TESTCASE MINIMISATION TOOL (TMT)

J-COMPONENT PROJECT REPORT

WINTER SEMESTER 2020-21

SUBMITTED BY:

- Kushagra Singhal (19BCE0716)
- Priyasha Thacker (19BCE2267)
- Manav Kulshrestha (19BCE0718)

In partial fulfilment for the award of the degree of

B.Tech

In

Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vellore – 632014, Tamil Nadu, India

ACKNOWLEDGEMENT:

In the present world of competition there is a race of existence in which those are having will to come forward succeed. Project is like a bridge between theoretical and practical working. With this willing we joined this particular project.

First of all, we are feeling oblige in taking the opportunity to sincerely thanks to honorable G. Viswanathan (Founder-Chancellor, VIT University). Next we would like to thank Dr. Saravanan R (DEAN, School of Computer Science and Engineering) and special thanks to Dr. Vairamuthu S (Head of Department, School of Computer Science and Engineering).

It is our pleasure to express with deep sense of gratitude to DR. SWATHI.J.N, Associate Professor, School of Computer Science and Engineering, Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding; more than all, she taught us patience in our endeavour. Our association with her is not confined to academics only, but it is a great opportunity on our part of work with an intellectual and expert in the field of Software Engineering.

We got to learn a lot from this project. At last, we would like to extend our heartfelt thanks to our parents because without their help this project would not have been successful.

EXECUTIVE SUMMARY:

Software testing is most expensive phase of development. It becomes unfeasible to execute all the test cases. Test case minimization techniques are used to minimize the testing cost in terms of execution time, resources etc. Main purpose of test case minimization techniques is to remove test cases that become redundant and obsolete over time.

Testcase Minimization Tool, as the name suggests, is designed with parallelization in mind. TMT was built to use strategies and techniques that dramatically speed up the minimization process. Minimization tools use various techniques to simplify the testcase, but the core algorithm is simply bisection. Bisection is an inherently serial process, you can't advance through the algorithm without knowing the result of each step. This data dependency problem can make minimization very slow, sometimes taking hours to complete while cpu cores sit idle.

TMT solves this problem using *pessimistic speculative execution*. We build a binary tree of all the possible bisection steps and then idle cores can speculatively test future steps ahead of our position in the algorithm. In many cases, the test results are already known by the time we need them. We call it *pessimistic*, because real workloads are characterized by long series of consecutive failures. We simply assume that tests are going to fail, and speculatively follow the failure path until proven wrong.

CONTENTS:

LIST OF FIGURES:

LIST OF TABLES:

ABBREVIATIONS:

Table of Definitions, Acronyms, and Abbreviations

Definition, Acronym, or Abbreviation	Description
SRS	Software Requirement Specification.
SDS	Software Design Specification
TMT	Testcase Minimisation Tool

1. INTRODUCTION:

1.1 OBJECTIVE:

Software testing is most expensive phase of development. It becomes unfeasible to execute all the test cases. Test case minimization techniques are used to minimize the testing cost in terms of execution time, resources etc. The main purpose of test case minimization techniques is to remove test cases that become redundant and obsolete over time. Lastly, the purpose of this document is to communicate the system attributes of the Testcase Minimisation Tool software. These system attributes include reliability, availability, scalability, maintainability, and portability.

1.2 MOTIVATION:

Every project begins with a requirement. Understanding the types of users and their requirements is the most important part of a project. As mentioned in the objective, it sometimes becomes unfeasible to execute all the testcases, there we opt for the Testcase Minimisation Tool.

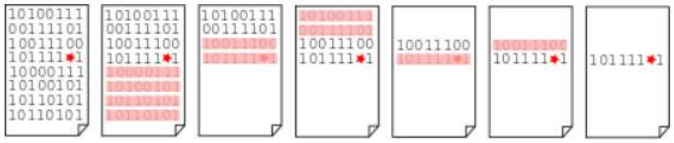
It is outside the scope of this document to describe certain technology or the general problem with test cases. It is also outside the scope of this document to describe in any detail at all how certain mentioned standards or technologies work and operate.

For future scope, the next version will allow the level of pessimism to be controlled at runtime.

1.3 BACKGROUND:

Fuzzers find inputs that trigger bugs, but understanding those bugs is easier when you remove as much extraneous data as possible. This is called *testcase minimization* or *delta debugging*.

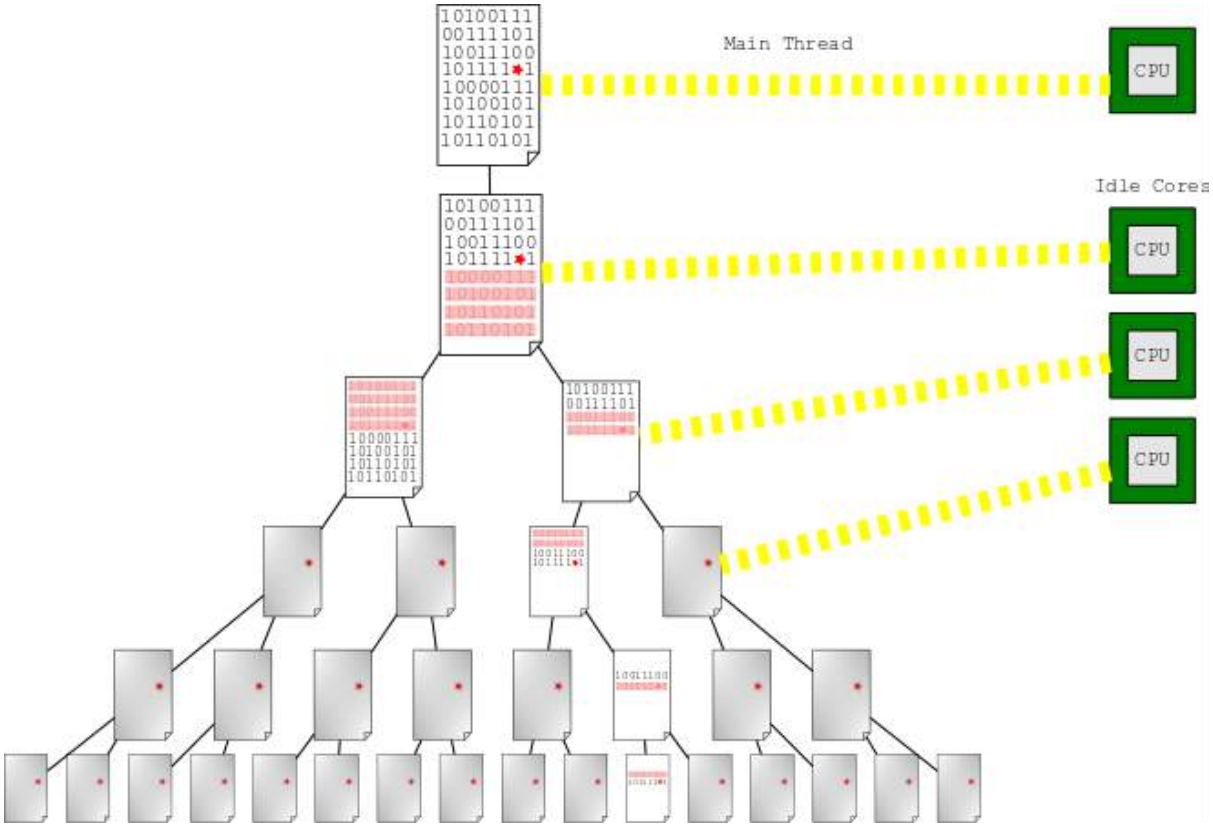
Minimization tools use various techniques to simplify the testcase, but the core algorithm is simply bisection. Bisection is an inherently serial process, you can't advance through the algorithm without knowing the result of each step. This data dependency problem can make minimization very slow, sometimes taking hours to complete while cpu cores sit idle.



In this diagram you can see we progressively remove parts of the file to determine which section is interesting.

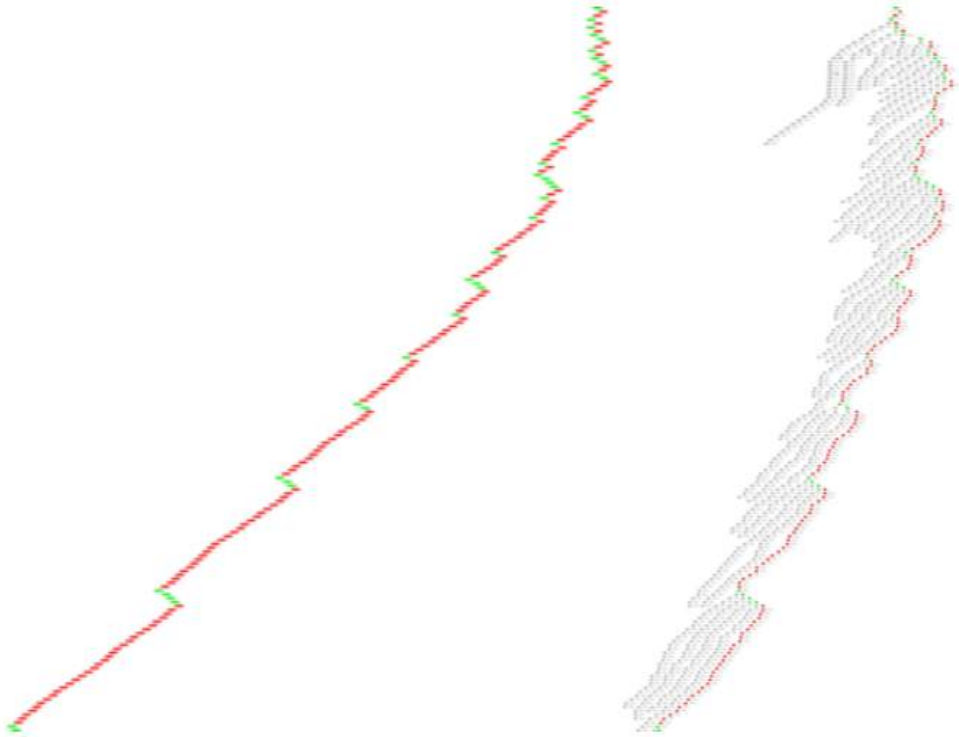
TMT solves this problem using *pessimistic speculative execution*. We build a binary tree of all the possible bisection steps and then idle cores can speculatively test future steps ahead of our position in the algorithm. In many cases, the test results are already known by the time we need them.

We call it *pessimistic*, because real workloads are characterized by long series of consecutive failures. We simply assume that tests are going to fail, and speculatively follow the failure path until proven wrong.



In this diagram, you can see we generated a binary tree of all possible outcomes, and now idle cores can speculatively work ahead of the main thread.

If you're fuzzing a target that takes more than a few seconds to run then parallelizing the minimization can dramatically speedup your workflow. Real fuzzing inputs that take several seconds to reproduce can take many hours to complete using serial bisection, but TMT can produce the same output in minutes.



This is a real minimization path from a fuzzer generated crash.

Testcase Minimisation Tool(TMT) generates a binary tree, and this graph shows the path through the tree from the root to the final leaf (discarded paths are hidden on the left to simplify the diagram).

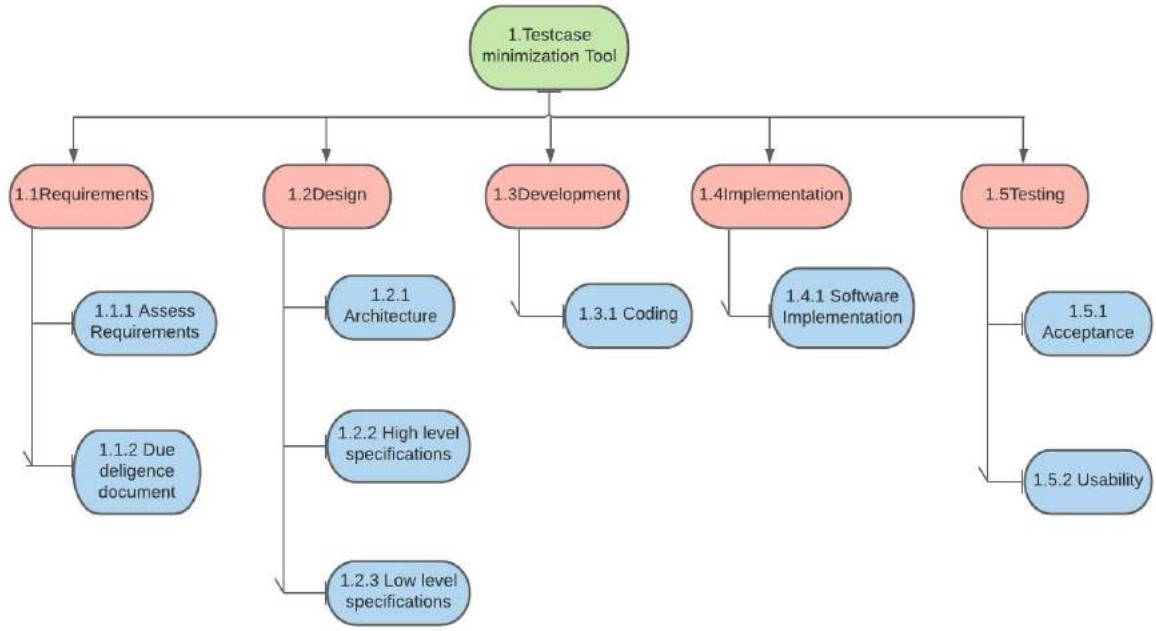
The green nodes were successful and the red nodes were failures. The grey nodes in the right were explored but discarded. Because all consecutive red nodes are executed in parallel, the actual wall clock time required to minimize the input was minimal.

Each crash took ~11 seconds to reproduce, requiring about 34 minutes of compute time - but TMT completed in just few seconds!

2. PROJECT DESCRIPTION AND GOALS:

The main distinguishing factor of Testcase Minimisation Tool is its uniqueness which is not following the traditional ways of computing and still getting the work done using the same traditional algorithms ensuring correct standpoints. If you're fuzzing a target that takes more than a few seconds to run then parallelizing the minimization can dramatically speedup your workflow. Real fuzzing inputs that take several seconds to reproduce can take many hours to complete using serial bisection, but Testcase Minimisation Tool can produce the same output in minutes.

2.1 WORK BREAKDOWN STRUCTURE:

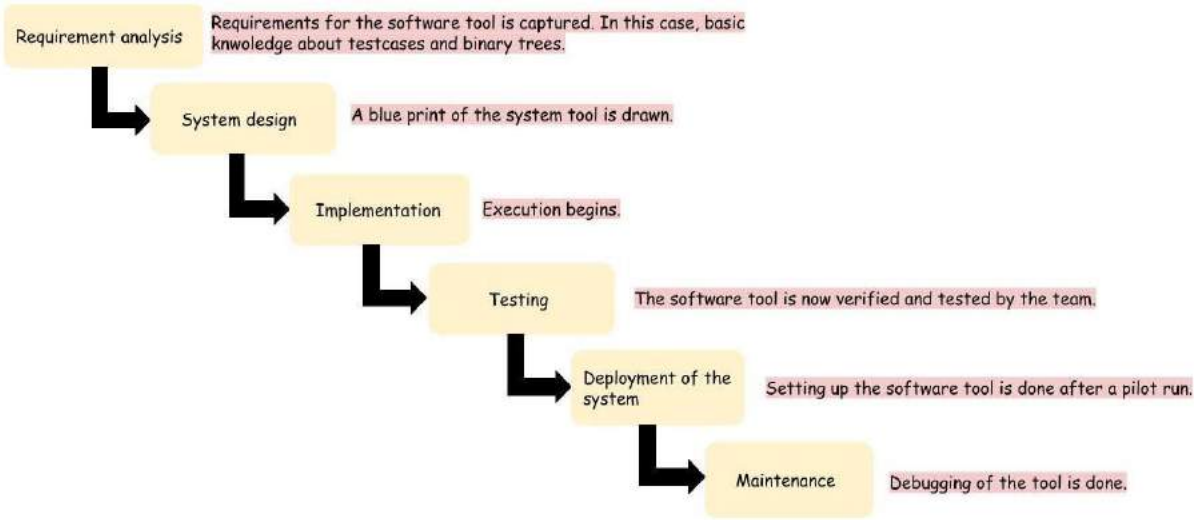


2.2 PROCESS MODEL:

We have chosen the Waterfall model for our project because the requirements are clear and easy to understand. The workflow is well and fine and the process aspects are clear.

This model is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The following illustration is a representation of the different phases of the Waterfall Model.



WHY THIS MODEL?

- For our project the Waterfall model was most appropriate because the requirements are well-documented, clear and fixed.
- It is a sequential approach which would be comfortable to work with.
- It is easy to manage due to the rigidity of the model.
- Each of its phase has some specific deliverables and a review process.
- Here, the phases are processed and completed one at a time.
- It works well for smaller projects where requirements are very well understood.
- The technology is easily understood and is not dynamic.
- There are no ambiguous requirements.
- The stages are clearly defined.
- The process and the results are well documented.

3. TECHNICAL SPECIFICATION:

3.1 System Features

3.1.1 Introduction

The Test case minimization tool shall allow a user to give input to the software. The user must first enter the test cases as an input file when asked.

3.1.2 Functional Requirements

Purpose: Taking a basic input file.

Input: Test cases

Processing: Appending the input file in the format accepted by software.

Output: Minimal output file.

3.1.3 Stimulus Response

A) User does not give an input file.

User Actions	System Actions
(1) Run without input file	
	(2) Shows error for no input
(3) Error in input file	
	(4) Shows error for wrong input

B) User gives an appropriate input file.

User Actions	System Actions
(1)Run the input	
	(2) Binary tree generation
	(3) Elimination of unnecessary test cases

3.1.4 Binary Tree generation

3.1.4.1 Introduction

The software regulates the test cases and generates a binary tree of all possible outcomes, and now idle cores can speculatively work ahead of the main thread.

3.1.4.2 Functional Requirements

Purpose: Receiving test cases and building a binary tree

Input: Test cases

Processing: Software minimizes the test cases and forms a binary tree

Output: Final binary tree

3.1.4.3 Stimulus Response

User Actions	**System Actions**
	(1) Receives input
	(2) Check if input is correct
	(3) Forms a binary tree
	(4) Verify test cases
	(5) Display minimized output
(6) User views the output	

3.2 Performance Requirements

The following tables list the performance requirements of the Test case minimization tool.

Performance Requirement	**Description**
Minimal execution time	Since all failures are executed parallelly , the actual wall clock time required to minimize the input is minimal.
Software Runtime Errors	The Test case minimization tool will handle the runtime errors consistently and as gracefully as possible.

3.3 Software System Attributes

3.3.1 Reliability

Reliability in the Testcase Minimisation Tool will be ensured by thorough unit, milestone, and release testing. Comprehensive test scenarios and acceptance criteria will be established to reflect the necessary level reliability required of the Testcase Minimisation Tool. The all delivered source code will be thoroughly tested using the established test scenarios until the acceptance criteria are satisfied by the Testcase Minimisation Tool.

3.3.2 Security

The Testcase Minimisation Tool will utilize Public / Private key encryption. This will provide test cases that are as secure as the public / private key encryption method is secure.

3.3.3 Maintainability

The Testcase Minimisation Tool is written in C programming language. C promotes good design practices due to the inherent structure of a C program.

Along with the well-formed programming enforced by C, best practice development conventions will be enforced for the construction of the Testcase Minimisation Tool. Consistent variable naming conventions will be used by all the programmers. Consistent spacing will be used in the source code by all the programmers. The design of the source code will use the principles of Object Oriented Design and the source code will be programmed using Object Oriented Programming. Object-Oriented Design and Object-Oriented programming will make the code easier to understand.

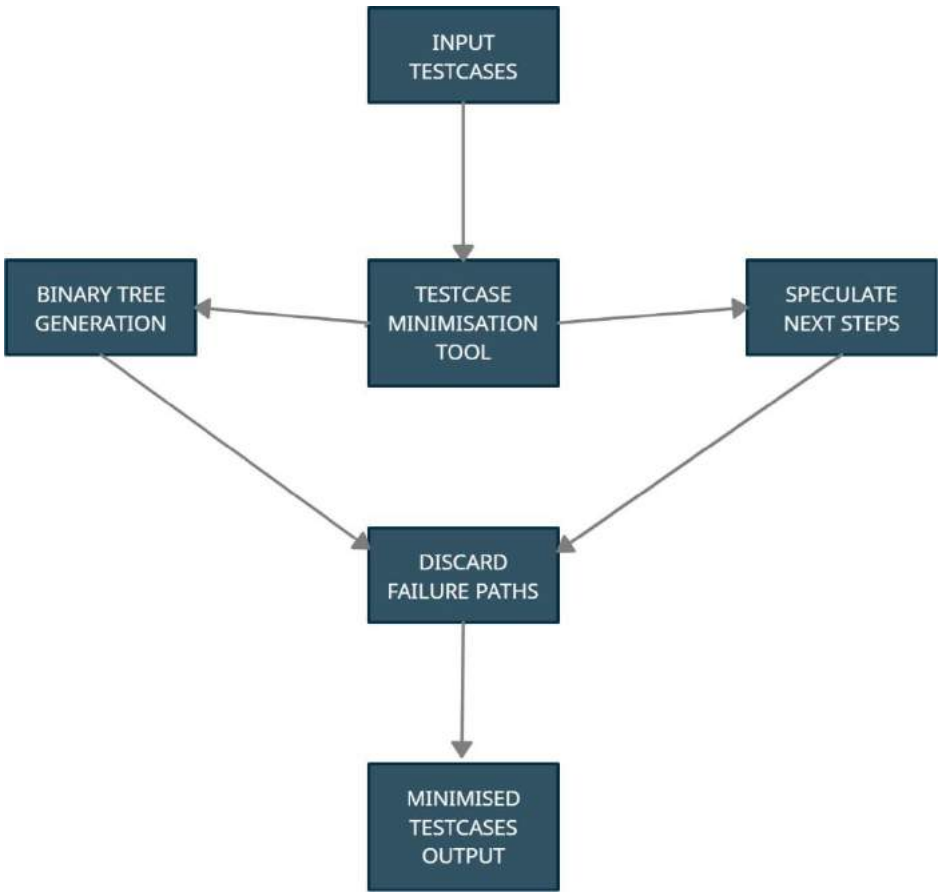
3.3.4 Portability

It is safe to say that the implementation of the Testcase Minimisation Tool will be able to be ported to other system platforms that accept C/C++ applications with little to no changes required. It is not safe to say that the Testcase Minimisation Tool will execute properly on the other system platforms with little or no change. Significant changes to the Testcase Minimisation Tool may be required to ensure proper execution on other system platforms.

