1)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{16,17,19,20}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {11}

resCov(t2)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{16,17,19,20}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {11}

resCov(t3)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{16,17,19,20}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {11}

resCov(t4)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{16,17,19,20}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {11}

resCov(t5)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{2,3,5,6}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {9}

resCov(t6)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{15,17,18,20}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {11}

resCov(t7)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{1,3,4,6}∩{1,2,3,4,5,6,7,8,11,14,15,19,20}]

= {9}

resCov(t9)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{2,3,4,7,8,9,12,14}∩{1,2,3,4,5,6,7,8,11,14, 15,19, 20}]

= {7}

resCov(t10)={1,2,3,4,5,6,7,8,11,14,15,19,20}\[{2,3,4,7,9,10,11,14}∩{1,2,3,4,5,6,7,8,11,14,15,19, 20}]

= {7}

In step 3.2, the residual coverage of all the test cases will be compared to find the test case having minimum residual coverage. Since t9 and t10 have minimum resCov so any of them can be selected randomly. t9 will be selected.
In step 3.3 the data will be updated again as follow:

PrT: {t8, t9}

T' = {t1, t2, t3, t4, t5, t6, t7, t10}

entitiesCov = {1, 5, 6, 11, 15, 19, 20}

Since the entitiesCov and T' are not empty thus the process will continue and step 3 will be repeated again until any of them becomes empty.

In step 4, the algorithm will yield the following prioritized list:

Mutation Based Priority List (PrT): {8, 9, 1, 5, 6, 7, 10, 2, 3, 4}

In the end, we will compare our prioritization technique with already existing white box prioritization technique. Since the branch coverage is the strongest prioritization approach we will use it for comparison.

Chapter 4 : Implementation

This chapter includes the implementation details of the proposed solution. For implementation, we have used MuCap to automate the process. The tool has three main components. First component automates the process of mutation testing, second component generates, executes the test cases, generates and displays report and third component prioritizes test cases based on the report produced by second component using greedy prioritization algorithm explained in previous chapter. The tool has been created using Netbeans IDE 8.2 and Java language.

The architecture of the tool is given below in figure 4.1:

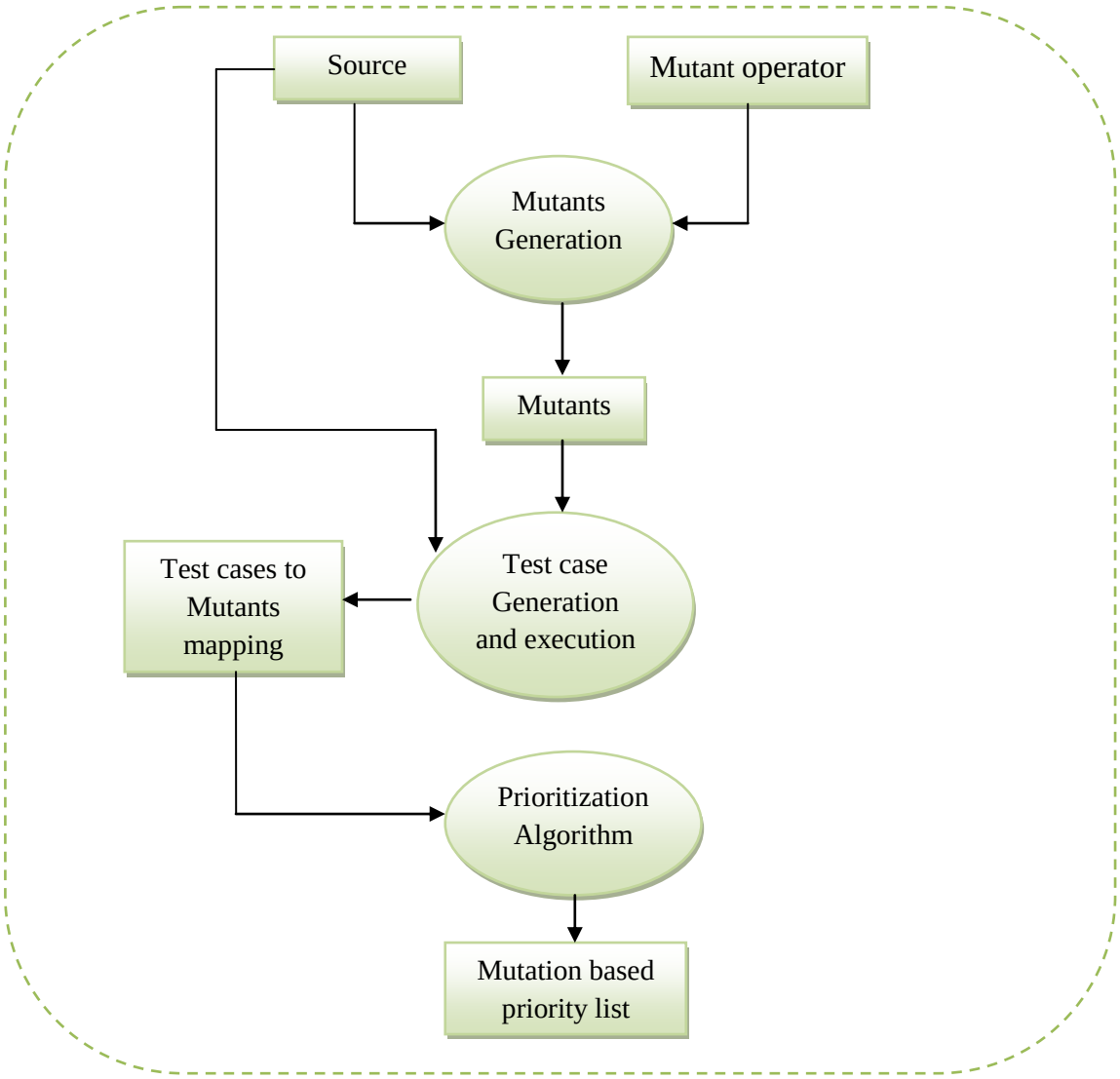

Figure 4.1: MuCap

The detailed architectural diagrams of each component are given below in Figure 4.2, 4.3 and 4.4 which show all the process involved in each component.

