LC 224, 227, 772 Basic Calculator
LC 336. Palindrome Pair
LC 751. IP to CIDR
LC 68. Text Justification
LC 10. Regular Expression Matching
LC 755. Pour Water
LC 759. Employee Free Time
LC 773. Sliding Puzzle
LC 198, 213, 337 House Robber
LC 787. Cheapest Flights within K Stops
LC 756. Pyramid Transition Matrix

Bank System
http://rextester.com/CSBN84060

Collatz Conjecture
http://rextester.com/GOFNW25685

Implement queue with fixed size arrayList
http://rextester.com/URTSW41685

2D List Iterator
http://rextester.com/SWJH94490

Round Price:
http://rextester.com/GFTSYM67828

Travel Buddy:
http://rextester.com/JVYR94608

File System
http://rextester.com/PCH74934

CSV Parser:
http://rextester.com/MVP52948

Max number of night you can accommodate
http://rextester.com/PAFG22591

10 Wizards
http://rextester.com/VHCBD37760

Guess Number 1
http://rextester.com/CAGUYI28432

Display Page (Pagination)
http://rextester.com/BZRU72622

Find Median in Large File of Integers
http://rextester.com/WQVXAE31958

Sliding Game N puzzle
http://rextester.com/QAOU59023

Minimum Vertices to Traverse Directed Graph
http://rextester.com/YYFDDY29192

Find Case Combinations of a String
http://rextester.com/REOSQ76912

Menu Combination Sum
http://rextester.com/LRDD64894
http://rextester.com/WMF99079