4. DESIGN APPROACH AND DETAILS:

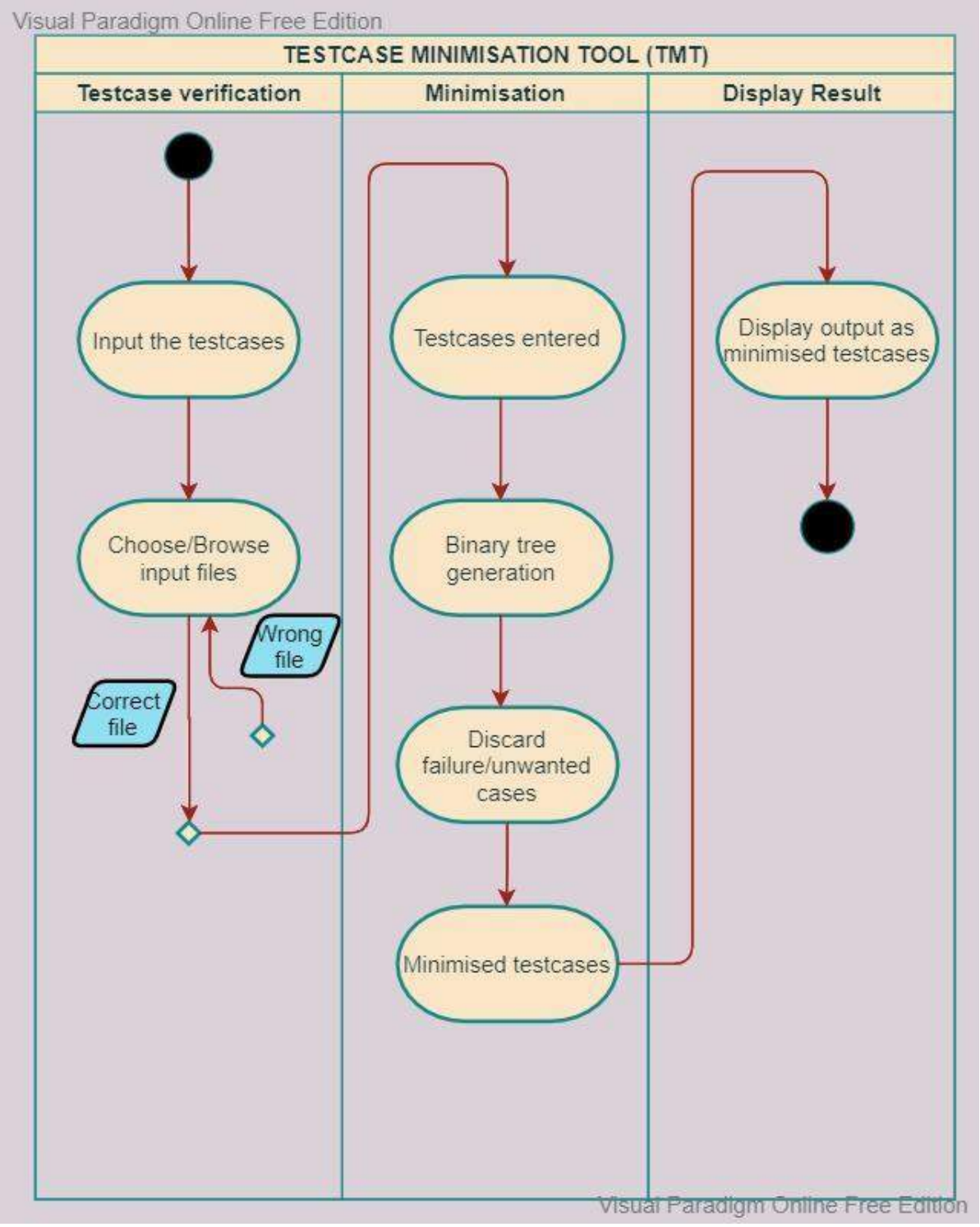
4.1 DESIGN APPROACH / MATERIALS & METHODS:

4.1.1 Architectural Diagram:

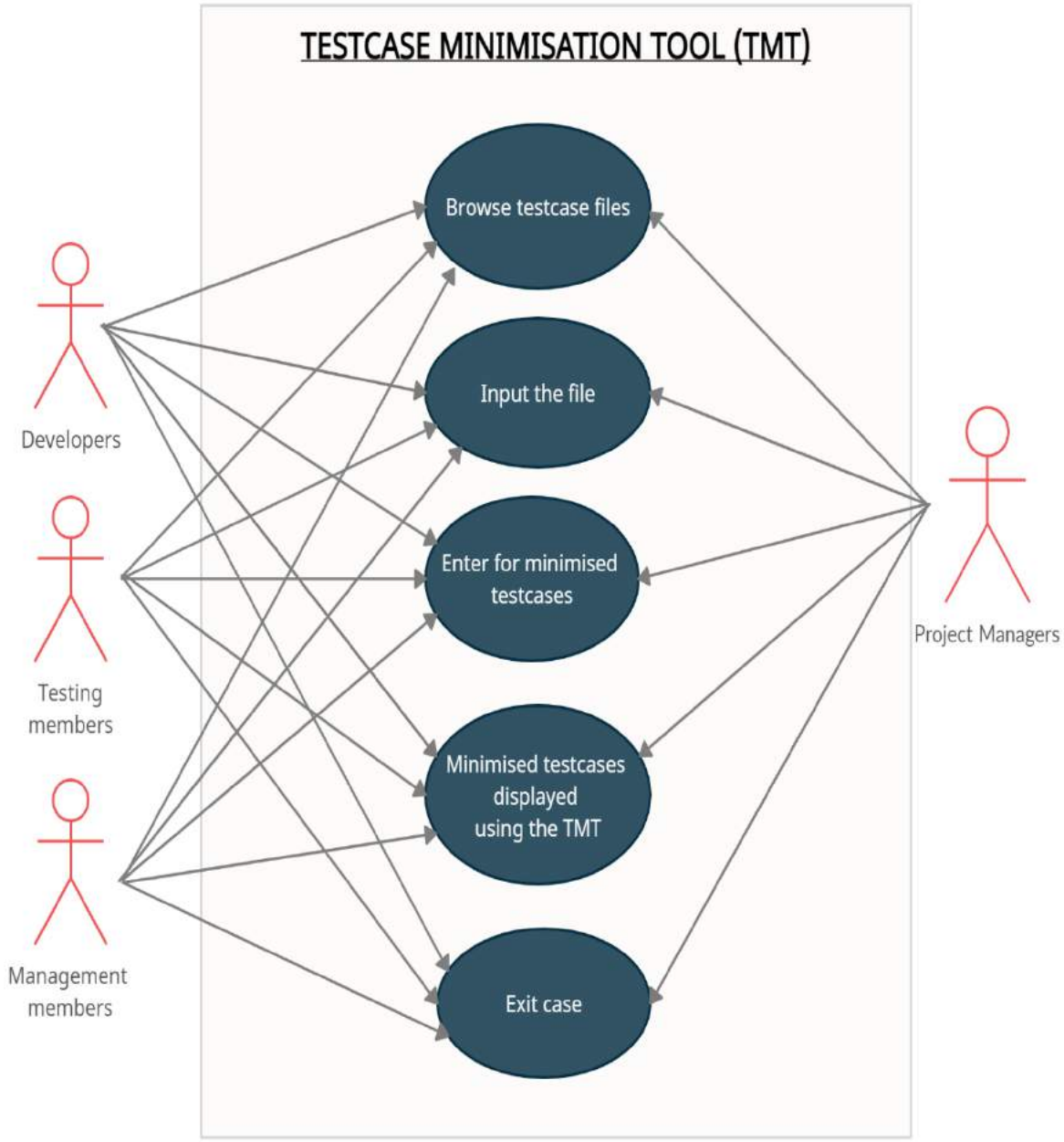


The above architectural diagram shows the graphical representation of set of concepts that are part of our Testcase Minimization Tool, including their principles, elements and components. This diagram is a data flow type of architectural system.

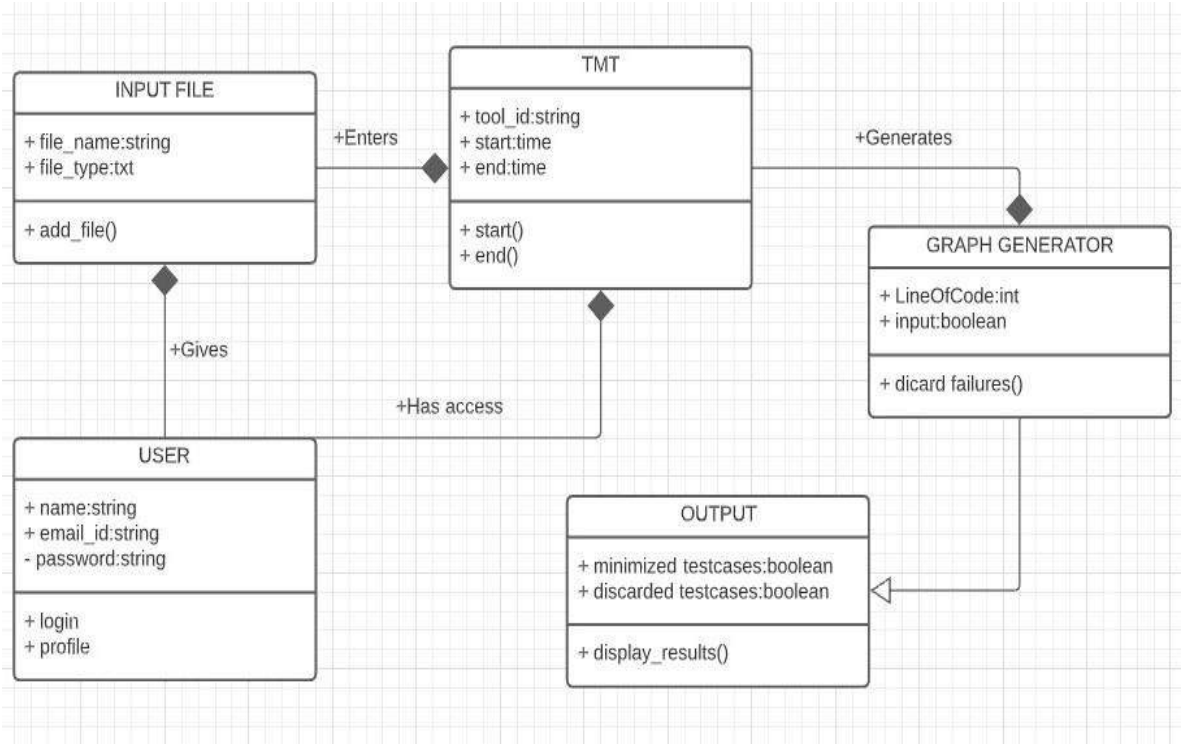
4.1.2 Swimlane Activity Diagram:



4.1.3 Use Case Diagram:

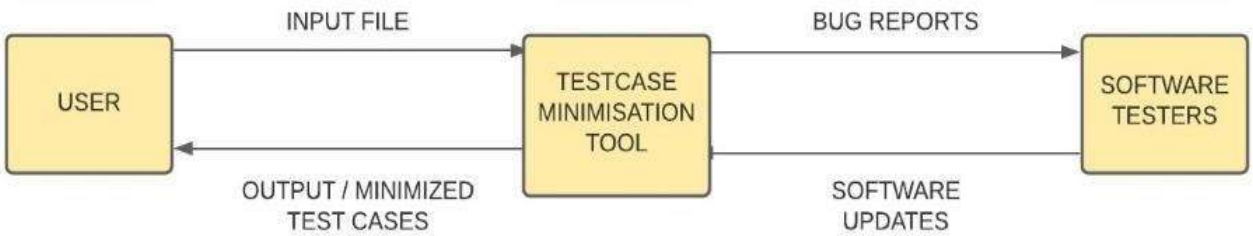


4.1.4 Class Diagram:

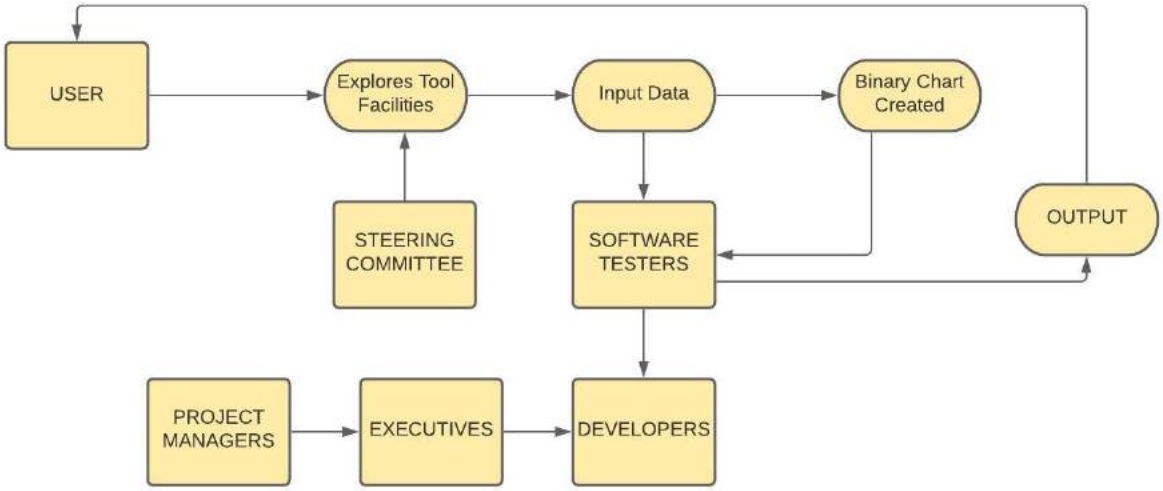


4.1.5 Data Flow Diagram:

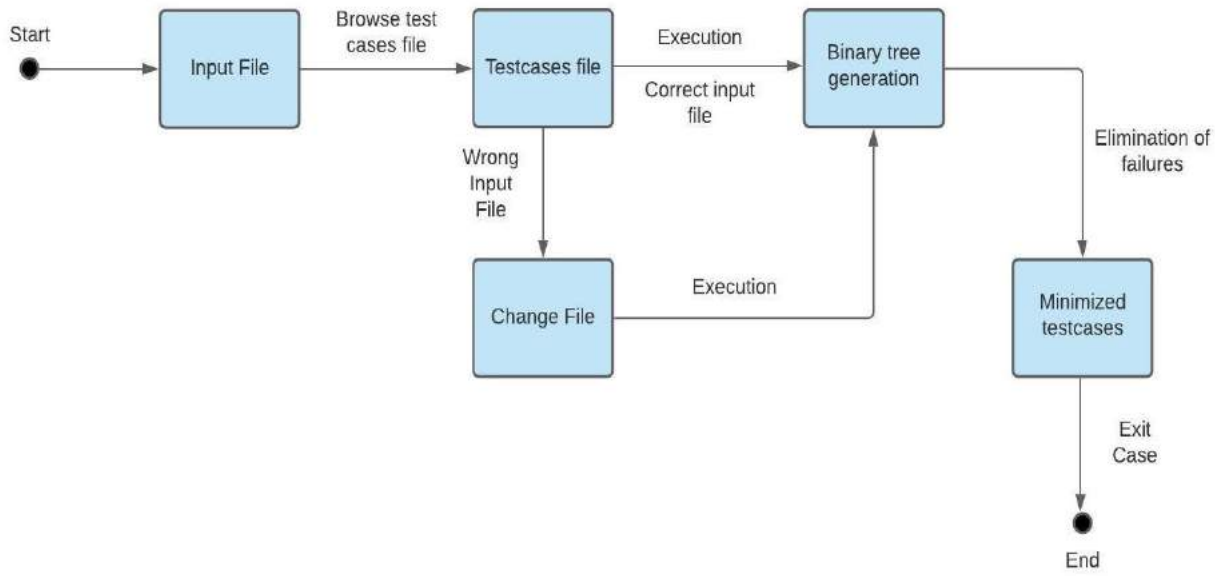
Level 0 DFD:



Level 1 DFD:



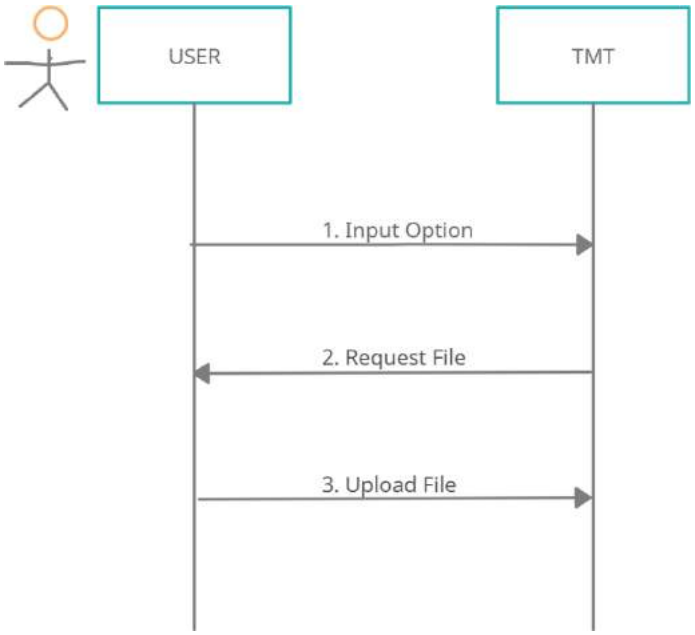
4.1.6 State Transition Diagram:



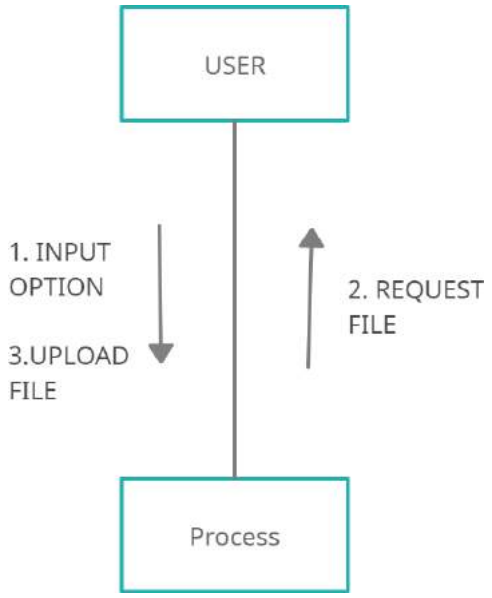
4.1.7 Sequence & Collaboration Diagrams:

4.1.7.1 Configuration Module:

SEQUENCE DIAGRAM-

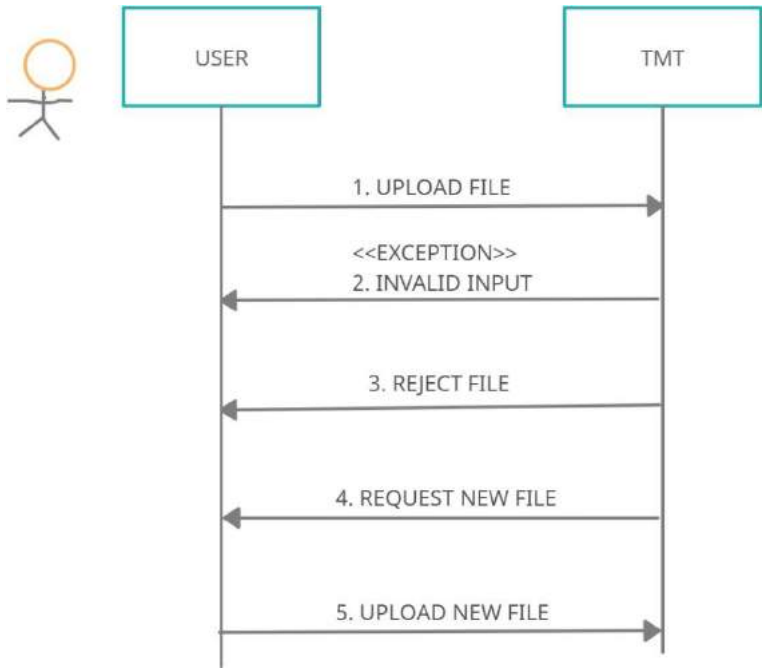


COLLABORATION DIAGRAM:

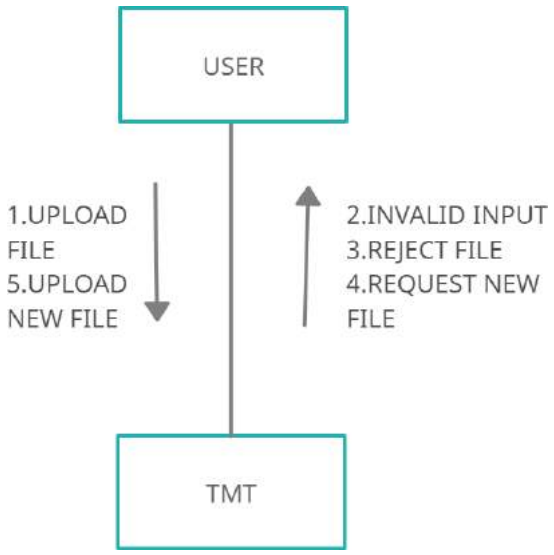


4.1.7.2 Input Testcases Module:

SEQUENCE DIAGRAM-

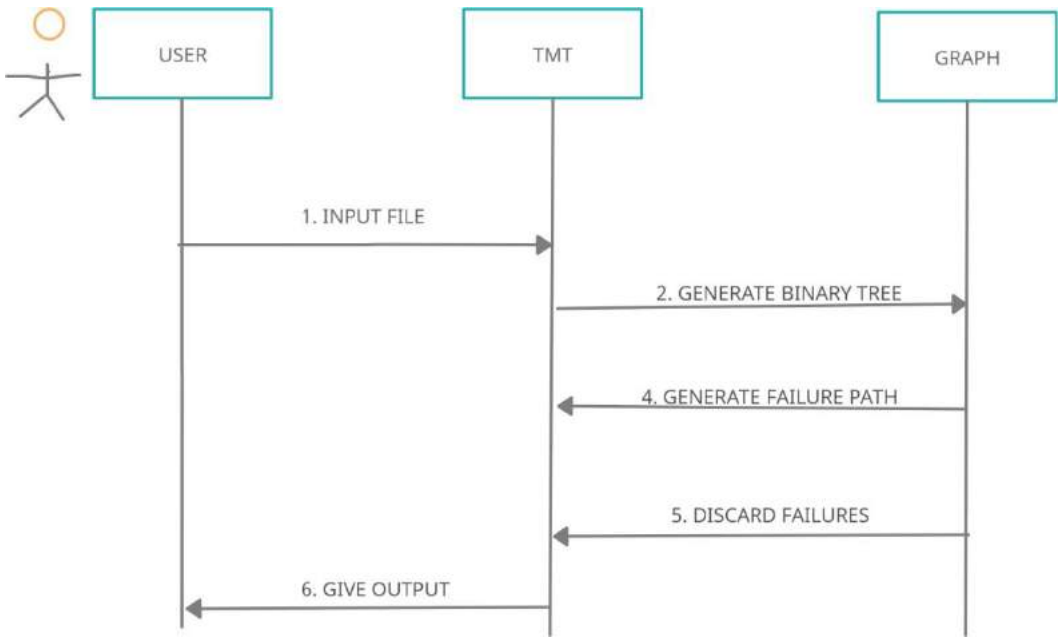


COLLABORATION DIAGRAM:

