```
//encode and decode with encode length
import java.io.*;
import java.util.*;

class Solution {

 public static void main(String[] args) {
  String input = "aaaaabbbb2222222222222233333333333333";

  System.out.println("Original String Length: " + input.length());
  Solution RLE = new Solution();
  String encodedStr = RLE.encode(input);
  System.out.println("Encoded String: " + encodedStr);
  System.out.println("Encoded String Length: " + encodedStr.length());

  String decodedStr = RLE.decode(encodedStr);
  System.out.println("Decoded String: " + decodedStr);
 }

 /**
  * 用Run-Length算法编码字符串
  * @param sourceStr 原始字符串
  * @return
  */
 public String encode(String sourceStr) {
  if(sourceStr == null || sourceStr.length() <= 1) {
   return sourceStr;
  }

  int len = sourceStr.length();
  StringBuilder resultBuilder = new StringBuilder();
  for(int i = 0; i < len; i++) {
   char cur = sourceStr.charAt(i);
   int runLength = 1;
   while((i+1) < len && sourceStr.charAt(i+1) == cur) {
    i++;
    runLength++;
   }

   if(runLength > 1) {
    resultBuilder.append(runLength + "*" + cur);
   }
```

```
   else {
    resultBuilder.append(cur);
   }
  }

  return resultBuilder.toString();
 }

 /**
  * 解码Run-Length编码的字符串
  * @param encodedStr
  * @return
  */
 public String decode(String encodedStr) {
  if(encodedStr == null || encodedStr.length() <= 1) {
   return encodedStr;
  }

  int len = encodedStr.length();
  StringBuilder resultBuilder = new StringBuilder();
  for(int i = 0; i < len; i++) {
    int num = 0;
   while(Character.isDigit(encodedStr.charAt(i))) {
    num = num * 10 + encodedStr.charAt(i) - '0';
    i++;
   }
    i++;
    char nextChar = encodedStr.charAt(i);
    for(int j = 0; j < num; j++) {
     resultBuilder.append(nextChar);
    }
  }

  return resultBuilder.toString();
 }

}
```

https://leetcode.com/problems/lru-cache/description/

https://leetcode.com/problems/word-ladder/description/

//Given a list of strings, output the most frequent characters that are in the same group as the letter.
For example, for string "abc", a, b,c are in the same group. "bcd" are in the same group.
for a: b,c . for b: c as c occured in two groups with letter b. duplicates are not considered. i.e "aabc" is the same as "abc"

input: a list of strings { "abc", "bcd", "def"}. Waral 鍗氬鏈夋涔澶氭椅绔嬧€,
output: a: b, c
b: c
c: b
d: b, c, e, f
//

```
import java.io.*;
import java.util.*;


class Solution {
 class Pair {
  char c;
  int n;
  public Pair(char c, int n) {
    this.c = c;
    this.n = n;
  }
 }
 public static void main(String[] args) {
  Solution RLE = new Solution();
  String[] encodedStr = {"abc", "bcd", "def"};
  List<String> decodedStr = RLE.mostFrequency(encodedStr);
   for(String s : decodedStr) {
    System.out.println(s);
   }
 }
 public List<String> mostFrequency(String[] strs) {
   List<Pair>[] arr = new List[26];

   for(String s : strs) {
    Set<Character> set = new HashSet<>();
    for(char ch : s.toCharArray()) {
      set.add(ch);
    }
    for(char ch : s.toCharArray()) {
      set.remove(ch);
      helper(arr, ch, set);
```

```
      set.add(ch);
    }
   }
   List<String> res = new ArrayList<>();
   for(int i = 0; i < arr.length; i++) {
     List<Pair> list = arr[i];
     if(list == null) continue;
     Collections.sort(list, new Comparator<Pair>(){
      public int compare(Pair p1, Pair p2) {
        return p2.n - p1.n;
      }
     });
     StringBuilder sb = new StringBuilder((char)('a' + i) + ":");

     int num = list.get(0).n;
     for(Pair p : list) {
      if(p.n == num) {
        sb.append(p.c + ",");
      }else {
        break;
      }
     }

     res.add(sb.toString());
   }
   return res;
 }

  public void helper(List<Pair>[] arr, char ch, Set<Character> set) {
    List<Pair> list = arr[ch - 'a'];

    if(list == null) {
      list = new ArrayList<Pair>();
    }
    for(char c : set) {
      Pair pOrigin = find(list, c);

      list.remove(pOrigin);
      Pair p = new Pair(c, pOrigin.n + 1);
      list.add(p);
    }
    arr[ch - 'a'] = list;
  }
```

```
  public Pair find(List<Pair> list, char c) {
    for(int i = 0; i < list.size(); i++) {
      Pair p = list.get(i);
      if(p.c == c) {
        return p;
      }
    }
    return new Pair(c, 0);
  }

}
```

