ECOLE NATIONALE D'INGENIEURS DE SOUSSE  
  
CLUB ACM ENISo

**ACM Notebook 2018**

1. ChessQueensProblems

1.1. EightQueensProblem  
  
public class EightQueensProblem {  
  
 private static int[] row = new int[8];  
 private static int a;  
 private static int b;  
 private static int lineCounter;  
  
 private static boolean place(int r, int c) {  
 for (int prev = 0; prev < c; prev++) {  
 if (row[prev] == r || Math.abs(row[prev] - r) == Math.abs(prev - c)) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 private static void backtrack(int c) {  
 if (c == 8 && row[b] == a) {  
 System.out.print(" " + (++lineCounter) + "\t" + (row[0] + 1));  
 for (int i = 1; i < 8; i++) {  
 System.out.print(" " + (row[i] + 1));  
 }  
 System.out.println();  
 }  
 for (int r = 0; r < 8; r++) {  
 if (place(r, c)) {  
 row[c] = r;  
 backtrack(c + 1);  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.in);  
 int tc = in.nextInt();  
 while (tc-- > 0) {  
 a = in.nextInt() - 1;  
 b = in.nextInt() - 1;  
 Arrays.fill(row, 0);  
 System.out.println("SOLN\t COLUMN");  
 System.out.println(" #\t1 2 3 4 5 6 7 8\n");  
 backtrack(0);  
 if (tc > 0) {  
 System.out.println();  
 }  
 }  
 }  
}

1.2. NQueensProblem  
  
public class NQueensProblem {  
  
 private static boolean[] rw = new boolean[30];  
 private static boolean[] ld = new boolean[30];  
 private static boolean[] rd = new boolean[30];  
 public static int n;  
 private static int ans = 0;  
 private static char[][] board = new char[15][15];  
  
 private static void backtrack(int c) {  
 if (c == n) {  
 ans++;  
 return;  
 }  
 for (int r = 0; r < n; r++) {  
 if (board[r][c] != '\*' && !rw[r]  
 && !ld[r - c + n - 1] && !rd[r + c]) {  
 rw[r] = ld[r - c + n - 1] = rd[r + c] = true;  
 backtrack(c + 1);  
 rw[r] = ld[r - c + n - 1] = rd[r + c] = false;  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));  
 Scanner in = new Scanner(System.in);  
 n = in.nextInt();  
 int j = 1;  
 while (n != 0) {  
 ans = 0;  
 Arrays.fill(rw, false);  
 Arrays.fill(ld, false);  
 Arrays.fill(rd, false);  
 for (int i = 0; i < n; i++) {  
 board[i] = in.next().toCharArray();  
 }  
 backtrack(0);  
 out.printf("Case %d: %d\n", j++, ans);  
 n = in.nextInt();  
 }  
 out.close();  
 }  
}

2. ClassicalDP

2.1. CoinChange  
  
public class CoinChange {  
  
 public static int n;  
 private static int[] coinValue;  
  
 private static int solve(int value) {  
 if (value == 0) {  
 return 0;  
 }  
 if (value < 0) {  
 return Integer.MAX\_VALUE;  
 }  
 int min = Integer.MAX\_VALUE;  
 for (int aCoinValue : coinValue) {  
 min = Integer.min(min, solve(value - aCoinValue));  
 }  
 return 1 + min;  
 }  
  
 private static int nbrWays(int i, int value) {  
 if (value == 0) {  
 return 1;  
 }  
 if (value < 0 || i == n) {  
 return 0;  
 }  
 return nbrWays(i + 1, value) + nbrWays(i, value - coinValue[i]);  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 4;  
 coinValue = new int[n];  
 coinValue[0] = 1;  
 coinValue[1] = 3;  
 coinValue[2] = 4;  
 coinValue[3] = 5;  
 out.println(solve(3));  
 out.println(nbrWays(0, 5));  
 out.close();  
 }  
  
