MathWorks

Math Question Part

1. Probability for a ball Selection: a bag has 3 white balls and 5 black balls. take two draws randomly, ask for the probability in second drawing with white ball?
   Answer: (3/8 * 2/7) + (1 – 3/8) * 3/7 = 21/56

2. Largest 32-bits Unsigned Integer
   Answer: $2^{31} - 1 = 2147483647$

3. Infinite Series: Evaluate $\sum_{i=1}^{\infty} 1/i$?
   Answer: Series doesn't converge

4. Tennis Tournament: 512 players participated in a tennis tournament. how many matches were producing?
   Answer: 511

5. Cycle-Free Graph: A graph have N vertices. What is the maximum number of edges it can have so that the undirected graph is cycle-free?
   Answer: N-1.

6. Probability for a Card Selection: From a pack of 52 cards, two cards are drawn to together at random. What is the probability that both cards are king?
   Answer: 1/221　(4/52)*(3/51)

7. Number Properties: How many positive integers are there between 100 and 999 inclusive are divisible by 3 or 4?
   Answer: 450

8. Ant Probability: There is an ant on each vertex of a pentagon. What is the probability of collision if they start walking on the edges of the pentagon?
   Answer: 15/16.

9. 口袋 A 有 1 个蓝球 4 个黄球，B 有 3 个蓝球 2 个黄球，随机拿一个球，问一直是蓝色球的情况下，球从 A 口袋里面出来的概率有多大？
   Answer: 1/4

10. 400 个人的房间里面任意挑选 2 个人的生日概率是多大？
    Answer: (据维基百科这是一个生日悖论问题，旦凡这个房间超过 50 个人则任意两个人的生日有超过 99.9%的概率相同)。第二个人与第一个人生日不同的公式是：$365!/365^n(365-n)!$

11. 50 张扑克牌背面朝下，第 1 次把所有牌依次翻面，第 2 次每隔 1 张牌翻一次面，第 3 次每隔 2 张牌翻一次面，。。。，第 50 次只翻最后一张。问 50 次之后有几张牌正面朝上？
    Answer: 7。

12. 还有一道题是从 1st ave，121 street 走到 2ed ave，1 street，哪里掉东西概率最大?
    Answer:我假设掉东西是柏松分布，得到答案是 2ed ave，40 几街

13. 一个小镇，有 tv 的人家 60 个，有 scooter 的人家 85 个，有冰箱的 70 个，有 radio set 的 95 家，已知有 130 家只有其中一个物件，问这个小镇最多有几户人家？
    Answer: 215

14. 120 个学生选课 学号 1-120 被 2 整除的选了 CS 被 5 整除的选了 ME 被 7 整除的选了 EE 求什么都没选的学生人数。
    Answer: 41

15. 12 人学了英语和德语，22 人学了德语，一共 40 人，若所有人至少学了英语或者德语，那么只学英语有几个人？
    Answer: 18

16. Drone delivery. 起点（1，121），终点（2，1）.问选项中最有可能掉落的地点？
    Answer: 找第一行上靠近（1，121）的点，或者第二行最靠近（2，1）的点。

17. 一种 tri bit， 能表达 0，1，2 问 8 位的 tribit 能最大?
    Answer: unsigned number 3^8-1

18. Jane, James 和 Josh 分别花 9,6,14 天各自独立完成一项任务，问 3 人同时完成这项任务并按各自付出比例瓜分 4400 块钱时，jane 得到多少钱？
    Answer:（1400）1/9 + 1/6 + 1/14 = 14 + 21 + 9/ 3327 = 22/ 33 * 7 = 1400