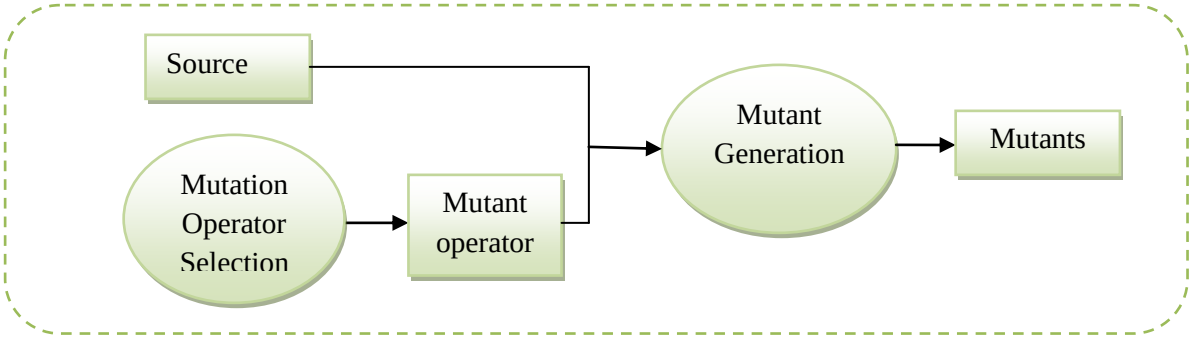

Figure 4.2: Architecture diagram of mutant generator

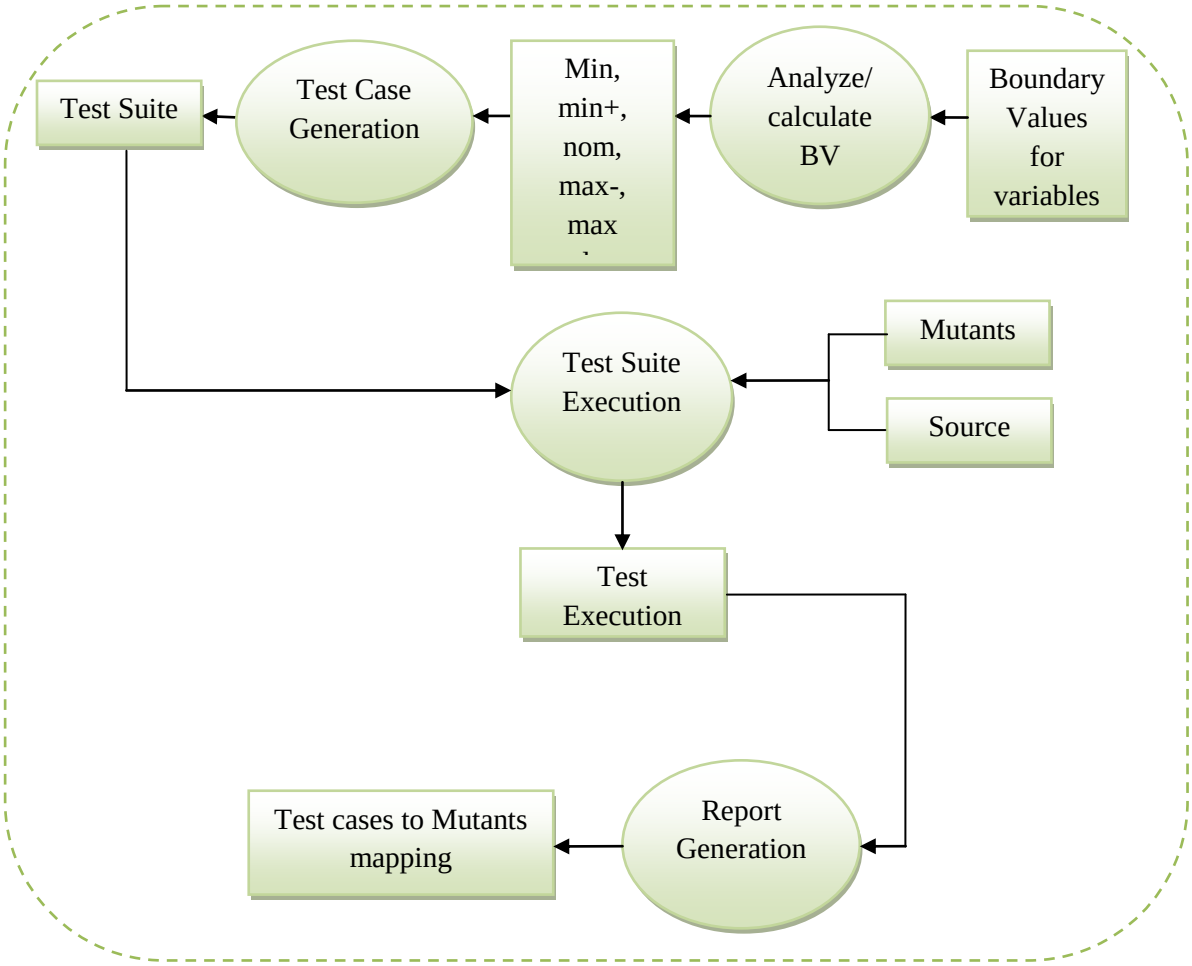

Figure 4.3: Architecture diagram of test suite generator and executer

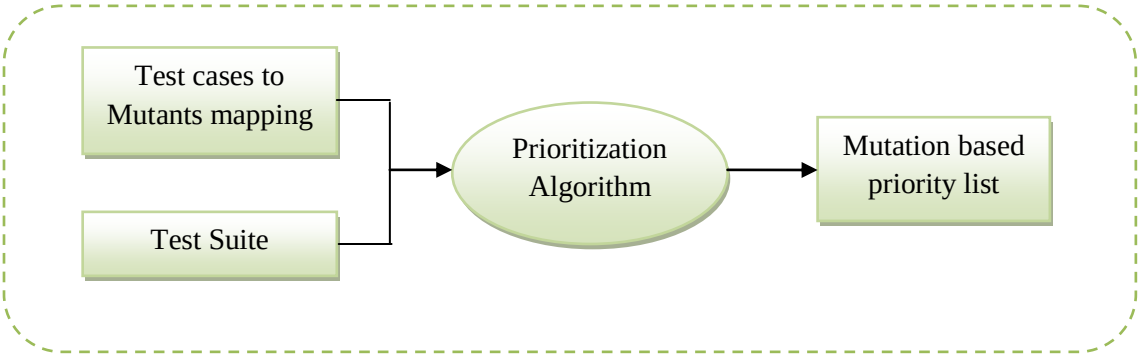

Figure 4.4: Architecture diagram of test suite prioritization

4.1 Implementation details

This section includes the implementation details of all three components of mutation testing tool. It describes all the classes and their methods used for each of the component. Class diagram of Mutation Testing Tool has been divided into two parts: The first diagram is of the mutant generation component. The second diagram shows the second component

The class diagrams of first and second components are shown below in Figure 4.5 and Figure 4.6 respectively:

4.1.1 Mutant Generation

This module has been implemented using 16 main classes: the first class is the controller class and other classes are for mutation operators.

- Mutation Operator Classes: For each operator, separate classes have been defined and different methods have been used that perform certain tasks according to requirements. For example, in ROR, the main method ***call_ROR( )***, is created to find the relational operator in the program and replace it with other 5 relational operators and whole condition with ‘true’ and ‘false’ values one by one. Similarly, in AOR, the main method that has been created finds the basic arithmetic operator and replaces it with other 4 arithmetic operators.

- Controller Class: In this module controller class has also been used that performs the main functionality. It has 15 main methods for 15 mutation operators, as shown in above class diagram, which are responsible for generating separate files for the mutants created by applying selected operator. When the user selects a mutation level operator, the controller class creates two objects, one for calling method level mutation operator class and other for calling one of the 15 methods of controller class according to the selection of mutation operator. For example, if the user selects ROR from the list of operators, then the controller class will create an object which will call ***call_ROR( )*** method of ROR class to generate mutants of the program, the other object will be used to call the ***ROR_Mutants( )*** method of controller class which will generate text files for storing each mutant generated against ROR separately named as Mutant 01, Mutant 02 and so on in the project's folder.

4.1.2 Test Case Execution

This module has 5 different classes for 5 different functionalities.

- GetBoundary: In this class, the main method which performs the core functionality is ***readFunctionParametersFile( )***. This method is responsible for getting the boundary values for each variable in the source program from the text boxes of user interface and generating other three values including min+, nom, max- for all variables. It also stores
- BoundaryConfirmation: In this class, method ***createTestCases( )*** is responsible for carrying out main functionality. This method is used to create test cases by using worst case boundary value testing technique which generates test cases by making all the combinations of all 5 values for all variables, so for 'n' variables there will be 5^n test cases (Jorgensen, 2010). Test cases after creation are stored in a text file for later use.
- Execute_Test_Cases: This class has four main methods for performing main functionalities. The first method ***executeTestCases( )*** is used to read each test case one by one from test cases' text file and then the method writes this test case in

original program and in each mutant one by one. A programming logic is used to find the exact location in original program and in mutants, where the function calling statement is written. When location of function call in a code file is found then the test case is written within the parameters of that function and then this file is saved with .java extension. The second method used is ***compileProgram( )*** which is responsible for compiling the .java file created by the previous method. The class file generated is then given to ***executeClassFile( )*** for execution. The fourth method ***programOutput( )*** is used to store the output of original program and mutants against each test case in a text file.

- GenerateReport: For report generation, the main method ***comparison( )*** has been implemented using a programming logic which reads file in which the outputs of original program and all mutants are written. Then a comparison is made to find out, against each test case, the number of killed mutants. If the output of mutant is different than the original program against a particular test case then the mutant is said to be killed. The other method ***shortReport( )*** is used to save each test cases with the mutants killed by it in a text file.
- DisplayReport: this class has different methods for displaying the report in the form of a table. The method ***findNoOfKilledMutants( )*** is used to find the total number of mutants killed by the test suite. Methods ***populateTable( )*** and ***setRowsinTable( )*** are used to generate table by setting columns and rows respectively. ***setRowsinTable( )***reads the report file and fills in the rows with test cases and its killed mutants.

4.1.3 Prioritizing Test Cases

The report generated by the previous tool will be used as an input to the program which is the coverage information of all the test cases. Based on this information, the prioritization algorithm which has already been discussed in the previous chapter will be used to assign priorities to the test cases. The algorithm will select a test case which kills the maximum number of mutants and will assign the highest priority to that test case, after that the coverage information will be updated to indicate mutants left unkilled. The algorithm will

then find the residual coverage of all the test cases for assigning priorities until all the mutants have been covered or the test suite has no unprioritized test case left.