}

2.2. Knapsack  
  
public class Knapsack {  
  
 public static int n;  
 private static int[] value;  
 private static int[] weight;  
  
 private static int solve(int i, int remW) {  
 if (remW == 0 || i == n) {  
 return 0;  
 }  
 if (weight[i] > remW) {  
 return solve(i + 1, remW);  
 } else {  
 return Integer.max(solve(i + 1, remW),  
 value[i] + solve(i + 1, remW - weight[i]));  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 4;  
 int s = 12;  
 value = new int[n];  
 value[0] = 100;  
 value[1] = 70;  
 value[2] = 50;  
 value[3] = 10;  
 weight = new int[n];  
 weight[0] = 10;  
 weight[1] = 4;  
 weight[2] = 6;  
 weight[3] = 12;  
 out.println(solve(0, s));  
 out.close();  
 }  
  
}

2.3. LIS  
  
public class LIS {  
  
 public static int n;  
 private static int[] tab;  
  
 private static int lis(int k) {  
 if (k == 0) {  
 return 1;  
 }  
 int max = 0;  
 for (int i = 0; i < k; i++) {  
 if (tab[i] < tab[k]) {  
 max = Integer.max(max, lis(i) + 1);  
 }  
 }  
 return max;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 8;  
 tab = new int[n];  
 tab[0] = -7;  
 tab[1] = 10;  
 tab[2] = 9;  
 tab[3] = 2;  
 tab[4] = 3;  
 tab[5] = 8;  
 tab[6] = 8;  
 tab[7] = 1;  
 int max = 0;  
 for (int i = 0; i < n; i++) {  
 max = Integer.max(max, lis(i));  
 }  
 out.println(max);  
 out.close();  
 }  
}

2.4. Max1DRangeSum  
  
public class Max1DRangeSum {  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int n = 9;  
 int tab[] = {4, -5, 4, -3, 4, 4, -4, 4, -5};  
 int s = 0;  
 int ans = 0;  
 for (int aTab : tab) {  
 s += aTab;  
 ans = Integer.max(ans, s);  
 s = (s < 0) ? 0 : s;  
 }  
 out.println(ans);  
 out.close();  
 }  
}

2.5. Max2DRangeSum  
  
public class Max2DRangeSum {  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 Scanner in = new Scanner(System.in);  
 int n = in.nextInt();  
 int[][] tab = new int[n][n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 tab[i][j] = in.nextInt();  
 if (i > 0) {  
 tab[i][j] += tab[i - 1][j];  
 }  
 if (j > 0) {  
 tab[i][j] += tab[i][j - 1];  
 }  
 if (j > 0 && i > 0) {  
 tab[i][j] -= tab[i - 1][j - 1];  
 }  
 }  
 }  
 int max = -127 \* n \* n;  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 for (int k = i; k < n; k++) {  
 for (int l = j; l < n; l++) {  
 int s = tab[k][l];  
 if (i > 0) {  
 s -= tab[i - 1][l];  
 }  
 if (j > 0) {  
 s -= tab[k][j - 1];  
 }  
 if (j > 0 && i > 0) {  
 s += tab[i - 1][j - 1];  
 }  
 max = Integer.max(max, s);  
 }  
 }  
 }  
 }  
 out.println(max);  
 out.close();  
 }  
}

2.6. TSP  
  
public class TSP {  
  
 public static int n;  
 private static int[][] dist;  
  
 private static int tsp(int i, int mask) {  
 if (mask == (1 << n) - 1) {  
 return dist[i][0];  
 }  
 int min = Integer.MAX\_VALUE;  
 for (int j = 0; j < n; j++) {  
 if (i != j && (mask & (1 << j)) == 0) {  
 min = Integer.min(min, dist[i][j]  
 + tsp(j, mask | (1 << j)));  
 }  
 }  
 return min;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 4;  
 dist = new int[n][n];  
 dist[0] = Arrays.copyOf(new int[]{0, 20, 42, 35}, n);  
 dist[1] = Arrays.copyOf(new int[]{20, 0, 30, 34}, n);  
 dist[2] = Arrays.copyOf(new int[]{42, 30, 0, 12}, n);  
 dist[3] = Arrays.copyOf(new int[]{35, 34, 12, 0}, n);  
 out.println(tsp(0, 1));  
 out.close();  
 }  
}