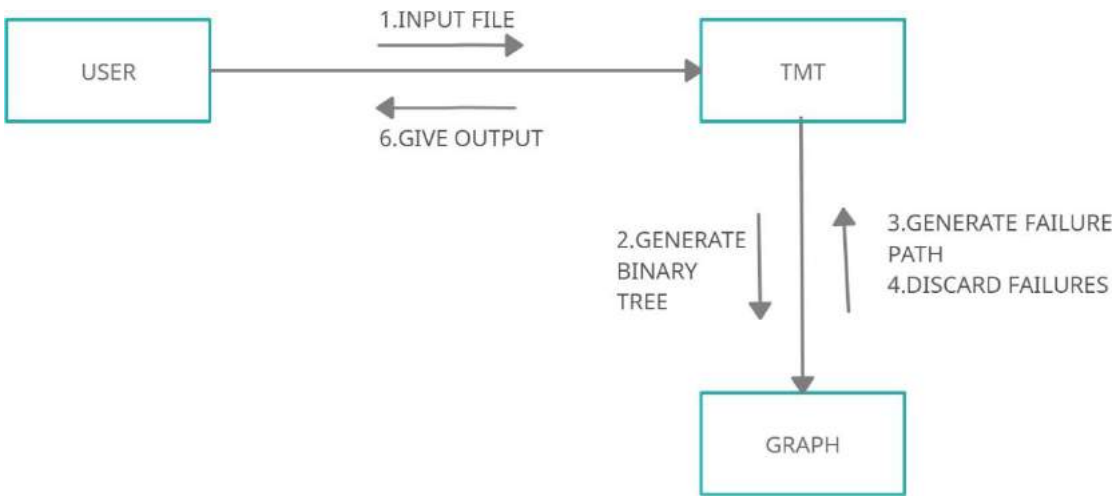


4.1.7.3 Processing Testcases Module:

SEQUENCE DIAGRAM-

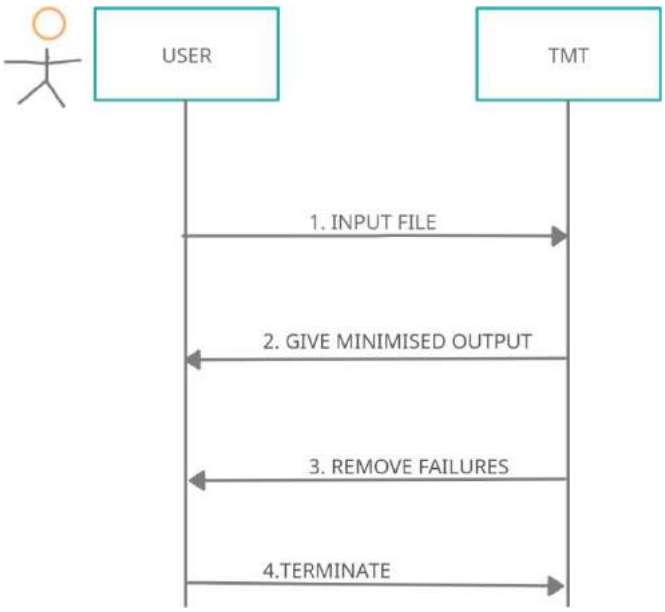


COLLABORATION DIAGRAM:

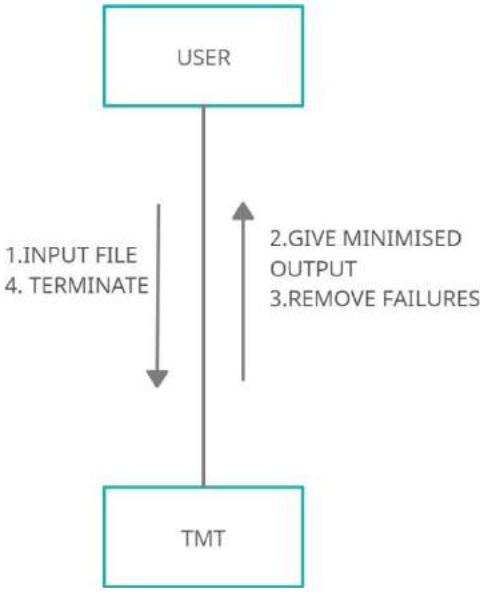


4.1.7.4 Output Testcases Module:

SEQUENCE DIAGRAM-



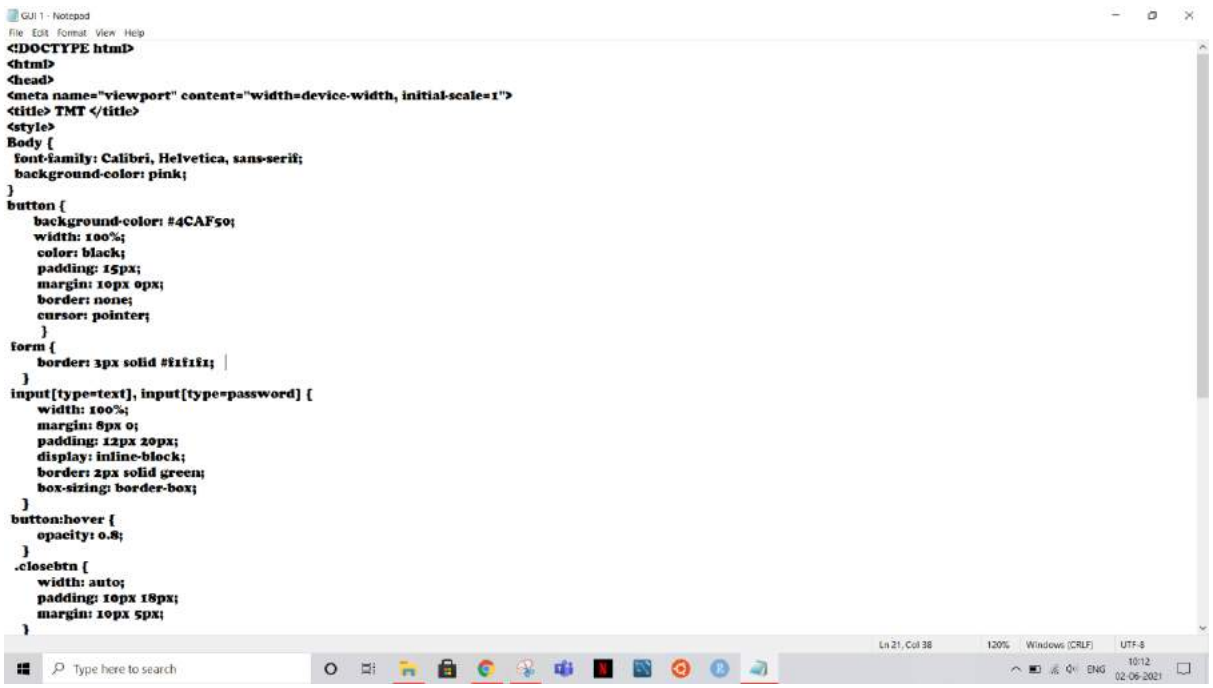
COLLABORATION DIAGRAM:



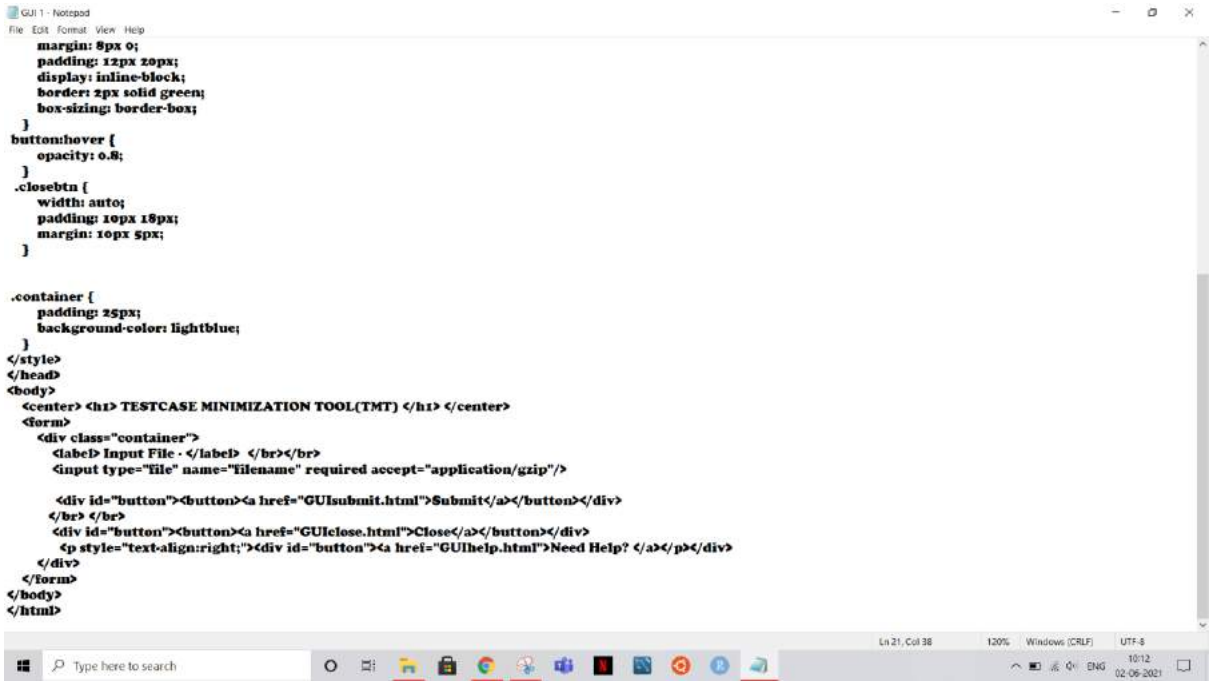
4.2 CODES AND STANDARDS:

4.2.1 FRONT END CODE SNIPPETS (FOR THE GUI):

MAIN PAGE:



```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title> TMT </title>
<style>
Body {
 font-family: Calibri, Helvetica, sans-serif;
 background-color: pink;
}
button {
      background-color: #4CAF50;
      width: 100%;
       color: black;
       padding: 15px;
       margin: 10px 0px;
       border: none;
       cursor: pointer;
        }
 form {
       border: 3px solid #f1f1f1;
   }
 input[type=text], input[type=password] {
       width: 100%;
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
   }
 button:hover {
       opacity: 0.8;
   }
 .closebtn {
       width: auto;
       padding: 10px 18px;
       margin: 10px 5px;
   }
```



```
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
   }
 button:hover {
       opacity: 0.8;
   }
 .closebtn {
       width: auto;
       padding: 10px 18px;
       margin: 10px 5px;
   }


 .container {
       padding: 25px;
       background-color: lightblue;
   }
</style>
</head>
<body>
   <center> <h1> TESTCASE MINIMIZATION TOOL(TMT) </h1> </center>
   <form>
      <div class="container">
         <label> Input File : </label>  </br></br>
         <input type="file" name="filename" required accept="application/gzip"/>

          <div id="button"><button><a href="GUIsubmit.html">Submit</a></button></div>
        </br> </br>
         <div id="button"><button><a href="GUIclose.html">Close</a></button></div>
          <p style="text-align:right;"><div id="button"><a href="GUIhelp.html">Need Help? </a></p></div>
      </div>
   </form>
</body>
</html>
```

CLOSE PAGE:

GUIclose - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title> TMT </title>
<style>
Body {
 font-family: Calibri, Helvetica, sans-serif;
 background-color: pink;
}
button {
      background-color: #4CAF50;
      width: 100%;
       color: black;
       padding: 15px;
       margin: 10px 0px;
       border: none;
       cursor: pointer;
        }
form {
       border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
       width: 100%;
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
  }
button:hover {
       opacity: 0.8;
  }

.container {
       padding: 25px;
       background-color: lightblue;
  }
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

Type here to search

ENG 10:14 02-06-2021

GUIclose - Notepad

File Edit Format View Help

```
button {
      background-color: #4CAF50;
      width: 100%;
       color: black;
       padding: 15px;
       margin: 10px 0px;
       border: none;
       cursor: pointer;
        }
form {
       border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
       width: 100%;
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
  }
button:hover {
       opacity: 0.8;
  }

.container {
       padding: 25px;
       background-color: lightblue;
  }
</style>
</head>
<body>
  <center> <h1> TESTCASE MINIMIZATION TOOL(TMT) </h1> </center>
</br>
<h1 style="color:blue;" ><p style="text-align:center;"> Thank You For Using Our Tool !!</p></h1>

</body>
</html>
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

Type here to search

ENG 10:14 02-06-2021

HELP PAGE:

GUIhelp - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title> TMT </title>
<style>
Body {
 font-family: Calibri, Helvetica, sans-serif;
 background-color: pink;
}
button {
      background-color: #4CAF50;
      width: 100%;
       color: black;
       padding: 15px;
       margin: 10px 0px;
       border: none;
       cursor: pointer;
        }
form {
       border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
       width: 100%;
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
  }
button:hover {
       opacity: 0.8;
  }

.container {
       padding: 25px;
       background-color: lightblue;
  }
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

Type here to search ENG 10:15 02-06-2021

GUIhelp - Notepad

File Edit Format View Help

```
       margin: 10px 0px;
       border: none;
       cursor: pointer;
        }
form {
       border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
       width: 100%;
       margin: 8px 0;
       padding: 12px 20px;
       display: inline-block;
       border: 2px solid green;
       box-sizing: border-box;
  }
button:hover {
       opacity: 0.8;
  }

.container {
       padding: 25px;
       background-color: lightblue;
  }
</style>
</head>
<body>
   <center> <h1> TESTCASE MINIMIZATION TOOL(TMT) </h1> </center>
</br>
<h4 style="color:purple;"><p>TMT is a new testcase minimization tool, designed with parallelization in mind. TMT was built to use strategies and techniques that
dramatically speed up the minimization process.If you're fuzzing a target that takes more than a few seconds to run then parallelizing the minimization can
dramatically speedup your workflow. Real fuzzing inputs that take several seconds to reproduce can take many hours to complete using serial bisection, but TMT can
produce the same output in minutes.</br></br>
An input file of .gz(gzip) format is recommended.</br></br>
First, create a shell script that when given your input on stdin, returns zero.</br></br>
Now simply run TMT with your input and it will find the smallest version that still returns zero.
</body>
</html>
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

Type here to search ENG 10:15 02-06-2021

SUBMIT PAGE:

GUIsubmit - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title> TMT </title>
<style>
Body {
 font-family: Calibri, Helvetica, sans-serif;
 background-color: pink;
}
button {
      background-color: #4CAF50;
      width: 100%;
      color: black;
      padding: 15px;
      margin: 10px 0px;
      border: none;
      cursor: pointer;
       }
form {
      border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
      width: 100%;
      margin: 8px 0;
      padding: 12px 20px;
      display: inline-block;
      border: 2px solid green;
      box-sizing: border-box;
  }
button:hover {
      opacity: 0.8;
  }
 .closebtn {
      width: auto;
      padding: 10px 18px;
      margin: 10px 5px;
  }
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

Type here to search ENG 10:16 02-06-2021

GUIsubmit - Notepad

File Edit Format View Help

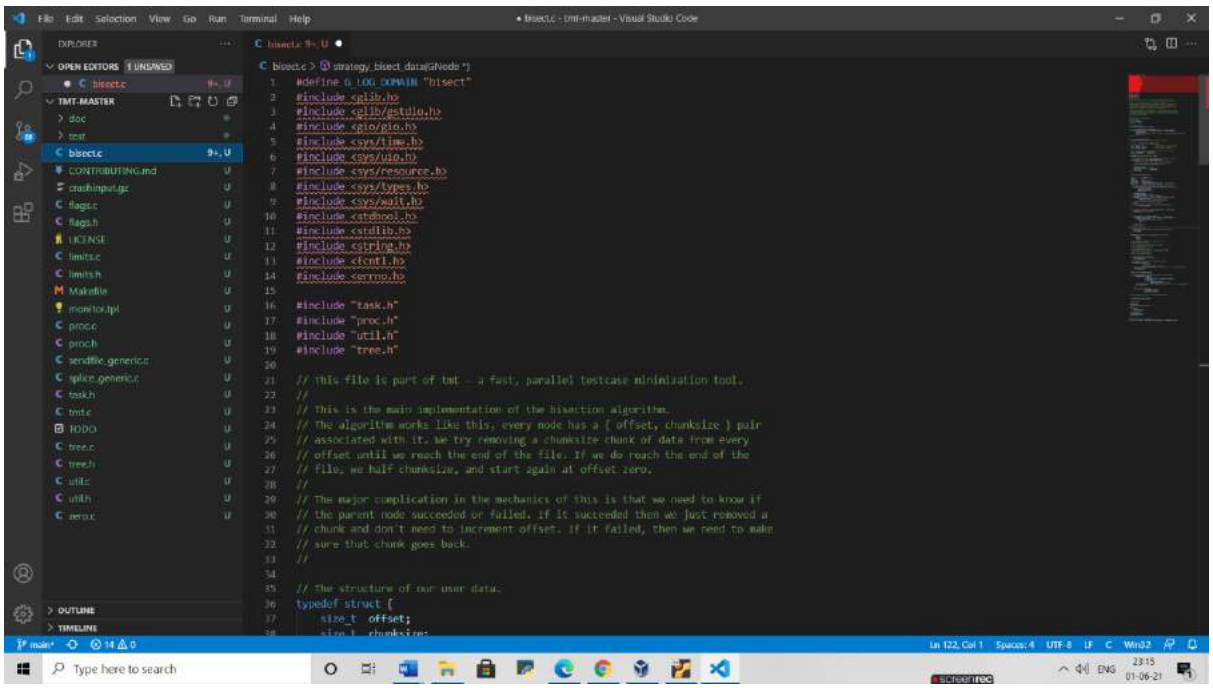
```
      cursor: pointer;
       }
form {
      border: 3px solid #f1f1f1;
  }
input[type=text], input[type=password] {
      width: 100%;
      margin: 8px 0;
      padding: 12px 20px;
      display: inline-block;
      border: 2px solid green;
      box-sizing: border-box;
  }
button:hover {
      opacity: 0.8;
  }
 .closebtn {
      width: auto;
      padding: 10px 18px;
      margin: 10px 5px;
  }


.container {
      padding: 25px;
      background-color: lightblue;
  }
</style>
</head>
<body>
   <center> <h1> TESTCASE MINIMIZATION TOOL(TMT) </h1> </center>
</br></br>
<h2 style="color:blue;">MINIMIZED TESTCASES : </h2>

</body>
</html>
```

Ln 1, Col 1 120% Windows (CRLF) UTF-8

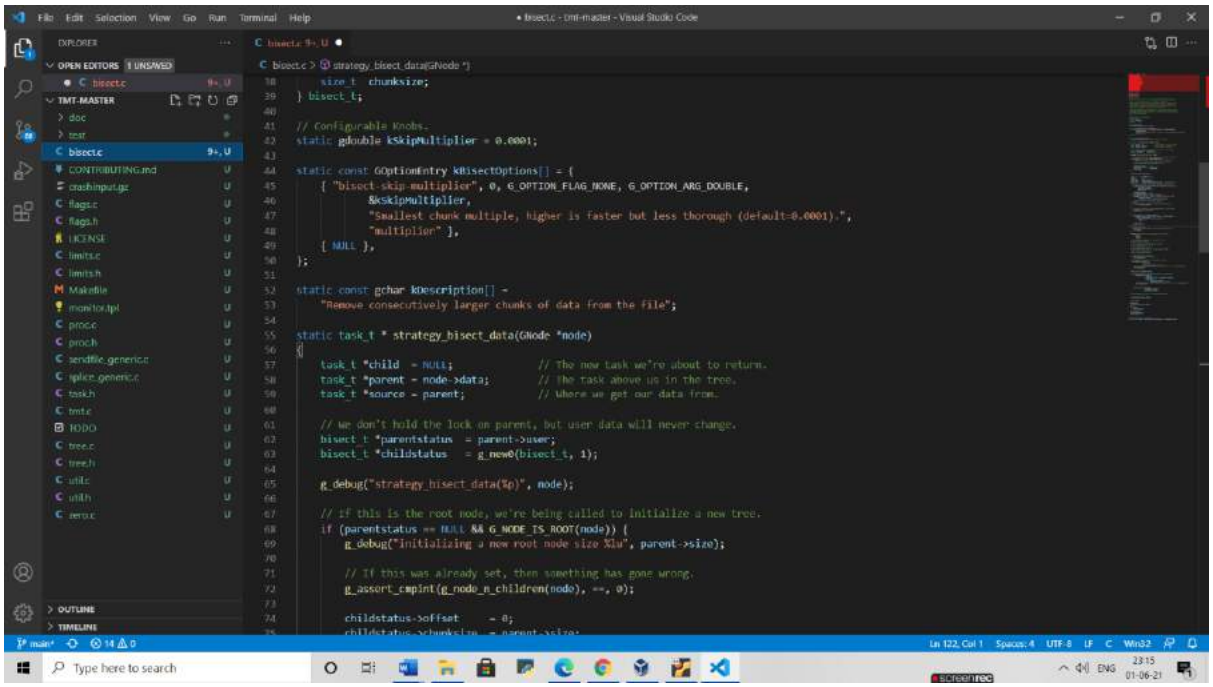
Type here to search ENG 10:16 02-06-2021

4.2.2 BACKEND CODE SNIPPETS:

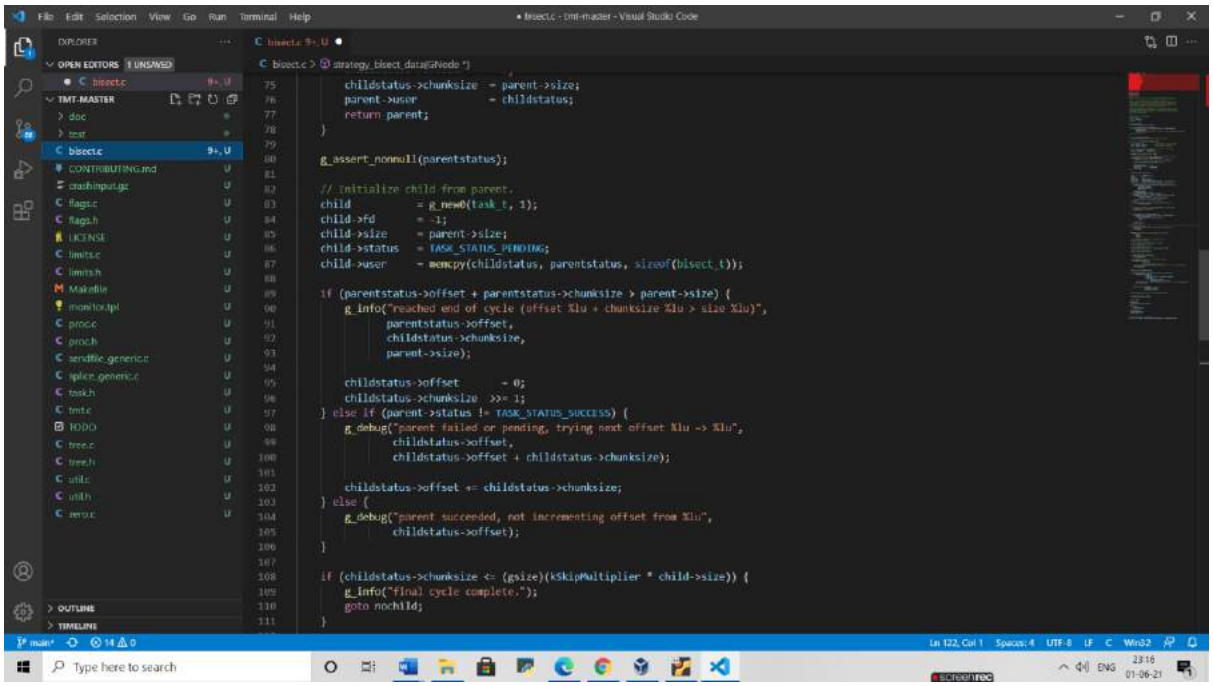
For the bisection of binary tree-



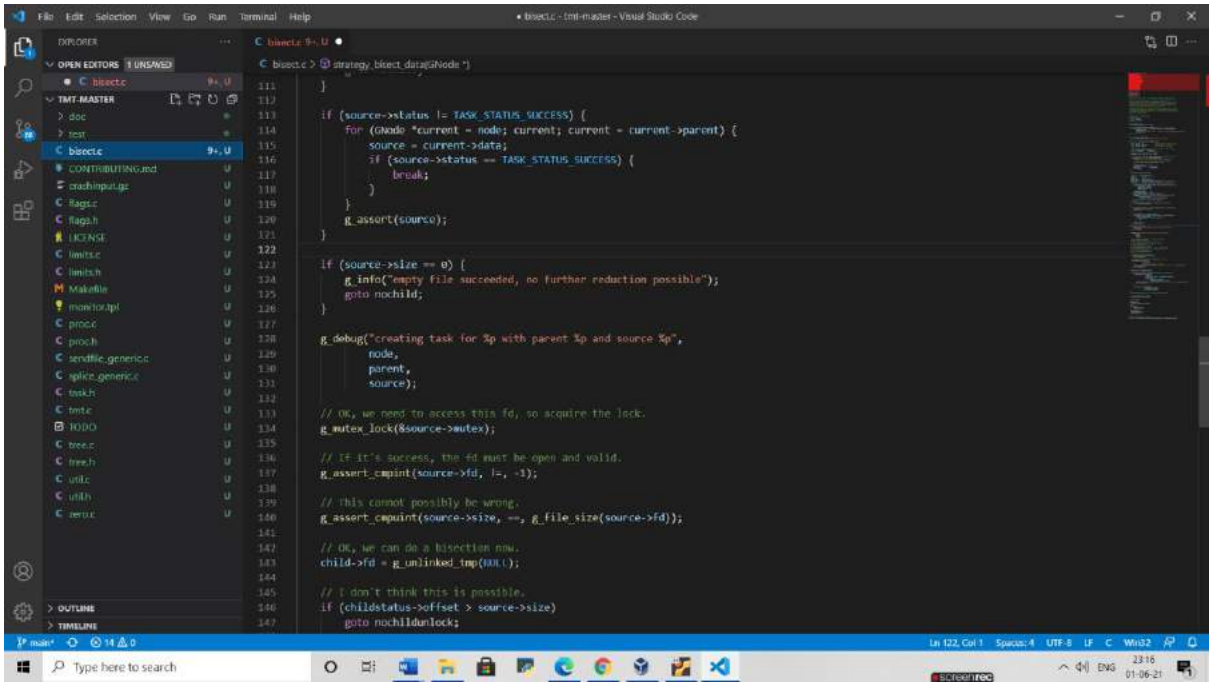
```c
#define G_LOG_DOMAIN "bisect"
#include <glib.h>
#include <glib/gstdio.h>
#include <gio/gio.h>
#include <sys/time.h>
#include <sys/wio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

#include "task.h"
#include "proc.h"
#include "util.h"
#include "tree.h"

// this file is part of tmt -- a fast, parallel testcase minimization tool.
//
// This is the main implementation of the bisection algorithm.
// The algorithm works like this, every node has a ( offset, chunksize ) pair
// associated with it. We try removing a chunksize chunk of data from every
// offset until we reach the end of the file. If we do reach the end of the
// file, we half chunksize, and start again at offset zero.
//
// The major complication in the mechanics of this is that we need to know if
// the parent node succeeded or failed. If it succeeded then we just removed a
// chunk and don't need to increment offset. If it failed, then we need to make
// sure that chunk goes back.
//

// The structure of our user data.
typedef struct {
    size_t  offset;
```