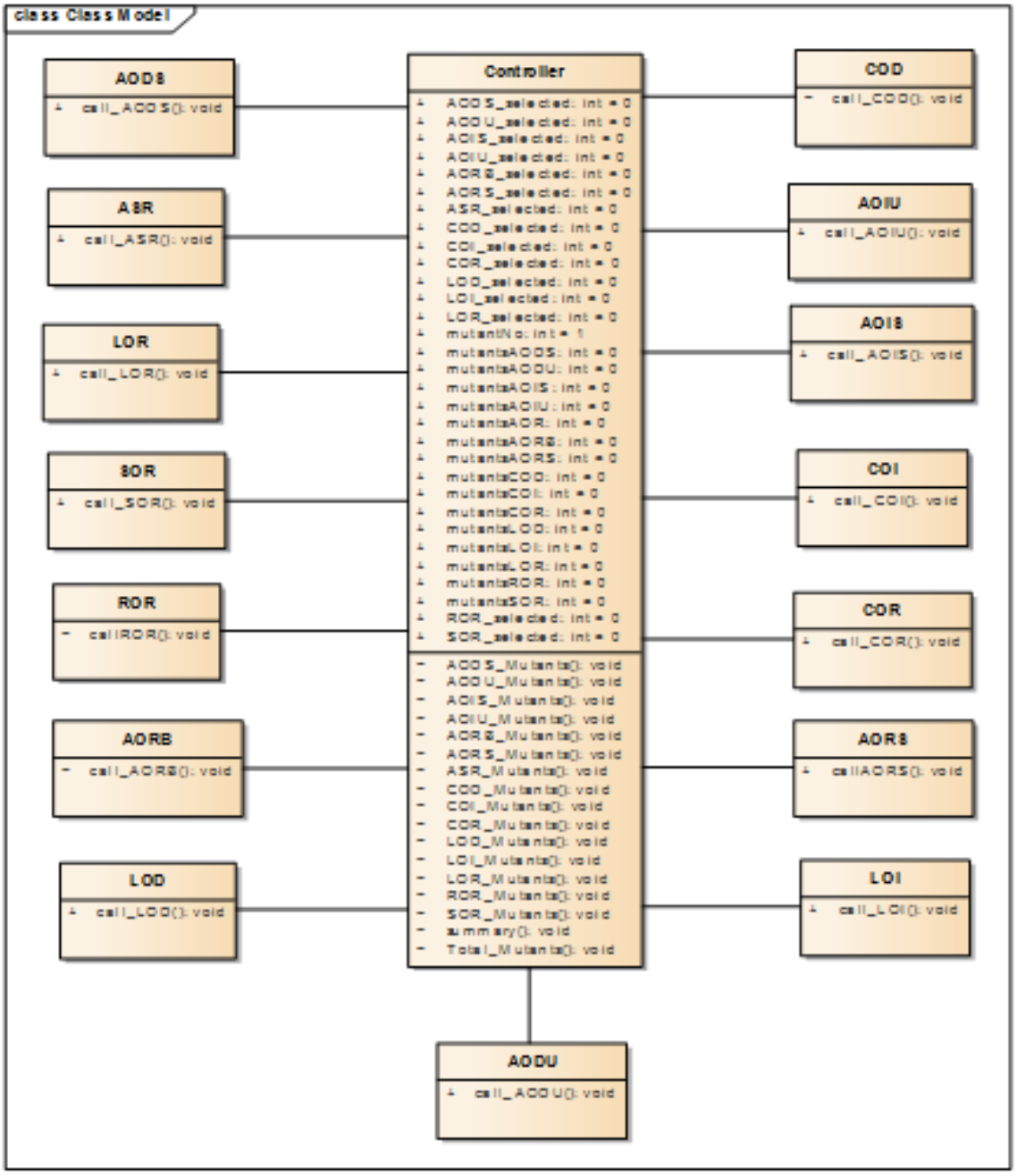

Figure 4.5: Class diagram of mutant generator

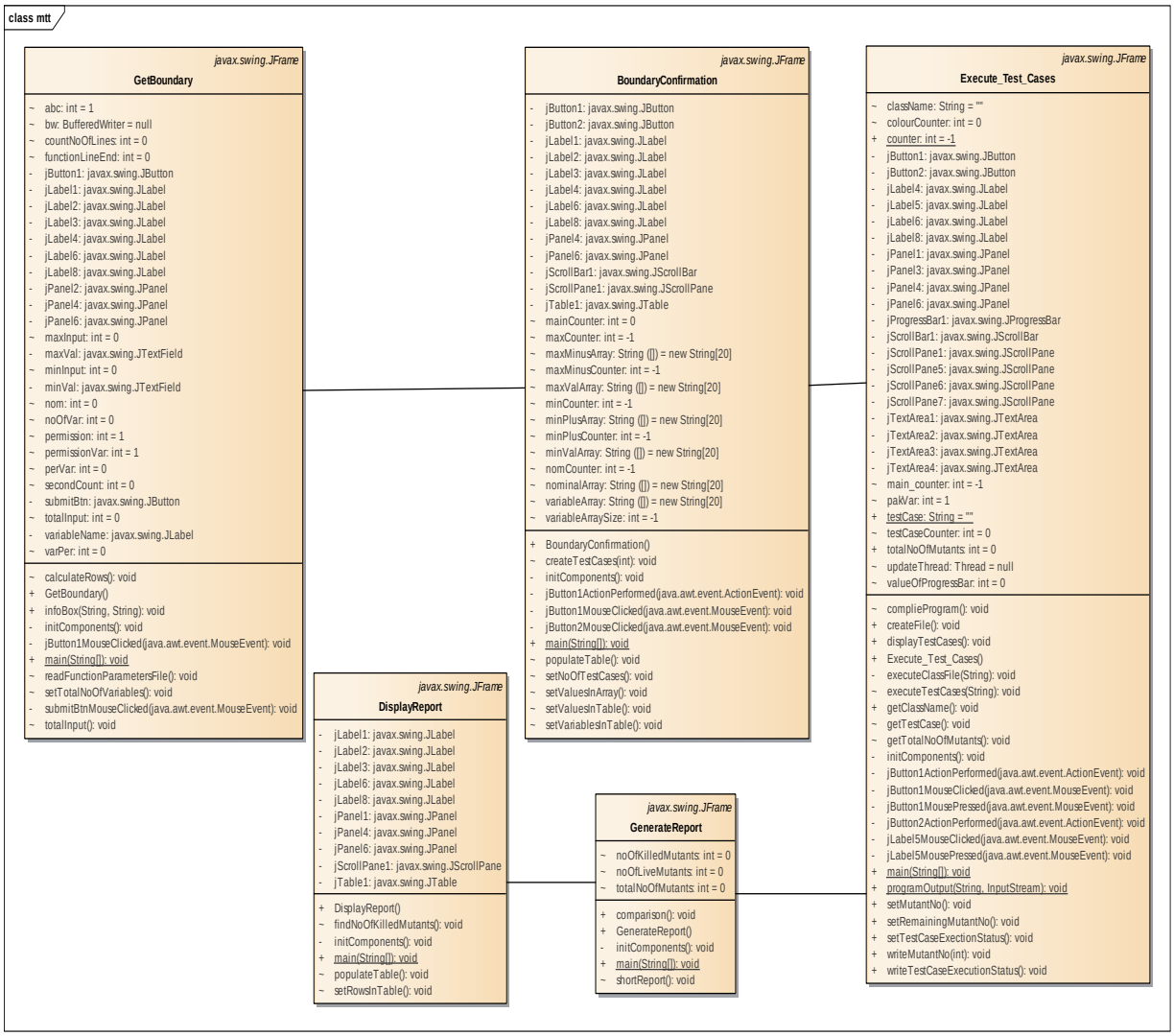

Figure 4.6: Class diagram of test suite generator and executer

4.2 User interfaces

This section includes all the user interfaces of the mutation testing tool.

4.2.1 Mutant Generation

The first step in this system is to submit a .java file of the program for which mutants are to be generated, this program can either be selected from already available list of programs or by browsing it from other location, the tool also provides an option of deleting file. The user interface is shown in Fig 4.7.

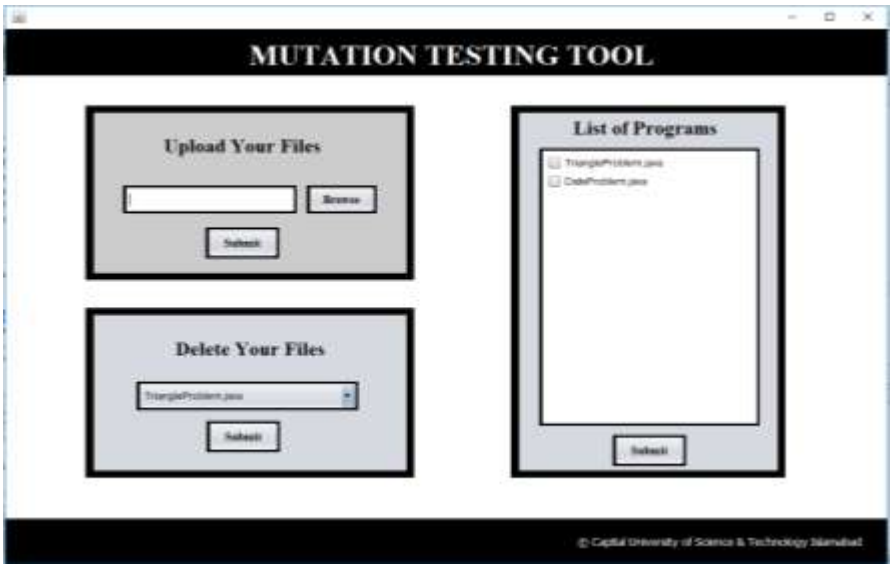

Figure 4.7: Source Code Selection

When the program is submitted to the system, a list of method level mutation operator appears in front of the user from which atleast one operator needs to be selected for generating mutants. User can also select multiple or all operators and submit it to the system. The user interface of this step is given in Fig 4.8.

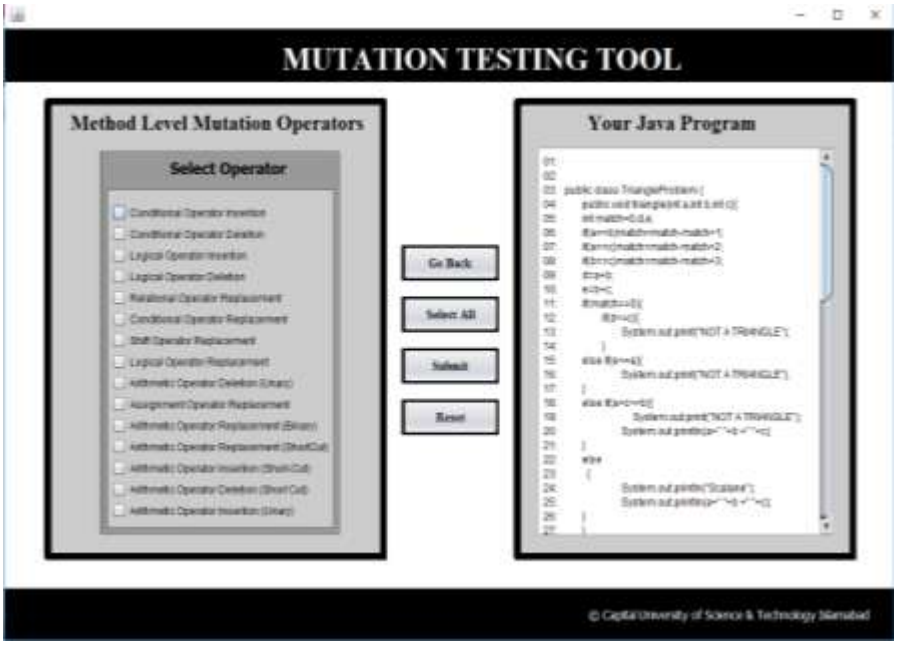

Figure 4.8: Mutation Operator Selection

Upon the selection of the mutation operator, this operator is applied on source code and mutants are generated and a user interface, as shown in Fig 4.9, is displayed which contains the summary of mutants' generation. It also provides an option of viewing mutants one by one by selecting a mutant and submitting it. If user clicks on 'Test Cases' button then the interface for the second module will open up.

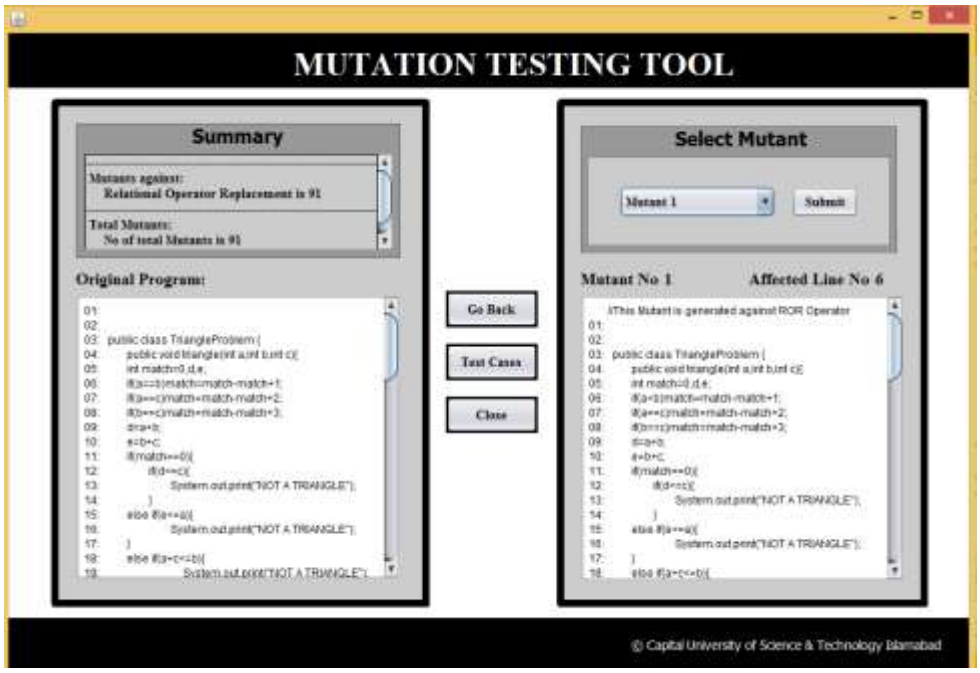

Figure 4.9: Mutants' Generation Summary

4.2.2 Test case Execution

The second module of the system is used to generate and execute test suite. For generating test suite, system takes the min and max values for each variable from user through a user interface which is shown in Figure 4.10.

After the submission of boundary values for all variables, the system will ask the user for confirmation of boundary values as shown in Figure 4.11. If user clicks on proceed then next window will open.

The system will display test cases with an option of test cases execution as shown in Figure 4.12.