713. Subarray Product Less Than K

```
    int numSubarrayProductLessThanK(vector<int>& nums, int k) {
        if (k <= 1) return 0;
        int n = nums.size(), prod = 1, ans = 0, left = 0;
        for (int i = 0; i < n; i++) {
            prod *= nums[i];
            while (prod >= k) prod /= nums[left++];
            ans += i - left + 1;
        }
        return ans;
    }
```

Prefix to Postfix Conversion

Stack
https://www.geeksforgeeks.org/prefix-postfix-conversion/

Recursion

```
import java.io.*;
import java.util.*;
import java.math.*;
public class Solution {
    public static void main(String[] args) {
      Solution s = new Solution();
      System.out.println(s.preToPost("*+AB*C-DE"));
      System.out.println(s.preToPost("*-A/BC-/AKL"));
      System.out.println(s.preToPost("+AB"));
      System.out.println(s.preToPost("*+AB-CD"));
```

```
    System.out.println(s.preToPost("*+ABC"));
    System.out.println(s.preToPost("+A*BC"));


  }

  public String preToPost(String input) {
    if(isOperator(input.charAt(0))) {
      if(1 < input.length() && isOperator(input.charAt(1))) {
        if(isOperator(input.charAt(3))) {
          return preToPost(input.substring(1, 6)) + preToPost(input.substring(6)) + input.charAt(0);
        }else {
          return preToPost(input.substring(1, 4)) + preToPost(input.substring(4)) + input.charAt(0);
        }

      }else {
        if(isOperator(input.charAt(2))) {
          return "" + input.charAt(1) + preToPost(input.substring(2)) + input.charAt(0);
        }else {
          return "" + input.charAt(1) + input.charAt(2) + input.charAt(0);
        }

      }
    }else {
      return "" + input.charAt(0);
    }
  }
 public boolean isOperator(char c) {
  if(c == '+' || c == '-' || c == '*' || c == '/') {
    return true;
  }
  return false;
 }

}
```

Postfix to Prefix Conversion

https://www.geeksforgeeks.org/postfix-prefix-conversion/

Inversion Count

https://www.geeksforgeeks.org/count-inversions-array-set-3-using-bit/

最大岛屿问题及变种

Largest island，0为海，非0为岛（不仅仅是1），求最大的岛的面积。 同时要求如果同一个岛上出现相同的数字，比如【1,2,1】，这个岛就invalid。

```
import java.io.*;
import java.util.*;


public class Solution {
  int area = 0;

  public int getNumOfIslands(int[][] matrix) {
     Map<Integer, Integer> map = new HashMap<>();
     int max = 0;
     int m = matrix.length;
     int 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 (!visited[i][j]) {
              dfs(i, j, matrix, visited);
              if(map.containsKey(matrix[i][j])) {
               map.remove(matrix[i][j]);
              }else {
               map.put(matrix[i][j], area);
              }
              area = 0;
           }
        }
     }
     for(int v : map.values()) {
      max = Math.max(v, max);
     }
     return max;
  }

  private void dfs(int i, int j, int[][] matrix, boolean[][] visited) {
     if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length || visited[i][j]) {
        return ;
     }
     visited[i][j] = true;
     area++;
     if (i + 1 < matrix.length && matrix[i + 1][j] == matrix[i][j]) {
        dfs(i + 1, j, matrix, visited);
     }
     if (i - 1 >= 0 && matrix[i - 1][j] == matrix[i][j]) {
        dfs(i - 1, j, matrix, visited);
     }
     if (j + 1 < matrix[0].length && matrix[i][j + 1] == matrix[i][j]) {
        dfs(i, j + 1, matrix, visited);
```

```
        }
        if (j - 1 >= 0 && matrix[i][j - 1] == matrix[i][j]) {
            dfs(i, j - 1, matrix, visited);
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[][] test = {
                    {1, 2, 3, 3},
                    {1, 2, 3, 4},
                    {1, 3, 2, 2},
                    {3, 3, 3, 3}};
        System.out.println(solution.getNumOfIslands(test));
    }
}
```