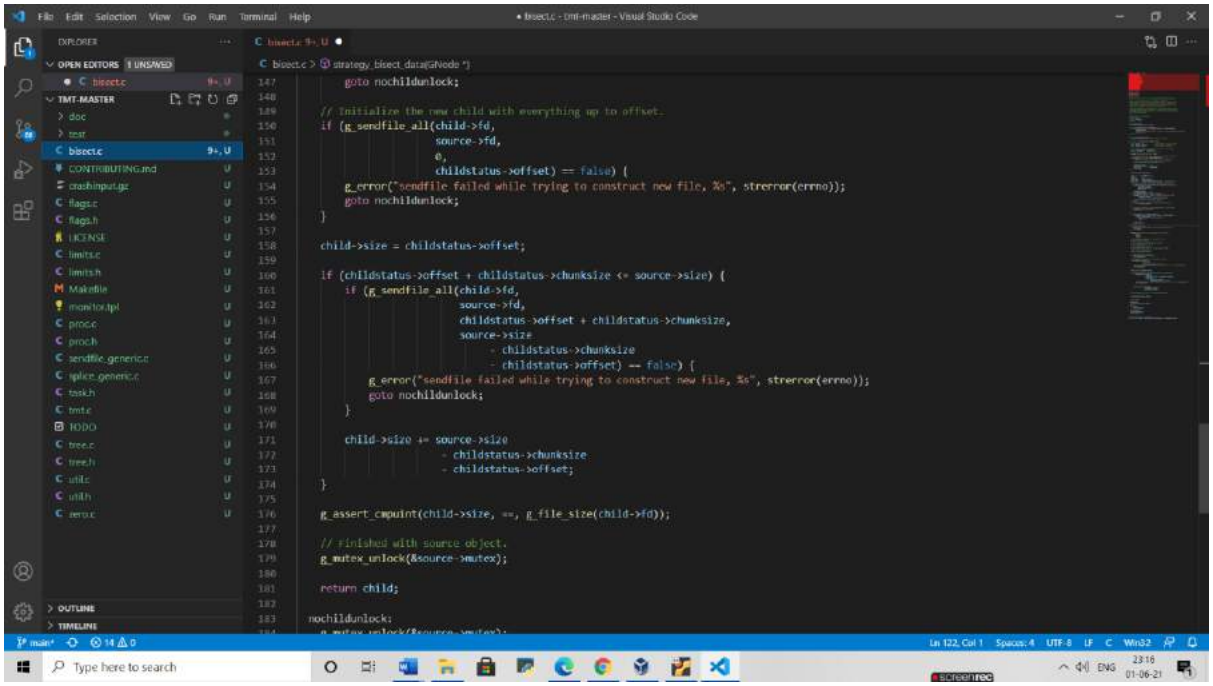
```c
    size_t  chunksize;
} bisect_t;

// Configurable Knobs.
static gdouble kSkipMultiplier = 0.0001;

static const GOptionEntry kBisectOptions[] = {
    { "bisect-skip-multiplier", 0, G_OPTION_FLAG_NONE, G_OPTION_ARG_DOUBLE,
            &kSkipMultiplier,
            "Smallest chunk multiple, higher is faster but less thorough (default=0.0001).",
            "multiplier" },
    { NULL },
};

static const gchar kDescription[] =
    "Remove consecutively larger chunks of data from the file";

static task_t * strategy_bisect_data(GNode *node)
{
    task_t *child  = NULL;          // The new task we're about to return.
    task_t *parent = node->data;    // The task above us in the tree.
    task_t *source = parent;        // Where we get our data from.

    // We don't hold the lock on parent, but user_data will never change.
    bisect_t *parentstatus  = parent->user;
    bisect_t *childstatus   = g_new0(bisect_t, 1);

    g_debug("strategy_bisect_data(%p)", node);

    // If this is the root node, we're being called to initialize a new tree.
    if (parentstatus == NULL && G_NODE_IS_ROOT(node)) {
        g_debug("initializing a new root node size %lu", parent->size);

        // If this was already set, then something has gone wrong.
        g_assert_cmpint(g_node_n_children(node), ==, 0);

        childstatus->offset     = 0;
```

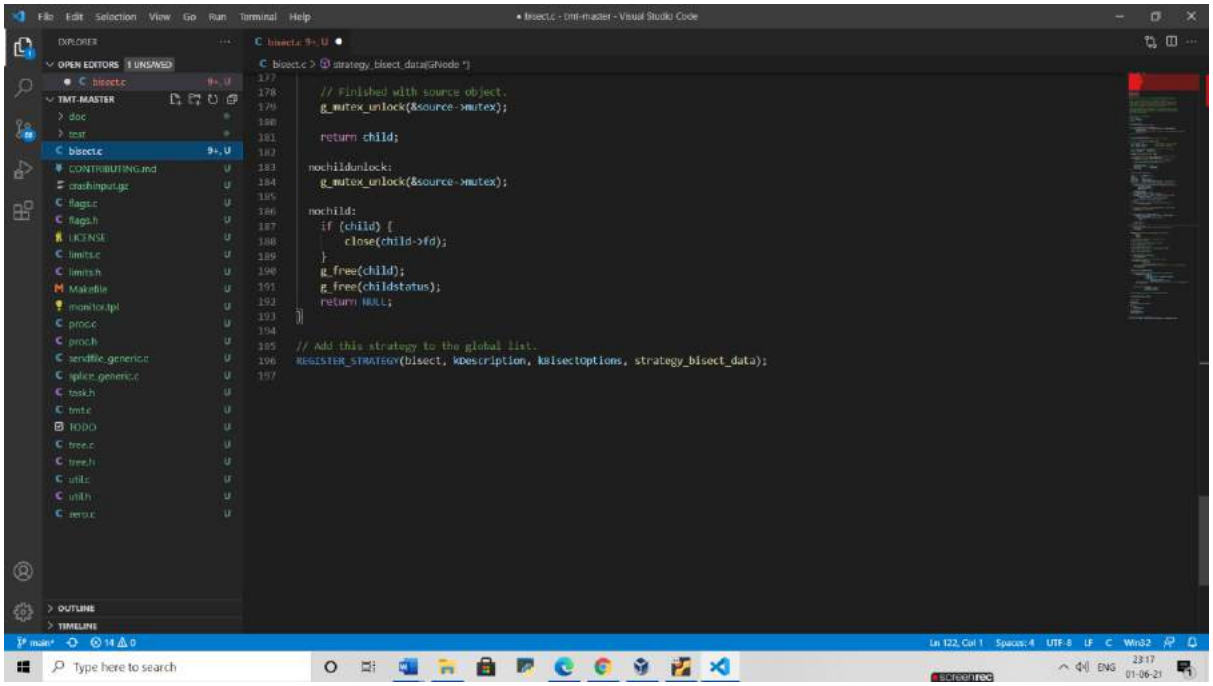


```
        childstatus->chunksize  = parent->size;
        parent->user            = childstatus;
        return parent;
    }

    g_assert_nonnull(parentstatus);

    // Initialize child from parent.
    child           = g_new0(task_t, 1);
    child->fd       = -1;
    child->size     = parent->size;
    child->status   = TASK_STATUS_PENDING;
    child->user     = memcpy(childstatus, parentstatus, sizeof(bisect_t));

    if (parentstatus->offset + parentstatus->chunksize > parent->size) {
        g_info("reached end of cycle (offset %lu + chunksize %lu > size %lu)",
               parentstatus->offset,
               childstatus->chunksize,
               parent->size);

        childstatus->offset       = 0;
        childstatus->chunksize  >>= 1;
    } else if (parent->status != TASK_STATUS_SUCCESS) {
        g_debug("parent failed or pending, trying next offset %lu => %lu",
                childstatus->offset,
                childstatus->offset + childstatus->chunksize);

        childstatus->offset += childstatus->chunksize;
    } else {
        g_debug("parent succeeded, not incrementing offset from %lu",
                childstatus->offset);
    }

    if (childstatus->chunksize <= (gsize)(kSkipMultiplier * child->size)) {
        g_info("final cycle complete.");
        goto nochild;
    }
```

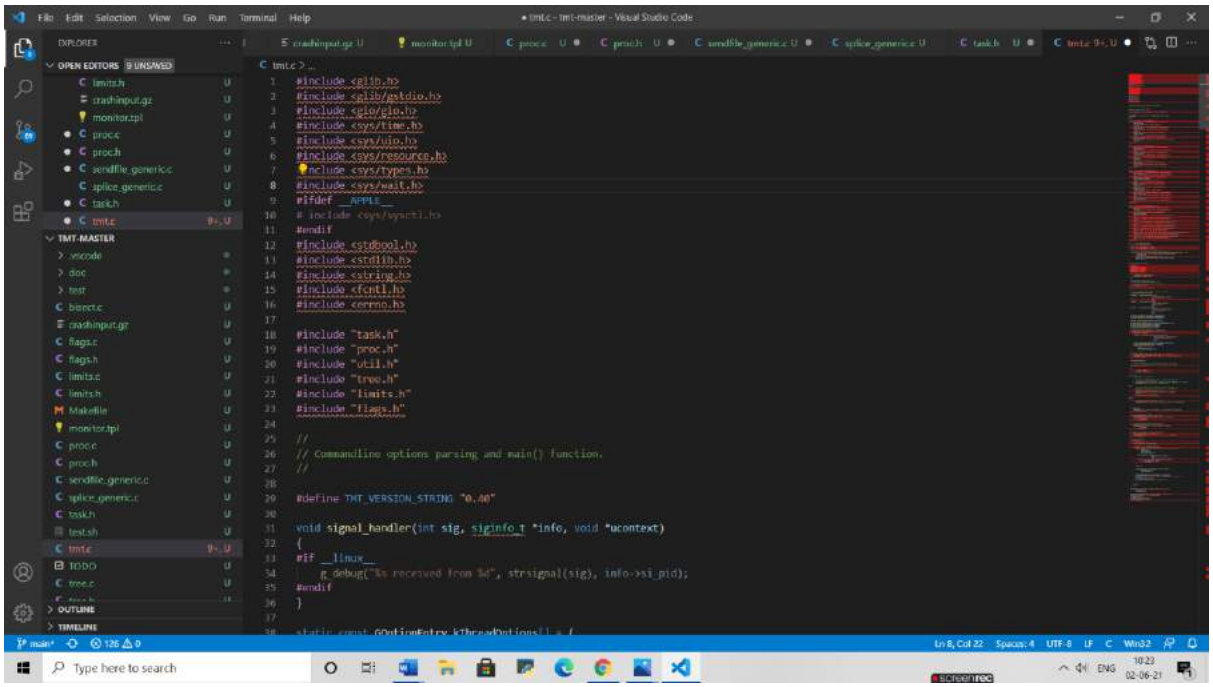


```
    }

    if (source->status != TASK_STATUS_SUCCESS) {
        for (GNode *current = node; current; current = current->parent) {
            source = current->data;
            if (source->status == TASK_STATUS_SUCCESS) {
                break;
            }
        }
        g_assert(source);
    }

    if (source->size == 0) {
        g_info("empty file succeeded, no further reduction possible");
        goto nochild;
    }

    g_debug("creating task for %p with parent %p and source %p",
            node,
            parent,
            source);

    // OK, we need to access this fd, so acquire the lock.
    g_mutex_lock(&source->mutex);

    // If it's success, the fd must be open and valid.
    g_assert_cmpint(source->fd, !=, -1);

    // This cannot possibly be wrong.
    g_assert_cmpuint(source->size, ==, g_file_size(source->fd));

    // OK, we can do a bisection now.
    child->fd = g_unlinked_tmp(NULL);

    // I don't think this is possible.
    if (childstatus->offset > source->size)
        goto nochildunlock;
```

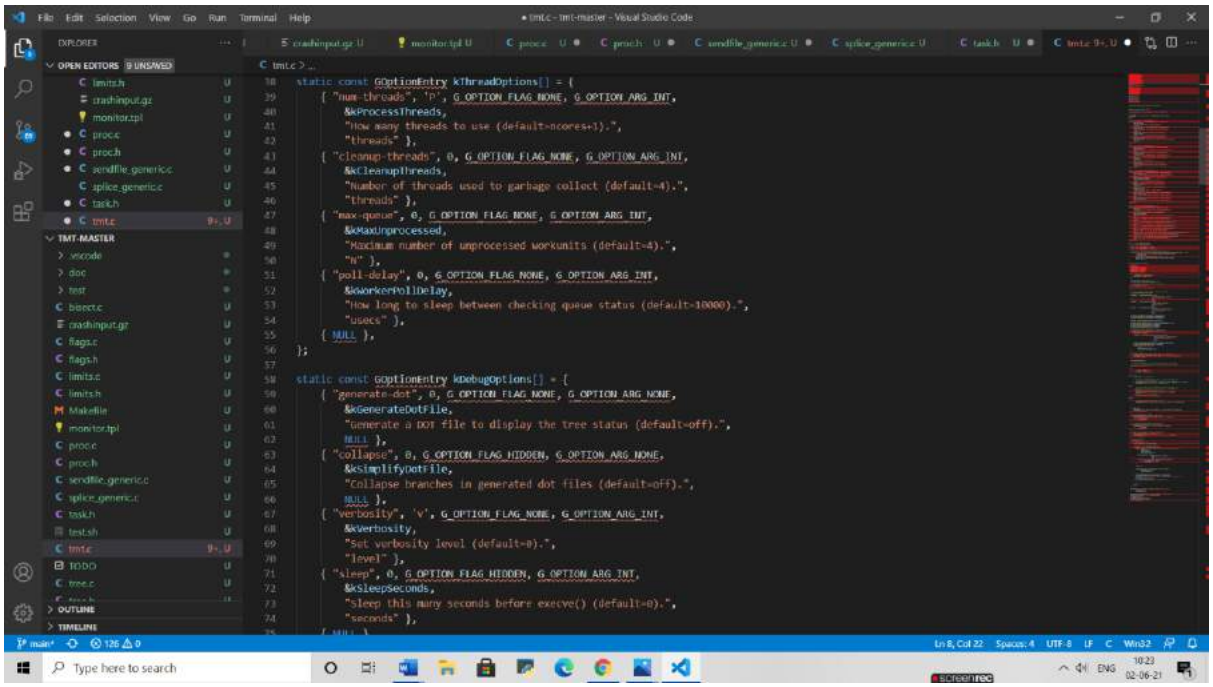

File Edit Selection View Go Run Terminal Help
• bisect.c - tmt-master - Visual Studio Code
goto nochildunlock;
// Initialize the new child with everything up to offset.
if (g_sendfile_all(child->fd,
source->fd,
0,
childstatus->offset) == false) {
g_error("sendfile failed while trying to construct new file, %s", strerror(errno));
goto nochildunlock;
}
child->size = childstatus->offset;
if (childstatus->offset + childstatus->chunksize <= source->size) {
if (g_sendfile_all(child->fd,
source->fd,
childstatus->offset + childstatus->chunksize,
source->size
- childstatus->chunksize
- childstatus->offset) == false) {
g_error("sendfile failed while trying to construct new file, %s", strerror(errno));
goto nochildunlock;
}
child->size += source->size
- childstatus->chunksize
- childstatus->offset;
}
g_assert_cmpuint(child->size, ==, g_file_size(child->fd));
// Finished with source object.
g_mutex_unlock(&source->mutex);
return child;
nochildunlock:
Type here to search


File Edit Selection View Go Run Terminal Help
• bisect.c - tmt-master - Visual Studio Code
// Finished with source object.
g_mutex_unlock(&source->mutex);
return child;
nochildunlock:
g_mutex_unlock(&source->mutex);
nochild:
if (child) {
close(child->fd);
}
g_free(child);
g_free(childstatus);
return NULL;
}
// Add this strategy to the global list.
REGISTER_STRATEGY(bisect, N_("description"), N_("bisect options"), strategy_bisect_data);
Type here to search

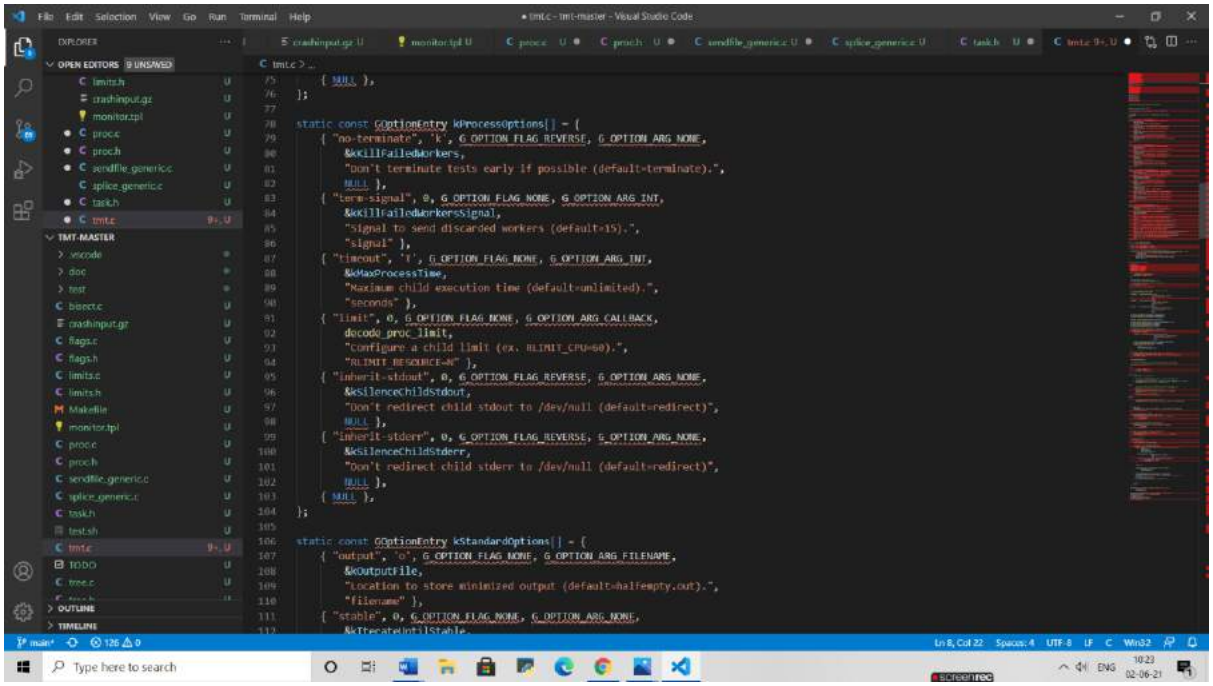
Main Code for the minimization tool-

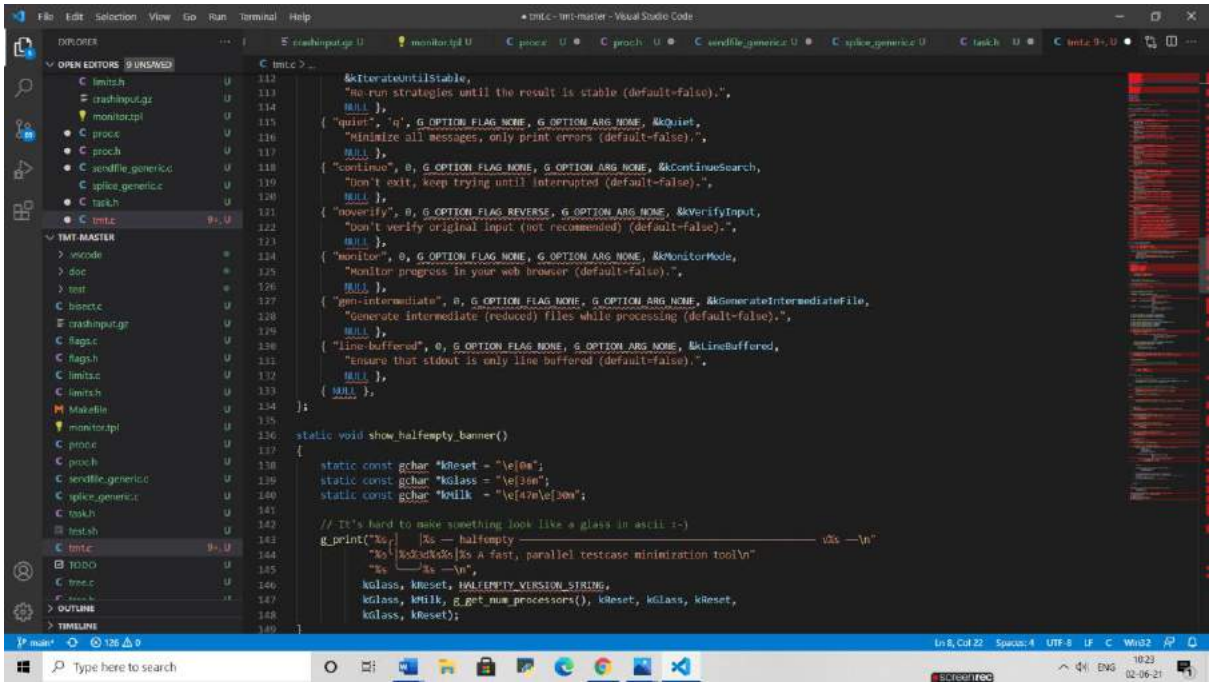


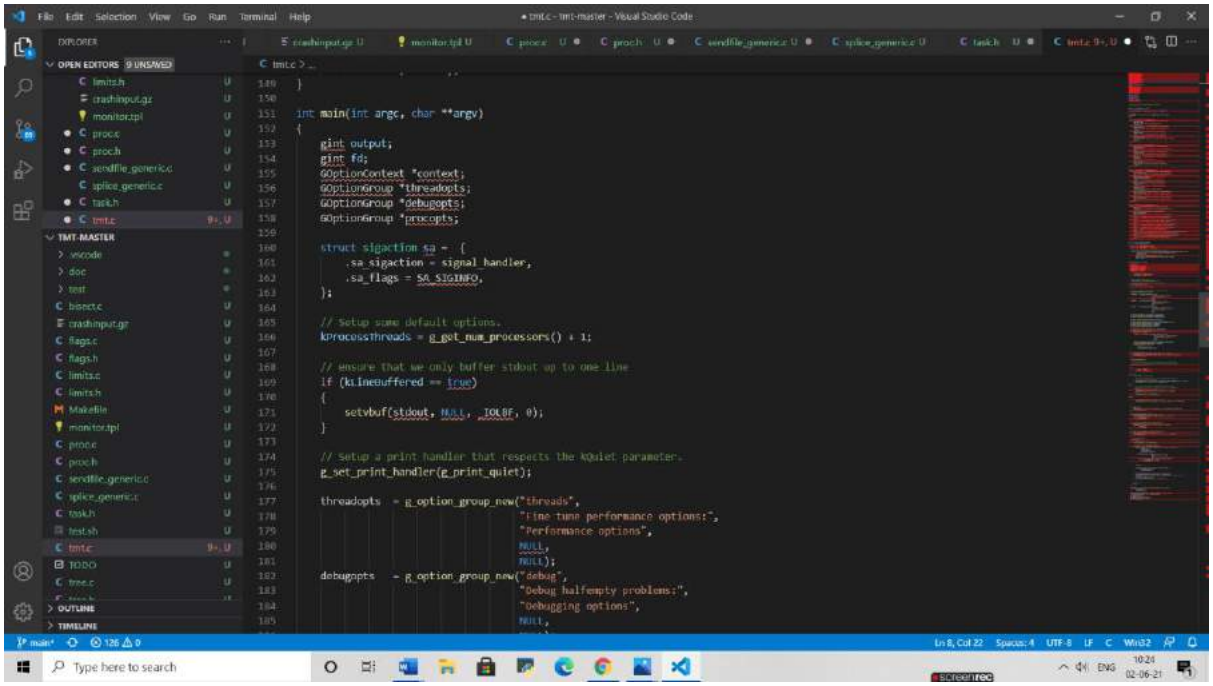
```
#include <glib.h>
#include <glib/gstdio.h>
#include <gio/gio.h>
#include <sys/time.h>
#include <sys/uio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/wait.h>
#ifdef __APPLE__
# include <sys/sysctl.h>
#endif
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

#include "task.h"
#include "proc.h"
#include "util.h"
#include "tree.h"
#include "limits.h"
#include "flags.h"

//
// Commandline options parsing and main() function.
//

#define TMT_VERSION_STRING "0.40"

void signal_handler(int sig, siginfo_t *info, void *ucontext)
{
#if __linux__
    g_debug("%s received from %d", strsignal(sig), info->si_pid);
#endif
}
```



