CO322: Data Structures and Algorithms

Lab 02: HR problems

WIJERATHNE E.S.G

E/18/397

---

01. Grading Students

```java
public static List<Integer> gradingStudents(List<Integer> grades) {
    for (int i = 0; i<grades.size(); i++){

            if (grades.get(i)>=38){

                if(((grades.get(i)+1) % 5) == 0 ){
                    // for values like 39, 44 , 49
                    grades.set(i,grades.get(i) +1)  ;

                }else if(((grades.get(i)+2) % 5) == 0 ){
                    // for values like 38, 43 , 48
                    grades.set(i,grades.get(i) +2)  ;

                }
                else{
                    //no change in values already divide by 5 and not in above if and else if parts
                    //grades[i] = grades[i];
                }
            }else{

                // no change values below 38
                //grades[i] = grades[i];

           }

    }
    return grades;

}
```

Here first I check for the value is greater than or equal to 38. If the value is less than 38, I let the value as it is. Otherwise, I check for the (value + 1) and (value + 2) modules is equal to 0. If it is I set that value to the nearest value which will be divided by 5. Otherwise, no change in the value.

02. The power sum

```java
public static int powerSum(int X, int N) {

    return powerSum(X, N, 1);
}

public static int powerSum(int X, int N, int start) {

    // base case
    if(X == 0)
        return 1;

    // variable to store count for each iteration
    int count = 0;

    // check for combinatons
    for (int i = start; Math.pow(i, N) <= X; i++) {
        count += powerSum((int) (X - Math.pow(i, N)), N, i+1);
    }
    return count;
}
```

Start with examples X=13 and N=2. First, we start values with 1^2. We subtract that value from the X and then go to the next step. Now X = 12 and start = 2. Then I consider $12 - 2^2$. Now I have X = 8 and the start value as 3. $3^2 = 9$ and we cannot make 13. So we start considering other combinations using recursion.

03. Caesar Cipher

```java
public static String caesarCipher(String s, int k) {

    String alphabet = "abcdefghijklmnopqrstuvwxyz";
    String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String output = "";
    for(int i = 0 ; i < s.length(); i ++){

        // to store alphabet positions
        int currentPositionSimple;
        int currentPositionCapital;

        // getting current position according to in alphabets. (simple and capital)
        currentPositionCapital = ALPHABET.indexOf(s.charAt(i));
        currentPositionSimple = alphabet.indexOf(s.charAt(i));

        // for capital letters in the string
        if(currentPositionCapital != -1 && currentPositionSimple == -1){
            int cypherPlaceCapital = (k + currentPositionCapital) % 26;
            output += ALPHABET.charAt(cypherPlaceCapital);
       }

       // for simpleletters in the string
        else if(currentPositionSimple != -1 && currentPositionCapital == -1){
           int cypherPlaceSimple = (k + currentPositionSimple) % 26;
           output += alphabet.charAt(cypherPlaceSimple);
       }

       // when not a leter
       else if(currentPositionSimple == -1 && currentPositionCapital == -1){
           output += s.charAt(i);
       }

   }

    return output;
}
```

I have two strings with the simple and capital alphabet to get the positions of each letter. Then I am iterating through each letter and get relevant character positions. Once find the position get the relevant rotated letter position and a new letter will be added to the return string.

04. Climbing the Leaderboard

```java
public static List<Integer> climbingLeaderboard(List<Integer> ranked, List<Integer> player) {

        List<Integer> out = new ArrayList<>();

        // getting set of ranked data (to stop duplicating same value)
        Set<Integer> targetSet = new HashSet<>(ranked);

        // lsit to store above data as sorted data
        List<Integer> sortedList = new ArrayList<>(targetSet);

        Collections.sort(sortedList);

        int i = 0;

        for (Integer play : player){
            // check for the ranks
             while (i<sortedList.size() && sortedList.get(i)<=play){
                 i+=1;
             }
            // adding new ranks
            out.add(sortedList.size() - i +1);

        }

        return out;

    }
```

First, get the ranked list values to a set to eliminate duplicate values. Then those values will be sorted. Then I iterate through new player values and check that the corresponding rank values are according to the old list. Once find a rank it will be added to the output list.