//Union Find动态更新岛屿数

```
class Solution {
    int[][] dirs = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        List<Integer> result = new ArrayList<>();
        if(m <= 0 || n <= 0) return result;

        int count = 0;                      // number of islands
        int[] roots = new int[m * n];       // one island = one tree
        Arrays.fill(roots, -1);

        for(int[] p : positions) {
            int root = n * p[0] + p[1];     // assume new point is isolated island
            roots[root] = root;             // add new island
            count++;

            for(int[] dir : dirs) {
                int x = p[0] + dir[0];
                int y = p[1] + dir[1];
                int nb = n * x + y;
                if(x < 0 || x >= m || y < 0 || y >= n || roots[nb] == -1) continue;

                int rootNb = findIsland(roots, nb);
                if(root != rootNb) {        // if neighbor is in another island
                    roots[root] = rootNb;   // union two islands
                    root = rootNb;          // current tree root = joined tree root
                    count--;
                }
```

```
            }

            result.add(count);
        }
        return result;
    }

    public int findIsland(int[] roots, int id) {
        while(id != roots[id]) id = roots[id];
        return id;
    }
}
```

///不单是1 or 0，0不是岛屿，其他数都是岛屿， 更新岛屿数

```
import java.io.*;
import java.util.*;
import java.lang.Iterable;

class Solution {
    int[][] dirs = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        int[][] matrix = new int[m][n];
        List<Integer> result = new ArrayList<>();
        if(m <= 0 || n <= 0) return result;

        int count = 0;                    // number of islands
        int[] roots = new int[m * n];       // one island = one tree
        Arrays.fill(roots, -1);

        for(int[] p : positions) {
            int root = n * p[0] + p[1];     // assume new point is isolated island
            int val = p[2];
            if(val == 0) {
              continue;
            }
            matrix[p[0]][p[1]] = val;
            roots[root] = root;             // add new island
            count++;

            for(int[] dir : dirs) {
                int x = p[0] + dir[0];
                int y = p[1] + dir[1];
                int nb = n * x + y;
                if(x < 0 || x >= m || y < 0 || y >= n || roots[nb] == -1) continue;
```

```
                int rootNb = findIsland(roots, nb);
                if(root != rootNb && matrix[x][y] == val) {        // if neighbor is in another island
                    roots[root] = rootNb;   // union two islands
                    root = rootNb;          // current tree root = joined tree root
                    count--;
                }
            }

            result.add(count);
        }
        return result;
    }

    public int findIsland(int[] roots, int id) {
        while(id != roots[id]) id = roots[id];
        return id;
    }
    public static void main(String[] args){
      Solution s = new Solution();
      int[][] pos = new int[5][3];
      pos[0] = new int[] {0,0,2};
      pos[1] = new int[] {0,1,2};
      pos[2] = new int[] {1,2,2};
      pos[3] = new int[] {2,2,2};
      pos[4] = new int[] {1,1,2};
      List<Integer> test = s.numIslands2(3,3,pos);
      for(int i : test) {
        System.out.println(i);
      }

    }
}
```

///PStack

```
public class Solution {

    static class PStack {
        private int size;
        private int element;
        private PStack previous;

        public PStack(PStack previous, int element, int size) {
            this.previous = previous;
            this.element = element;
```

```
        this.size = size;
    }

    public PStack() {
        this.previous = this;
        this.size = 0;
    }

    public PStack pop() {
        return previous;
    }

    public PStack push(int val) {
        return new PStack(this, val, this.size + 1);
    }

    public int peek() {
        return element;
    }

    public int getSize() {
        return size;
    }
}

public static void main(String args[]) {
    PStack pStack = new PStack();

    PStack s1 = pStack.push(1);
    PStack s2 = pStack.push(2);
    PStack s3 = pStack.push(3);
    PStack s4 = pStack.pop();

    System.out.println(pStack.getSize());
    System.out.println("====");
    System.out.println(s1.peek());
    System.out.println(s1.getSize());
    System.out.println("====");
    System.out.println(s2.peek());
    System.out.println(s2.getSize());
    System.out.println("====");
    System.out.println(s3.peek());
    System.out.println(s3.getSize());
    System.out.println("====");
    System.out.println(s4.peek());
```

```
        System.out.println(s4.getSize());
    }
}
```