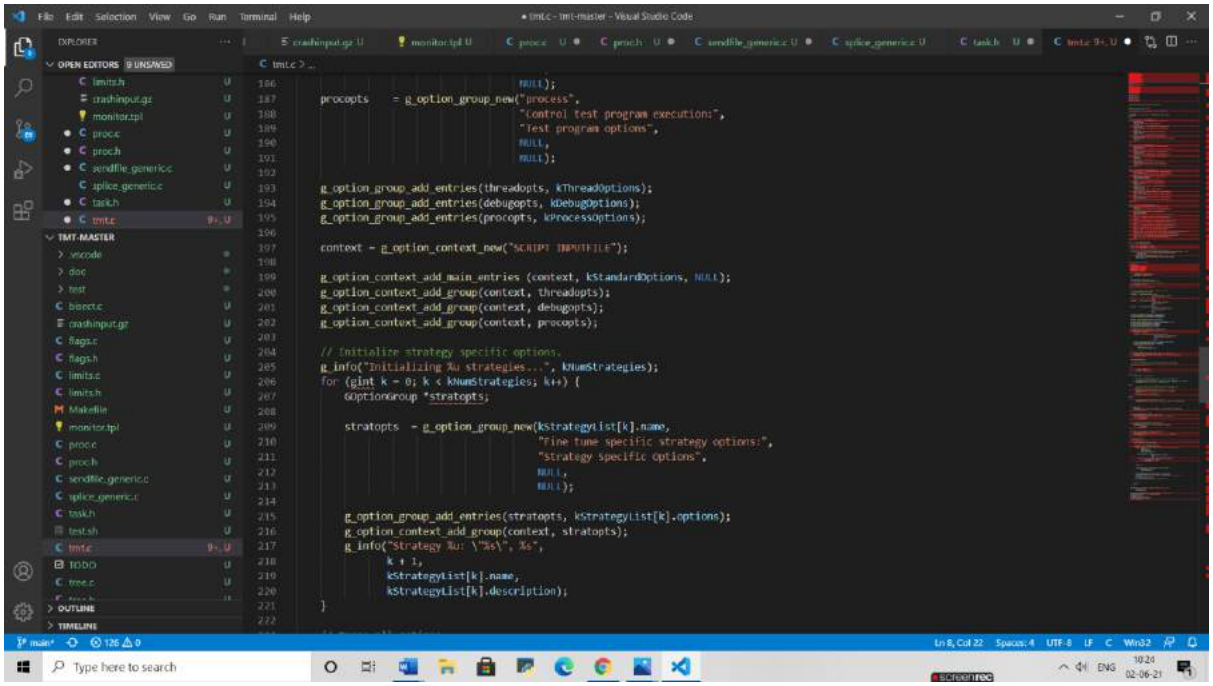
```
static const GOptionEntry kThreadOptions[] = {
    { "num-threads", 'p', G_OPTION_FLAG_NONE, G_OPTION_ARG_INT,
        &kProcessThreads,
        "How many threads to use (default=ncores+1).",
        "threads" },
    { "cleanup-threads", 0, G_OPTION_FLAG_NONE, G_OPTION_ARG_INT,
        &kCleanupThreads,
        "Number of threads used to garbage collect (default=4).",
        "threads" },
    { "max-queue", 0, G_OPTION_FLAG_NONE, G_OPTION_ARG_INT,
        &kMaxUnprocessed,
        "Maximum number of unprocessed workunits (default=4).",
        "n" },
    { "poll-delay", 0, G_OPTION_FLAG_NONE, G_OPTION_ARG_INT,
        &kWorkerPollDelay,
        "How long to sleep between checking queue status (default=10000).",
        "usecs" },
    { NULL },
};

static const GOptionEntry kDebugOptions[] = {
    { "generate-dot", 0, G_OPTION_FLAG_NONE, G_OPTION_ARG_NONE,
        &kGenerateDotFile,
        "Generate a DOT file to display the tree status (default=off).",
        NULL },
    { "collapse", 0, G_OPTION_FLAG_HIDDEN, G_OPTION_ARG_NONE,
        &kSimplifyDotFile,
        "Collapse branches in generated dot files (default=off).",
        NULL },
    { "verbosity", 'v', G_OPTION_FLAG_NONE, G_OPTION_ARG_INT,
        &kVerbosity,
        "Set verbosity level (default=0).",
        "level" },
    { "sleep", 0, G_OPTION_FLAG_HIDDEN, G_OPTION_ARG_INT,
        &kSleepSeconds,
        "Sleep this many seconds before execve() (default=0).",
        "seconds" },
```



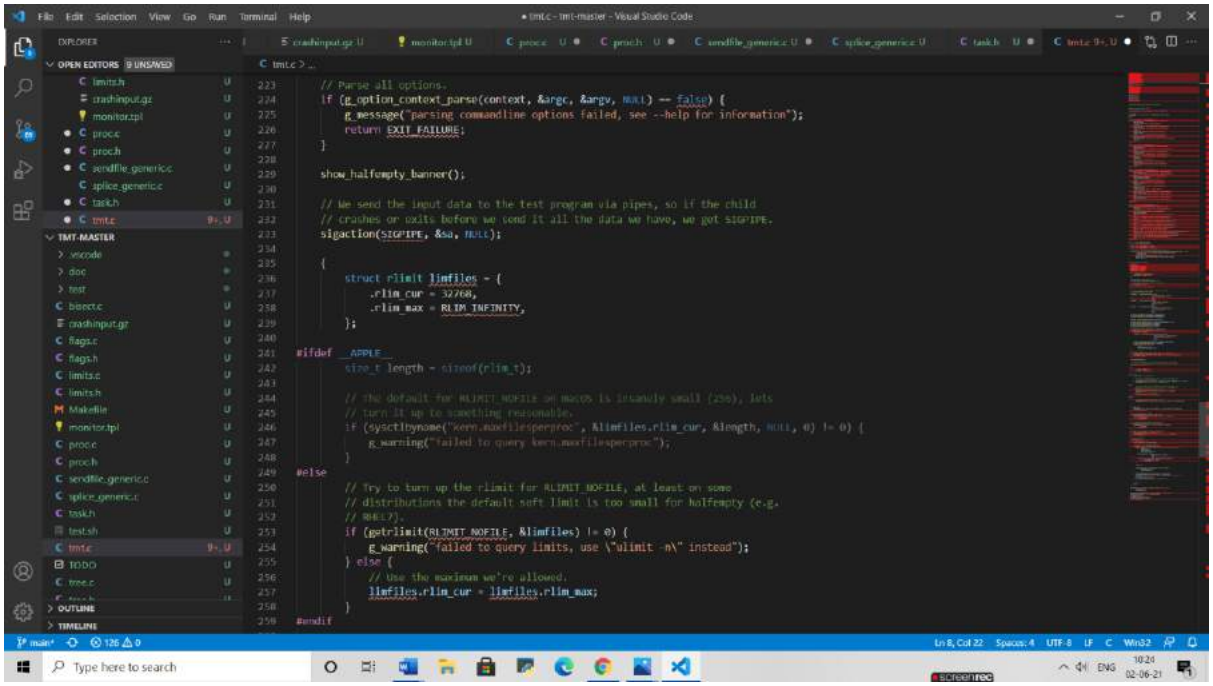




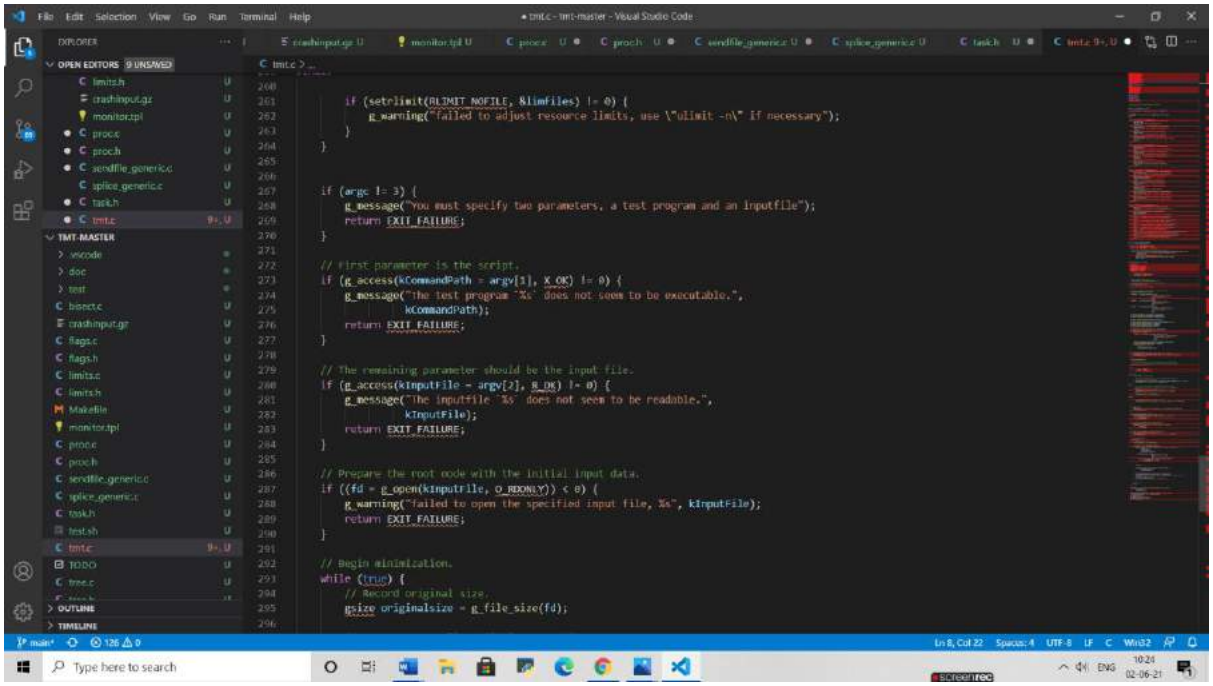
```
}

int main(int argc, char **argv)
{
    gint output;
    gint fd;
    GOptionContext *context;
    GOptionGroup *threadopts;
    GOptionGroup *debugopts;
    GOptionGroup *procopts;

    struct sigaction sa = {
        .sa_sigaction = signal_handler,
        .sa_flags = SA_SIGINFO,
    };

    // Setup some default options.
    kProcessThreads = g_get_num_processors() + 1;

    // ensure that we only buffer stdout up to one line
    if (kLineBuffered == true)
    {
        setvbuf(stdout, NULL, _IOLBF, 0);
    }

    // Setup a print handler that respects the kQuiet parameter.
    g_set_print_handler(g_print_quiet);

    threadopts  = g_option_group_new("threads",
                                     "Fine tune performance options:",
                                     "Performance options",
                                     NULL,
                                     NULL);
    debugopts   = g_option_group_new("debug",
                                     "Debug halfempty problems:",
                                     "Debugging options",
                                     NULL,
```



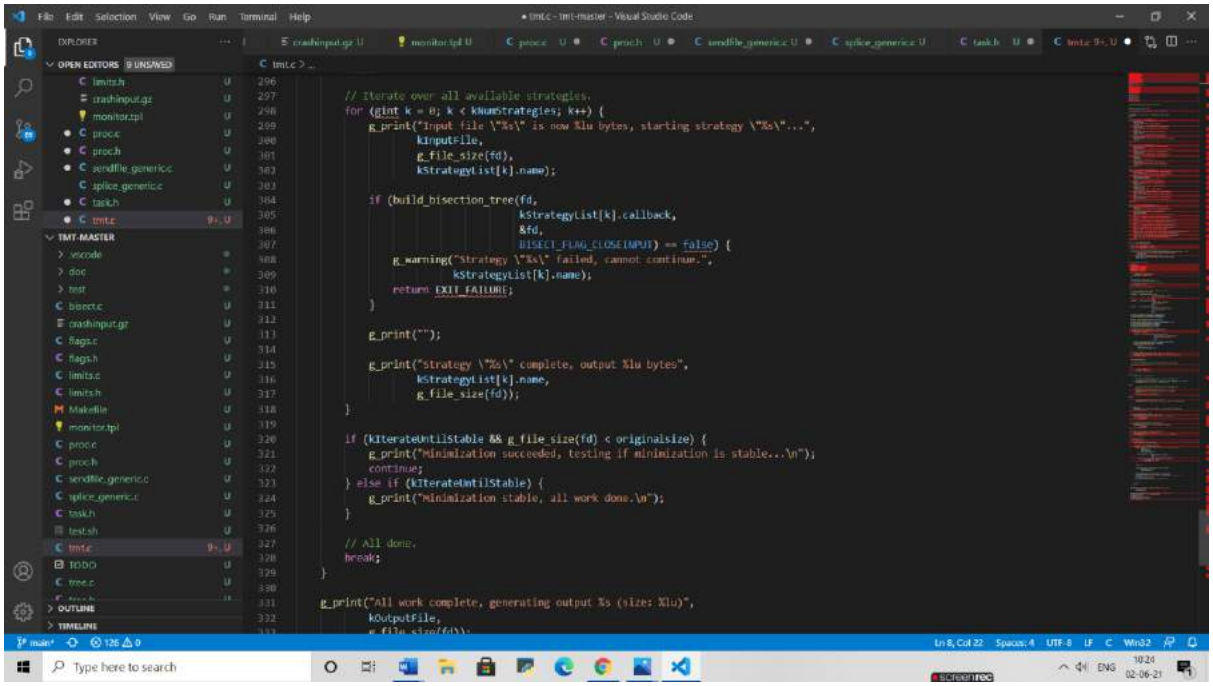
```
                                     NULL);
    procopts    = g_option_group_new("process",
                                     "Control test program execution:",
                                     "Test program options",
                                     NULL,
                                     NULL);

    g_option_group_add_entries(threadopts, kThreadOptions);
    g_option_group_add_entries(debugopts, kDebugOptions);
    g_option_group_add_entries(procopts, kProcessOptions);

    context = g_option_context_new("SCRIPT INPUTFILE");

    g_option_context_add_main_entries (context, kStandardOptions, NULL);
    g_option_context_add_group(context, threadopts);
    g_option_context_add_group(context, debugopts);
    g_option_context_add_group(context, procopts);

    // Initialize strategy specific options.
    g_info("Initializing %u strategies...", kNumStrategies);
    for (gint k = 0; k < kNumStrategies; k++) {
        GOptionGroup *stratopts;

        stratopts  = g_option_group_new(kStrategyList[k].name,
                                        "Fine tune specific strategy options:",
                                        "Strategy specific options",
                                        NULL,
                                        NULL);

        g_option_group_add_entries(stratopts, kStrategyList[k].options);
        g_option_context_add_group(context, stratopts);
        g_info("Strategy %u: \"%s\", %s",
               k + 1,
               kStrategyList[k].name,
               kStrategyList[k].description);
    }
```



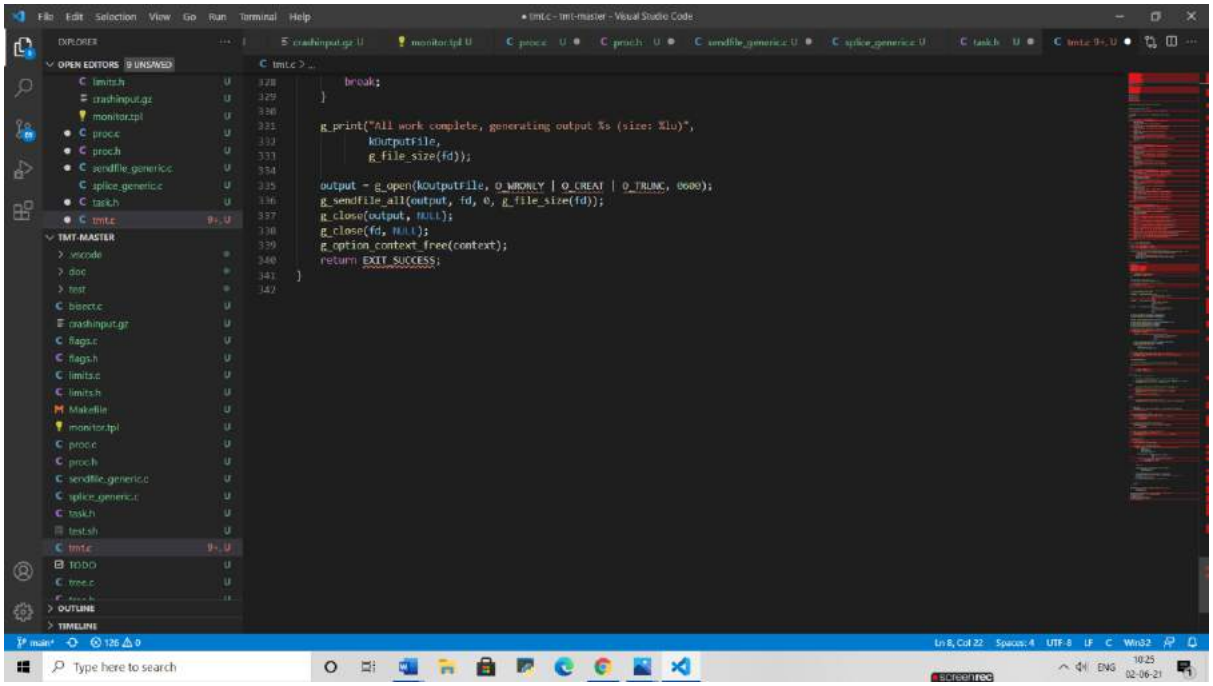
```
// Parse all options.
if (g_option_context_parse(context, &argc, &argv, NULL) == false) {
    g_message("parsing commandline options failed, see --help for information");
    return EXIT_FAILURE;
}

show_halfempty_banner();

// We send the input data to the test program via pipes, so if the child
// crashes or exits before we send it all the data we have, we get SIGPIPE.
sigaction(SIGPIPE, &sa, NULL);

{
    struct rlimit limfiles = {
        .rlim_cur = 32768,
        .rlim_max = RLIM_INFINITY,
    };

#ifdef __APPLE__
    size_t length = sizeof(rlim_t);

    // The default for RLIMIT_NOFILE on macOS is insanely small (256), lets
    // turn it up to something reasonable.
    if (sysctlbyname("kern.maxfilesperproc", &limfiles.rlim_cur, &length, NULL, 0) != 0) {
        g_warning("failed to query kern.maxfilesperproc");
    }
#else
    // Try to turn up the rlimit for RLIMIT_NOFILE, at least on some
    // distributions the default soft limit is too small for halfempty (e.g.
    // RHEL7).
    if (getrlimit(RLIMIT_NOFILE, &limfiles) != 0) {
        g_warning("failed to query limits, use \"ulimit -n\" instead");
    } else {
        // Use the maximum we're allowed.
        limfiles.rlim_cur = limfiles.rlim_max;
    }
#endif
```



```
    if (setrlimit(RLIMIT_NOFILE, &limfiles) != 0) {
        g_warning("failed to adjust resource limits, use \"ulimit -n\" if necessary");
    }
}


if (argc != 3) {
    g_message("You must specify two parameters, a test program and an inputfile");
    return EXIT_FAILURE;
}

// First parameter is the script.
if (g_access(kCommandPath = argv[1], X_OK) != 0) {
    g_message("The test program `%s` does not seem to be executable.",
              kCommandPath);
    return EXIT_FAILURE;
}

// The remaining parameter should be the input file.
if (g_access(kInputFile = argv[2], R_OK) != 0) {
    g_message("The inputfile `%s` does not seem to be readable.",
              kInputFile);
    return EXIT_FAILURE;
}

// Prepare the root node with the initial input data.
if ((fd = g_open(kInputFile, O_RDONLY)) < 0) {
    g_warning("failed to open the specified input file, %s", kInputFile);
    return EXIT_FAILURE;
}

// Begin minimization.
while (true) {
    // Record original size.
    gsize originalsize = g_file_size(fd);
```

