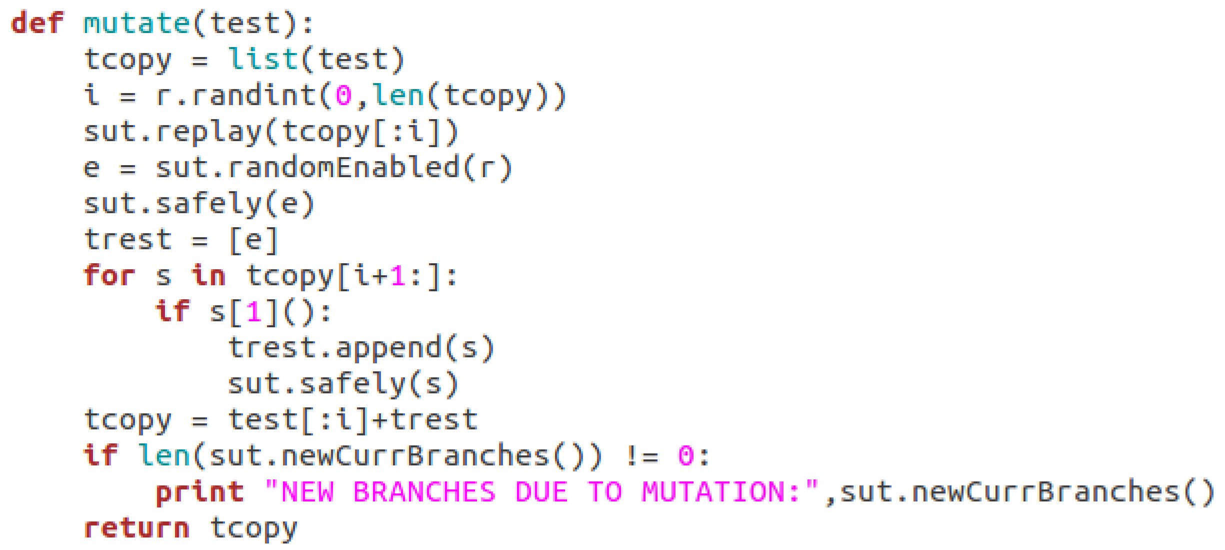
Competitive Milestone 2

Omkar Thakur

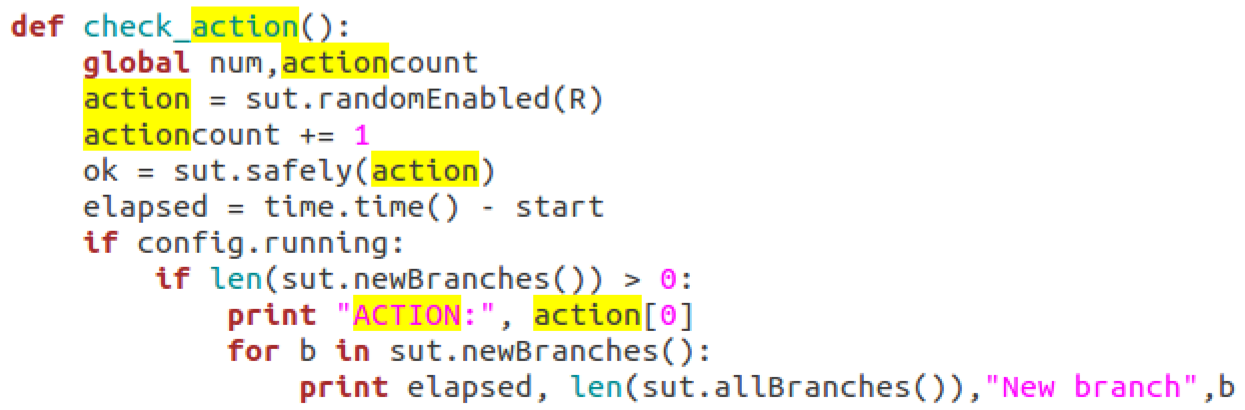
932704163

The main idea behind this algorithm is to cover as many branches and statements as far as possible. But due to addition of some mutation loops, the coverage started to decrease considerably. But it is yet to find whether it can produce qualitative results.

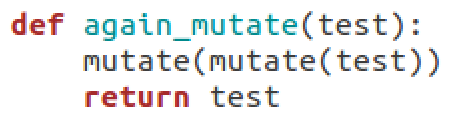
In my approach the FCFS and the mirror adaptive random testing was going to be implemented. But it was revealed that the implementation may not work in all the aspects. My main aim is also to improve random testing with respect to combination logic. The following new things I have added in project.



```
def mutate(test):
    tcopy = list(test)
    i = r.randint(0,len(tcopy))
    sut.replay(tcopy[:i])
    e = sut.randomEnabled(r)
    sut.safely(e)
    trest = [e]
    for s in tcopy[i+1:]:
        if s[1]():
            trest.append(s)
            sut.safely(s)
    tcopy = test[:i]+trest
    if len(sut.newCurrBranches()) != 0:
        print "NEW BRANCHES DUE TO MUTATION:",sut.newCurrBranches()
    return tcopy
```



```
def check_action():
    global num,actioncount
    action = sut.randomEnabled(R)
    actioncount += 1
    ok = sut.safely(action)
    elapsed = time.time() - start
    if config.running:
        if len(sut.newBranches()) > 0:
            print "ACTION:", action[0]
            for b in sut.newBranches():
                print elapsed, len(sut.allBranches()),"New branch",b
```



```
def again_mutate(test):
    mutate(mutate(test))
    return test
```

Bugs were not found in this part of the implementation but a number of branches were covered effectively. The basic implementation of FCFS was tried. First the system is restarted with the help of the sut.restart(). Then after each action of range, the random action is taken place. The efficiency of the algorithm has also to be seen. The current implementation does not look as efficient. The next milestone of this implementation will be tried to be more efficient.

Current Work:

I have used some mutation algorithm in order to gain some quantity of branches. The current approach uses mutation algorithm. In one condition, the branches and statements covered went to be 4 and 5 respectively.

The results were as follows:

Branch count is 187

Statement count is 144

The failure file has also been added to the current implementation. Some function changes were done in the tester1.py and tester2.py but the results were drastically different that of tester2.py

How to run:

```
python tester2.py 30 1 100 1 0 1 1 100
```

References:

http://www.cs.cmu.edu/~agroce/nfm15.pdf

http://link.springer.com/chapter/10.1007%2F978-3-319-17524-9_15#page-1