3. FenwickTree  
  
public class FenwickTree {  
  
 private int[] ft;  
  
 public FenwickTree(int n) {  
 ft = new int[n + 1];  
 }  
  
 private int LSOne(int s) {  
 return s & (-s);  
 }  
  
 public int rsq(int b) {  
 int s = 0;  
 for (; b > 0; b -= LSOne(b)) {  
 s += ft[b];  
 }  
 return s;  
 }  
  
 public int rsq(int a, int b) {  
 return rsq(b) - (a == 1 ? 0 : rsq(a - 1));  
 }  
  
 public void adjust(int k, int v) {  
 for (; k < ft.length; k += LSOne(k)) {  
 ft[k] += v;  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int f[] = {2, 4, 5, 5, 6, 6, 6, 7, 7, 8, 9};  
 FenwickTree ft = new FenwickTree(1000000);  
 for (int i = 0; i < f.length; i++) {  
 ft.adjust(f[i], 1);   
 }  
 out.println(ft.rsq(1,1));  
 out.println(ft.rsq(1,2));  
 out.println(ft.rsq(1,6));  
 out.println(ft.rsq(1,10));  
 out.println(ft.rsq(3,6));  
 ft.adjust(5, 2);  
 out.println(ft.rsq(1,10));  
 out.close();  
 }  
  
}

4. GoodMethods  
  
public class GoodMethods {  
  
 public static List<int[]> list = new LinkedList<>();  
  
 /\*\*  
 \* Permute donne tous les permutations d'un ensemble des entiers,  
 \* charactères, etc ...  
 \*  
 \* @param input  
 \* @param startindex  
 \*/  
 public static void Permute(int[] input, int startindex) {  
 int size = input.length;  
 if (size == startindex + 1) {  
 int[] tab = new int[size];  
 for (int i = 0; i < tab.length; i++) {  
 tab[i] = input[i];  
 }  
 list.add(tab);  
 } else {  
 for (int i = startindex; i < size; i++) {  
 int temp = input[i];  
 input[i] = input[startindex];  
 input[startindex] = temp;  
 Permute(input, startindex + 1);  
 int temp2 = input[i];  
 input[i] = input[startindex];  
 input[startindex] = temp2;  
 }  
 }  
 }  
  
 /\*\*  
 \* Méthode de radix sort d'un tableau des entiers  
 \*  
 \* @param f  
 \* @return  
 \*/  
 public static int[] radixSort(int[] f) {  
 return radixSort(f, f.length);  
 }  
  
 public static int[] radixSort(int[] f, int n) {  
 int[] to = new int[n];  
 {  
 int[] b = new int[65537];  
 for (int i = 0; i < n; i++) {  
 b[1 + (f[i] & 0xffff)]++;  
 }  
 for (int i = 1; i <= 65536; i++) {  
 b[i] += b[i - 1];  
 }  
 for (int i = 0; i < n; i++) {  
 to[b[f[i] & 0xffff]++] = f[i];  
 }  
 int[] d = f;  
 f = to;  
 to = d;  
 }  
 {  
 int[] b = new int[65537];  
 for (int i = 0; i < n; i++) {  
 b[1 + (f[i] >>> 16)]++;  
 }  
 for (int i = 1; i <= 65536; i++) {  
 b[i] += b[i - 1];  
 }  
 for (int i = 0; i < n; i++) {  
 to[b[f[i] >>> 16]++] = f[i];  
 }  
 int[] d = f;  
 f = to;  
 to = d;  
 }  
 return f;  
 }  
  
 public static double bisection(double i, double v, int m) {  
 double a = 0.01;  
 double b = (1 + i) \* v;  
 double d = (a + b) / 2;  
 double fa = f(a, m, v, i);  
 double fb = f(b, m, v, i);  
 double fd = f(d, m, v, i);  
 int j = 0;  
 while (Math.abs(fd) > 1e-10) {  
 if (fa \* fd < 0) {  
 b = d;  
 d = (a + b) / 2;  
 fb = fd;  
 fd = f(d, m, v, i);  
 } else {  
 a = d;  
 d = (a + b) / 2;  
 fa = fd;  
 fd = f(d, m, v, i);  
 }  
 }  
 return d;  
 }  
  
 public static double f(double d, int m, double v, double i) {  
 if (m == 1) {  
 return v \* (1.0 + i) - d;  
 }  
 return f(d, m - 1, v \* (1.0 + i) - d, i);  
 }  
}