//add and mul

```
import java.io.*;
import java.util.*;

public class Solution {
  //"( add 1 2 3 4 )"
  //"( mul ( mul 3 -2 5 ) -3 ( add 1 ( add 1 2 ) ) )"\
  class Pair {
    String s;
    int n;
    public Pair(String s, int n){
      this.s = s;
      this.n = n;
    }
  }
  public int parse (String exp) {
    Stack<Pair> op = new Stack<>();
    Stack<Pair> num = new Stack<>();
    String[] arr = exp.split(" ");
    int res = 0;
    int cnt = 0;
    for(int i = 0; i < arr.length; i++) {
      if(arr[i].equals("(")) {
        if(i != 0) {
          num.push(new Pair("" + res, cnt));
        }
        cnt++;
      }else if(arr[i].equals("add") || arr[i].equals("mul")){
        op.push(new Pair(arr[i], cnt));
        if(arr[i].equals("add")) {
          res = 0;
        }else {
          res = 1;
        }
      }else if(isNumeric(arr[i])) {
        if(op.peek().s.equals("add")) {
          res += Integer.parseInt(arr[i]);
        }else {
          res *= Integer.parseInt(arr[i]);
        }
      }else if(arr[i].equals(")")) {
        while(!num.isEmpty() && num.peek().n == cnt) {
          if(op.peek().n == cnt) {
```

```
            if(op.peek().s.equals("add")) {
              res += Integer.parseInt(num.pop().s);
            }else {
              res *= Integer.parseInt(num.pop().s);
            }
          }
        }
        op.pop();
        cnt--;
      }
    }
    return res;

  }
  public boolean isNumeric(String s) {
    return s != null && s.matches("[-+]?\\d*\\.?\\d+");
  }


  public static void main(String[] args){
    Solution s = new Solution();
    int test = s.parse("( add 1 2 3 4 )");
    System.out.println(test);

  }
}
```

//升级版本

Leetcode 736. Parse Lisp Expression

//futuretask 数组加和

```
public class Solution {

   static class SumArrayCallable implements Callable<Integer> {

       private int[] arr;

       public SumArrayCallable(int[] arr) {
           this.arr = arr;
       }

       @Override
       public Integer call() throws Exception {
           int sum = 0;
```

```
            for (int num : arr) {
                sum += num;
            }

            return sum;
        }
    }

    public static void main(String args[]) {

        int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};

        List<Callable<Integer>> callables = new ArrayList<>();

        int numOfThreads = 2;
        int partSize = (int)Math.ceil((double)arr.length / numOfThreads);
        int sum = 0;

        for (int i = 0; i < arr.length; i += partSize) {
            callables.add(new SumArrayCallable(Arrays.copyOfRange(arr, i, Math.min(i + partSize,
arr.length))));
        }

        ExecutorService executorService = Executors.newFixedThreadPool(numOfThreads);

        try {
            List<Future<Integer>> futures = executorService.invokeAll(callables);

            for (Future<Integer> future : futures) {
                sum += future.get();
            }

            executorService.shutdown();

            System.out.println(sum);
        } catch (Exception e) {
            System.out.println("Exception while calculating sum");
        }
    }
}
```

https://leetcode.com/problems/trapping-rain-water/description/

Given an array with different integers that simulates a terrain topology height.