19. How many four-digit numbers divisible by 11 are not palindromes?
    Answer: 729

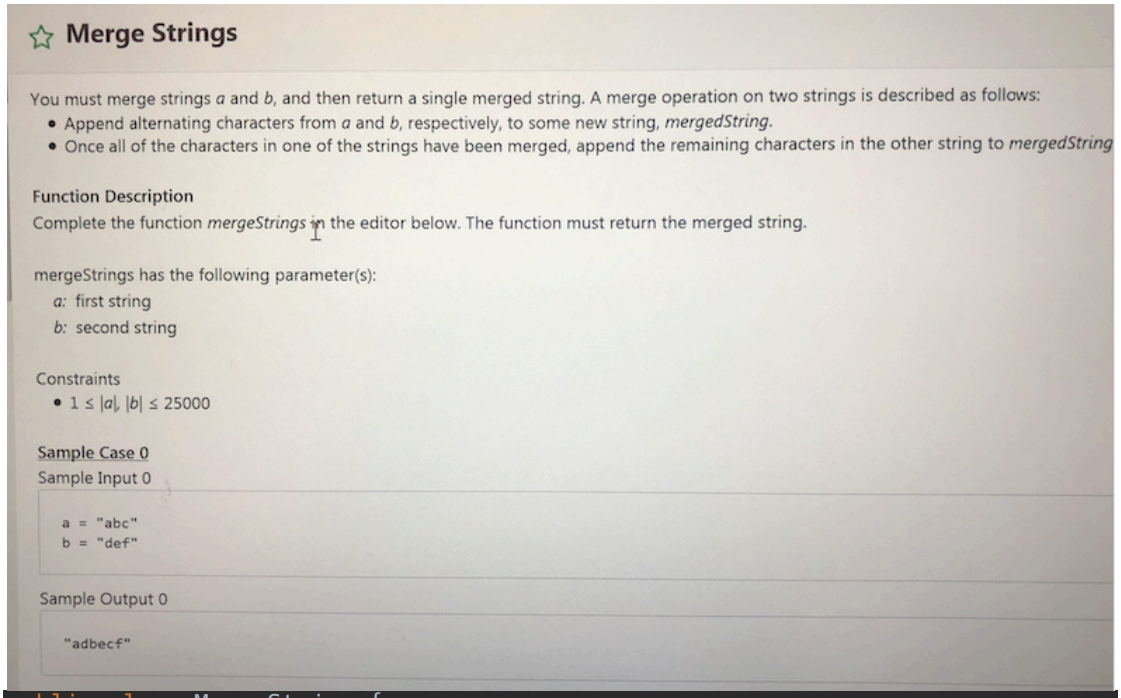
20. BANANA permutation:
    Answer: 60

21. -5, -8, -11… 101th number is?
    Answer: -305

22. initial value of f（X）= e^(0.5X)，当 X = 0 时，f（0）是多少?
    Answer: 0.5^n

Algorithm Part

1. Merge String



Merge Strings

You must merge strings *a* and *b*, and then return a single merged string. A merge operation on two strings is described as follows:

- Append alternating characters from *a* and *b*, respectively, to some new string, *mergedString*.
- Once all of the characters in one of the strings have been merged, append the remaining characters in the other string to *mergedString*

Function Description

Complete the function *mergeStrings* in the editor below. The function must return the merged string.

mergeStrings has the following parameter(s):

a: first string
b: second string

Constraints

- $1 \leq |a|, |b| \leq 25000$

Sample Case 0

Sample Input 0

```
a = "abc"
b = "def"
```

Sample Output 0

```
"adbecf"
```

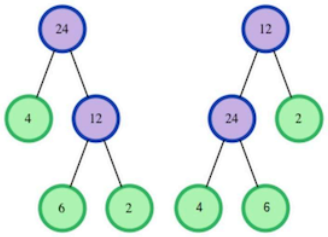
```java
public class MergeString {
    private static String mergeString(String str1, String str2){
        if(str1 == null || str1.length() == 0){
            return str2 == null ? "" : str2;
        }
        int index = 0;
        StringBuilder sb = new StringBuilder();
        if(str1.length() >= str2.length()){
            for(int i = 0; i < str2.length(); i++){
                sb.append(str1.charAt(i)).append(str2.charAt(i));
                index++;
            }
            if(index < str1.length()){
                sb.append(str1.substring(index));
            }

        } else{
            for(int i = 0; i < str1.length(); i++){
                sb.append(str1.charAt(i)).append(str2.charAt(i));
                index++;
            }
            if(index < str2.length()){
                sb.append(str2.substring(index));
            }
        }
        return sb.toString();
    }
}
```

2. The Cost Of a Tree

Given an array of integers, construct a tree. Each node of the tree has either two children or none, in which case it is a leaf node. A leaf node costs *0* to construct. The cost to build a parent node is the product of the maximum leaf values in its left and right sub-trees. Partition the array to minimize the cost of building the entire tree.

For example, there are *n = 3* elements in the array: *arr = [4, 6, 2]*. There are two possible choices to split the array: *{4}, {6,2}* and *{4, 6}, {2}*. The array elements can not be reordered. Leaves are shown in green, and parent nodes are in blue.



