**CS330 Programming Assignment Memo**

**Adam Sadek**

## Part 2 intruder estimation:

The probability that the intruder inputs the first correct digit is 1/9. Then, the probability that the next correct digit is inputted is 1/9 * 1/9. For the sequence of all 6 correct digits being inputted, the probability is $(\frac{1}{9})^6$ which is equal to $\frac{1}{531,441}$ or 0.000001881676423. This is clearly a very slim chance. This means that on average, it would take around 531,441 attempts before the correct 6 digit sequence is inputted. If, after each digit, the intruder must wait one second to see if the lock has been unlocked, the intruder will wait on average 531,441 seconds or 147.6225 hours or 6.1509375 days.

## How the Programs Work:

Sadek_Adam_CS330_Programming_Assignment.py:

The lock starts in a locked state, as shown by the attribute *status*. There is a while loop that executes infinitely until the user exits the program. In this while loop, the program takes an input string from the user. The program first removes all characters from the string that are not digits between 0-9 by use of the regular expression module *re* in python that is used to check if a string contains a specified search pattern – in this case, a character that is not a digit between 0-9.

If the length of the string after non-digit removal is less than the length of the lock or unlock code, the program just outputs the current status of the lock and prompts the user to input another string.

If the length of the string is equal to or greater than the length of the unlock or lock code, then a for loop starting from 5 and going until the length of the input string is initiated. This is done so as to not encounter an indexing error.

If it is known that there are at least 6 characters, then the element at index “element - 5” will be the first element in the inputted string. On the last loop of the for loop, where element is equal to the length of the input string, the element at index “element - 5” will be the 6th last element (i.e. the first element in the last possible spot where the lock/unlock code could be placed in the input string). This method ensures that no indexing error can occur no matter the length of the input string the user inputs. The indexed elements are compared with the elements of the lock and unlock codes.

If there is a match with the unlock code, the status is changed to ‘Unlocked.’, and if there is a match with the lock code, the status is changed to ‘Locked.’. After going through the whole for loop, the status of the lock is printed to screen and the while loop reiterates.

Sadek_Adam_CS330_Programming_Assignment_Pt2.py:

Both the time and randint modules are imported. The time module is used to time how long it takes for the machine to find the access code through a random number generator made possible through randint. A function titled generator returns a string that contains one integer between 0-9. This program appends a random number to a growing list of numbers until the unlock code ‘024451’ is found in the list of

numbers. This process is timed, and 20 trials of the process are done. The average, maximum, and minimum times and symbols needed are then outputted onto the screen as follows from this example below:

Times to find access code (sec):
    Average:  1.5059203505516052
    Minimum:  0.03596901893615723
    Maximum:  6.5189208984375
Symbols needed to find access code:
    Average:  1014482.05
    Minimum:  25471
    Maximum:  4313510

The attribute time_taken is used to store the total amount of time taken for the machine to find the unlock code 20 times. The list_of_times attribute is a list that stores the time it took for the machine to find the unlock code for each of the 20 trials. When divided by 20, this gives the average time taken to find the unlock code,stored in the attribute average_time. The list_of_numbers attribute is a list that stores the number of symbols (digits 0-9) used by the machine before the unlock code was found for each of the 20 trials.

The total_count attribute is used to store the total number of symbols used throughout the 20 trials. When divided by 20, this gives the average number of symbols used, stored in the attribute average_counter. The for loop is run 20 times, once for each trial. Each loop begins by initializing the empty list of numbers generated by the machine (rand_num starts as an empty list’’). The counter which holds the number of symbols generated by the machine is also set to 0, the boolean attribute found is set to False, and is only True once the unlock code is found in the rand_num attribute.

Then, the timer starts, and a while loop which executes until found is True, checks to see if the length of the rand_num is greater than or equal to the length of the unlock code. If it isn't, then another random number is appended to the rand_num attribute, the symbol counter is upped by 1, and the while loop iterates again. Once the length of rand_num is greater than or equal to the length of the unlock code, the last digit of rand_num is compared with the last digit of the unlock code, and moving backwards until the 6th last digit is compared with the 6th last (or first) digit of the unlock code.