Hilbert Curve
http://rextester.com/YBER46164

Finding Ocean
http://rextester.com/INFCL54630

Boggle Game
https://lintcode.com/submission/15759767/

K Edit Distance
https://lintcode.com/submission/15651597/

Preference List
http://rextester.com/WFY24661

```
//
//Bank System
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        BankSystem bs = new BankSystem();
        System.out.println(bs.withdraw(0, 100, 0));  // false
        bs.deposite(0, 100, 1);
        bs.deposite(1, 250, 2);
        bs.withdraw(0, 30, 3);
        System.out.println(bs.check(0, 0, 2)[0]);  // 0
        System.out.println(bs.check(0, 0, 2)[1]);  // 100
        bs.deposite(1, 5, 7);
        System.out.println(bs.check(1, 3, 9)[0]);  // 250
        System.out.println(bs.check(1, 3, 9)[1]);  // 255
    }
}

/**
 * 设计一个银行帐户系统，实现:
 * 存钱（帐户id，存钱数目，日期）
 * 取钱（帐户id，存钱数目，日期）
 * 查账（帐户id，起始日期，结束日期）： 只需要返回两个数值，一个是起始日期的balance，一个是结束日期的balance。
 * 描述就是这么多，剩下的自己发挥。钱的类型用integer，日期什么的自定义，我直接拿了integer
 */
class BankSystem {
  Map<Integer, Integer> accountBalance;    // id -> balance
  Map<Integer, Map<Long, Integer>> accountStatement; // id -> timestamp -> balance
  public BankSystem() {
    this.accountBalance = new HashMap<>();
    this.accountStatement = new HashMap<>();
  }

  public void deposite(int id, int amount, long timestamp) {
    if (!accountBalance.containsKey(id)) {
      accountBalance.put(id, 0);
      accountStatement.put(id, new HashMap<>());
    }
    accountBalance.put(id, accountBalance.get(id) + amount);
    accountStatement.get(id).put(timestamp, accountBalance.get(id));
  }

  public boolean withdraw(int id, int amount, long timestamp) {
    if (!accountBalance.containsKey(id) || accountBalance.get(id) < amount) {
      return false;
    }

    accountBalance.put(id, accountBalance.get(id) - amount);
    accountStatement.get(id).put(timestamp, accountBalance.get(id));
    return true;
  }

  public int[] check(int id, long startTime, long endTime) {
    if (!accountBalance.containsKey(id)) {
      return new int[0];
    }

    int[] res = new int[2];
    Map<Long, Integer> statement = accountStatement.get(id);
    List<Long> timestamps = new ArrayList<>(statement.keySet());
    Collections.sort(timestamps);

    if (statement.containsKey(startTime)) {
      res[0] = statement.get(startTime);
    } else {
      int index = -(Collections.binarySearch(timestamps, startTime) + 1);
      res[0] = index == 0 ? 0 : statement.get(timestamps.get(index - 1));
    }
    if (statement.containsKey(endTime)) {
      res[1] = statement.get(endTime);
    } else {
      int index = -(Collections.binarySearch(timestamps, endTime) + 1);
      res[1] = index == 0 ? 0 : statement.get(timestamps.get(index - 1));
    }

    return res;
  }
}

//
// Round Price
//
import java.util.*;
import java.lang.*;

class Node {
    double num;
    double diffToFloor;
    int index;
    public Node(double num, double diffToFloor, int index) {
        this.num = num;
        this.diffToFloor = diffToFloor;
        this.index = index;
    }
}

class Rextester
{
    public static void main(String args[])
    {
        double[] input = new double[100];
        StringBuilder sb = new StringBuilder();
        Random rand = new Random();
        for (int i = 0; i < 100; i++) {
            int num1 = rand.nextInt(100);
            double num2 = rand.nextDouble();
            double num = (double) num1 + num2;
            input[i] = num;
            sb.append(num).append(",");
        }
        System.out.println(sb.toString());
        sb.setLength(0);
        int[] result = roundUp(input);
        for (int i = 0; i < result.length; i++) {
            sb.append(result[i]).append(",");
        }
        System.out.println(sb.toString());
    }

    public static int[] roundUp(double[] input) {
        int n = input.length;
        double sum = 0;
        int sumFloor = 0;
        Node[] arr = new Node[n];
        for (int i = 0; i < n; i++) {
            sum += input[i];
            int floor = (int) input[i];
            sumFloor += floor;
            arr[i] = new Node(input[i], input[i] - floor, i);
        }
        Arrays.sort(arr, new Comparator<Node>(){
            public int compare(Node n1, Node n2) {
                return Double.compare(n2.diffToFloor, n1.diffToFloor);
            }
        });
        int sumRound = (int) Math.round(sum);
        int diff = sumRound - sumFloor;
        int[] result = new int[n];
        for (int i = 0; i < n; i++) {
            int index = arr[i].index;
            if (i < diff) {
                result[index] = (int) arr[i].num + 1;
            } else {
                result[index] = (int) arr[i].num;
            }
        }
        return result;
    }
}

//
// Sliding Game N puzzle
//
import java.util.*;
import java.lang.*;

class Rextester
```

```
{
    private static int[] dirX = {0, 0, 1, -1};
    private static int[] dirY = {1, -1, 0, 0};

    public static void main(String args[])
    {

        int[][] board = {{4,1,2},{5,0,3}};
        int[][] board2 = {{1,2,3},{4,5,0}};
        System.out.println(slidingGame(board, board2));

    }

    public static int slidingGame(int[][] board, int[][] board2) {
        if (board == null || board.length == 0 || board[0].length == 0) {
            return -1;
        }
        int n = board.length;
        int m = board[0].length;
        String start = "";
        String target = "";
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                start += board[i][j];
                target += board2[i][j];
            }
        }
        Set<String> set = new HashSet<>();
        Queue<String> queue = new LinkedList<>();
        /*推广到n宫格之后，难度在于String的index和board中坐标x,y的转换关系
          index = x * m + y;
          x = index / m;
          y = index % m;
        */
        set.add(start);
        queue.offer(start);
        int steps = -1;
        while (!queue.isEmpty()) {
            steps++;
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String crt = queue.poll();
                if (crt.equals(target)) {
                    return steps;
                }
                int index = crt.indexOf('0');
                int x = index / m;
                int y = index % m;
                for (int j = 0; j < 4; j++) {
                    int next_x = x + dirX[j];
                    int next_y = y + dirY[j];
                    if (!inBound(next_x, next_y, n, m)) {
```

```
                    continue;
                }
                String next = swap(crt, index, next_x * m + next_y);
                if (set.add(next)) {
                    queue.offer(next);
                }
            }
        }
    }
    return -1;
}

private static String swap(String s, int i, int j) {
    char[] arr = s.toCharArray();
    char temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    return new String(arr);
}

private static boolean inBound(int x, int y, int n, int m) {
    return x >= 0 && x < n && y >= 0 && y < m;
}
}

//
//Collatz Conjecture
//
import java.util.*;
import java.lang.*;

class Rextester{

    public static void main(String args[])
    {
        System.out.println(findLongestSteps(1));
    }

    private static int findSteps(int num, Map<Integer, Integer> map) {
        if (num <= 1) {
            return 1;
        }
        if (map.containsKey(num)) {
            return map.get(num);
        }
        int result = 0;
        if (num % 2 == 0) {
            result =  findSteps(num / 2, map) + 1;
        } else {
            result = findSteps(3 * num + 1, map) + 1;
        }
        map.put(num, result);
        return result;
    }
    // recursion solution
    private static int findLongestSteps(int limit) {
        if (limit < 1) {
            return 0;
        }
        Map<Integer, Integer> map = new HashMap<>();
        int longest = 0;
        for (int i = 1; i <= limit; i++) {
            longest = Math.max(longest, findSteps(i, map));
        }
        return longest;
    }
     //iteration solution
     private static int findLongestSteps2(int limit) {
        int longest = 0;
        int[] memo = new int[limit + 1];
        Arrays.fill(memo, -1);
        memo[1] = 1;