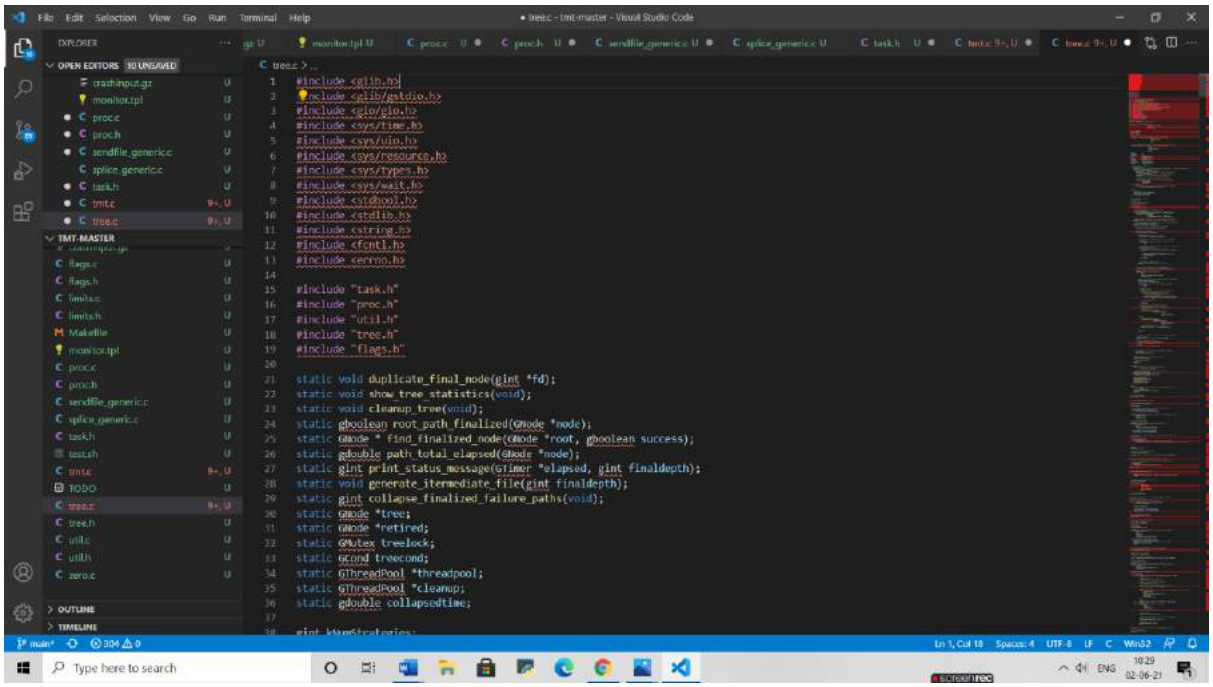


```
        // Iterate over all available strategies.
        for (gint k = 0; k < kNumStrategies; k++) {
            g_print("Input file \"%s\" is now %lu bytes, starting strategy \"%s\"...",
                    kInputFile,
                    g_file_size(fd),
                    kStrategyList[k].name);

            if (build_bisection_tree(fd,
                                     kStrategyList[k].callback,
                                     &fd,
                                     BISECT_FLAG_CLOSE(INPUT) == false) {
                g_warning("Strategy \"%s\" failed, cannot continue.",
                          kStrategyList[k].name);
                return EXIT_FAILURE;
            }

            g_print("");

            g_print("Strategy \"%s\" complete, output %lu bytes",
                    kStrategyList[k].name,
                    g_file_size(fd));
        }

        if (kIterateUntilStable && g_file_size(fd) < originalsize) {
            g_print("Minimization succeeded, testing if minimization is stable...\n");
            continue;
        } else if (kIterateUntilStable) {
            g_print("Minimization stable, all work done.\n");
        }

        // All done.
        break;
    }

    g_print("All work complete, generating output %s (size: %lu)",
            kOutputFile,
```

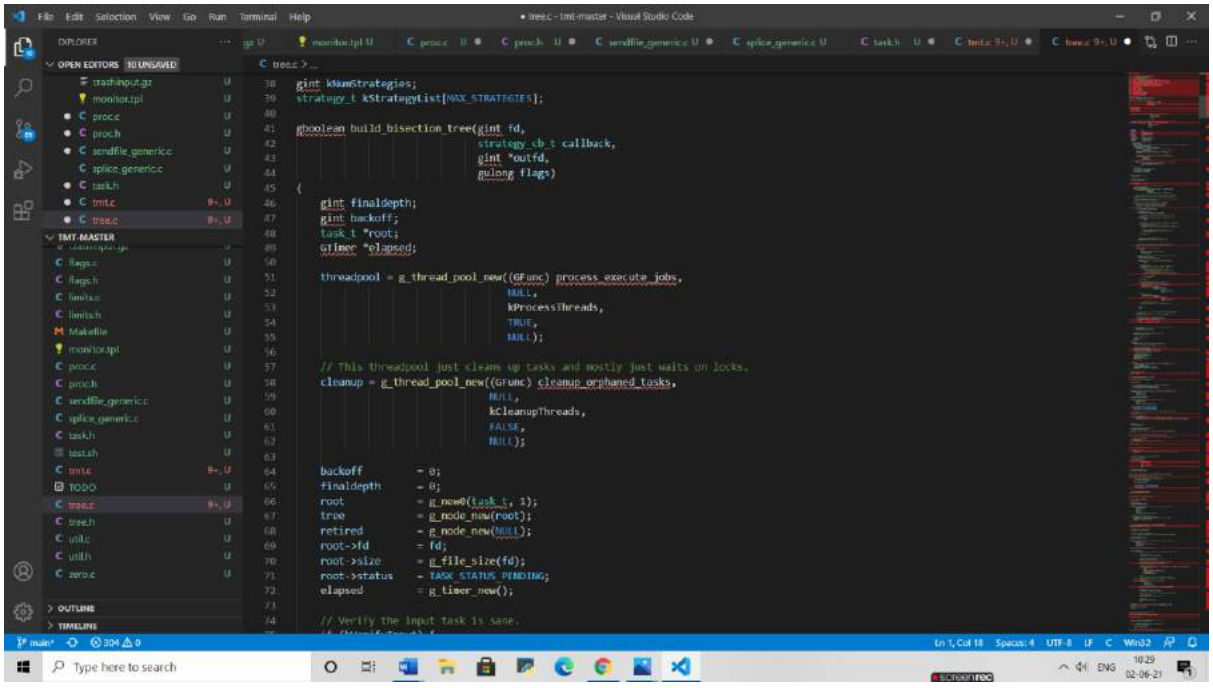


```
        break;
    }

    g_print("All work complete, generating output %s (size: %lu)",
            kOutputFile,
            g_file_size(fd));

    output = g_open(kOutputFile, O_WRONLY | O_CREAT | O_TRUNC, 0600);
    g_sendfile_all(output, fd, 0, g_file_size(fd));
    g_close(output, NULL);
    g_close(fd, NULL);
    g_option_context_free(context);
    return EXIT_SUCCESS;
}
```

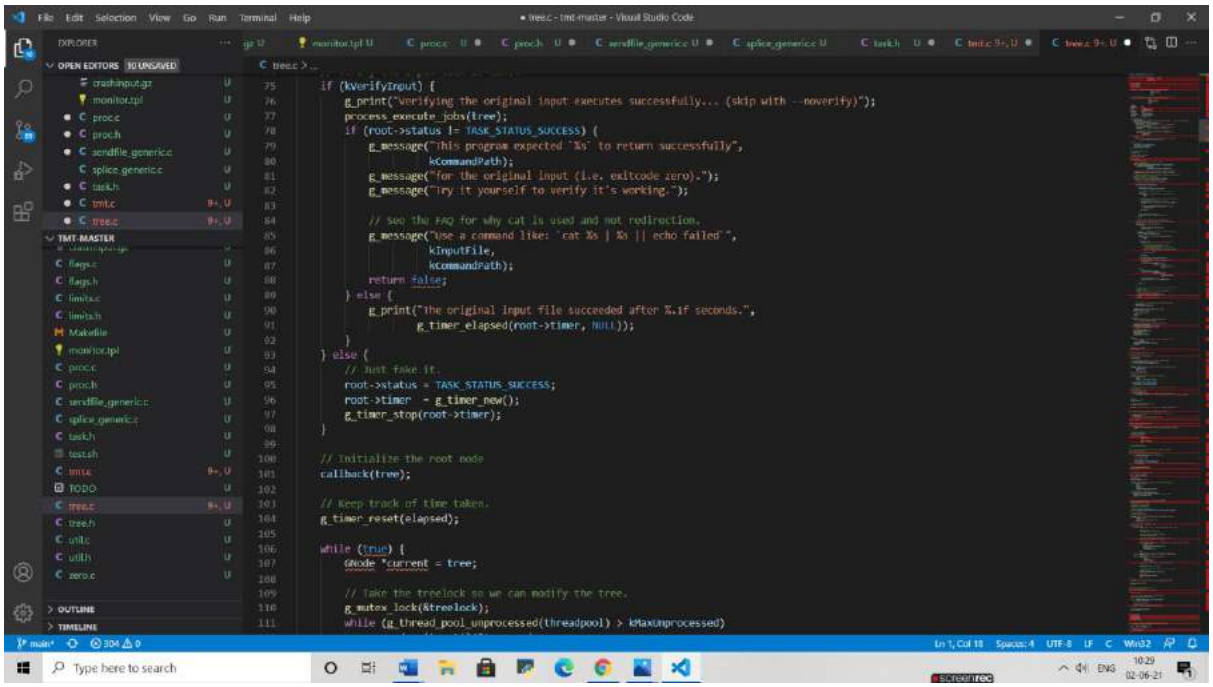
To generate the binary tree used to predict future steps-

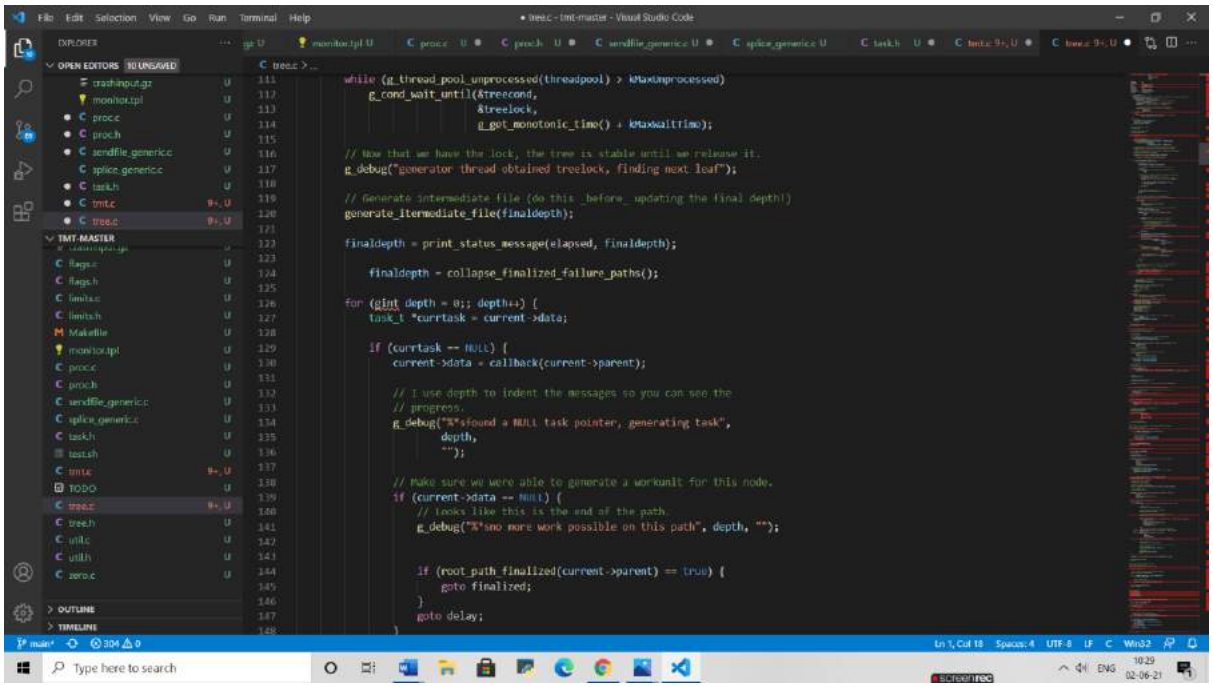


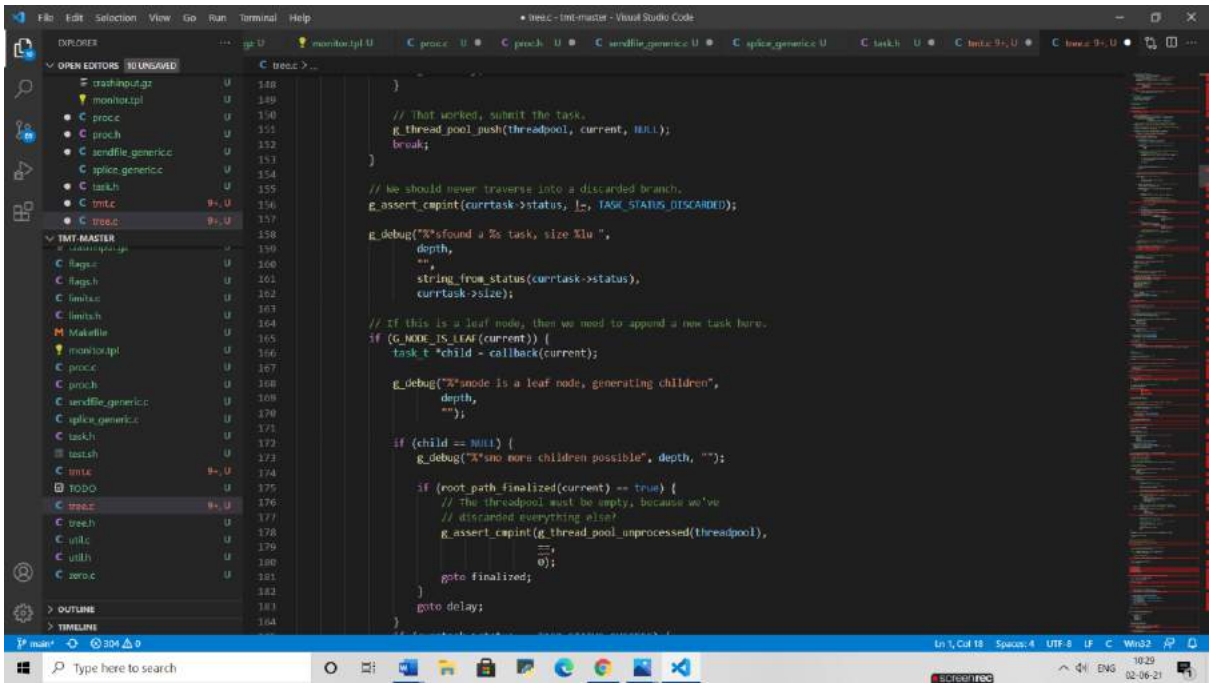
```
#include <glib.h>
#include <glib/gstdio.h>
#include <gio/gio.h>
#include <sys/time.h>
#include <sys/uio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>

#include "task.h"
#include "proc.h"
#include "util.h"
#include "tree.h"
#include "flags.h"

static void duplicate_final_node(gint *fd);
static void show_tree_statistics(void);
static void cleanup_tree(void);
static gboolean root_path_finalized(GNode *node);
static GNode * find_finalized_node(GNode *root, gboolean success);
static gdouble path_total_elapsed(GNode *node);
static gint print_status_message(GTimer *elapsed, gint finaldepth);
static void generate_itermediate_file(gint finaldepth);
static gint collapse_finalized_failure_paths(void);
static GNode *tree;
static GNode *retired;
static GMutex treelock;
static GCond treecond;
static GThreadPool *threadpool;
static GThreadPool *cleanup;
static gdouble collapsedtime;
```



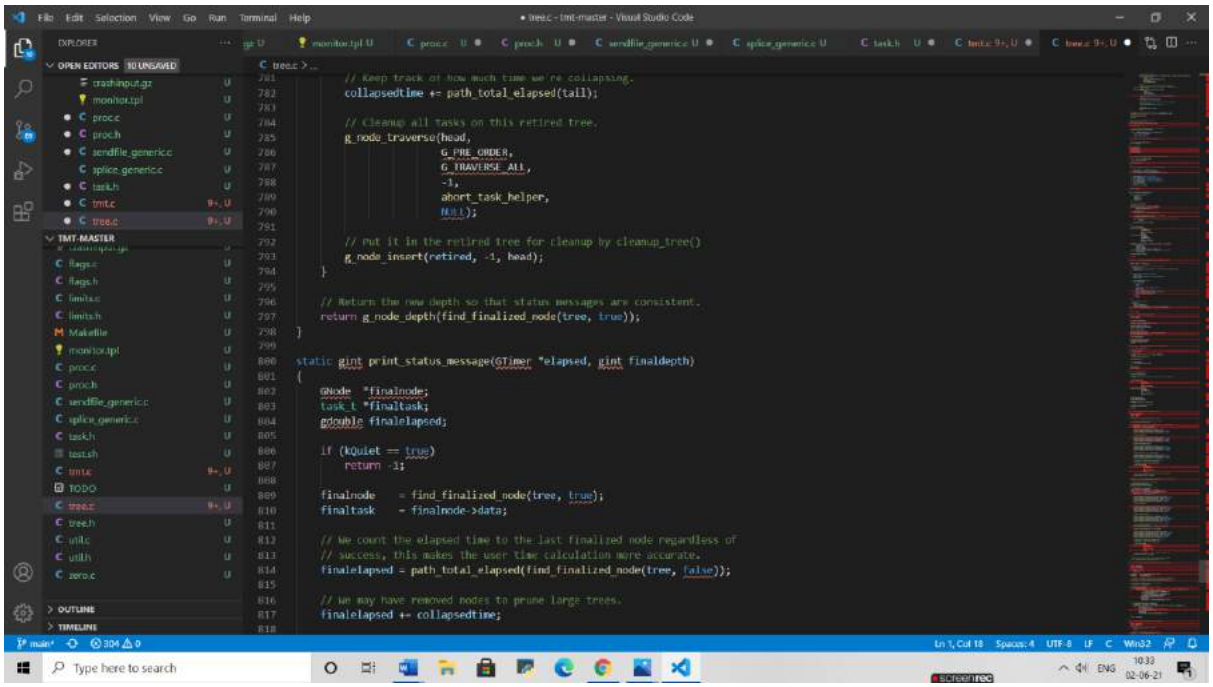
```
gint kNumStrategies;
strategy_t kStrategyList[MAX_STRATEGIES];

gboolean build_bisection_tree(gint fd,
                              strategy_cb_t callback,
                              gint *outfd,
                              gulong flags)
{
    gint finaldepth;
    gint backoff;
    task_t *root;
    GTimer *elapsed;

    threadpool = g_thread_pool_new((GFunc) process_execute_jobs,
                                   NULL,
                                   kProcessThreads,
                                   TRUE,
                                   NULL);

    // This threadpool just cleans up tasks and mostly just waits on locks.
    cleanup = g_thread_pool_new((GFunc) cleanup_orphaned_tasks,
                                NULL,
                                kCleanupThreads,
                                FALSE,
                                NULL);

    backoff      = 0;
    finaldepth   = 0;
    root         = g_new0(task_t, 1);
    tree         = g_node_new(root);
    retired      = g_node_new(NULL);
    root->fd     = fd;
    root->size   = g_file_size(fd);
    root->status = TASK_STATUS_PENDING;
    elapsed      = g_timer_new();

    // Verify the input task is sane.
```


File Edit Selection View Go Run Terminal Help
• tree.c - tmt-master - Visual Studio Code
EXPLORER
OPEN EDITORS 10 UNSAVED
TMT-MASTER
OUTLINE
TIMELINE
if (kVerifyInput) {
g_print("Verifying the original input executes successfully... (skip with --noverify)");
process_execute_jobs(tree);
if (root->status != TASK_STATUS_SUCCESS) {
g_message("this program expected '%s' to return successfully",
kCommandPath);
g_message("for the original input (i.e. exitcode zero).");
g_message("try it yourself to verify it's working.");
// see the FAQ for why cat is used and not redirection.
g_message("Use a command like: 'cat %s | %s || echo failed'",
kInputFile,
kCommandPath);
return false;
} else {
g_print("the original input file succeeded after %.1f seconds.",
g_timer_elapsed(root->timer, NULL));
}
} else {
// Just fake it.
root->status = TASK_STATUS_SUCCESS;
root->timer = g_timer_new();
g_timer_stop(root->timer);
}
// Initialize the root node
callback(tree);
// Keep track of time taken.
g_timer_reset(elapsed);
while (true) {
GNode *current = tree;
// Take the treelock so we can modify the tree.
g_mutex_lock(&treelock);
while (g_thread_pool_unprocessed(threadpool) > kMaxUnprocessed)
Ln 1, Col 18 Spaces: 4 UTF-8 LF C Win32
Type here to search
ENG
10:29
02-06-21

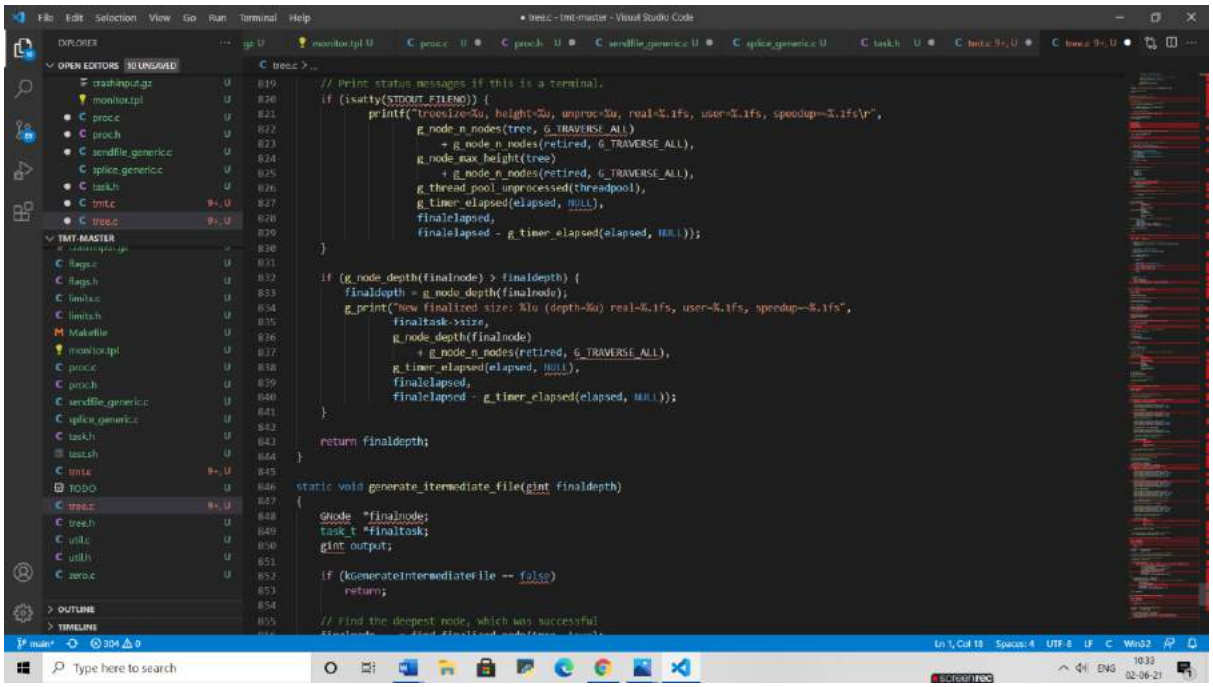

File Edit Selection View Go Run Terminal Help
• tree.c - tmt-master - Visual Studio Code
EXPLORER
OPEN EDITORS 10 UNSAVED
TMT-MASTER
OUTLINE
TIMELINE
while (g_thread_pool_unprocessed(threadpool) > kMaxUnprocessed)
g_cond_wait_until(&treecond,
&treelock,
g_get_monotonic_time() + kMaxWaitTime);
// Now that we have the lock, the tree is stable until we release it.
g_debug("generator thread obtained treelock, finding next leaf");
// Generate intermediate file (do this _before_ updating the final depth!)
generate_intermediate_file(finaldepth);
finaldepth = print_status_message(elapsed, finaldepth);
finaldepth = collapse_finalized_failure_paths();
for (gint depth = 0;; depth++) {
task_t *currtask = current->data;
if (currtask == NULL) {
current->data = callback(current->parent);
// I use depth to indent the messages so you can see the
// progress.
g_debug("%*sfound a NULL task pointer, generating task",
depth,
"");
// Make sure we were able to generate a workunit for this node.
if (current->data == NULL) {
// looks like this is the end of the path.
g_debug("%*sno more work possible on this path", depth, "");
if (root_path_finalized(current->parent) == true) {
goto finalized;
}
goto delay;
}
Ln 1, Col 18 Spaces: 4 UTF-8 LF C Win32
Type here to search
ENG
10:29
02-06-21