5. Graphs

5.1. ArticulationPointsandBridges  
  
public class ArticulationPointsandBridges {  
  
 public static int n;  
 private static int dfsRoot;  
 private static int rootChildren;  
 private static int dfsNumberCounter;  
 private static int[] visited = new int[100000];  
 private static int[] low = new int[100000];  
 private static boolean[] articulation\_vertex = new boolean[100000];  
 private static int[] parent = new int[100000];  
 private static ArrayList<Integer>[] AdjList = new ArrayList[100000];  
  
 private static void articulationPointAndBridge(int u, PrintWriter out) {  
 low[u] = visited[u] = dfsNumberCounter++;  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (visited[v] == -1) {  
 parent[v] = u;  
 if (u == dfsRoot) {  
 rootChildren++;  
 }  
 articulationPointAndBridge(v, out);  
 if (low[v] >= visited[u]) {  
 articulation\_vertex[v] = true;  
 }  
 if (low[v] > visited[u]) {  
 out.printf("Edge (%d, %d) is a bridge\n", u, v);  
 }  
 low[u] = Integer.min(low[u], low[v]);  
 } else if (v != parent[u]) {  
 low[u] = Integer.min(low[u], visited[v]);  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 6;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(4);  
 AdjList[1].add(3);  
 AdjList[1].add(5);  
 AdjList[2].add(1);  
 AdjList[3].add(1);  
 AdjList[4].add(1);  
 AdjList[4].add(5);  
 AdjList[5].add(1);  
 AdjList[5].add(4);  
 Arrays.fill(visited, -1);  
 out.println("Bridges :");  
 for (int i = 0; i < n; i++) {  
 if (visited[i] == -1) {  
 dfsRoot = i;  
 rootChildren = 0;  
 articulationPointAndBridge(i, out);  
 articulation\_vertex[dfsRoot] = rootChildren > 1;  
 }  
 }  
 out.println("Articulation Points:");  
 for (int i = 0; i < n; i++) {  
 if (articulation\_vertex[i]) {  
 out.println("Vertex "+i);  
 }  
 }  
 out.close();  
 }  
}

5.2. BFS  
  
public class BFS {  
  
 public static int n;  
 private static ArrayList<Integer>[] AdjList = new ArrayList[1000];  
  
 private static void bfs(int s, PrintWriter out) {  
 int[] d = new int[1000];  
 Arrays.fill(d, -1);  
 d[s] = 0;  
 Queue<Integer> q = new LinkedList<>();  
 q.add(s);  
 while (!q.isEmpty()) {  
 int u = q.poll();  
 out.print("Layer "+d[u]+":");  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (d[v] == -1) {  
 d[v] = d[u] + 1;  
 out.print(" "+v);  
 q.add(v);  
 }  
 }  
 out.println();  
 }  
 out.flush();  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int n = 13;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[0].add(4);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(5);  
 AdjList[2].add(1);  
 AdjList[2].add(3);  
 AdjList[2].add(6);  
 AdjList[3].add(2);  
 AdjList[3].add(7);  
 AdjList[4].add(0);  
 AdjList[4].add(8);  
 AdjList[5].add(1);  
 AdjList[5].add(6);  
 AdjList[5].add(10);  
 AdjList[6].add(2);  
 AdjList[6].add(5);  
 AdjList[6].add(11);  
 AdjList[7].add(3);  
 AdjList[7].add(12);  
 AdjList[8].add(4);  
 AdjList[8].add(9);  
 AdjList[9].add(8);  
 AdjList[9].add(10);  
 AdjList[10].add(5);  
 AdjList[10].add(9);  
 AdjList[10].add(11);  
 AdjList[11].add(6);  
 AdjList[11].add(10);  
 AdjList[11].add(12);  
 AdjList[12].add(7);  
 AdjList[12].add(11);  
 bfs(5, out);  
 out.close();  
 }  
}