        for (int i = 1; i <= limit; i++) {
            int num = i;
            int count = 0;
            while (num >= limit || memo[num] == -1) {
                count++;
                if (num % 2 == 0) {
                    num /= 2;
                } else {
                    num = num * 3 + 1;
                }
            }
            int result = count + memo[num];
            memo[i] = result;
            longest = Math.max(longest, result);
        }
        return longest;
    }

}

//
//CSV Parser
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        System.out.println("c:\\windows\\system32");
        String s = "\"Alexandra \"\"Alex\"\"\",Menendez,alex.menendez@gmail.com,Miami,1
```

```
\"\"\"Alexandra Alex\"\"\"";
        System.out.println(parseCSV(s));
    }

    //"Alexandra ""Alex""",Menendez,alex.menendez@gmail.com,Miami,1 """Alexandra Alex"""
    public static String parseCSV(String str) {
        List<String> list = new ArrayList<>();
        StringBuilder sb = new StringBuilder();
        boolean inQuote = false;
        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (inQuote) {
                if (c == '\"') {
                    if (i + 1 < str.length() && str.charAt(i + 1) == '\"') {
                        sb.append('\"');
                        i++;
                    } else {
                        inQuote = false;
                    }
                } else {
                    sb.append(c);
                }
            } else {
                if (c == '\"') {
                    inQuote = true;
                } else if (c == ',') {
                    list.add(sb.toString());
                    sb.setLength(0);
                } else {
                    sb.append(c);
                }
            }
        }
        if (sb.length() > 0) {
            list.add(sb.toString());
        }
        return String.join("|", list);
    }
}

//
//Max number of night you can accommodate
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        int[] nums = {4, 10, 3, 1, 5};
        System.out.println(maxSum(nums));
    }

    private static int maxSum(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        if (nums.length == 1) {
            return nums[0];
        }
        int n = nums.length;
        int[] dp = new int[n + 1];
        dp[0] = 0;
        dp[1] = nums[0];
        for (int i = 2; i <= n; i++) {
            dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i - 1]);
        }
        return dp[n];
    }
}

//
//10 Wizards
//
import java.util.*;
import java.lang.*;

class Route {
    int wizard;
    int fromWizard;
    int cost;
    public Route(int wizard, int fromWizard, int cost) {
        this.wizard = wizard;
        this.fromWizard = fromWizard;
        this.cost = cost;
    }
}

class Rextester
{
    public static void main(String args[])
    {
        List<List<Integer>> wizards = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            List<Integer> list = new ArrayList<>();
            if (i == 0) {
                list.add(1);
                list.add(2);
            } else if (i == 1) {
                list.add(3);
            } else if (i == 2) {
                list.add(3);
                list.add(4);
            } else if (i == 3) {
                list.add(4);
            }
            wizards.add(list);
        }
        List<Integer> path = getShortestPath(wizards, 0, 4);
        for (int i = 0; i < path.size(); i++) {
            System.out.println(path.get(i));
        }
    }

    public static List<Integer> getShortestPath(List<List<Integer>> wizards, int source, int
target) {
        List<Integer> path = new ArrayList<>();
        if (wizards == null || wizards.size() == 0) {
            return path;
        }
        int n = wizards.size();
        Route[] from = new Route[n];
        PriorityQueue<Route> pq = new PriorityQueue<Route>(new Comparator<Route>(){
            public int compare(Route r1, Route r2) {
                return r1.cost - r2.cost;
            }
        });
        List<Integer> nextWizards = wizards.get(source);
        for (int next: nextWizards) {
            pq.offer(new Route(next, source, (next - source) * (next - source)));
        }
        while (!pq.isEmpty()) {
            Route route = pq.poll();
            if (from[route.wizard] != null) {
                continue;
            }
            from[route.wizard] = route;
            if (route.wizard == target) {
                getPath(from, source, target, path);
                return path;
            }
            for (int next : wizards.get(route.wizard)) {
                pq.offer(new Route(next, route.wizard, (route.wizard - next) * (route.wizard
- next)));
            }
        }
        return path;
    }

    private static void getPath(Route[] from, int source, int target, List<Integer> path) {
        int wizard = target;
        while (wizard != source) {
            path.add(wizard);
            wizard = from[wizard].fromWizard;
        }
        path.add(source);
        Collections.reverse(path);
    }
}

//
//Guess Number 1
//
import java.util.*;
import java.lang.*;

class Rextester
{
    private static String target = "3536";
    public static void main(String args[])
    {
        System.out.println(guess());
    }

    //这个版本只能handle每次server只返回一位数字，表明guess中有几位存在于target中，因此最后只能求出target中的数字的组合，并不能知道排列
    //如果需要Permutation的话，需要返回2位数字，类似于Bulls and Cows
    private static int guessServer(String guess) {
        int res = 0;
        Map<Character, Integer> targetMap = new HashMap<>();
        for (char c : target.toCharArray())
            targetMap.put(c, targetMap.getOrDefault(c, 0) + 1);
        Map<Character, Integer> guessMap = new HashMap<>();
        for (char c : guess.toCharArray())
            guessMap.put(c, guessMap.getOrDefault(c, 0) + 1);
        for (char k : guessMap.keySet()) {
            if (targetMap.containsKey(k)) {
                res += Math.min(guessMap.get(k), targetMap.get(k));
            }
        }
        return res;
    }

    private static String genNumber(List<Integer> guessed, int c) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < guessed.size(); i++) {
            sb.append(guessed.get(i));
        }
        for (int i = guessed.size(); i < 4; i++) {
            sb.append(c);
        }
        return sb.toString();
    }

    private static String genNumber(List<Integer> guessed) {
        if (guessed == null || guessed.size() == 0) return "";
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < guessed.size(); i++) {
            sb.append(guessed.get(i));
        }
        return sb.toString();
    }

    public static String guess() {
        List<Integer> res = new ArrayList<>();
        List<Integer> cands = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
         System.out.println("\nstart to guess " + target + " ...");
         System.out.println("res: " + res);
         System.out.println("candList: " + cands);
        int counter = 0;
        Iterator<Integer> iter = cands.iterator();
        while (iter.hasNext() && res.size() < 4) {
            int cand = iter.next();
            counter++;
            int guessedCount = res.size();
            String guessCand = genNumber(res, cand);
            int guessRes = guessServer(guessCand);
             System.out.println("cand: " + cand);
             System.out.println("guessRes: " + guessRes);
            if (guessRes == guessedCount) {
                iter.remove();
            } else if (guessRes > guessedCount) {
                for (int i=guessedCount; i< guessRes; i++) {
                    res.add(cand);
                }
                iter.remove();
            } else {
                // something wrong here
                return genNumber(res);
            }
        }
        //System.out.println(res.size());
        if (res.size() < 4) {
            for (int i = res.size(); i < 4; i++) {
                res.add(6);
            }
        }
        // System.out.println("guessed " + counter + " times");
        return genNumber(res);
    }
}

//
//Guess Number II
//

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