Working through the first choice, the left sub-tree is *4* and the right sub-tree is *{6,2}*. The maxima are *4* and *6*, so the root node costs *4 * 6 = 24* to create. The left node is a leaf so it has a cost of *0*. Now create leaves for *{6, 2}*, with their parent costing *6 * 2 = 12* to construct. The entire tree costs *24 + 12 = 36*.

If the same analysis is performed on the second choice, the root node costs *max({4, 6}) * 2 = 12*, and the node below that on the left costs *4 * 6 = 24*. Again, the total cost is *36*. If these were different, the minimum would be chosen. The answer is *36*.

Function Description

Complete the function *calculateCost* in the editor below. The function must return an integer that represents the minimum cost to construct the tree.

calculateCost has the following parameter(s):
arr[arr[0],...arr[n-1]]: an array of integers

Constraints

- *2 ≤ n ≤ 50*
- *1 ≤ arr[i] ≤ 1000*

```java
public class CostOfATree {
    public static int buildTreeMinCosr(int[] array){
    if(array == null || array.length <= 1){
        return 0;
    }
    int result = 0;
    int r_boundary = array.length - 1;
    while(r_boundary > 0){
        int pair_start = 0;
        int min_cost = array[0] * array[1]; // default min product from a pair

        //get pair with min Product
        for(int i = 0; i < r_boundary; ++i){
            int cur_cost = array[i] * array[i+1];
            if(min_cost > cur_cost) {
                pair_start = i;
                min_cost = cur_cost;
            }
        }
        int single = pair_start + 1;
        array[pair_start] = Math.max(array[pair_start], array[pair_start+1]);
        for(int i = pair_start + 2; i <= r_boundary; ++i){ //O(n)
            array[single] = array[i];
        }

        result += min_cost;
        r_boundary--;
    }
    return result;
}
    }
```

3. reverse words (similar to Leetcode151 Reverse Words in a String)
   code:

```
public String reverseWords(String s) {
    if(s.length() == 0 || s == null) return "";
    s = s.trim();
    String[] sa = s.split("\\s+");
    StringBuilder sb = new StringBuilder();
    for(int i = sa.length - 1; i >= 0; --i){
       sb.append(sa[i]).append(" ");
    }
    return sb.toString().trim();
  }
```

4. Factorial Trailing Zeroes

```
public int trailingZeroes(int n) {
    return n == 0 ? 0 : n/5 + trailingZeroes(n/5);
 }
```

5. Substrings

```
public class SubStrings {
    public static void findSubstrings(String s){
        if(s == null || s.length() == 0) return;
        s.toLowerCase();
        Set<Character> vowelSet = new HashSet<>();

vowelSet.add('a');vowelSet.add('e');vowelSet.add('i');vowelSet.add('o');vowe
lSet.add('u');
        Set<Character> consonantSet = new HashSet<>();

consonantSet.add('b');consonantSet.add('c');consonantSet.add('d');consonantS
et.add('f');consonantSet.add('g');

consonantSet.add('h');consonantSet.add('j');consonantSet.add('k');consonantS
et.add('l');consonantSet.add('m');

consonantSet.add('n');consonantSet.add('w');consonantSet.add('p');consonantS
et.add('q');consonantSet.add('r');

consonantSet.add('s');consonantSet.add('t');consonantSet.add('v');consonantS
et.add('x');consonantSet.add('y');
        consonantSet.add('z');

        List<String> substrings = new ArrayList<>();
        generate(substrings, s);
        List<String> sLists = new ArrayList<>();

        for(int i = 0; i < substrings.size(); i++){
            if(!substrings.get(i).equals("")
                    && vowelSet.contains(substrings.get(i).charAt(0))
                    &&
consonantSet.contains(substrings.get(i).charAt(substrings.get(i).length()-
1))){
                sLists.add(substrings.get(i));
            }
        }
        String[] results = new String[sLists.size()];
        for(int i = 0; i < sLists.size(); i++){
            results[i] = sLists.get(i);
        }
        Arrays.sort(results, new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                return o1.compareTo(o2);
            }
        });
        System.out.println(results[0] + ", " + results[results.length-1]);
    }

    public static void generate(List<String> substrings,String word) {
        if (word.length() == 1) {
            substrings.add(word);
            return;
        }else{
            substrings.add(word);
            generate(substrings,word.substring(0, word.length()-1));
            generate(substrings,word.substring(1, word.length()));
        }
    }
}
```