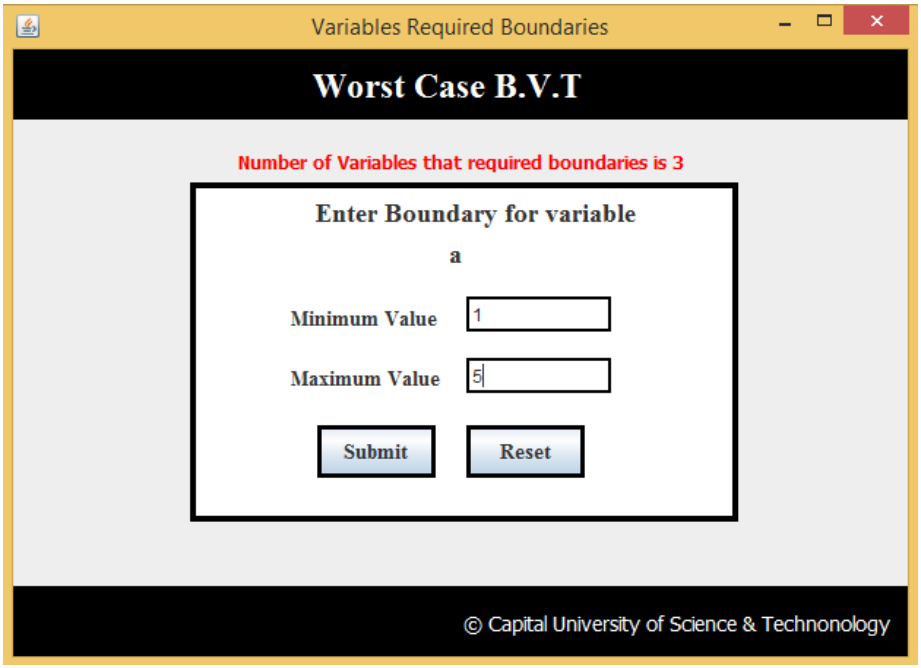

Figure 4.10: Taking Boundary values

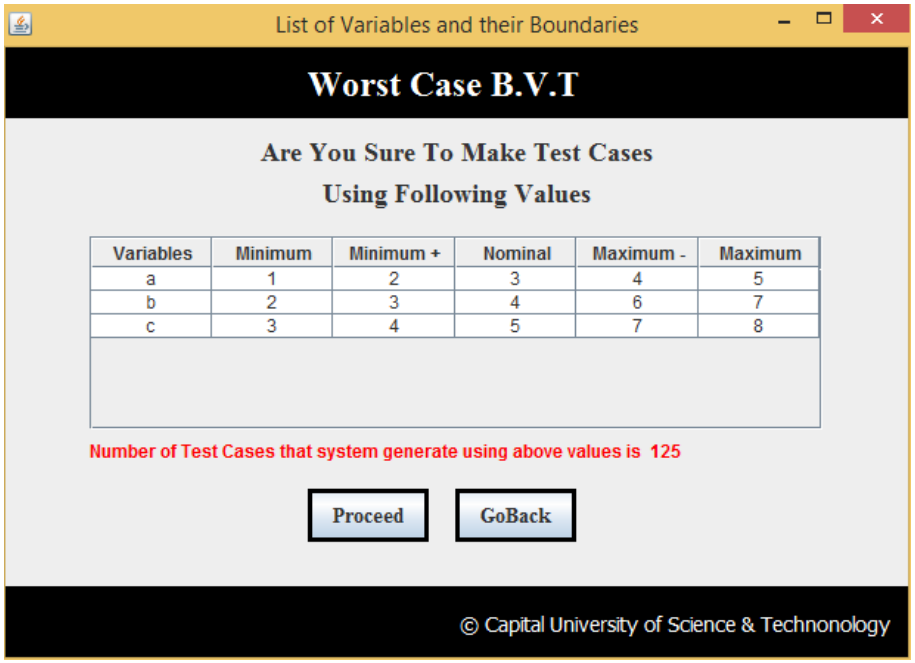

Figure 4.11: Displaying all five values

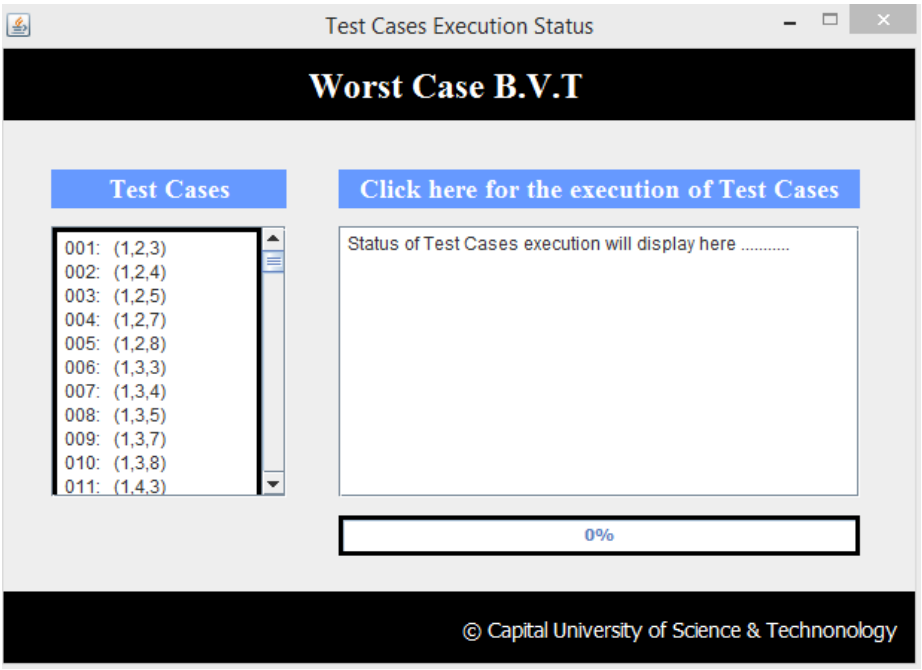

Figure 4.12: Displaying test cases

If user clicks chooses to execute test cases, then the execution will begin and the execution status will be displayed on the user interface as shown in Figure 4.13.

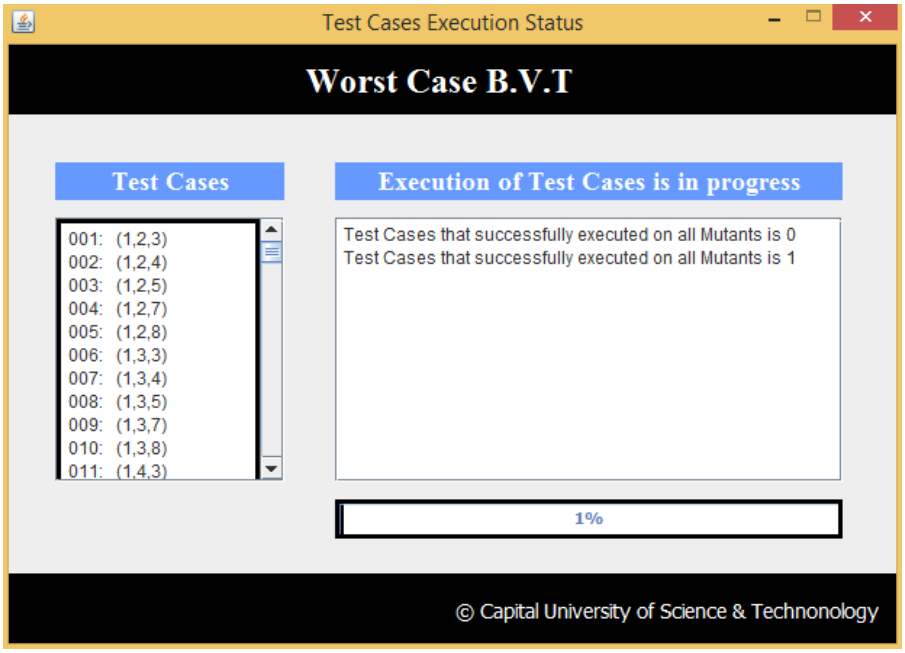

Figure 4.13: Execution of test cases

After the execution of all test cases on all mutants, the system will display the final report in the form of a table as shown in Figure 4.14.

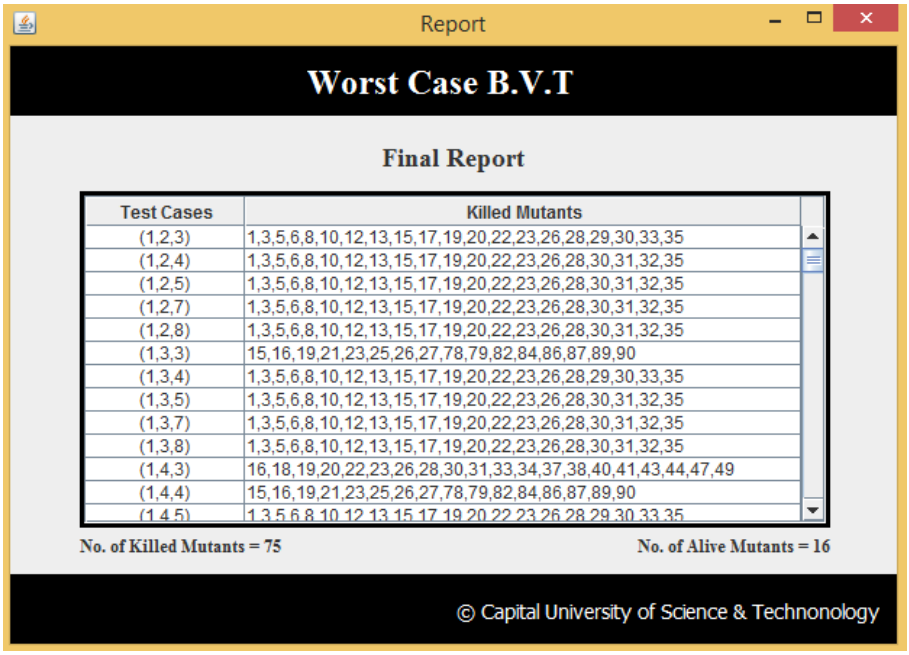

Figure 4.14: Test cases to mutants mapping

Chapter 5 : Results and Discussion

In this chapter, we have discussed experiments' results which we have performed on different subject programs. By using ROR mutation operator, we generated mutants of the original program and then using the data of test cases to mutants mapping, we generated mutation based prioritized list for each program. The existing criteria which we have used for comparison is Additional Branch Coverage since it is considered to be strongest prioritization technique. Both of the techniques including mutation based prioritization and branch coverage based prioritization are then compared using APFD.

For evaluating our technique, we have used three different programs. For selection of subject programs, different sources were searched including open source project repositories, and SIR (Software Infrastructure Repository). Since our proposed prioritization technique applies on method level, we looked for reasonably sized methods in the repositories. However, even in larger projects, the methods were either not too complex or had few lines of code. Therefore, we selected the subject programs from software testing literature, as these programs are good examples for application of software testing techniques. The source codes of these programs are given as an input to the tool described in Chapter 4 to generate mutants using ROR operator and for execution of original and mutant programs using the test suite generated by Worst case boundary value analysis. The data generated is then given to the prioritization algorithm for generating prioritization lists. Branch coverage information is also collected for each program using its respective test suite and branch coverage based prioritization lists are also generated.

5.1 Subject programs

We have used Triangle problem, Commission problem and Date problem as subject programs for evaluating our approach. A brief description of each program is given below (Jorgensen, 2010):

- **Triangle problem:** Triangle program takes three input variable a, b and c which represents the sides of a triangle. The input variables must satisfy the following conditions:

  C1. $a < b + c$

  C2. $b < a + c$

  C3. $c < a + b$

  There are three types of triangle which are equilateral, Isosceles and Scalene. Triangle problem returns the type of the triangle on the basis of input variables' values if the above conditions are satisfied. If these values do not meet any of the above conditions then the program returns "Not a Triangle" as an output.
- **Commission problem:** Locks, stocks and barrels made by a gunsmith were sold by a salesperson. The price of lock is $45, $30 for stock, $25 for barrels. Atleast one lock, one stock and one barrel were to be sold per month by the salesperson and maximum of 70 locks, 80 stocks and 90 barrels per month. After visiting each town, the salesperson sent a telegram to gunsmith informing him about the number of locks, stocks and barrels sold in that town. At the end of the month, the gunsmith calculated salesperson's commission as: 10% commission on sales up to $1000, 15% on the next $800, and 20% on any sales in excess of $1800.
- **Date problem:** Date problem takes three input: month, day and year and based on the current date calculates the next date. In case of February, 28, the date program checks whether the year is a leap or non leap year and calculates the date respectively.