```
class Rextester
{

    private static String target = "4361";
    public static void main(String args[])
    {
        System.out.println(client());

    }

    public static int check(String guess) {
        int count = 0;
        for (int i = 0; i < 4; i++) {
            if (target.charAt(i) == guess.charAt(i)) {
                count++;
            }
        }
        return count;
    }

    private static String client(){
        char[] result = new char[4];
        Arrays.fill(result, '0');
        String base = "1111";
        System.out.println("Server call: " + base);
        int baseResult = check(base);
        if (baseResult == 4) {
            return base;
        }
        for (int i = 0; i < 4; i++) {
            for (int j = 2; j < 6; j++) {
                String newS = replace(base, i, (char) (j + '0'));
                System.out.println("Server call: " + newS);
                int newResult = check(newS);
                if (newResult != baseResult) {
                    result[i] = baseResult > newResult ? '1' : (char) (j + '0');
                    break;
                }
            }
            if (result[i] == '0') {
                result[i] = '6';
            }
        }
        return new String(result);
    }

    private static String replace(String s, int index, char c) {
        char[] arr = s.toCharArray();
        arr[index] = c;
        return new String(arr);
    }
}
```

```
//
//Display Page (Pagination)
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        List<String> input = new ArrayList<>();
        //input.add("1,28,310.6,SF");
        //input.add("4,5,204.1,SF");
        //input.add("20,7,203.2,Oakland");
        //input.add("6,8,202.2,SF");
        //input.add("6,10,199.1,SF");
        //input.add("1,16,190.4,SF");
        input.add("6,29,185.2,SF");
        //input.add("7,20,180.1,SF");
        input.add("6,21,162.1,SF");
        //input.add("2,18,161.2,SF");
        //input.add("2,30,149.1,SF");
        //input.add("3,76,146.2,SF");
        input.add("2,14,141.1,San Jose");

        List<String> result = displayPages(input, 5);
        for (int i = 0; i < result.size(); i++) {
            System.out.println(result.get(i));
        }