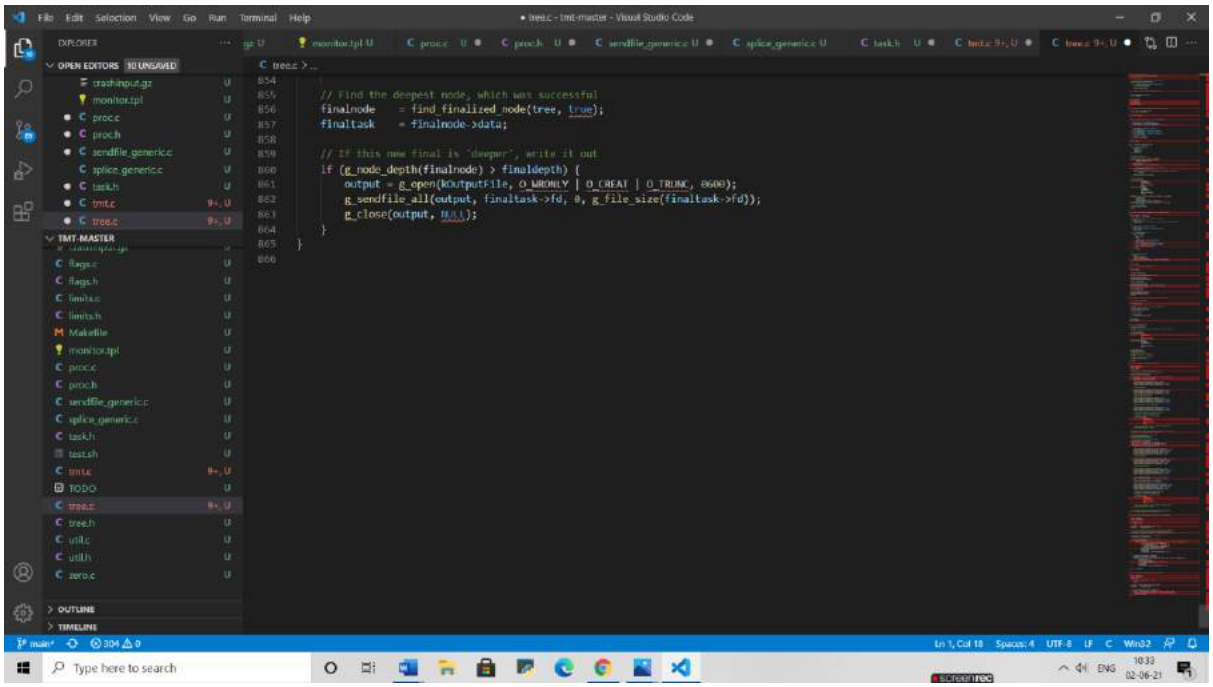
```
                }

                // That worked, submit the task.
                g_thread_pool_push(threadpool, current, NULL);
                break;
            }

            // We should never traverse into a discarded branch.
            g_assert_cmpint(currtask->status, !=, TASK_STATUS_DISCARDED);

            g_debug("%*sfound a %s task, size %lu ",
                    depth,
                    "",
                    string_from_status(currtask->status),
                    currtask->size);

            // If this is a leaf node, then we need to append a new task here.
            if (G_NODE_IS_LEAF(current)) {
                task_t *child = callback(current);

                g_debug("%*snode is a leaf node, generating children",
                        depth,
                        "");

                if (child == NULL) {
                    g_debug("%*sno more children possible", depth, "");

                    if (root_path_finalized(current) == true) {
                        // The threadpool must be empty, because we've
                        // discarded everything else?
                        g_assert_cmpint(g_thread_pool_unprocessed(threadpool),
                                        ==,
                                        0);
                        goto finalized;
                    }
                    goto delay;
                }
```



```
        // Keep track of how much time we're collapsing.
        collapsedtime += path_total_elapsed(tail);

        // Cleanup all tasks on this retired tree.
        g_node_traverse(head,
                        G_PRE_ORDER,
                        G_TRAVERSE_ALL,
                        -1,
                        abort_task_helper,
                        NULL);

        // Put it in the retired tree for cleanup by cleanup_tree()
        g_node_insert(retired, -1, head);
    }

    // Return the new depth so that status messages are consistent.
    return g_node_depth(find_finalized_node(tree, true));
}

static gint print_status_message(GTimer *elapsed, gint finaldepth)
{
    GNode  *finalnode;
    task_t *finaltask;
    gdouble finalelapsed;

    if (kQuiet == true)
        return -1;

    finalnode    = find_finalized_node(tree, true);
    finaltask    = finalnode->data;

    // We count the elapsed time to the last finalized node regardless of
    // success, this makes the user time calculation more accurate.
    finalelapsed = path_total_elapsed(find_finalized_node(tree, false));

    // We may have removed nodes to prune large trees.
    finalelapsed += collapsedtime;
```



```
    // Print status messages if this is a terminal.
    if (isatty(STDOUT_FILENO)) {
            printf("treesize=%u, height=%u, unproc=%u, real=%.1fs, user=%.1fs, speedup=%.1fs\r",
                    g_node_n_nodes(tree, G_TRAVERSE_ALL)
                        + g_node_n_nodes(retired, G_TRAVERSE_ALL),
                    g_node_max_height(tree)
                        + g_node_n_nodes(retired, G_TRAVERSE_ALL),
                    g_thread_pool_unprocessed(threadpool),
                    g_timer_elapsed(elapsed, NULL),
                    finalelapsed,
                    finalelapsed - g_timer_elapsed(elapsed, NULL));
    }

    if (g_node_depth(finalnode) > finaldepth) {
        finaldepth = g_node_depth(finalnode);
        g_print("New finalized size: %lu (depth=%u) real=%.1fs, user=%.1fs, speedup=%.1fs",
                finaltask->size,
                g_node_depth(finalnode)
                    + g_node_n_nodes(retired, G_TRAVERSE_ALL),
                g_timer_elapsed(elapsed, NULL),
                finalelapsed,
                finalelapsed - g_timer_elapsed(elapsed, NULL));
    }

    return finaldepth;
}

static void generate_itermediate_file(gint finaldepth)
{
    GNode  *finalnode;
    task_t *finaltask;
    gint output;

    if (kGenerateIntermediateFile == false)
        return;

    // Find the deepest node, which was successful
```



```
    // Find the deepest node, which was successful
    finalnode    = find_finalized_node(tree, true);
    finaltask    = finalnode->data;

    // If this new final is "deeper", write it out
    if (g_node_depth(finalnode) > finaldepth) {
        output = g_open(kOutputFile, O_WRONLY | O_CREAT | O_TRUNC, 0600);
        g_sendfile_all(output, finaltask->fd, 0, g_file_size(finaltask->fd));
        g_close(output, NULL);
    }
}
```

4.3 CONSTRAINTS AND TRADEOFFS:

4.3.1 CONSTRAINTS-

Table for Constraints –

Constraints	Description
Programming Language	The software should be such that it can be used with input of different programming languages. The language used in this project is C and shell script
OS Support	This project cannot be considered as a very platform friendly software due to its preliminary Mac/Linux OS support.
Interruptions and errors	If input program intercepts signals or creates process groups, it might be difficult to cleanup.

4.3.2 ALTERNATIVES-

As a whole tool, there might not be a proper alternative, but in future releases the scope of different programming language that can be used as an input file is vast.

If the cost for making or improving this project can be increased then it would allow implementing more features and would be a good alternative.

Moreover, investing more time on this tool would definitely help in improving the overall look and adding new features.

4.3.3 TRADEOFFS-

While designing this tool including an option for generating testcases was also discussed, but the load and cost of implementing it was not feasible. So keeping in mind the main goal and available resources, only testcase minimization tool was designed and implemented.

This tool will help software developers and crash maintainers to minimize their testcases and save time.

The developers/maintainers can't put in the required time to test all the testcases and also analyze the results, but can generate the required testcases beforehand.

Hence, their efficiency is increased.

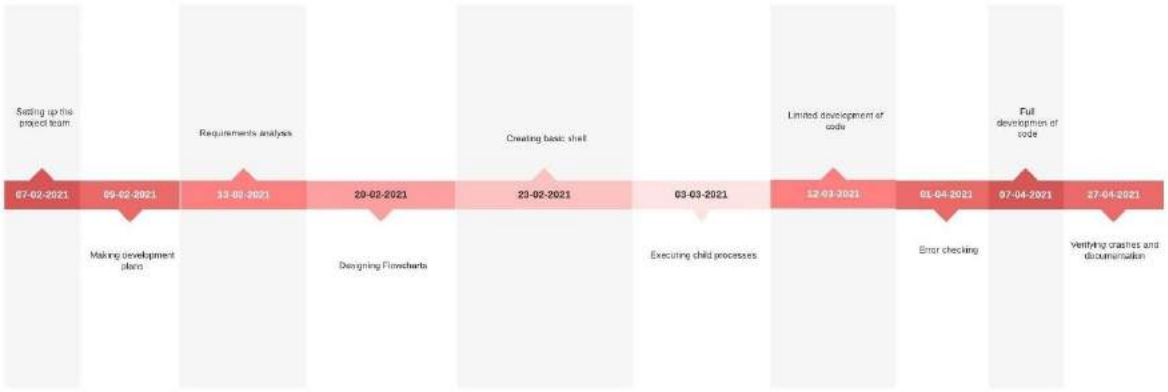
5. SCHEDULES, TASKS AND MILESTONES

5.1 GANTT CHART:

Table:

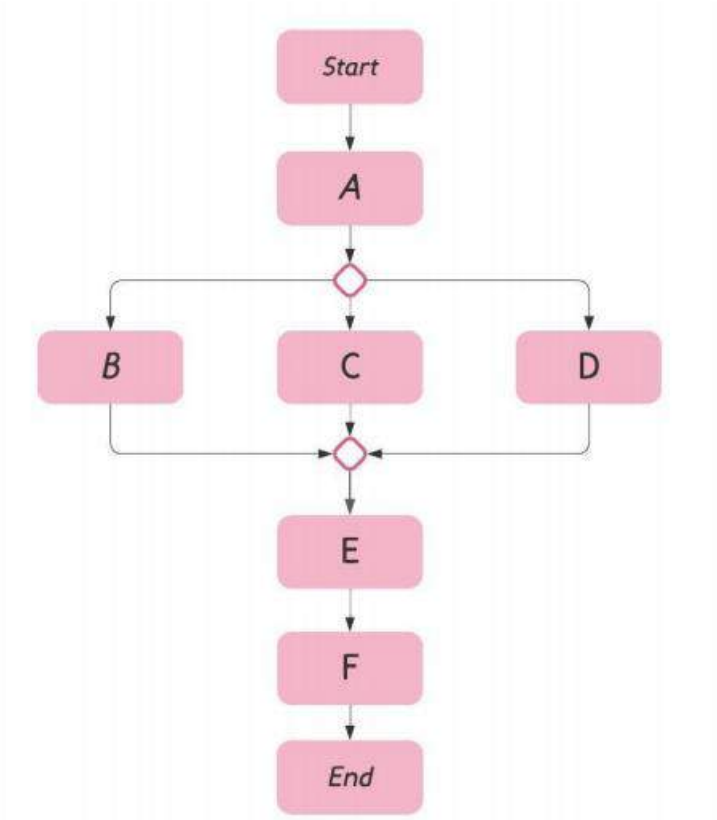
Title	T/M	Start	End		%
Setting up project team	T	07/02/2021	08/02/2021	1 day	%
Making development plans	T	09/02/2021	11/02/2021	3 days	%
Requirement analysis	T	13/02/2021	19/02/2021	5 days	%
Designing Flowcharts	T	20/02/2021	23/02/2021	2 days	%
Creating basic shell	T	23/02/2021	03/03/2021	7 days	%
Executing Child processes	T	03/03/2021	11/03/2021	7 days	%
Limited development of code	T	12/03/2021	01/04/2021	15 days	%
Error checking	T	01/04/2021	06/04/2021	4 days	%
Full development of code	T	07/04/2021	26/04/2021	14 days	%
Verifying Crashes and documentation	T	27/04/2021	03/05/2021	5 days	%

5.2 TIMELINE NETWORK:



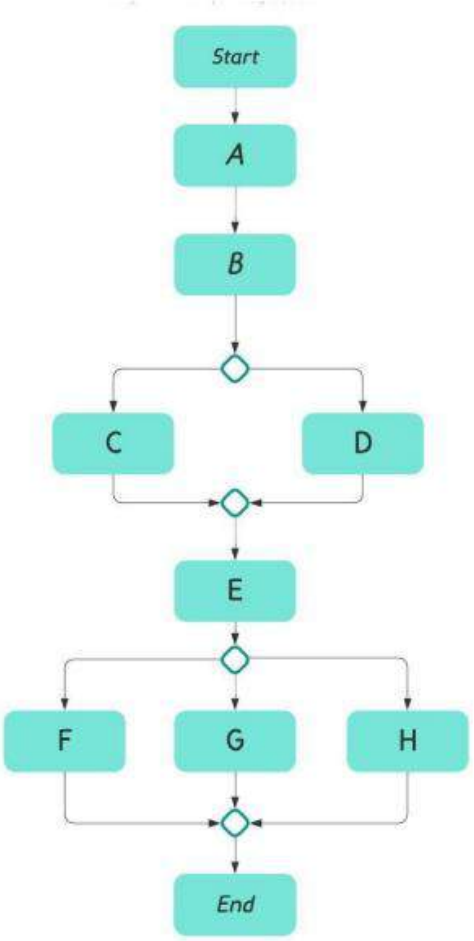
5.3 ACTIVITY NETWORK DIAGRAM:

PROCESS BASED:



TASK	LABEL	PREDECESSOR
Requirement analysis	A	
Design	B	A
Implementation	C	A,B
Critical review	D	C
Verification and testing	E	B,C,D
Deployment	F	E

PRODUCT BASED:

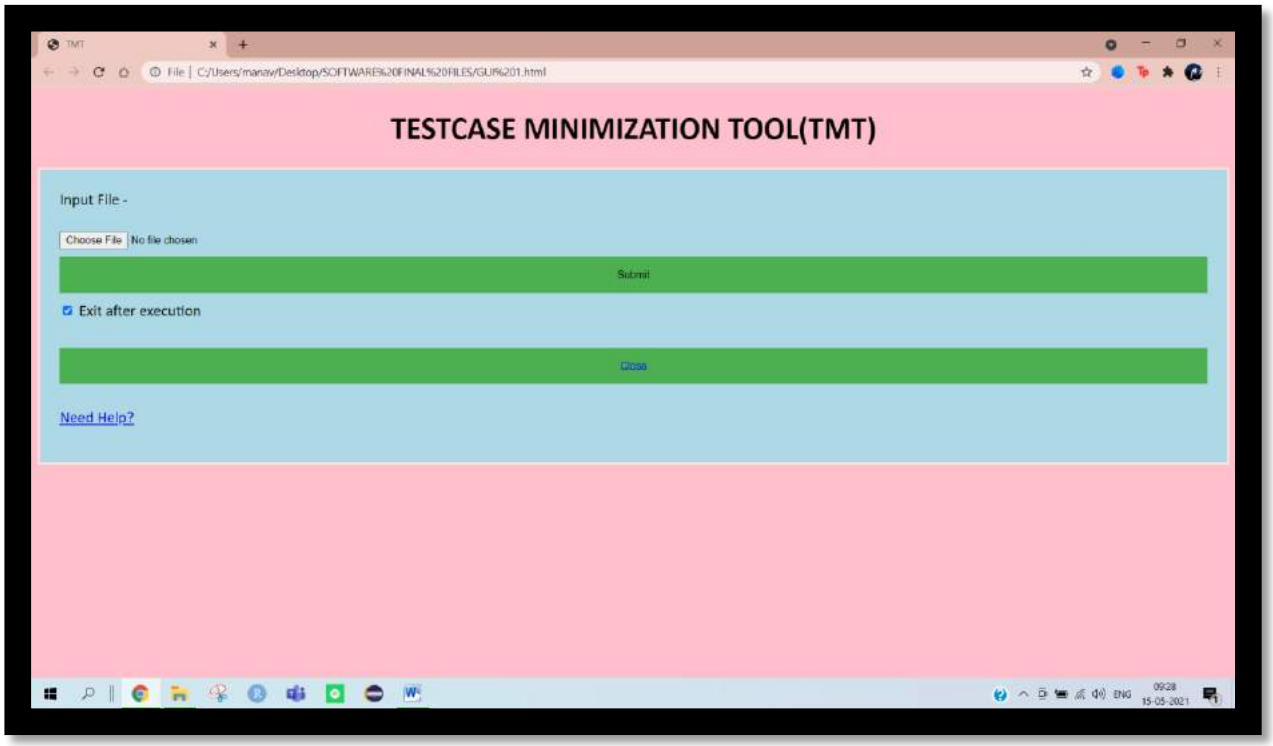


TASK	LABEL	PREDECESSOR
Creating a basic shell	A	
Execution	B	A
Give inputs and run the TMT	C	B
TMT finds smallest version that returns 0	D	B,C
O/p file in tmt.out	E	D
Monitor TMT using --monitor mode	F	E
Realtime graphs	G	E,F
TMT generates a URL, open it and view the data	H	E,F,G

6. PROJECT DEMONSTRATION:

Project demonstration will include the implementation of our tool.

GUI MAIN PAGE:



DESCRIPTION:

It is the first page which is shown when the test case minimization tool is opened.

Consists of all the buttons and links.

1)"choose file" button which when clicked takes to the computers directories to select input file for the tool.

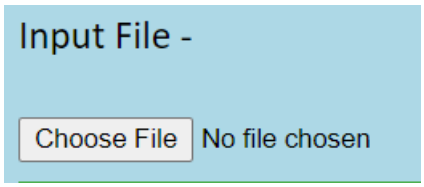
2)"submit" button that when clicked starts to minimize the input file by initiating the tool.

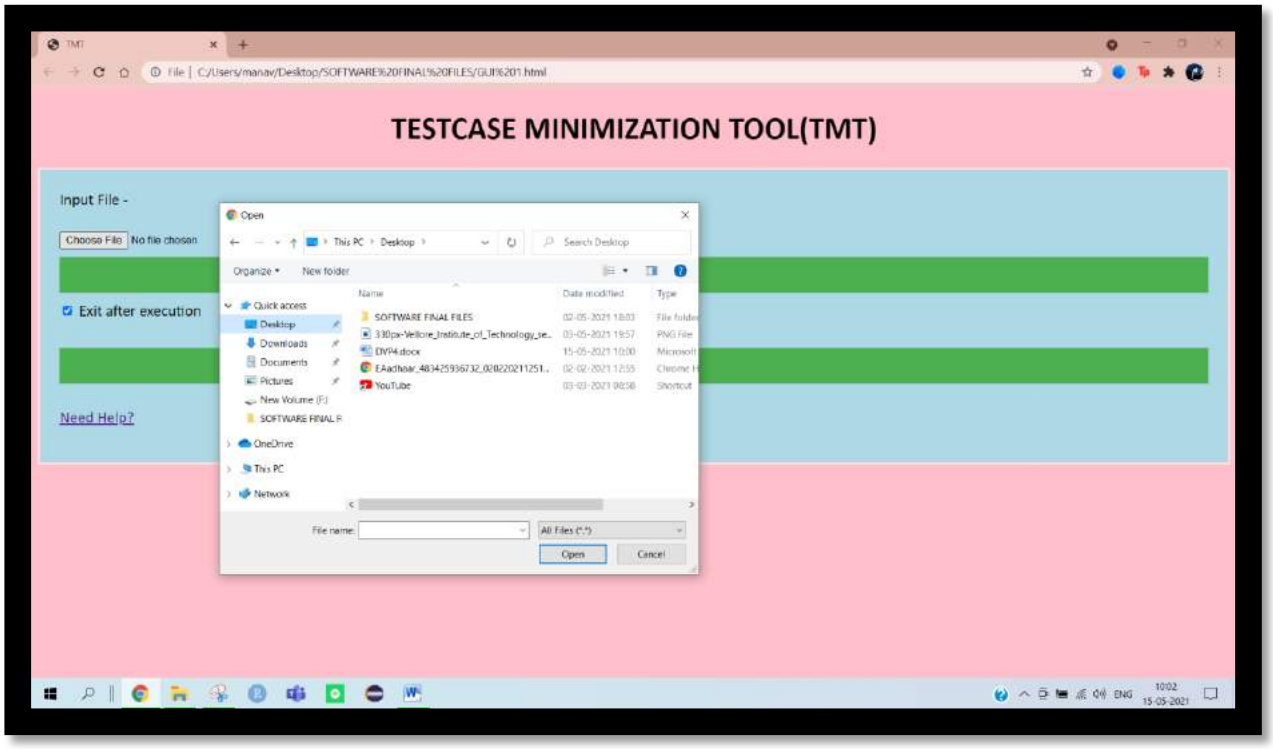
3)An optional button called "exit after execution" to tick/untick.

4)"close" button to exit the tool.

5)"need help" button to know about the tool.

GUI INPUT PAGE:



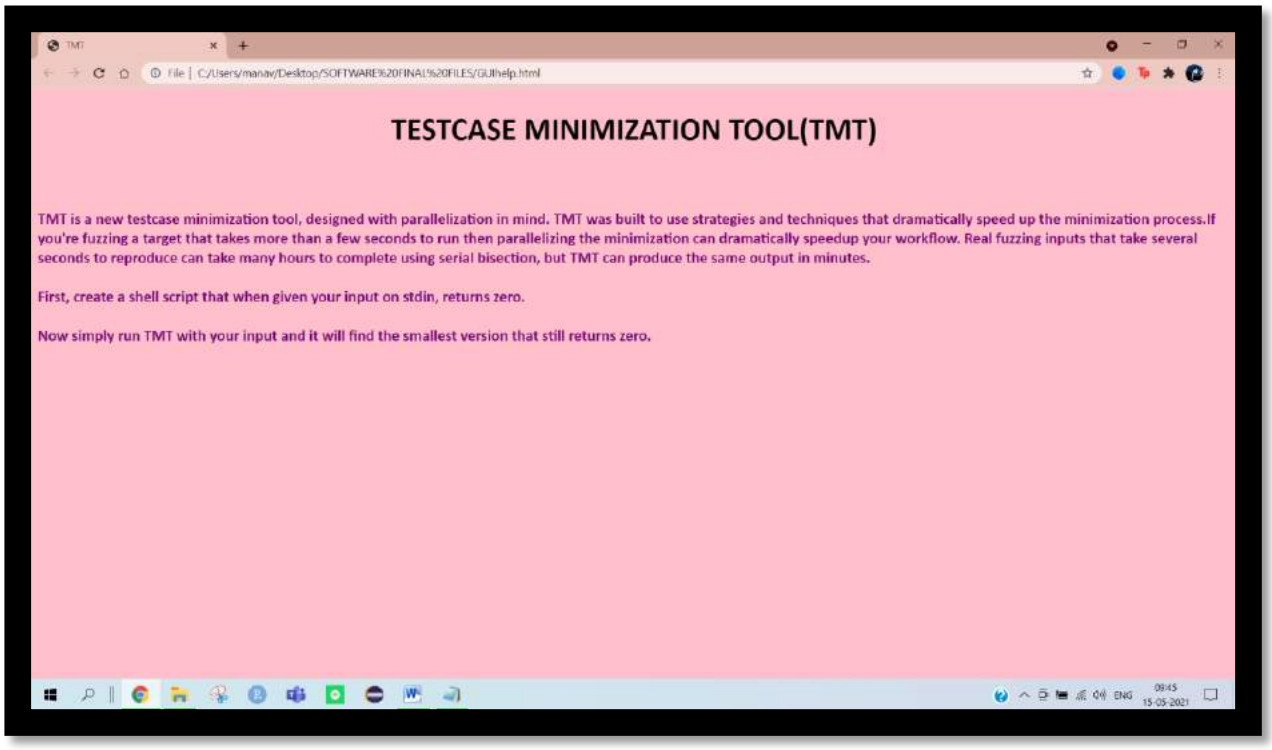


DESCRIPTION:

1) Takes the input file from the user as per requirement.