The source codes of these programs are given in appendix A. Different characteristics of these subjects programs are given below in Table 5.1.

For each decision in the program, 7 mutants are generated. Since switch statement is used in Date program, thus the number of relational operators is reduced leaving behind less number of mutants.

Table 5.1: Subject Programs summary

Program	LOC	No. of inputs	No. of decisions	No. of mutants using ROR	No. of branches
Triangle Problem	80	3	13	91	26
Commission problem	46	3	2	14	4
Date problem	80	3	20	63	40

For each of these programs, two priority lists are generated, one is branch coverage based and other is mutation based priority list, using the data given in appendix B where each program's data is given in tabular form. The first column of the tables shows the mutants killed by each test case, second column indicates the total number of mutants killed, third column lists all the branches covered by each test case, forth column shows the total number of branches covered and last column indicates the errors detected by each test case used for APFD calculation. Table 5.2 shows both priority lists for all 3 programs.

Table 5.2: Subject Programs' Priority Lists

Program	Mutation based priority list	Branch coverage based priority list
1. Triangle Problem	{t32, t6, t26, t51, t101, t16, t1, t66,t2, t18, t71, t3, t4, t5, t7, t8, t9,t10, t11, t12, t13, t14, t15, t17, t19, t20, t21, t22, t23, t24, t25, t27, t28, t29, t30, t31, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t67, t68, t69, t70, t72, t73, t74, t75, t76, t77, t78, t79, t80, t81, t82, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t102, t103, t104, t105, t106, t107, t108, t109, t110, t111,	{t6, t11, t26, t51, t1, t32, t71, t101, t2, t3, t4, t5, t7, t8, t9, t10, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24, t25, t27, t28, t29, t30, t31, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t66, t67, t68, t69, t70, t72, t73, t74, t75, t76, t77, t78, t79, t80, t81, t82, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t102, t103,t104,t105, t106, t107, t108, t109, t110, t111, t112,

		t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}	t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}
2.	**Commission problem**	{t1, t4, t2, t3, t5, t6, t7, t8, t9, t10,t11, t12, t13, t14, t15, t16, t17,t18, t19, t20, t21, t22, t23, t24, t25, t26, t27, t28, t29, t30, t31, t32, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t51, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t66, t67, t68, t69, t70, t71, t72, t73, t74, t75, t76, t77, t78, t79, t80, t81, t82, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t101, t102, t103, t104, t105, t106, t107, t108, t109, t110, t111, t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}	{t1, t4, t2, t3, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24, t25, t26, t27, t28, t29, t30, t31, t32, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t51, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t66, t67, t68, t69, t70, t71, t72, t73, t74, t75, t76, t77, t78, t79, t80, t81, t82, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t101, t102, t103, t104, t105, t106, t107, t108, t109, t110, t111, t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}
3.	**Date problem**	{t107, t81, t1, t11, t21, t85, t6, t2, t3, t4, t5, t7, t8, t9, t10, t12, t13, t14, t15, t16, t17, t18, t19, t20, t22, t23, t24, t25, t26, t27, t28, t29, t30, t31, t32, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t51, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t66, t67, t68, t69, t70, t71, t72, t73, t74, t75, t76, t77, t78, t79, t80, t82, t83, t84, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t101, t102, t103, t104, t105, t106, t108, t109, t110, t111, t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}	{t106, t1, t11, t21, t81, t6, t16, t82, t2, t3, t4, t5, t7, t8, t9, t10, t12, t13, t14, t15, t17, t18, t19, t20, t22, t23, t24, t25, t26, t27, t28, t29, t30, t31, t32, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t51, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t66, t67, t68, t69, t70, t71, t72, t73, t74, t75, t76, t77, t78, t79, t80, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t101, t102, t103, t104, t105, t107, t108, t109, t110, t111, t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}

5.2 Comparison

Both of the prioritization techniques are then compared using APFD; which is the standard criteria for evaluation of prioritization techniques. For APFD calculation, we seeded the faults in the original program using mutation since the mutation faults are the representative of real faults. Hand seeded faults can be problematic for validity of results (Do et al., 2005).

We have used AORB (Arithmetic Operator Replacement Binary) mutation operator for generating such faulty versions of the program. Errors detected by test cases are shown in column 5 of the each program's tables given in appendix B. APFD is calculated using following formula:

$$APFD = 1 - \frac{TF_1 + \cdots \ldots . TF_m}{nm} + \frac{1}{2n}$$

Where T is the test suite containing n test cases and F is the set of m faults revealed by T. For ordering T', let TFi be the order of the first test case that reveals the *ith* fault. For APFD calculation, only those faults which are detected by the test suite are considered and undetected faults are ignored.

For example, consider the Triangle problem where:

F= {5, 6, 7, 8, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48}

T (mutation based)= {t32, t6, t26, t51, t101, t16, t1, t66, t2, t18, t71, t3, t4, t5, t7, t8, t9, t10, t11, t12, t13, t14, t15, t17, t19, t20, t21, t22, t23, t24, t25, t27, t28, t29, t30, t31, t33, t34, t35, t36, t37, t38, t39, t40, t41, t42, t43, t44, t45, t46, t47, t48, t49, t50, t52, t53, t54, t55, t56, t57, t58, t59, t60, t61, t62, t63, t64, t65, t67, t68, t69, t70, t72, t73, t74, t75, t76, t77, t78, t79, t80, t81, t82, t83, t84, t85, t86, t87, t88, t89, t90, t91, t92, t93, t94, t95, t96, t97, t98, t99, t100, t102, t103, t104, t105, t106, t107, t108, t109, t110, t111, t112, t113, t114, t115, t116, t117, t118, t119, t120, t121, t122, t123, t124, t125}

n= 125; m= 36

$$APFD = 1 - \frac{3+3+3+3+4+4+4+4+59+59+2+2+2+\cdots+4+2+2+2}{125(36)} + \frac{1}{2(125)}$$

$$APFD = 1 - \frac{328}{4500} + \frac{1}{250}$$

$$APFD = 93.11\%$$

Using this data APFD for mutation based prioritization for Triangle problem will be calculated.

APFDs of all three programs for both prioritization techniques are given below.

Table 5.3: Subject Programs' APFD

Program	No. of test cases	No. of faults seeded	No. of faults detected	APFD for mutation based prioritization	APFD for branch coverage prioritization
1. Triangle Problem	125	48	36	93.11%	92.51%
2. Commission Problem	125	68	58	99.39%	99.39%
3. Date Problem	125	40	24	96.93%	85.86%

For triangle problem, the difference between APFDs is only 0.6% because both priority lists include almost identical test cases with their positions varying. In mutation based priority list the test case t32 is given the highest priority because of its highest fault exposing potential, but in branch based priority list t32 is on 6^{th} position and out of 36 errors 8 are detected by t32. Since t32 in both priority lists is among higher priority test cases with small difference between its positions in both lists, thus the difference between both APFDs is only 0.6%.The graphical representation of fault detection of test cases for triangle problem is given below in Figure 5.1.

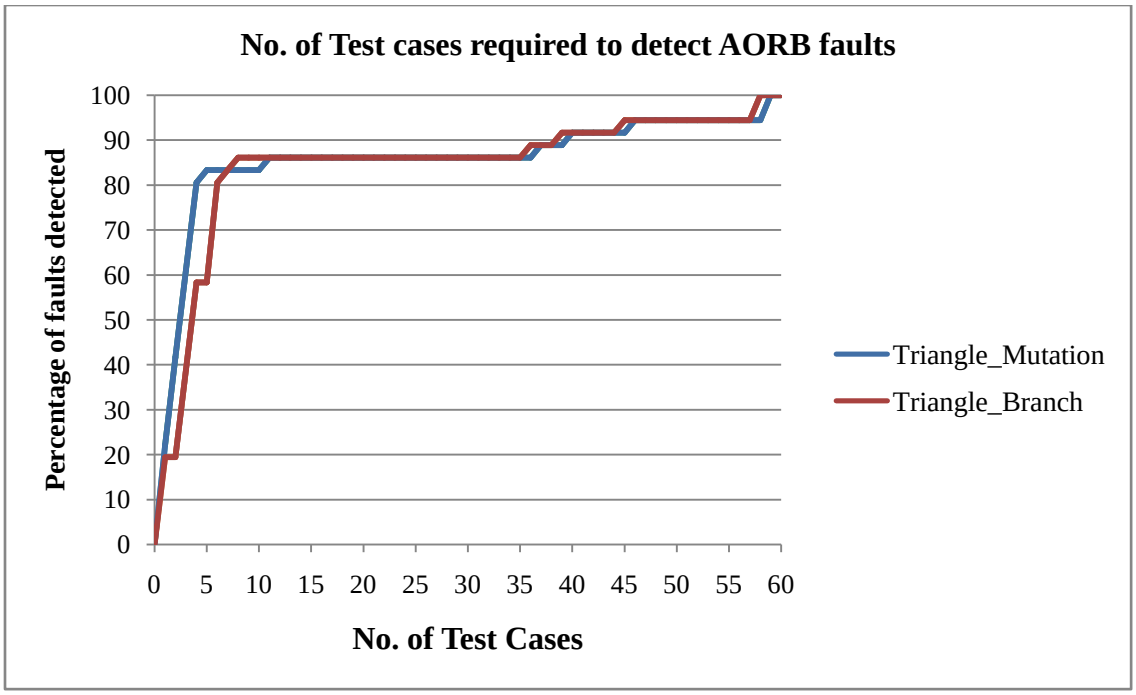

Figure 5.1: Graphical representation of fault detection of test cases for Triangle Problem

In commission problem, the number of branches and number of mutants are less than the other two example programs. Only two test cases are sufficient to kill all mutants and cover all branches. The test case that kills the maximum number of mutants covers the maximum number of branches as well, making both the prioritized lists exactly same, so there is no difference between both APFDs. The graphical representation of fault detection of test cases for commission problem is given below in Figure 5.2.

In case of date program, there is a more than 10% difference between the APFDs of both lists because of two test cases, i.e., t107 and t85. In mutation based priority list t107 is given highest priority and t85 in given 6^{th} priority but in branch based priority they both are treated as redundant test cases. Out of 24 errors used for APFD calculation, 4 are detected only by t107 and t82, but in branch based priority list t107 is among redundant test cases and among the highest priority test cases t82 is given the lowest priority. The other 4 errors are detected only by t85 but in branch based priority list it is among the redundant test cases, which makes the APFD of mutation based prioritization much

higher than the branch based prioritization. The graphical representation of fault detection of test cases for date problem is given below in Figure 5.3.