    }

     //用MaxHeap实现的版本，可以做到每个page内部都是排序
    //类似K路归并的思想
    //以上说法不对，用maxHeap的话会优先point，导致本来可以不重复hostId的情况也去重复hostId了
    //用queue就好，相同hostId的出队列之后，它的后续节点直接被扔到队尾了
    public static List<String> displayPages2(List<String> input, int pageSize) {
        List<String> result = new ArrayList<>();
        //把hostID相同的归类到一个List里面
        Map<String, List<String>> map = new HashMap<>();
        for (String record : input) {
            String hostId = record.split(",")[0];
            if (!map.containsKey(hostId)) {
                map.put(hostId, new ArrayList<>());
            }
            map.get(hostId).add(record);
        }/*
        PriorityQueue<Node> maxHeap = new PriorityQueue<Node>(new Comparator<Node>(){
            //maxHeap根据point排序
            public int compare (Node n1, Node n2) {
                return Double.compare(Double.parseDouble(n2.s.split(",")[2]), 
Double.parseDouble(n1.s.split(",")[2]));
            }
        });
        */
        Queue<Node> queue = new LinkedList<>();
        for (Map.Entry<String, List<String>> entry : map.entrySet()) {
            String hostId = entry.getKey();
            List<String> list = entry.getValue();
            queue.offer(new Node(hostId, 0, list.get(0)));
        }
        int total = input.size();
        int count = 0;
        List<Node> temp = new ArrayList<>();
        while (!queue.isEmpty()) {

            Node head = queue.poll();
            temp.add(head);
            count++;
            total--;
            if (head.j + 1 != map.get(head.hostId).size()) {
                queue.offer(new Node(head.hostId, head.j + 1, 
map.get(head.hostId).get(head.j + 1)));
            }
            if (count == pageSize) {
                copy(temp, result);
                count = 0;
            }
        }
        if (!temp.isEmpty()) {
            copy(temp, result);
        }
        return result;
    }

    private static void copy(List<Node> temp, List<String> result) {
        Collections.sort(temp, new Comparator<Node>(){
                    public int compare (Node n1, Node n2) {
                        return Double.compare(Double.parseDouble(n2.s.split(",")[2]), 
Double.parseDouble(n1.s.split(",")[2]));
                    }
                });
        for (Node node : temp) {
            result.add(node.s);
        }
        result.add("");
        temp.clear();
    }


    //O(n)时间复杂度的solution
    public static List<String> displayPages(List<String> input, int pageSize) {
        List<String> result = new ArrayList<>();
        Map<String, Set<String>> map = new HashMap<>();
        for (String record : input) {
            String hostId = record.split(",")[0];
            if (!map.containsKey(hostId)) {
                map.put(hostId, new LinkedHashSet<>());
            }
            map.get(hostId).add(record);
        }
        Set<String> set = new LinkedHashSet<>();
        for (String record : input) {
            set.add(record);
        }
        int count = 0;
        while (!set.isEmpty()) {
            //第一页不加分页符“”，其他页都要加
            if (result.size() != 0) {
                result.add("");
            }
            //先从分号类的里面去加
            for (Map.Entry<String, Set<String>> entry : map.entrySet()) {
                if (entry.getValue().size() == 0) continue;
                if (count < pageSize) {
                    String record = entry.getValue().iterator().next();
                    result.add(record);
                    count++;
                    sync(set, map, record);
                } else {
                    break;
                }
            }
            Iterator<String> it = set.iterator();
            while (count < pageSize && it.hasNext()) {
                String record = it.next();
                result.add(record);
                sync(set, map, record);
                count++;
            }
            count = 0;
        }
        return result;
    }

    private static void sync(Set<String> set, Map<String, Set<String>> map, String del) {
        String hostId = del.split(",")[0];
        set.remove(del);
        map.get(hostId).remove(del);
    }
}

//
```

```
//Find Median in Large File of Integers
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        int[] nums = new int[100];
        Random rand = new Random();
        for (int i = 0; i < 100; i++) {
            int num = rand.nextInt(100);
            nums[i] = num;
        }
        Arrays.sort(nums);
        for (int i = 0; i < 100; i++) {
            System.out.print(nums[i] + ",");
        }
        System.out.println(nums[49]);
        System.out.println(nums[50]);
        System.out.println(findMedian(nums));
    }

    private static long findKth(int[] nums, int k, long left, long right) {
        if (left >= right) { return left;
        }
        long largestLessThanGuess = left;
        long guess = left + (right - left) / 2;
        int count = 0;
        for (int num : nums) {
        if (num <= guess) {
            count++;
            largestLessThanGuess = Math.max(largestLessThanGuess, num); }
        }
        if (count == k) {
            return largestLessThanGuess;
        } else if (count < k) {
            //如果下面用left - largestLessThanGuess的范围的话，这里必须用guess + 1(注意+1） -
right
            //否则两边都用guess: left - guess  guess - right
            return findKth(nums, k, guess + 1, right);
        } else {
            return findKth(nums, k, left, largestLessThanGuess);
        }
}


    private static double findMedian(int[] nums) {
        int len = 0;
        for (int i = 0; i < nums.length; i++) {
            len++;
        }
        if (len % 2 == 0) {
            return (double) (findKth(nums, len / 2, Integer.MIN_VALUE, Integer.MAX_VALUE) +
findKth(nums, len / 2 + 1, Integer.MIN_VALUE, Integer.MAX_VALUE)) / 2;
        }
        return (double) findKth(nums, len / 2 + 1, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }
}


//
//Minimum Vertices to Traverse Directed Graph
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        int[][] edges = {{2,9},{3,3},{3,5},{3,7},{4,8},{5,8},{6,6},{7,4},{8,7},{9,3},{9,6}};
        List<Integer> result = getMin(edges, 10);
        for (int num : result) {
            System.out.println(num);
        }
    }

    public static List<Integer> getMin(int[][] edges, int n) {
        List<Integer> result = new ArrayList<>();
        boolean[] visited = new boolean[n];
        Map<Integer, List<Integer>> map = new HashMap<>();
        int[] inDegree = new int[n];
        for (int[] edge : edges) {
            if (!map.containsKey(edge[0])) {
                map.put(edge[0], new ArrayList<>());
            }
            map.get(edge[0]).add(edge[1]);
            inDegree[edge[1]]++;
        }
        for (int i = 0; i < n; i++) {
            if (inDegree[i] == 0) {
                result.add(i);
                dfs(i, map, visited);
            }
        }
        //如果用HashSet去重的话，本循环要注意concurrent modification error，边循环边修改，
        //会不会用HashSet的iterator安全一点
        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                result.add(i);
                dfs(i, map, visited);
            }
        }
        return result;
    }

    private static void dfs(int crt, Map<Integer, List<Integer>> map, boolean[] visited) {
        visited[crt] = true;
        if (map.containsKey(crt)) {
            for (int next : map.get(crt)) {
                if (visited[next]) {
                    continue;
                }
                dfs(next, map, visited);
            }
        }
    }
}