6. Traveling is Fun

```
static int[] connectedCities(int n, int g, int[] originCities, int[] destinationCities) {
        int[] root = new int[n + 1];
        int[] ids = new int[n + 1];
        for (int i = 0; i <= n; i++) {
            root = i;
            ids = 1;
        }
        for (int i = g + 1; i <= n; i++)
            for (int j = 2 * i; j <= n; j += i)
                unionFind(j, i, root, ids);
        int[] res = new int[originCities.length];
        int i = 0;
        int j = 0;
        int k = 0;
        while (i < originCities.length && j < destinationCities.length) {
            if (getRoot(originCities, root) == getRoot(destinationCities[j], root)) {
                res[k] = 1;
            } else {
                res[k] = 0;
            }
            i++;
            j++;
            k++;
        }
        return res;
    }
    private static void unionFind(int a, int b, int[] root, int[] ids) {
        int aRoot = getRoot(a, root);
        int bRoot = getRoot(b, root);
        if (aRoot == bRoot)
            return;
        if (ids[aRoot] < ids[bRoot]) {
            root[aRoot] = root[bRoot];
            ids[bRoot] += ids[aRoot];
        } else {
            root[bRoot] = root[aRoot];
            ids[aRoot] += ids[bRoot];
        }
    }
    private static int getRoot(int a, int[] root) {
        while (a != root[a])
            a = root[a];
        return a;
    }
```

```
}
```

7. Reverse a Linked List

```
public ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode current = head;
    while(current != null) {
      ListNode nextTemp = current.next;
      current.next = prev;
      prev = current;
      current = nextTemp;
    }
    return prev;
  }
```

8. Redundancy in a linked list

```
public Node removeDup(Node head){
        HashMap<Integer, Integer> ht = new HashMap<Integer, Integer>();
        if(head==null) return null;
        Node currNode = head.next;
        Node prevNode = head;
        Node temp;
        ht.put(head.data, 1);
        while(currNode!=null){
                int data = currNode.data;
                if(ht.containsKey(data)){
                        prevNode.next = currNode.next;
                        temp= currNode;
                        currNode = currNode.next;
                        temp.next = null;
                }else{
                        ht.put(data, 1);
                        prevNode = currNode;
                        currNode = currNode.next;
                }
        }
        return head;
  }
```

9. Maximum difference in an array
   给一个 int list，当前的每个 int 值和前面所有的值比较，如果比前面的大就计算差值，找出 list 中最大的差值。 For a given array, find the maximum value of a[j] - a for all i, j where 0<= i < j < n and a < a[j], if there are no lower indexed smaller items for all the items, return -1.

```
/* The function assumes that there are at least two

   elements in array.

   The function returns a negative value if the array is

   sorted in decreasing order.

   Returns 0 if elements are equal */

int maxDiff(int arr[], int arr_size)

{

   int max_diff = arr[1] - arr[0];

   int i, j;

   for (i = 0; i < arr_size; i++)

   {

      for (j = i + 1; j < arr_size; j++)

      {

         if (arr[j] - arr[i] > max_diff)

            max_diff = arr[j] - arr[i];

      }

   }

   return max_diff;

}
```

10. Build office

思路：a. 先把公共有几种 build office 的情况写出来。b. 对每一种情况使用 BFS 来求距离。 c.然后把所有的 matrix 中的距离计算出来，得到最大的距离。

BFS 的方法（Leetcode 542）

```
public int[][] updateMatrix(int[][] matrix) {
        Queue<int[]> queue = new LinkedList<>();
        int m = matrix.length, n = matrix[0].length;
        boolean[][] visited = new boolean[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] == 0) {
                    queue.offer(new int[]{i, j});
                    visited[i][j] = true;
                }
            }
        }
        int[][] dir = new int[][]{{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
        while (!queue.isEmpty()) {
            int[] cur = queue.poll();
            for (int i = 0; i < 4; i++) {
                int row = cur[0] + dir[i][0];
                int col = cur[1] + dir[i][1];
                if (row < 0 || row >= m || col < 0 || col >= n ||
visited[row][col]) {
                    continue;
                }
                visited[row][col] = true;
                matrix[row][col] = matrix[cur[0]][cur[1]] + 1;
                queue.offer(new int[]{row, col});
            }
```

```
        }

        return matrix;

    }
```