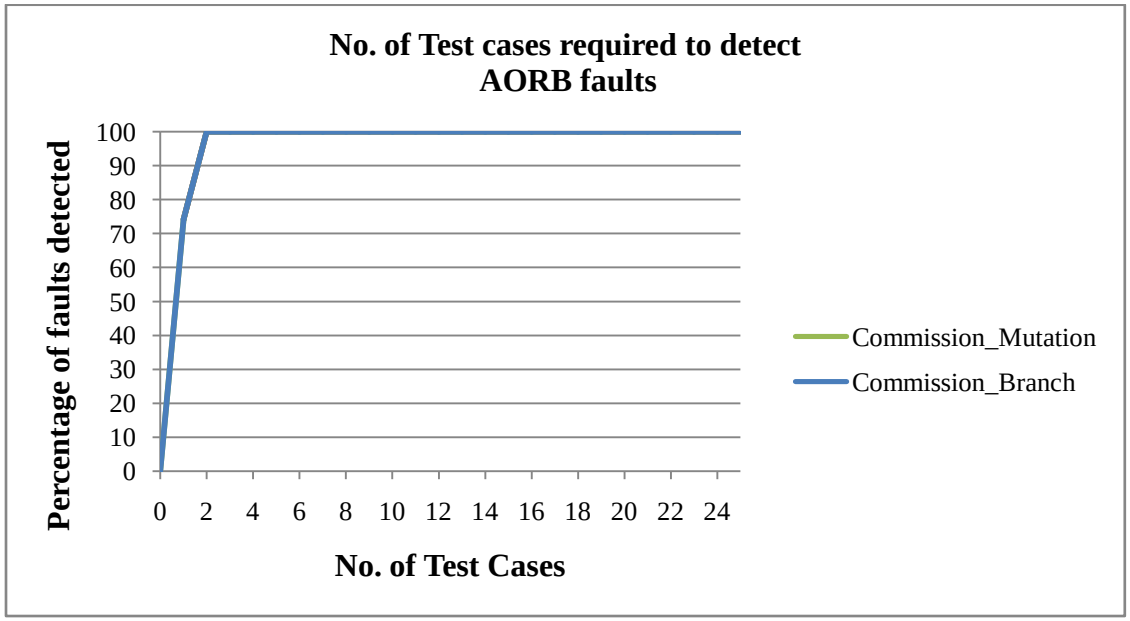

Figure 5.2: Graphical representation of fault detection of test cases for Commission problem

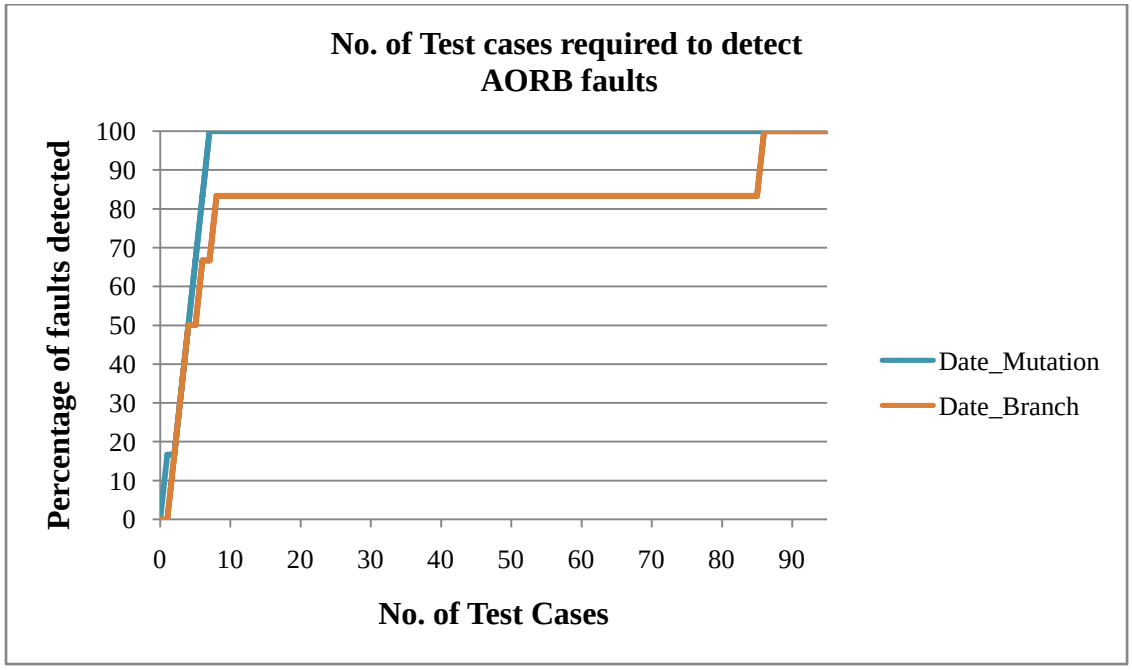

Figure 5.3: Graphical representation of fault detection of test cases for Date problem

By comparing the APFD of mutation based prioritization with branch coverage based prioritization, it can be concluded that it is clearly higher than the latter.

In coverage based prioritization, the assumption used is that coverage will maximize the fault detection rate, but from the above results, it can be seen that this assumptions does not always hold. The test cases which have higher fault detection potential can be given low priorities or they can be treated as redundant test cases. Since the mutation based prioritization assigns priorities to test cases based on their fault detection ability, thus it performs better than coverage based prioritization techniques in terms of APFD.

Chapter 6 : Conclusion and Future Work

After reviewing literature, we concluded that topic of test suite prioritization is of increasing importance because of it reducing the cost of regression testing. A lot of work has been done in this field and significant results have been achieved. To overcome the drawbacks of other regression techniques, test case prioritization is more commonly used. Through the detailed literature survey and experimentation, we are able to answer our research questions described in Chapter 1 as follows:

RQ. 1: What are the existing and most commonly used test case prioritization techniques?

A number of white box and black box prioritization techniques have been proposed to solve the problem of test suite prioritization. White box techniques use code coverage information for prioritization while black box techniques use specification in order to prioritize test cases. We have also seen that white box prioritization techniques, specifically those proposed by Rothermel et al. (2001) including branch, statement and function coverage based, are more commonly used than black box prioritization. Many black box prioritization techniques are also highly effective and competitive, and the difference between the APFDs of black box and white box techniques ranges from 2% to 5%, yet they are not commonly used because of the lack of structural information. Overall, White box prioritization techniques outperform black box prioritization techniques in 50 to 60% cases.

RQ. 2: What are the gaps in the existing prioritization techniques?

Although existing prioritization techniques perform well, but most of these prioritization techniques are not fault based techniques. They all prioritize test suites based on some coverage criteria including assuming that the coverage will maximize rate of fault detection. There are only few prioritization techniques which are fault based including

total FEP and additional FEP. FEP based techniques use fault detection probability of test cases in order to prioritize them which cannot be calculated accurately.

RQ. 3: How well does the mutation based prioritization technique compare with the well studied white box and black box prioritization techniques in terms of rate of fault detection?

The technique proposed in this research work involves the simple use of mutation testing in order to prioritize the test cases based on the number of additional mutants killed. The test case that exposes the maximum number of faults in the original program is given highest priority. Using three different subject programs, we generated priority lists for both mutation and branch coverage and we concluded that the test case that exposes maximum number of faults was given lower priority in branch coverage which is the major drawback of the technique. Mutation based prioritization addressed this drawback and assigned higher priorities to those test cases. The main objective of the proposed technique was to increase the average percentage of fault detection (APFD) of test suite. After generating the mutation based prioritization list, we have seen that the APFD of our proposed technique is more than the strongest coverage based prioritization technique which is branch coverage. The difference between the APFDs of branch coverage and mutation based coverage ranges from 0.5 to 11%.

6.1 Future work

After successful experimentation of the proposed technique using ROR operator, we plan to use more mutation operators for prioritization of test suite in near future. More than one mutation operator can also be used together to generate more than errors in each faulty version. Moreover, mutation testing can also be combined with some existing coverage based techniques by which we can solve the problem of random selection in case of tie. Mutation testing can be used as primary criteria for prioritization and in case of tie between test cases, we can use the coverage information of those test cases as a secondary criteria and the test case that achieves the maximum coverage can be assigned

higher priority instead of random selection. We also plan to perform experiments with larger case studies.

References

Arafeen, M. J., Do, H. (2013). Test case prioritization using requirements based clustering. International Conference on Software Testing. IEEE Computer Society Press, pp, 312–321.

Baresi,L., Pezze, M. (2006). An introduction to software testing. Electronic Notes in Theoretical Computer Science, vol.148, pp, 89-111.

Bryce, R. C., Memon, A. M. (2007). Test suite prioritization by interaction coverage. Proceedings of the Workshop on Domain Specific Approaches to Software Test Automation (DOSTA), ACM, pp, 1–7.

Burnstein, I. (2003). Practical software testing: a process-oriented approach. Springer-Verlag, New York, Inc.

Chen, T.Y., Lau, M.F. (1996). Dividing strategies for the optimization of a test suite. Information Processing Letters, 60(3), pp, 135– 141.

Do, H., Rothermel, G. (2005). A controlled experiment assessing test case prioritization techniques via mutation faults. Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society Press, pp, 411–420.

Do, H., Rothermel, G. (2006). On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering, 32(9), pp, 733–752.

Elbaum, S.G., Malishevsky, A.G., Rothermel, G.(2001). Prioritizing test cases for regression testing. Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2000), ACM Press, 2000; 102–112.

Elbaum, S.G., Malishevsky, A. G., Rothermel, G. (2002). Test case prioritization: a family of empirical studies. IEEE Transactions on Software Engineering, 28(2), pp, 159–182.

Fazlalizadeh,Y., Khalilian, A., Azgomi, M., Parsa,S. (2009). Prioritizing test cases for resource constraint environments using historical test case performance data. 2nd IEEE International Conference on Computer Science and Information Technology, pp, 190–195.

Garey, M.R., Johnson, D.S. (1979). Computers and Intractability: A guide to the theory of NP-Completeness. W. H. Freeman and Company: New York, NY.

Github, www.github.com

Harrold, M.J. (1999). Testing evolving software. The Journal of Systems and Software, 47(2–3), pp, 173–181.

Harrold, M.J., Gupta, R., Soffa, M.L. (1993). A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology, 2(3), pp, 270–285.

Henard, C., Papadakis, M., Harman, M., Jia, Y. and Traon, Y.L. (2016). Comparing White-box and Black-box Test Prioritization. In Proceedings of the 38th International Conference on Software Engineering (ICSE). ACM.

Horgan, J., London, S. (1992). ATAC: A data flow coverage testing tool for c. Proceedings of the Symposium on Assessment of Quality Software Development Tools, IEEE Computer Society Press, pp, 2–10.

Jorgensen, P.C. (2010). Software Testing: A Craftsman's Approach. 2nd Edition. Boca Raton, FL: CRC Press. ISBN: 13: 978-1-4665-6069-7.

Kaur,A., Goyal,S. (2011). A genetic algorithm for regression test case prioritization using code coverage. International Journal of Advanced Trends in Computer Science and Engineering.