//
// Boggle Game
//
class TrieNode {
    TrieNode[] map;
    String word;
    public TrieNode() {
        this.map = new TrieNode[26];
    }
}

public class Solution {
    /*
     * @param board: a list of lists of character
     * @param words: a list of string
     * @return: an integer
     */
    private int n;
    private int m;
    private final int[] dirX = {0, 0, 1, -1};
    private final int[] dirY = {1, -1, 0, 0};
    private TrieNode root;
    private int max = 0;

    public int boggleGame(char[][] board, String[] words) {
        // write your code here
        if (board == null || board.length == 0 || board[0].length == 0 || words == null ||
words.length == 0) {
            return this.max;
        }
        this.n = board.length;
        this.m = board[0].length;
        this.root = new TrieNode();
        for (String word : words) {
            insert(word);
        }
        for (int i = 0; i < this.n; i++) {
            for (int j = 0; j < this.m; j++) {
                search(board, i, j, this.root, new ArrayList<String>(), new
boolean[this.n][this.m]);
            }
        }
        return this.max;
    }

    private void search(char[][] board, int x, int y, TrieNode node, List<String> list,
boolean[][] visited) {
        if (!inBound(x, y) || visited[x][y] || node.map[board[x][y] - 'a'] == null) {
            return;
        }
        node = node.map[board[x][y] - 'a'];
        visited[x][y] = true;
        if (node.word != null) {
            list.add(node.word);
            //每次加入新的word之后max可能会变化，需要check一下
            this.max = Math.max(this.max, list.size());
            //因为当前词已经加入进来了，所有要重新开始，从别的没有被marked的点开始找新的词
            for (int i = 0; i < this.n; i++) {
                for (int j = 0; j < this.m; j++) {
                    search(board, i, j, this.root, list, visited);
                }
            }
            //回溯之前把当前词扔掉，回到上一步找别的方向继续
            list.remove(list.size() - 1);
            //这里是不用mark有点triky，拿example作为例子，如果abc有词了，那么所有以a或者ab为
prefix的搜索都mark c为visited，因为别的词肯定比abc长，因为
            //绕过别的字母了，会造成总数更少，如果不mark的话会有超时的风险，比如有一个测试例是aaa
aa的那个。
            //visited[x][y] = false;
            return;
        }
        for (int i = 0; i < 4; i++) {
            int next_x = x + this.dirX[i];
            int next_y = y + this.dirY[i];
            search(board, next_x, next_y, node, list, visited);
        }
        visited[x][y] = false;
    }

    private boolean inBound(int x, int y) {
        return x >= 0 && x < this.n && y >= 0 && y < this.m;
    }

    private void insert(String word) {
        TrieNode node = this.root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (node.map[c - 'a'] == null) {
                node.map[c - 'a'] = new TrieNode();
            }
            node = node.map[c - 'a'];
        }
        node.word = word;
    }
}

//
// K Edit Distance
//

class TrieNode {
    TrieNode[] map;
    String word;
    public TrieNode() {
        this.map = new TrieNode[26];
    }
}

public class Solution {
    /**
     * @param words: a set of stirngs
     * @param target: a target string
     * @param k: An integer
     * @return: output all the strings that meet the requirements
     */
    private TrieNode root;
    private String target;
    private int k;
    private int n;
    public List<String> kDistance(String[] words, String target, int k) {
        // write your code here
        List<String> result = new ArrayList<>();
        if (words == null || words.length == 0) {
            return result;
        }
        this.root = new TrieNode();
        this.target = target;
        this.k = k;
        for (int i = 0; i < words.length; i++) {
            insert(words[i]);
        }
        this.n = target.length();
        //initially, this dp array is for empty string ""
        //dp represents how many edit distance for empty string "" to form first i element
of target
        int[] dp = new int[n + 1];
        for (int i = 0; i <= n; i++) {
            dp[i] = i;
        }
        search(root, dp, result);
        return result;
    }

    private void insert(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (node.map[c - 'a'] == null) {
                node.map[c - 'a'] = new TrieNode();
            }
            node = node.map[c - 'a'];
        }
        node.word = word;
    }

    private void search(TrieNode node, int[] prevDp, List<String> result) {
        if (node.word != null && prevDp[n] <= k) {
            result.add(node.word);
        }
        for (int i = 0; i < 26; i++) {
            if (node.map[i] == null) {
                continue;
            }
            int[] dp = new int[n + 1];
            //dp[0]是构建空字符串，如果之前的dp构建需要prevDp[0]步的话，现在多了一个字符，所以要删
除多出的这个字符才能构建出空字符串，所以需要+1步
            dp[0] = prevDp[0] + 1;
            for (int j = 1; j <= n; j++) {
                //意思是target里的第j个字符等于当前对应的字符
                if (target.charAt(j - 1) - 'a' == i) {
                    dp[j] = prevDp[j - 1];
                } else {
                    /* 如果dp[i][j]表示当前trie树所构成的prefix字符串的前i个字符和target字符串的
前j个字符的编辑距离的话，按照以前的做法，如果当前第i个和第j个字符不相同的话，则有目前的对应关系：
                        dp[i - 1][j - 1] + 1 replace   => prevDp[j - 1] + 1
                        dp[i][j - 1] + 1 insert            => dp[j - 1] + 1
                        dp[i - 1][j] + 1 delete           => prevDp[j] + 1

                    */
                    dp[j] = Math.min(prevDp[j - 1], Math.min(prevDp[j], dp[j - 1])) + 1;
                }
            }
            search (node.map[i], dp, result);
        }
    }
}