If there is a total match, found = True and the while loop ends, also ending the timer. If there is not a complete match, another digit is added to rand_num and the symbol counter is upped by 1 again. Once found eventually equals True, the time taken can be found by subtracting the end time attribute by the start time attribute. This value is appended to the list_of_times list which will hold all of the times taken for all 20 trials.

The symbol counter for that particular trial is then also appended to the list_of_numbers list. The time_taken attribute is then added to by the time taken for this particular trial and the total_count attribute is then added to by the symbol count of this particular trial. This occurs for each of the 20 trials.

After the 20 trials, average time and average symbol count are found by dividing their total values by 20. The max and min times and symbol counts are also found by using the max() and min() functions on the list_of_numbers and list_of_times lists. Then, all of this data is outputted to the screen.

The reason why I chose to do 20 trials of the random number generator is because there is significant variance between trials as I found out after running each trial individually. Even between different trial runs of 20 there were noticeable differences in terms of average time and average symbol count. I chose to stick with 20 trials however because a larger amount of trials takes too long to run and a smaller amount of trials does not give a good enough idea of real averages/max and min values.

**My findings, the language I chose and the regular expression:**

Findings in Sadek_Adam_CS330_Programming_Assignment.py:

- Inputting a string smaller than length of lock or unlock immediately results in the program outputting the current lock status
- Inputting a string with an unlock code and then a lock code later in the string results in the status changing to unlocked and then locked (only the final status is shown on screen)
- Inputting a string with an lock code and then an unlock code later in the string results in the status changing to locked and then unlocked
- Inputting a string with a lock or unlock code with non-digit characters intermixed within it still locks or unlocks respectively, as the non-digit characters are ignored before comparing strings
- If the status is locked and a string is inputted that does not include a lock or unlock code, the status remains locked
- If the status is unlocked and a string is inputted that does not include a lock or unlock code, the status remains unlocked

Findings in Sadek_Adam_CS330_Programming_Assignment_Pt2.py

- I found that because of the nature of the random generator, it is not easy to determine how long the machine will take to find the access code. Sometimes the machine gets lucky and finds it very quickly, and sometimes it takes it very long to unlock the lock.
- I found that even with 20 trial runs, the averages achieved through different groups of 20 trial runs differed significantly. For instance, these are two runs of 20 trial runs:

| | |
|---|---|
| Times to find access code (sec):<br>Average: 1.5059203505516052<br>Minimum: 0.03596901893615723<br>Maximum: 6.5189208984375<br>Symbols needed to find access code:<br>Average: 1014482.05<br>Minimum: 25471<br>Maximum: 4313510 | Times to find access code (sec):<br>Average: 1.8751803636550903<br>Minimum: 0.1769399642944336<br>Maximum: 4.727880954742432<br>Symbols needed to find access code:<br>Average: 1254888.3<br>Minimum: 122116<br>Maximum: 2957632 |

In the example on the left, the minimum time is much less than the minimum time in the example on the right, and the maximum time is much greater than the maximum time in the example on the right. They have similar average times but the example on the left has greater variance. The example on the left has a

smaller average number of symbols needed (about 240,406 less characters on average). However, the maximum number of symbols needed is much greater in the left example than the right example (1,355,878 more symbols needed). The minimum number of symbols needed is also much less in the left example than the right example (96,645 symbols less in the minimum case). This shows that there can be varying levels of variance between the data, which makes sense since the machine is essentially just trying to 'get lucky' with the correct string of digits – some trials will be luckier than others. I would like to note however that there does seem to be a solid range where the vast majority of trials fall within the range of 1-4 seconds and 800,000-1,500,000 symbols needed.

In this assignment I used python3 as my programming language, and the regular expression of python is re, which was used in Sadek_Adam_CS330_Programming_Assignment.py to check if the user-inputted string contains a character that is not a digit between 0 and 9.

The language of the FSM is the set of digits {0,1,2,3,4,5,6,7,8,9} and the regular expression is 66457(1|4).

**State transition diagram:**