Kim, J., Porter, A. A. (2002). A history-based test prioritization technique for regression testing in resource constrained environments. Proceedings of the 24th International Conference on Software Engineering (ICSE), ACM Press, pp, 119–129.

Kumar, H., Pal, V., Chauhan, N. (2013). A hierarchical system test case prioritization technique based on requirements. 13th Annual International Software Testing Conference, pp, 4–5

Leung, H.K.N., White, L. (1989). Insight into regression testing. Proceedings of Interntional Conference on Software Maintenance (ICSM), IEEE Computer Society Press, pp, 60–69.

Li, Z., Harmanand, M., Hierons, R. (2007). Search Algorithms for Regression Test Case Prioritization. Proceedings of IEEE Transactions on Software Engineering, Vol. 33, pp, 225-237.

Lin, S. (1965). Computer Solutions of the Travelling Salesman Problem. Bell System Technical Journal, pp, 2245-2269.

Ma, Y. S., Offutt, A. J., Kwon, Y. R. (2006). MuJava: A Mutation System for Java. Proceedings of the 28th international Conference on Software Engineering (ICSE), IEEE Computer Society Presspp. 827–830.

Maheswari,R. U., Mala, D. J. (2015). Combined Genetic and Simulated Annealing Approach for Test Case Prioritization. Indian Journal of Science and Technology, Vol 8(35).

Marre, M., Bertolino, A. (2003). Using spanning sets for coverage testing. IEEE Transactions on Software Engineering; 29(11), pp, 974–984.

Offutt, A. J., Pan, J., Voas, J. (1995). Procedures for reducing the size of coverage-based test sets. Proceedings of the 12th International Conference on Testing Computer Software, ACM Press, pp, 111–123.

Offutt, A. J., Rothermel, G., Zapf, C. (1993). An experimental evaluation of selective mutation. Proceedings of the 15th International Conference on Software Engineering, pp, 100–107. IEEE Computer Society Press.

Qu, B., Nie, C., Xu, B., Zhang, X. (2007). Test case prioritization for black box testing. In Computer Software and Applications Conference, pp. 465–474.

Rothermel, G, Harrold M.J. (1994). A framework for evaluating regression test selection techniques. Proceedings of the 16th International Conference on Software Engineering (ICSE), IEEE Computer Society Press, pp, 201–210.

Rothermel, G. Untch, R.H., Chu, C., Harrold, M.J. (1999). Test case prioritization: An empirical study. Proceedings of International Conference on Software Maintenance (ICSM), IEEE Computer Society Press, pp, 179–188.

Software Infrastructure Repository, www.sir.unl.edu

Srikanth, H., Williams, L., Osborne,J. (2005). Towards the Prioritization of system test cases. North Carolina State University TR-44.

Srivastava, P. R. (2008). Test Case prioritization. Journal of Theoretical and Applied Information Technology (JATIT), BITS Pilani, India333031.

Wong, W. E., Horgan, J. R., London, S., Agrawal, H. (1997). A study of effective regression testing in practice, Proceedingsofthe8th International Symposium on Software Reliability Engineering (ISSRE), IEEE Computer Society, pp, 264–275.

Yoo, S., Harman, M. (2012). Regression testing: minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 22(2), pp, 67–120.

Appendix A: Source Codes of example programs

Source codes of examples programs:

Triangle Program:

```
public class TriangleProbelm {
    String newline = System.getProperty("line.separator");
    public void triangle(int a,int b,int c){
    int match=0,d,e;
    if(a==b)
        match = match - match + 1;
    if(a==c)
        match=  match - match + 2;
    if(b==c)
        match=  match - match + 3;
    d = a + b;
    e = b + c;
    if(match==0)
    {
        if(d<=c)
        {
            System.out.println(a+" "+b +" "+c);
            System.out.print("NOT A TRIANGLE" +newline);

        }
    else if(e<=a)
    {
          System.out.println(a+" "+b +" "+c);
          System.out.print("NOT A TRIANGLE" +newline);
    }
    else if (a + c <= b)
    {
          System.out.println(a+" "+b +" "+c );
          System.out.print("NOT A TRIANGLE" +newline);
    }
    else
    {
          System.out.println(a+" "+b +" "+c );
            System.out.println("Scalane" +newline);
    }
    }
    else if(match==1){
        if (a + c <= b)

        {
          System.out.println(a+" "+b +" "+c );
            System.out.println("NOT A TRIANGLE" +newline);}

        else
        {
          System.out.println(a+" "+b +" "+c );
            System.out.println("Isoscles" +newline);
        }
```

```
    }
    else if (match==2){
        if(a + c <= b)
        {
          System.out.println(a+" "+b +" "+c );
            System.out.print("NOT A TRIANGLE" +newline);
            }
        else
        {
          System.out.println(a+" "+b +" "+c );
            System.out.print("Isoscles" +newline);
            }
    }
    else if(match==3){
        if(b + c <= a)
        {
          System.out.println(a+" "+b +" "+c );
            System.out.print("NOT A TRIANGLE" +newline);
        }
        else
            {
          System.out.println(a+" "+b +" "+c );
            System.out.print("Isoscles" +newline);
            }
    }
    else
    {
      System.out.println(a+" "+b +" "+c);
      System.out.println("Equilaterial" +newline);
    }
   }
  }
```

Commission Problem:

```
public class Commission {
    String newline = System.getProperty("line.separator");
    public void commission(int locks, int stocks, int barrels){
    int totalLocks,totalStocks,totalBarrels;
    double lockPrice, stockPrice, barrelPrice;
    double lockSales, stockSales, barrelSales, sales, commission;
    lockPrice = 45.0;
    stockPrice = 30.0;
    barrelPrice = 25.0;
    totalLocks = 0;
    totalStocks = 0;
    totalBarrels = 0;
    totalLocks = totalLocks + locks;
    totalStocks = totalStocks + stocks;
    totalBarrels = totalBarrels + barrels;
    System.out.println("Locks sold:" +totalLocks);
    System.out.println("Stocks sold:" +totalStocks);
    System.out.println("Barrels sold:" +totalBarrels);
    lockSales = lockPrice * totalLocks;
    stockSales = stockPrice * totalStocks;
```

```
    barrelSales = barrelPrice * totalBarrels;
    sales = lockSales + stockSales + barrelSales;
    System.out.println("Total Sales:" +sales);
    if(sales > 1800.0) //1
    {
      commission = 0.10 * 1000.0;
      commission = commission + 0.15 * 800.0;
      commission = commission + 0.20 * (sales - 1800.0);
    }
    else if (sales > 1000.0) //2
      {
      commission = 0.10 * 1000.0;
      commission = commission + 0.15 * (sales - 1000.0);
      }
       else
      {
      commission = 0.10 * sales;
      }
    System.out.println("Commission:" +commission +newline);
    }
    public static  void main(String[] args )
    {
        Commission object=new Commission();
        object.commission(1, 1, 1);
 }
    }
```

Date Problem:

```
public class nextdate {
      public void nextday(int day,int month,int year){
            int tomorrowDay =
day,tomorrowMonth=month,tomorrowYear=year;
            boolean leapyear;
            leapyear= ((year % 4 == 0) && (year % 100 != 0 || year %
400 == 0));
            switch(month)
            {
            case 1: case 3: case 5: case 7: case 8: case 10:
                  if (day < 31)
                  {
                  tomorrowDay = day + 1;
                  }
                  else
                  {
                  tomorrowDay = 1;
                  tomorrowMonth = month + 1;
                  }
                 break;
            case 4: case 6: case 9: case 11:
                  if (day < 30)
                  {
                  tomorrowDay = day + 1;
                  }
                  else
```

```
                {
                tomorrowDay = 1;
                tomorrowMonth = month + 1;
                }
                break;
          case 12:
                if(day < 31)
                     {
                   tomorrowDay = day + 1;
                     }
                else
                     {
                     tomorrowDay = 1;
                     tomorrowMonth = 1;
                     tomorrowYear = year + 1;
                     }
                break;
          case 2:
                if (day < 28)
                {
                     tomorrowDay = day + 1;
                }
                else if( day == 28)
                  {
                        if (leapyear)
                           tomorrowDay = 29;
                        else
                        {
                           tomorrowDay = 1;
                         tomorrowMonth = 3;
                        }
                  }
                else
                {
                      if (day == 29)
                      {
                            if(leapyear)
                            {
                                  tomorrowDay = 1;
                               tomorrowMonth = 3;
                            }
                            else
                                  System.out.println("February
cannot have days" +day);

                      }
                }
                break;
          }
      System.out.printf("Date is:" +tomorrowDay + "-"  +tomorrowMonth
+  "-" +tomorrowYear);
}
    public static  void main(String[] args )
    {
        nextdate object=new nextdate();
```

```
        object.nextday(1,1,1);
    }
}
```

Appendix B: Test data for example programs

Test data for example programs:

Triangle Problem

Test case	Mutants killed	Cov(t) for Mutants	Branches covered	Cov(t) for branches	Errors detected
t1	1,3,5,6,8,10,12,13,15, 17,19,20,22,23,26,28,29, 30,33,35	20	2,4,6,7,9	5	
t2	1,3,5,6,8,10,12,13,15, 17,19,20,22,23,26,28,30, 31,32,35	20	2,4,6,7,9	5	
t3	1,3,5,6,8,10,12,13,15, 17,19,20,22,23,26,28,30, 31,32,35	20	2,4,6,7,9	5	
t4	1,3,5,6,8,10,12,13,15, 17,19,20,22,23,26,28,30, 31,32,35	20	2,4,6,7,9	5	
t5	1,3,5,6,8,10,12,13,15, 17,19,20,22,23,26,28,30, 31,32,35	20	2,4,6,7,9	5	
t6	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t7	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	
t8	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t9	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t10	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t11	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,10,12,13	7	
t12	15,16,19,21,23,25,26,27,	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47,

	78,79,82,84,86,87,89,90				48
t13	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	
t14	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t15	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t16	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	
t17	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	
t18	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 43,44,47,49	20	2,4,6,7,10,12,13	7	
t19	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29, 30,33,35	20	2,4,6,7,9	5	
t20	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t21	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	
t22	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	
t23	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	
t24	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t25	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	
t26	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t27	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t28	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t29	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t30	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40

t31	15,16,19,21,23,25,26,27,78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47,48
t32	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,32,34,35,36
t33	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,29,30,33,35	20	2,4,6,7,9	5	25
t34	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,32,35	20	2,4,6,7,9	5	
t35	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,32,35	20	2,4,6,7,9	5	
t36	1,3,5,6,8,10,12,13,16,18,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32,34,35,36
t37	15,16,19,21,23,25,26,27,78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47,48
t38	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,32,34,35,36
t39	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,32,35	20	2,4,6,7,9	5	25
t40	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,32,35	20	2,4,6,7,9	5	
t41	16,18,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,46,49	20	2,4,6,7,10,12,13	7	
t42	16,18,19,20,22,23,26,28,30,31,33,34,37,38,40,41,43,44,47,49	20	2,4,6,7,10,12,13	7	33
t43	1,3,5,6,8,10,12,13,16,18,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32,34,35,36
t44	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,30,31,33,34,37,38,40,41,44,45,47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,32,34,35,36
t45	1,3,5,6,8,10,12,13,15,17,19,20,22,23,26,28,29,30,33,35	20	2,4,6,7,9	5	25
t46	16,18,19,20,22,23,26,28,	20	2,4,6,7,10,12,13	7	