//
// Find Case Combinations of a String
//
```

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

class Rextester
{
    public static void main(String args[])
    {
       String input = "aBc";
       List<String> result = caseCombination2(input);
        for (String word : result) {
            System.out.println(word);
        }
    }

    // Bit Manipulation
    private static List<String> caseCombination(String input) {
        List<String> result = new ArrayList<>();
        char[] arr = input.toCharArray();
        int n = input.length();
        for (int i = 0; i < (1 << n); i++) {
            StringBuilder sb = new StringBuilder();
            for (int j = 0; j < n; j++) {
                int bit = (i >> j) & 1;
                sb.append(bit == 1 ? Character.toUpperCase(arr[j]) :
Character.toLowerCase(arr[j]));
            }
            result.add(sb.toString());
        }
        return result;
    }

    // DFS
    private static List<String> caseCombination2(String input) {
        List<String> result = new ArrayList<>();
        dfs(input, 0, new StringBuilder(), result);
        return result;
    }

    private static void dfs(String input, int i, StringBuilder sb, List<String> result) {
        if (i == input.length()) {
            result.add(new String(sb.toString()));
            return;
        }
        //不需要for循环和startIndex
        char c = input.charAt(i);
        sb.append(Character.toUpperCase(c));
        dfs(input, i + 1, sb, result);
        sb.deleteCharAt(sb.length() - 1);
        sb.append(Character.toLowerCase(c));
        dfs(input, i + 1, sb, result);
        sb.deleteCharAt(sb.length() - 1);
    }
}

//
// Menu Combination Sum
//
import java.util.*;
import java.lang.*;

class Rextester
{
    private static final double eps = 1.0E-1;
    public static void main(String args[])
    {
        double[] prices = {2.40, 0.01, 6.00, 2.58};
        List<List<Double>> result = getCombos(prices, 2.50);
        for (int i = 0; i < result.size(); i++) {
            System.out.println(i + "th result:");
            for (int j = 0; j < result.get(i).size(); j++) {
                System.out.println(result.get(i).get(j));
            }
        }
    }

    private static List<List<Double>> getCombos(double[] prices, double target) {
        List<List<Double>> result = new ArrayList<>();
        if (prices == null || prices.length == 0 ||  target <= 0) {
            return result;
        }
        Arrays.sort(prices);
        dfs(prices, 0, target, new ArrayList<>(), result);
        return result;
    }

    private static void dfs(double[] prices, int startIndex, double target, List<Double>
combo, List<List<Double>> result) {
        if (Math.abs(target) < eps) {
            result.add(new ArrayList<>(combo));
            return;
        }
        for (int i = startIndex; i < prices.length; i++) {
            if (i != startIndex && prices[i] == prices[i - 1]) {
                continue;
            }
            if (prices[i] - target > eps) {
                break;
            }
            combo.add(prices[i]);
            dfs(prices, i + 1, target - prices[i], combo, result);
            combo.remove(combo.size() - 1);
        }
    }
}
```

```
//
// Hilbert Curve
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        System.out.println(hilbertCurve(1, 1, 2));
        System.out.println(hilbertCurve(0, 1, 1));
        System.out.println(hilbertCurve(2, 2, 2));
    }
    /*
     顺带提及的知识点：
     如果坐标需要做rotation和reflection，所需要相乘的坐标都是以下：{{1, 1}, {1, -1}, {-1, 1}, {-
1, -1}};
     如果是reflection的话，原坐标不变，(x, y) => (x, y)然后乘以以上
     如果是rotation的话，原坐标需要互换，(x, y) => (y, x)然后乘以以上
    */
    private static int hilbertCurve(int x, int y, int iter) {
        if (iter == 0) {
            return 1;
        }
        //count代表一个板块包含了多少步
        int count = 1 << (2 * (iter - 1));
        //len代表了一个板块的边长是多少
        int len = 1 << (iter - 1);

        if (x >= len && y >= len) {
            //右上角板块，从起点板块需要跨过两个板块到达
            return 2 * count + hilbertCurve(x - len, y - len, iter - 1);
        } else if (x < len && y >= len) {
            //左上角板块，从起点板块需要跨过一个板块到达
            return count + hilbertCurve(x, y - len, iter - 1);
        } else if (x < len && y < len) {
            //左下角起点板块，不需要跨过板块了，只需要rotation一下
            return hilbertCurve(y, x, iter - 1);
        } else {
            //右下角板块，从起点需要跨过三个板块到达，同时还需要rotation
            //用y = -x 对称，然后移动会起点
            //做法是（x, y) => (-y, -x) => 向右移动len - 1，向上移动2* len - 1来移动回应该在的范
围，其实就是rotation
            return 3 * count + hilbertCurve(len - 1 - y, 2 * len - 1 - x, iter - 1);
        }
    }
}

//
```

```
// Finding Ocean
//
import java.util.*;
import java.lang.*;

class Point {
    int x;
    int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class Rextester
{
    private final static int[] dirX = {0, 0, 1, -1};
    private final static int[] dirY = {1, -1, 0, 0};
    private static int n;
    private static int m;

    public static void main(String args[])
    {
        char[][] graph = {
            {'W','W','W','L','L','L','W'},
            {'W','W','L','L','L','W','W'},
            {'W','L','L','L','L','W','W'},
        };
        markOcean(graph, 0, 1, 'W', 'O');
        for (int i = 0; i < graph.length; i++) {
            System.out.println();
            for (int j = 0; j < graph[0].length; j++) {
                System.out.print(graph[i][j]);
                System.out.print(",");
            }
        }
    }

    private static void markOcean(char[][] graph, int _x, int _y, char water, char ocean) {
        if (graph == null || graph.length == 0 || graph[0].length == 0) {
            return;
        }
        n = graph.length;
        m = graph[0].length;
        Queue<Point> queue = new LinkedList<>();
        queue.add(new Point(_x, _y));
        graph[_x][_y] = ocean;
        while (!queue.isEmpty()) {
            Point curt = queue.poll();
            for (int i = 0; i < 4; i++) {
                int next_x = curt.x + dirX[i];
                int next_y = curt.y + dirY[i];
                if (!inBound(next_x, next_y) || graph[next_x][next_y] != water) {
                    continue;
                }
                graph[next_x][next_y] = ocean;
                queue.offer(new Point(next_x, next_y));
            }
        }
    }

    private static boolean inBound(int x, int y) {
        return x >= 0 && x < n && y >= 0 && y < m;
    }
}

//
//Implement queue with fixed size arrayList
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        QueueWithFixedArray queue = new QueueWithFixedArray(5);
        System.out.println(queue.poll());//null
        queue.offer(1);
        queue.offer(1);
        queue.offer(2);
        queue.offer(3);

        queue.offer(4);
        queue.offer(5);
        queue.offer(6);
        System.out.println(queue.poll());//1
        System.out.println(queue.poll());//1
        System.out.println(queue.poll());//2
        System.out.println(queue.poll());//3
        System.out.println(queue.poll());//4
        queue.offer(7);
        System.out.println("size: " + queue.size());//size:3
        System.out.println(queue.poll());//5
        System.out.println(queue.poll());//6
        System.out.println(queue.poll());//7
        System.out.println(queue.poll());//null
    }
}

class QueueWithFixedArray {
    private int fixedSize;
    private int count;
    private int head;
    private int tail;
    List<Object> headList;
    List<Object> tailList;

    public QueueWithFixedArray(int size) {
        this.fixedSize = size;
        this.count = 0;
        this.head = 0;
        this.tail = 0;
        this.headList = new ArrayList<Object>();
        this.tailList = this.headList;
    }

    public void offer(int num) {
        //到了末尾就先处理一下再继续
        if (tail == fixedSize - 1) {
            List<Object> newList = new ArrayList<>();
            tailList.add(newList);
            tailList = newList;
            tail = 0;
        }
        tailList.add(num);
        tail++;
        count++;
    }

    public Integer poll() {
        if (count == 0) {
            return null;
        }
        //到了末尾就先处理一下再继续
        if (head == fixedSize - 1) {
            //注意cast
            headList = (List<Object>) headList.get(head);
            head = 0;
        }
        count--;
        //注意需要cast
        return (int) headList.get(head++);
    }

    public int size() {
        return this.count;
    }
}

//
//2D List Iterator
//
import java.util.*;
import java.lang.*;
```

```
class Rextester
{
    public static void main(String args[])
    {
        List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2));
        List<Integer> list2 = new ArrayList<>(Arrays.asList(3));
        List<List<Integer>> vec2d = new ArrayList<>();
        vec2d.add(list1);
        vec2d.add(list2);
        //1,2
        //3
        MyIterator myIter = new MyIterator(vec2d);
        System.out.println(myIter.hasNext());
        System.out.println(myIter.next());
        System.out.println(myIter.hasNext());
        myIter.remove();
        System.out.println(myIter.next());
        for (int i = 0; i < vec2d.size(); i++) {
            if (vec2d.get(i) == null) continue;
            for (int j = 0; j < vec2d.get(i).size(); j++) {
                System.out.println("List" + i + ":" + vec2d.get(i).get(j));
            }

        }
        System.out.println(myIter.hasNext());
        System.out.println(myIter.next());
    }
}

class MyIterator implements Iterator<Integer> {
    private Iterator<List<Integer>> i;
    private Iterator<Integer> j;

    public MyIterator(List<List<Integer>> vec2d) {
        this.i = vec2d.iterator();
        this.j = null;
    }

    public boolean hasNext() {
        //注意这里是while不是if
        while ((j == null || !j.hasNext()) && i.hasNext()) {
            //如果有需要，比如List里面含有null元素，则需要特殊处理一下handle有null的情况
            //否则下面j = i.next().iterartor()就会null pointer
           j  = i.next().iterator();
        }
        return j != null && j.hasNext();
    }

    public Integer next() {
        if (!hasNext()) {
            return null;
        }
        return j.next();
    }

    public void remove() {
        //注意这里!j.hasNext()无所谓，只要判断j不是null就行
        while (j == null && i.hasNext()) {