2) The input file is restricted to .gz format.

GUI HELP PAGE:

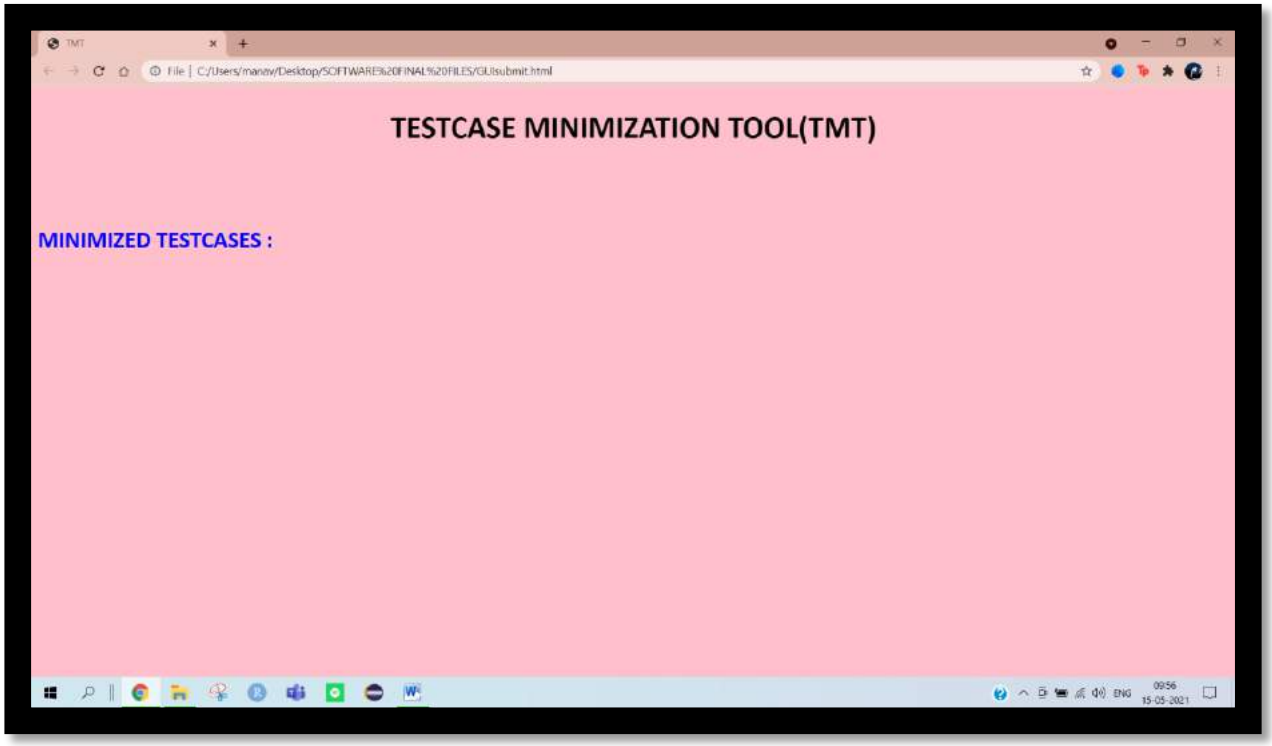


DESCRIPTION:

1)This page is opened when the “need help?” Button is clicked on the main page.

2)Explains the tool and guides the user.

GUI SUBMIT PAGE:

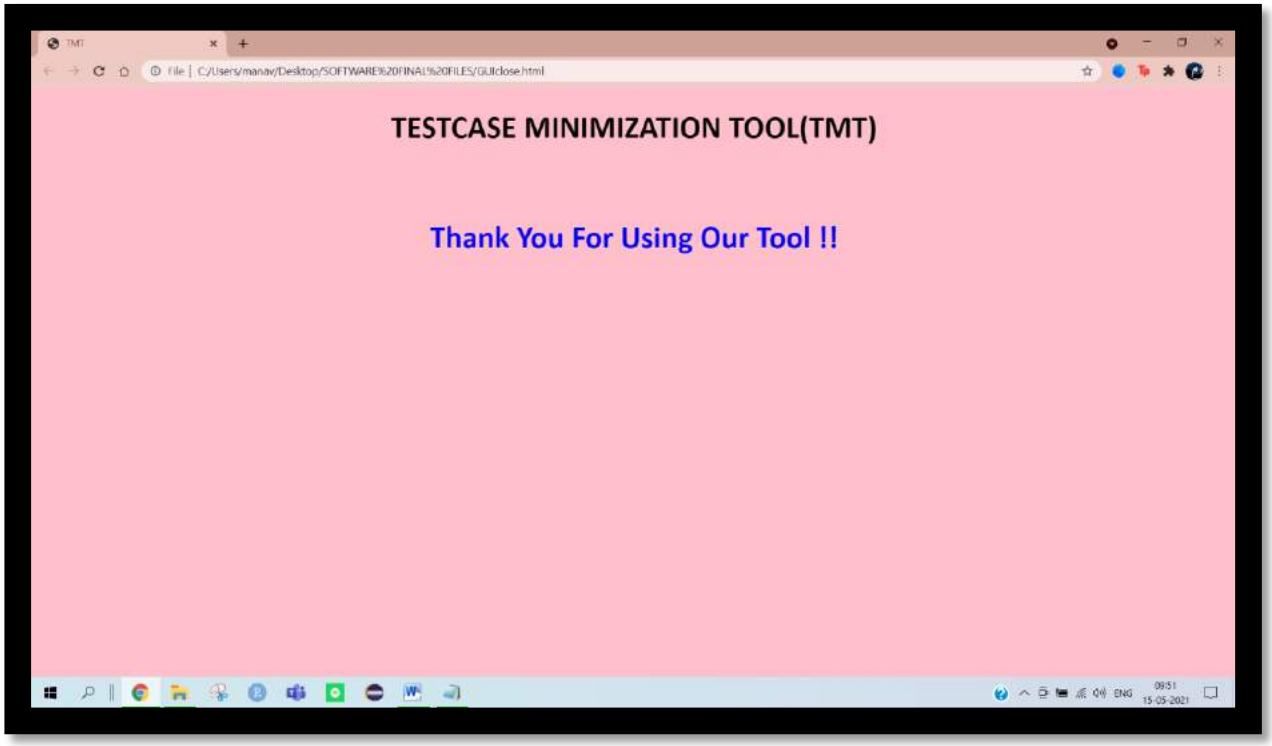


DESCRIPTION:

1)This page is opened when the user clicks the “submit” button in order to run the tool and obtain the minimized test cases.

2)Shows the minimized test cases that are obtained after running through the tool.

GUI CLOSE PAGE:



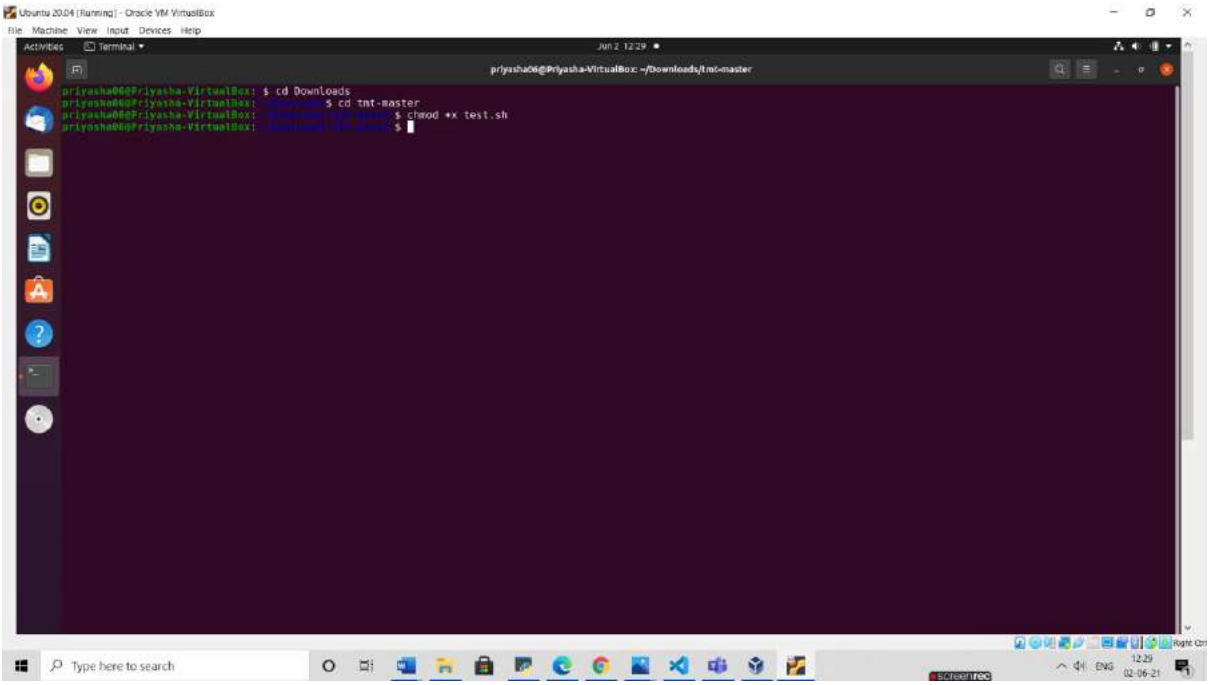
DESCRIPTION:

1)This page is opened when the user clicks the “close” button in order to exit the tool.

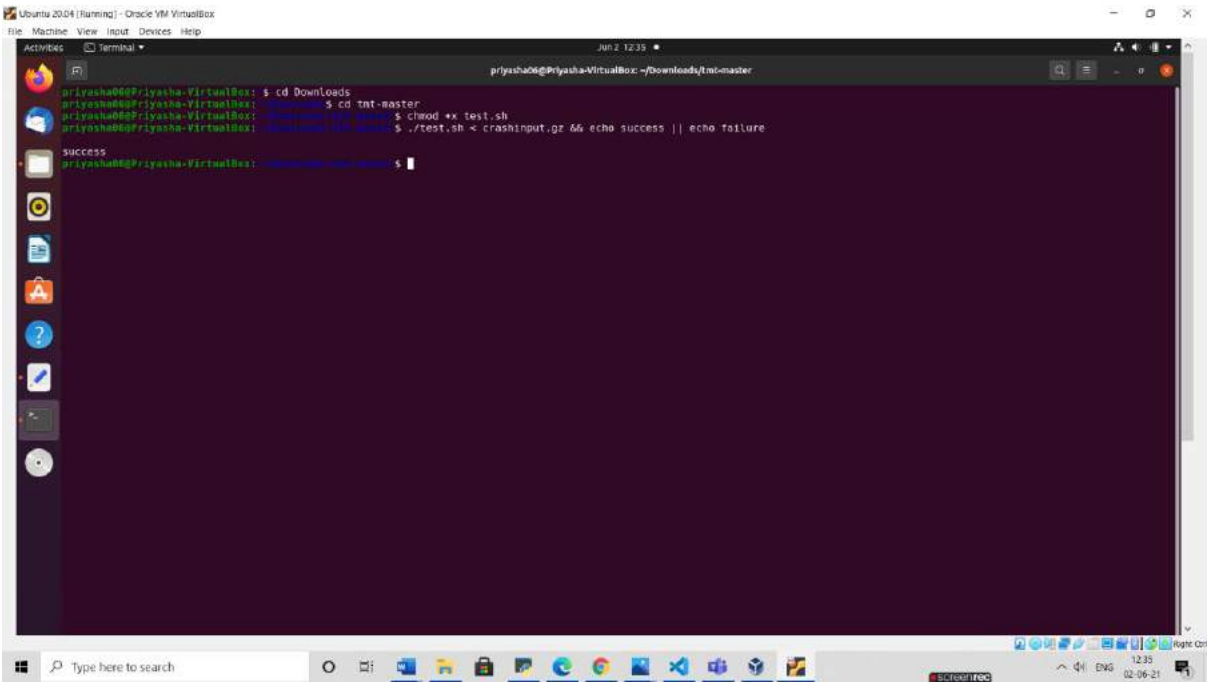
2)Displays a thank you message for the user.

CODE EXECUTION:

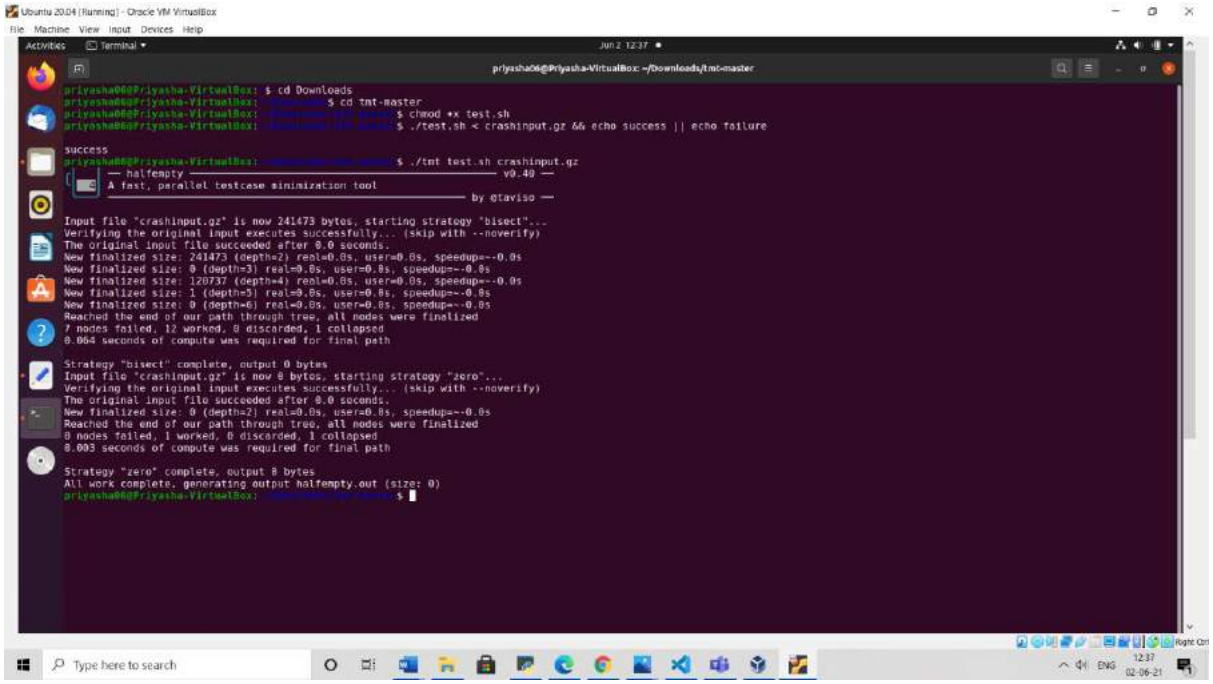
Create a shell script using the 'nano test.sh' command in the terminal(it opens the text editor) that when given your input on stdin, returns zero. To execute the test.sh file.



To verify the input file and see it its executable by the tool:



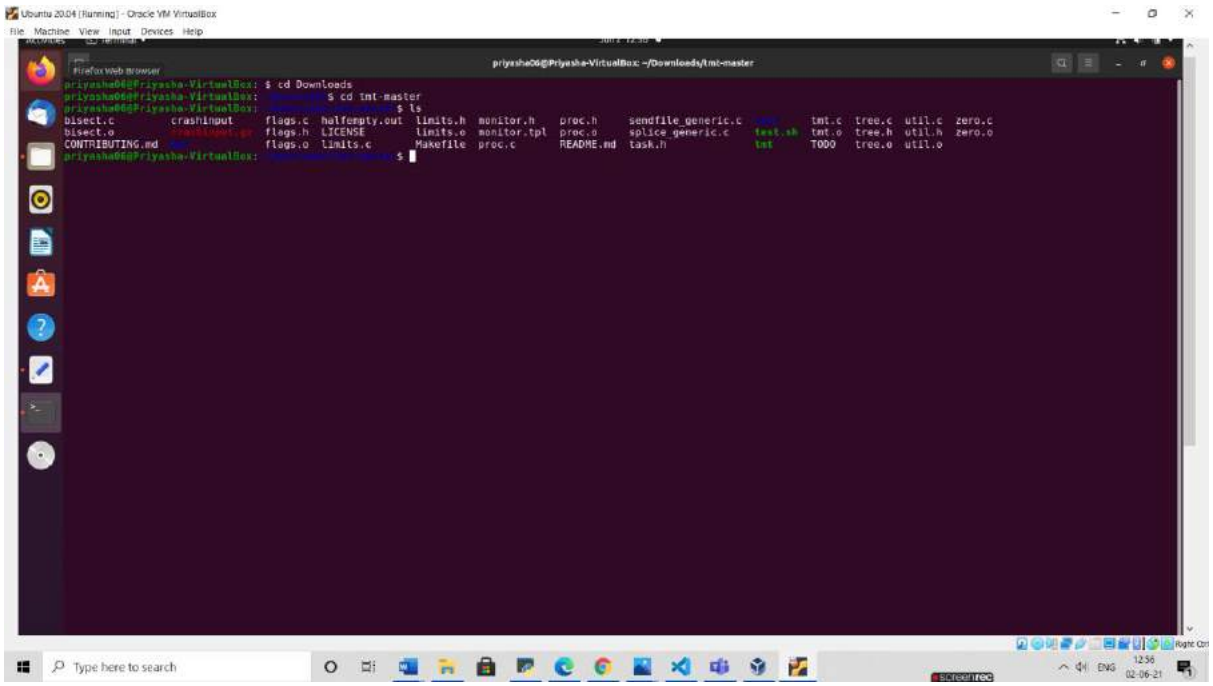
To get the desired output:



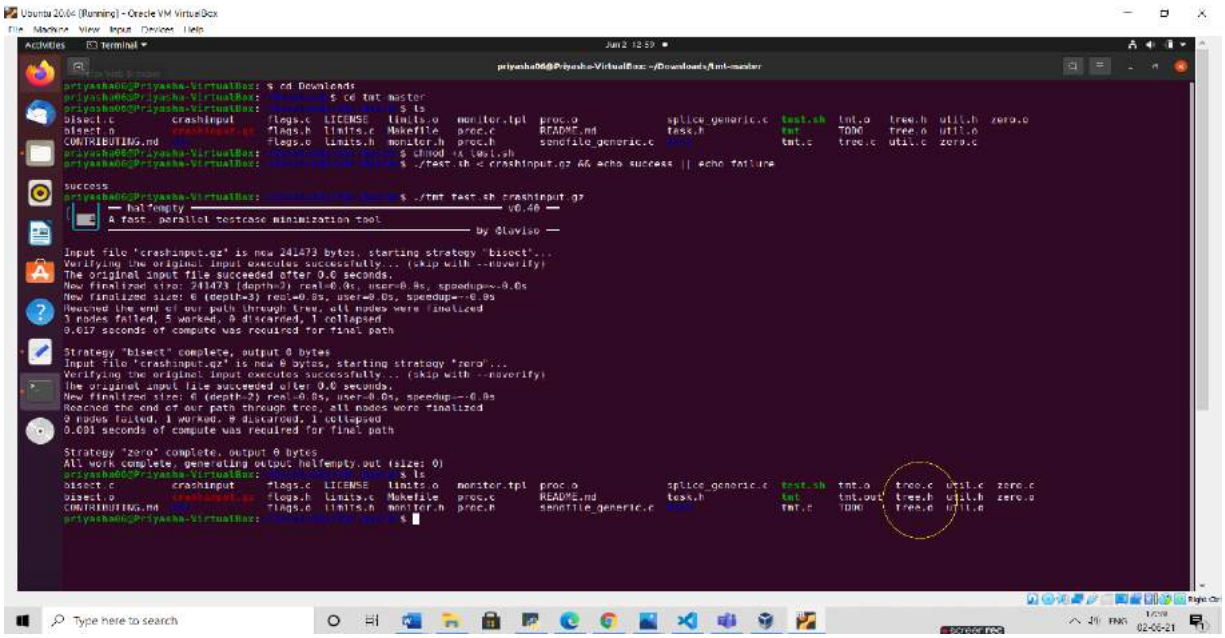
Tmt.out (output file) will be automatically generated in the folder which is opened.

7. RESULTS AND DISCUSSION:

Our problem statement was to minimise the number of testcases and we have successfully executed that, as it is shown below.

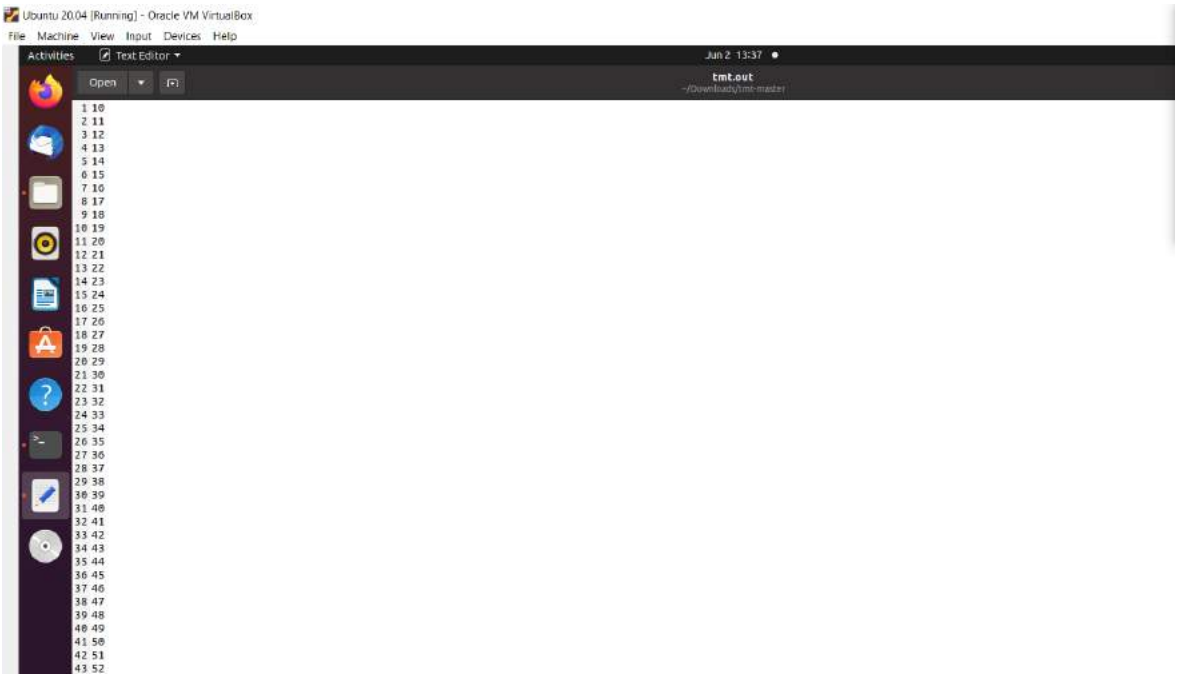


As we can see above, the list of files is shown before execution of the code.



As we can see here, a new file tmt.out has been generated automatically after execution.

It is encircled in yellow above as we can see.



This is the tmt.out file which was generated and it is of 270 bytes sizes (original size:241473 bytes).

Therefore, testcase minimisation uses bisection algorithm keeping parallisation in mind.

For future scope, the next version will allow the level of pessimism to be controlled at runtime.