	30,31,33,34,37,38,40,41, 44,45,46,49				
t47	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 44,45,46,49	20	2,4,6,7,10,12,13	7	33
t48	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 43,44,47,49	20	2,4,6,7,10,12,13	7	33
t49	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t50	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,32, 34,35,36
t51	8,9,12,14,23,25,26,27,64 ,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t52	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34, 36
t53	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	25
t54	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t55	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t56	23,25,26,27,78,79,82,84, 86,87,89,90	12	1,3,5,8,16,20,23,26	8	17,20,21,22,23,24, 46,47,48
t57	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t58	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t59	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t60	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t61	8,9,12,14,23,25,26,27,64 ,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t62	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t63	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,32,34, 35,36
t64	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30,	20	2,4,6,7,9	5	25

	33,35				
t65	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	25
t66	16,18,19,20,64,65,68,70, 71,72,75,77	12	2,3,6,8,16,19,22	7	16,41
t67	1,3,5,6,8,10,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t68	1,3,5,6,8,10,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t69	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t70	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t71	16,18,19,20,64,65,68,70, 72,73,74,77	12	2,3,6,8,16,19,21	7	16,41
t72	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41, 43,44,47,49	20	2,4,6,7,10,12,13	7	
t73	1,3,5,6,8,10,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t74	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t75	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t76	2,4,5,6,9,11,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t77	8,9,12,14,23,25,26,27,64 , 65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t78	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t79	2,4,5,6,8,10,12,13,15,17,	20	2,4,6,7,9	5	25

	19,20,22,23,26,28,30,31, 32,35				
t80	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	
t81	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t82	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t83	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t84	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	25
t85	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	25
t86	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	20	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t87	23,25,26,27,78,79,82,84, 86,87,89,90	12	1,3,5,8,16,20,23,26	8	17,20,21,22,23,24, 46,48,49
t88	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t89	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t90	1,2,5,7,23,25,26,27,50, 51,54,56,58,59,61,62	16	1,4,6,8,15,18	6	5,6,7,8,38,39,40
t91	1,3,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t92	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t93	1,3,5,6,8,10,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t94	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t95	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t96	16,18,19,20,22,23,26,28, 30,31,33,34,37,38,40,41,	20	2,4,6,7,10,12,13	7	33

	43,44,47,49				
t97	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t98	1,3,5,6,8,10,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t99	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t100	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t101	2,4,5,6,9,11,12,13,15,17, 19,20,22,23,26,28,36,37, 40,42	20	2,4,6,7,10,11	6	29,30
t102	2,4,5,6,9,11,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t103	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t104	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	25
t105	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 32,35	20	2,4,6,7,9	5	25
t106	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t107	2,4,5,6,9,11,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t108	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t109	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t110	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,29,30, 33,35	20	2,4,6,7,9	5	25
t111	2,4,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t112	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48

t113	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t114	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t115	2,4,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t116	1,3,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t117	1,3,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t118	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t119	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t120	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t121	1,3,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t122	1,3,5,6,9,11,12,13,16,18, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36
t123	8,9,12,14,23,25,26,27, 64,65,68,70,72,73,75,76	16	2,3,6,8,16,19,22	7	13,14,15,16,42,43, 44
t124	15,16,19,21,23,25,26,27, 78,79,82,84,86,87,89,90	16	2,4,5,8,16,20,23,26	8	21,22,23,24,46,47, 48
t125	1,3,5,6,8,10,12,13,15,17, 19,20,22,23,26,28,30,31, 33,34,37,38,40,41,44,45, 47,48	28	2,4,6,7,10,12,14	7	26,27,28,30,31,32, 34,35,36

Commission Problem

Test case	Mutants killed	Cov(t) for mutants	Branches covered	Cov(t) for branches	Errors detected
t1	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, 16,17,18,19,20,21,22,23,24,25,26,27, 28,29,30,31,32,33,34,35,36,57,58,59, 60,61,62,63
t2	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t3	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t4	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t5	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t6	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t7	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t8	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t9	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t10	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t11	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t12	1,2,5,6,7,8,9,	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

	11				,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,57,58,59,60,61,62,63
t13	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t14	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t15	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t16	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t17	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t18	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t19	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t20	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t21	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t22	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t23	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t24	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,

					29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t25	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t26	1,2,5,6,7,8,9, 11	8	2,3	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t27	1,2,5,6,7,8,9, 11	8	2,3	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t28	1,2,5,6,7,8,9, 11	8	2,3	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t29	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t30	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t31	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t32	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t33	8,9,11	3	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60, 61,62,63
t34	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t35	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t36	1,2,5,6,7,8,9, 11	8	2,3	2	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,57,58,59,60,

					61,62,63
t37	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t38	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t39	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t40	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t41	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t42	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t43	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t44	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t45	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t46	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t47	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t48	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51

t49	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t50	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t51	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t52	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t53	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t54	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t55	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t56	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t57	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t58	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t59	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t60	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t61	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

					,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t62	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t63	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t64	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t65	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t66	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t67	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t68	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t69	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t70	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t71	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t72	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t73	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28,

					29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t74	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t75	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t76	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t77	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t78	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t79	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t80	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t81	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t82	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t83	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t84	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t85	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40,

					41,42,43,44,45,46,47,48,49,50,51
t86	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t87	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t88	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t89	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t90	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t91	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t92	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t93	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t94	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t95	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t96	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t97	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51

t98	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t99	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t100	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t101	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t102	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t103	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t104	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t105	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t106	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t107	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t108	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t109	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
t110	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

					,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t111	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t112	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t113	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t114	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t115	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t116	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t117	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t118	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t119	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t120	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t121	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t122	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28,

					29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t123	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t124	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51
t125	1,2,4	3	1	1	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 ,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40, 41,42,43,44,45,46,47,48,49,50,51

Date Problem

Test case	Mutants killed	Cov(t) for mutants	Branches covered	Cov(t) for branches	Errors detected
t1	22,24,25,28	4	1,13	2	13,14,15,16
t2	22,24,25,28	4	1,13	2	13,14,15,16
t3	22,24,25,28	4	1,13	2	13,14,15,16
t4	22,24,25,28	4	1,13	2	13,14,15,16
t5	22,24,25,28	4	1,13	2	13,14,15,16
t6	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t7	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t8	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t9	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t10	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t11	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t12	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t13	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t14	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t15	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t16	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t17	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t18	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t19	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t20	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t21	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t22	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t23	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t24	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t25	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32

t26	22,24,25,28	4	1,13	2	13,14,15,16
t27	22,24,25,28	4	1,13	2	13,14,15,16
t28	22,24,25,28	4	1,13	2	13,14,15,16
t29	22,24,25,28	4	1,13	2	13,14,15,16
t30	22,24,25,28	4	1,13	2	13,14,15,16
t31	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t32	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t33	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t34	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t35	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t36	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t37	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t38	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t39	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t40	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t41	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t42	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t43	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t44	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t45	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t46	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t47	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t48	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t49	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t50	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t51	22,24,25,28	4	1,13	2	13,14,15,16
t52	22,24,25,28	4	1,13	2	13,14,15,16
t53	22,24,25,28	4	1,13	2	13,14,15,16
t54	22,24,25,28	4	1,13	2	13,14,15,16
t55	22,24,25,28	4	1,13	2	13,14,15,16
t56	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t57	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t58	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t59	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t60	43,45,46,49	4	2,4,6,8,10,12,16,18,20,22,26,29,31	13	37,38,39,40
t61	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t62	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t63	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t64	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t65	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t66	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t67	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t68	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t69	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t70	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t71	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t72	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t73	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32

t74	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t75	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t76	22,24,25,28	4	1,13	2	13,14,15,16
t77	22,24,25,28	4	1,13	2	13,14,15,16
t78	22,24,25,28	4	1,13	2	13,14,15,16
t79	22,24,25,28	4	1,13	2	13,14,15,16
t80	22,24,25,28	4	1,13	2	13,14,15,16
t81	2,4,5,6,44,45,46, 48,50, 51,54,56	12	22,4,6,8,10,12,16,18,20,22,26,29,3 2,33,36	15	
t82	1,2,5,7,8,10,12,1 4,50,51,54,56	12	2,4,6,8,10,12,16,18,20,22,26,29,32 ,33,35	15	1,2,3,4
t83	2,4,5,6,44,45,46, 48,50, 51,54,56	12	2,4,6,8,10,12,16,18,20,22,26,29,32 ,33,36	15	
t84	2,4,5,6,44,45,46, 48,50, 51,54,56	12	2,4,6,8,10,12,16,18,20,22,26,29,32 ,33,36	15	
t85	1,2,5,7,15,16,19, 21,50, 51,54,56	12	2,4,6,8,10,12,16,18,20,22,26,29,32 ,33,3	15	1,2,3,4,9,10,11, 12
t86	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t87	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t88	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t89	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t90	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t91	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t92	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t93	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t94	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t95	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t96	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t97	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t98	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t99	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t100	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t101	22,24,25,28	4	1,13	2	13,14,15,16
t102	22,24,25,28	4	1,13	2	13,14,15,16
t103	22,24,25,28	4	1,13	2	13,14,15,16
t104	22,24,25,28	4	1,13	2	13,14,15,16
t105	22,24,25,28	4	1,13	2	13,14,15,16
t106	2,4,5,6,43,45,47, 48,51,53,54,55, 57,58,61,63	16	2,4,6,8,10,12,16,18,20,22,26,29,32 ,34,37,40	16	
t107	1,2,5,7,8,10,12, 14,43,45,47,48, 51,53,54,55,57, 58,61,63	20	2,4,6,8,10,12,16,18,20,22,26,29,32 ,34,37,40	16	1,2,3,4

t108	2,4,5,6,43,45,47,48,51,53,54,55,57,58,61,63	16	2,4,6,8,10,12,16,18,20,22,26,29,32,34,37,40	16	
t109	2,4,5,6,43,45,47,48,51,53,54,55,57,58,61,63	16	2,4,6,8,10,12,16,18,20,22,26,29,32,34,37,40	16	
t110	1,2,5,7,15,16,19,21,43,45,47,48,51,53,54,55,57,58,61,63	20	2,4,6,8,10,12,16,18,20,22,26,29,32,34,37,39	16	1,2,3,4,9,10,11,12
t111	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t112	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t113	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t114	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t115	29,31,32,35	4	2,4,6,8,10,12,16,17,23	9	21,22,23,24
t116	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t117	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t118	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t119	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t120	29,31,32,35	4	2,4,6,8,10,12,16,18,20,21,23	11	21,22,23,24
t121	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t122	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t123	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t124	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32
t125	36,38,39,42	4	2,4,6,8,10,12,16,18,20,22,25,27	12	29,30,31,32