            j  = i.next().iterator();
        }
        if (j != null) {
            j.remove();
        }
    }
}

//
//Travel Buddy
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        Set<String> myList = new HashSet<>(Arrays.asList("A", "B", "C", "D"));

        Set<String> peter = new HashSet<>(Arrays.asList("A", "B", "E", "F"));
        Set<String> john = new HashSet<>(Arrays.asList("A", "B", "D", "G"));
        Set<String> casy = new HashSet<>(Arrays.asList("X", "B", "A", "D", "Q"));
        Set<String> jason = new HashSet<>(Arrays.asList("A", "B", "C", "D", "P", "Q"));
        Set<String> ken = new HashSet<>(Arrays.asList("A", "X", "Y", "Z"));

        Map<String, Set<String>> friendLists = new HashMap<>();
        friendLists.put("peter", peter);
        friendLists.put("john", john);
        friendLists.put("casy", casy);
        friendLists.put("jason", jason);
        friendLists.put("ken", ken);


        Solution solution = new Solution(myList, friendLists);
        List<Buddy> buddies = solution.findBuddies();
        for (int i = 0; i < buddies.size(); i++) {
            Buddy b = buddies.get(i);
            System.out.println("Name: " + b.name + " sim: " + b.sim);
        }

        Set<String> cities = solution.recommend(5);
        for (String city : cities) {
            System.out.println(city);
        }
    }
```

```
}

class Solution {

    private List<Buddy> buddies;
    private Set<String> myList;
    private Map<String, Set<String>> friendLists;

    public Solution(Set<String> myList, Map<String, Set<String>> friendLists) {
        this.myList = myList;
        this.friendLists = friendLists;
    }

    public List<Buddy> findBuddies() {
        List<Buddy> buddies = new ArrayList<>();
        for (Map.Entry<String, Set<String>> entry : friendLists.entrySet()) {
            Set<String> common = new HashSet<>(myList);
            String name = entry.getKey();
            Set<String> wishList = entry.getValue();
            common.retainAll(wishList);
            if (common.size() >= myList.size() / 2) {
                buddies.add(new Buddy(name, common.size(), wishList));
            }
        }
        Collections.sort(buddies, new Comparator<Buddy>(){
            public int compare(Buddy b1, Buddy b2) {
                return b2.sim - b1.sim;
            }
        });
        this.buddies = buddies;
        return buddies;
    }

    public Set<String> recommend(int k){
        int count = 0;
        Set<String> result = new LinkedHashSet<>();
        for (Buddy buddy : buddies) {
            //注意是从buddy的list中remove我的list，而不是从我的list中remove他们的
            Set<String> diff = new HashSet<>(buddy.list);
            diff.removeAll(myList);
            for (String city : diff) {
                if (count < k) {
                    if (result.add(city)) {
                        count++;
                    }
                } else {
                    return result;
                }
            }
        }
        return result;
    }
```

```
}
//Buddy类写Solution类外面，要不然访问不到
class Buddy {
        String name;
        int sim;
        Set<String> list;
        public Buddy(String name, int sim, Set<String> list) {
            this.name = name;
            this.sim = sim;
            this.list = list;
        }
}

//
//File System
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        Solution solution = new Solution();
        solution.create("/a", 1);
        System.out.println(solution.get("/a"));
        solution.create("/a/b", 2);
        System.out.println(solution.get("/a/b"));
        solution.create("/c/d", 3);
        System.out.println(solution.get("/c"));
        solution.set("/a/b", 4);
        System.out.println(solution.get("/a/b"));
        solution.watch("/a", "/a call back triggerred");
        solution.watch("/a/b", "/a/b call back triggerred");
        solution.set("/d", 5);
        solution.create("/a/b/c", 10);
        solution.set("/a/b/c", 11);
    }
}

// HashMap Solution
class Solution {
    Map<String, Integer> map;
    Map<String, Runnable> callbackMap;

    public Solution() {
        this.map = new HashMap<>();
        map.put("", 0);
        this.callbackMap = new HashMap<>();
    }

    public boolean create(String key, int value) {
        if (map.containsKey(key)) {
            return false;
        }
        String prefix = key.substring(0, key.lastIndexOf("/"));
        if (!map.containsKey(prefix)) {
            return false;
        }
        map.put(key, value);
        return true;
    }

    public boolean set(String key, int value) {
        if (!map.containsKey(key)) {
            return false;
        }
        map.put(key, value);
        String curt = key;
        while (curt.length() > 0) {
            if (callbackMap.containsKey(curt)) {
                callbackMap.get(curt).run();
            }
            curt = curt.substring(0, curt.lastIndexOf("/"));
        }
        return true;
    }

    public int get(String key) {
        if (!map.containsKey(key)) {
            return -1;
        }
        return map.get(key);
    }

    public void watch(String path, String alert) {
        Runnable runnable = new Runnable() {
            public void run() {
                System.out.println(alert);
            }
        };
        callbackMap.put(path, runnable);
    }
}

// Trie Solution, tricky
class Solution2 {
    private TrieNode root;

    public Solution() {
        this.root = new TrieNode("", 0, null, new HashMap<>());
    }

    public boolean create(String key, int value) {
        //    /a/b/c/d -> "" a b c
        TrieNode node = root;
        String[] arr = key.split("/");
        for (int i = 1; i < arr.length - 1; i++) {
            String toCheck = arr[i];
            if (!node.map.containsKey(toCheck)) {
                return false;
            }
            node = node.map.get(toCheck);
        }
        String toAdd = arr[arr.length - 1];
        if (node.map.containsKey(toAdd)) {
            return false;
        }
        TrieNode newNode = new TrieNode(toAdd, value, null, new HashMap<>());
        node.map.put(toAdd, newNode);
        return true;
    }

    public int get(String key) {
        TrieNode node = root;
        String[] arr = key.split("/");
        for (int i = 1; i < arr.length; i++) {
            String next = arr[i];
            if (!node.map.containsKey(next)) {
                return -1;
            }
            node = node.map.get(next);
        }
        return node.value;
    }

    public boolean set(String key, int value) {
        TrieNode node = root;
        String[] arr = key.split("/");
        for (int i = 1; i < arr.length; i++) {
            String toCheck = arr[i];
            if (!node.map.containsKey(toCheck)) {
                return false;
            }
            node = node.map.get(toCheck);
        }
        node.value = value;
        return true;
    }

    public void watch(String key, String alert) {
         //关于watch方法，需要询问未建立的节点是否能添加watch，如果可以的话，就需要用HashMap做；
        //如果只在当前的存在的节点上添加watch的话，就直接按照路径找到节点，然后添加即可.以下做的是在
现有的路径上添加watch的版本
        //另外如果需要没有error的时候才执行callback函数的话，那么需要在每个节点添加父亲节点，只有在
没有error成功create和set的时候才通过parent节点反向向上执行callback
```

```
        TrieNode node = root;
        String[] arr = key.split("/");
        for (int i = 1; i < arr.length; i++) {
            String toCheck = arr[i];
            if (!node.map.containsKey(toCheck)) {
                return;
            }
            node = node.map.get(toCheck);
        }
        Runnable runnable = new Runnable(){
            public void run() {
                System.out.println(alert);
            }
        };
        node.callback = runnable;
    }
}

class TrieNode {
    String key;
    int value;
    Runnable callback;
    Map<String, TrieNode> map;
    public TrieNode(String key, int value, Runnable callback, Map<String, TrieNode> map) {
        this.key = key;
        this.value = value;
        this.callback = callback;
        this.map = map;

    }
}

//
//Preference List
//
import java.util.*;
import java.lang.*;

class Rextester
{
    public static void main(String args[])
    {
        List<Integer> list1 = new ArrayList<>(Arrays.asList(3, 5, 7, 9));
        List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 3, 8));
        List<Integer> list3 = new ArrayList<>(Arrays.asList(5,8));
        List<List<Integer>> input = new ArrayList<>();
        input.add(list1);
        input.add(list2);
        input.add(list3);

        List<Integer> result = getPreference(input);
        for (int i = 0; i < result.size(); i++) {
            System.out.print(result.get(i) + ",");
        }
    }

    public static List<Integer> getPreference(List<List<Integer>> prefers) {
        List<Integer> result = new ArrayList<>();
        //如果有平手出现，就按照入queue的顺序，那么edges中的用List就比较好，因为它维持顺序，或者
LinkedHashSet也可以
        Map<Integer, List<Integer>> edges = new HashMap<>();
        Map<Integer, Integer> inDegrees = new HashMap<>();
        for (List<Integer> list : prefers) {
            for (int i = 0; i < list.size() - 1; i++) {
                int from = list.get(i);
                int to = list.get(i + 1);
                //indegree要包含from节点，否则下面BFS无从开始!
                if (!inDegrees.containsKey(from)) {
                    inDegrees.put(from, 0);
                }
                inDegrees.put(to, inDegrees.getOrDefault(to, 0) + 1);
                if (!edges.containsKey(from)) {
                    edges.put(from, new ArrayList<>());
                }
                edges.get(from).add(to);
            }
        }
        Queue<Integer> queue = new LinkedList<>();
        for (Map.Entry<Integer, Integer> entry : inDegrees.entrySet()) {
            if (entry.getValue() == 0) {
                queue.offer(entry.getKey());
            }
        }
        while (!queue.isEmpty()) {
            int head = queue.poll();
            result.add(head);
            if (edges.containsKey(head)) {
                for (int next : edges.get(head)) {
                    inDegrees.put(next, inDegrees.get(next) - 1);
                    if (inDegrees.get(next) == 0) {
                        queue.offer(next);
                    }
                }
            }
        }
        return result;
    }
}
```