**Unit 3 Algorithmics**

**Submit Task – Week 3**

Choosing ADTs

Describe which ADT you would use for each problem, providing a short justification.

1. Employees of a bank are allowed to access certain functions depending on their roles. Functions include CCTV control, monitoring ATM transactions and changing customer details. How could you store the details of which employees are allowed to access which functions?
   1. **Dictionaries could be used to map users to their roles, where keys represent the employee and the values contain a role.**
2. At a conference, people queue to use elevators. We want to get people to their rooms as quickly and as fairly as possible. If each person is allocated a ticket when they arrive at the lobby, how could you use ADTs to work out who gets on the next elevator?
   1. **A queue can be used to represent the elevator queue. Each person arriving at the lobby is allocated a ticket and added to the end of the queue. The first person in the queue will be the next to enter the elevator when it arrives.**
3. Reverse Polish Notation works by stating arguments and then the operation. For example:

3 5 add 3+5

2 6 add 7 times (2 + 6) × 7

2 4 add 5 3 sub exp (2 + 4)(5-3)

Which ADT is most suitable for interpreting these expressions?

1. **A stack is the most suitable datatype as operands are stated before the operation. When an operator is encountered, it operates on the top elements of the stack and the result is pushed back onto the stack.**

Prison Door Problem

A prison contains a large number of cells, numbered sequentially from 1 to 500. One night, when the prisoners are asleep, a bored guard unlocks every cell. Then, he returns to the start. He stops at every cell that is a multiple of two. If the cell is unlocked, he locks it. If it is locked, he unlocks it. He repeats this process for multiples of three, then four, and so on.

This table shows what happens for the first 7 cells.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Pass 1 | Unlocked | Unlocked | Unlocked | Unlocked | Unlocked | Unlocked | Unlocked |
| Pass 2 | Unlocked | Locked | Unlocked | Locked | Unlocked | Locked | Unlocked |
| Pass 3 | Unlocked | Locked | Locked | Locked | Unlocked | Unlocked | Unlocked |
| Pass 4 | Unlocked | Locked | Locked | Unlocked | Unlocked | Unlocked | Unlocked |

Your aim is to find out which cells are unlocked after the final pass.

1. Describe which ADTs you will use to model this problem.

For the most basic implementation an array could be used to reference the room numbers using indices of each “room” in the array along with a “True” or “False” status for each based on if its locked or not.

1. Write some pseudocode to perform the necessary steps.

**Define cells <- ADT Array**

**cells <- [False] \* 501**

**FOR i<-2 to 501 DO**

**FOR j<-I TO 501 STEP BY i**

**IF cells[j] == True:**

**cells[j] <- False**

**ELSE:**

**Cells[j]<-True**

**ENDFOR**

**ENDFOR**

**Sum <- COUNT(Cells == True)**

**DISPLAY (Sum)**

1. Translate your pseudocode into Python.
2. cells = [False] \* 501
3. for i in range(2, 501):
4. for j in range(i, 501, i):
5. cells[j] = not cells[j]
6. print("Locked Cells: ", sum(1 for cell in cells if cell))
7. Run your code to identify which doors are left open.

22 are left open

Hint: You should find that 225 and 484 are open, but 170 and 499 are closed.

There is a mathematical reason behind this problem, but even if you figure it out, make sure you model the process as an algorithmics problem.

**I don’t really know what this was referring too but I think has something to do with factors maybe? Factors of 6 for eg are 1,2,3 and 6 while the factors of something like 12 are 1,2,3,4,6 and 12. Aka cell 6 will be locked as its only toggled by 2 and 3 but 12 will be toggled by 2,3 AND 4 resulting it being unlocked? So I guess one could just count the number of factors and if its odd its locked and if its even its unlocked? Idk if this works for all numbers so I could just be offtrack and ranting but there may be a more efficient way of solving this.**

**EDIT: Nevermind I figured it out, perfect squares remain open!**