Calculate how much water can you trap in the valleys in integer sum.

```
public class Solution {
    public int trap(int[] height) {
        if(height.length==0) return 0;
        int[] left=new int[height.length];
        int[] right=new int[height.length];
        left[0]=0;
        right[height.length-1]=0;
        int sum=0;
        for(int i=1;i<height.length;i++){
            left[i]=Math.max(height[i-1],left[i-1]);
        }

        for(int i=height.length-2;i>=0;i--){
            right[i]=Math.max(height[i+1],right[i+1]);
        }
        for(int i=1;i<height.length-1;i++){
            if((Math.min(left[i],right[i])-height[i])>0){
                sum=sum+Math.min(left[i],right[i])-height[i];
            }
        }
        return sum;
    }
}
```

//Recursive and iterative way to flatten a list, given [1, [2,3], [[[4]]]], return [1,2,3,4].

```
//Recursive
import java.io.*;
import java.util.*;

public class Solution {

  static class NestedInteger {
      List<NestedInteger> list;
      Integer i;

      public NestedInteger(Integer i) {
        this.list = null;
        this.i = i;
```

```
    }

    public NestedInteger(List<NestedInteger> list) {
      this.i = null;
      this.list = list;
    }

    /** @return true if this NestedInteger holds a single integer, rather than a nested list */
    public boolean isInteger() {
       if(i == null)
         return(false);

       return(true);
    }

    /** @return the single integer that this NestedInteger holds, if it holds a single integer
    * Return null if this NestedInteger holds a nested list */
    public Integer getInteger() {
      if(isInteger())
        return(i);

      return(null);
    }

    /** @return the nested list that this NestedInteger holds, if it holds a nested list
    * Return null if this NestedInteger holds a single integer */
    public List<NestedInteger> getList() {
      if(!isInteger())
        return(list);

      return(null);
    }
}

public List<Integer> depthSum(List<NestedInteger> nestedList) {
  List<Integer> res = new ArrayList<>();
  helper(nestedList, res);
  return res;
}

private void helper(List<NestedInteger> list, List<Integer> res)
```

```
  {
     for (NestedInteger e: list)
     {
       if(e.isInteger()) {
         res.add(e.getInteger());
       }else {
         helper(e.getList(),res);
       }
     }

  }
  public static void main(String[] args){
    Solution s = new Solution();
    NestedInteger ni1 = new NestedInteger(1);
    NestedInteger ni2 = new NestedInteger(2);
    List<NestedInteger> ones = new ArrayList<NestedInteger>();
    ones.add(ni1);
    ones.add(ni2);
    NestedInteger niOne = new NestedInteger(ones);

    List<NestedInteger> input = new ArrayList<NestedInteger>();
    input.add(ni2);
    input.add(ni1);
    input.add(niOne);
    List<Integer> test = s.depthSum(input);
    for(int i : test) {
      System.out.println(i);
    }
  }
}
```

//iterative

```
  public List<Integer> depthSum(List<NestedInteger> nestedList) {
    List<Integer> res = new ArrayList<>();
    Stack<List<NestedInteger>> stack = new Stack<>();
    for(NestedInteger e : nestedList) {
      if(e.isInteger()) {
        res.add(e.getInteger());
      }else {
```

```
        stack.push(e.getList());
        while(!stack.isEmpty()) {
          List<NestedInteger> cur = stack.pop();
          for(NestedInteger ele : cur) {
            if(ele.isInteger()) {
              res.add(ele.getInteger());
            }else {
              List<NestedInteger> next = ele.getList();
              stack.push(next);
            }
          }
        }
      }
    return res;
  }
```

//读写锁

```
1. public class ReadWriteLock{
2.
3.   private int readers       = 0;
4.   private int writers       = 0;
5.   private int writeRequests = 0;
6.
7.   public synchronized void lockRead() throws InterruptedException{
8.     while(writers > 0 || writeRequests > 0){
9.       wait();
10.    }
11.    readers++;
12.  }
13.
14.  public synchronized void unlockRead(){
15.    readers--;
16.    notifyAll();
17.  }
18.
```

```
19.   public synchronized void lockWrite() throws
    InterruptedException{
20.     writeRequests++;
21.
22.     while(readers > 0 || writers > 0){
23.       wait();
24.     }
25.     writeRequests--;
26.     writers++;
27.   }
28.
29.   public synchronized void unlockWrite() throws
    InterruptedException{
30.     writers--;
31.     notifyAll();
32.   }
33.}
```

//trie prefix and suffix, combine method, avoid repeated code