5.3. BipartiteGraph  
  
public class BipartiteGraph {  
  
 public static int n;  
 private static ArrayList<Integer>[] AdjList = new ArrayList[1000];  
  
 private static boolean isBipartite(int s) {  
 int[] color = new int[1000];  
 Arrays.fill(color, -1);  
 color[s] = 0;  
 Queue<Integer> q = new LinkedList<>();  
 q.add(s);  
 while (!q.isEmpty()) {  
 int u = q.poll();  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (color[v] == -1) {  
 color[v] = 1 - color[u];  
 q.add(v);  
 } else if (color[v] == color[u]) {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int n = 13;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[0].add(4);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(5);  
 AdjList[2].add(1);  
 AdjList[2].add(3);  
 AdjList[2].add(6);  
 AdjList[3].add(2);  
 AdjList[3].add(7);  
 AdjList[4].add(0);  
 AdjList[4].add(8);  
 AdjList[5].add(1);  
 AdjList[5].add(6);  
 AdjList[5].add(10);  
 AdjList[6].add(2);  
 AdjList[6].add(5);  
 AdjList[6].add(11);  
 AdjList[7].add(3);  
 AdjList[7].add(12);  
 AdjList[8].add(4);  
 AdjList[8].add(9);  
 AdjList[9].add(8);  
 AdjList[9].add(10);  
 AdjList[10].add(5);  
 AdjList[10].add(9);  
 AdjList[10].add(11);  
 AdjList[11].add(6);  
 AdjList[11].add(10);  
 AdjList[11].add(12);  
 AdjList[12].add(7);  
 AdjList[12].add(11);  
 out.println(isBipartite(0));  
 out.close();  
 }  
}

5.4. DFS  
  
public class DFS {  
  
 public static int n;  
 private static boolean[] visited = new boolean[100000];  
 private static ArrayList<Integer>[] AdjList = new ArrayList[100000];  
  
 private static String dfs(int u) {  
 StringBuilder ch = new StringBuilder(u + "");  
 visited[u] = true;  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (!visited[v]) {  
 ch.append(" ").append(dfs(v));  
 }  
 }  
 return ch.toString();  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int n = 9;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(3);  
 AdjList[2].add(1);  
 AdjList[2].add(3);  
 AdjList[3].add(1);  
 AdjList[3].add(2);  
 AdjList[3].add(4);  
 AdjList[4].add(3);  
 AdjList[6].add(7);  
 AdjList[6].add(8);  
 AdjList[7].add(6);  
 AdjList[8].add(6);  
 int numCC = 0;  
 for (int i = 0; i < n; i++) {  
 if (!visited[i]) {  
 out.println("CC " + ++numCC + ": " + dfs(i));  
 }  
 }  
 out.close();  
 }  
}

5.5. DFSSpanningTree  
  
public class DFSSpanningTree {  
  
 public static int n;  
 private static int[] visited = new int[100000];  
 private static int[] parent = new int[100000];  
 private static ArrayList<Integer>[] AdjList = new ArrayList[100000];  
  
 private static void graphCheck(int u, PrintWriter out) {  
 visited[u] = 1;  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 switch (visited[v]) {  
 case -1:  
 parent[v] = u;  
 graphCheck(v, out);  
 break;  
 case 1:  
 if (v == parent[u]) {  
 out.printf("Tow ways (%d, %d)-(%d, %d)\n", u, v, v, u);  
 } else {  
 out.printf("Back Edge (%d, %d) (Cycle)\n", u, v);  
 }  
 break;  
 default:  
 out.printf("Forward/Cross Edge (%d, %d)\n", u, v);  
 break;  
 }  
 }  
 visited[u] = 2;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 9;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(3);  
 AdjList[2].add(1);  
 AdjList[2].add(3);  
 AdjList[3].add(1);  
 AdjList[3].add(2);  
 AdjList[3].add(4);  
 AdjList[4].add(3);  
 AdjList[6].add(7);  
 AdjList[6].add(8);  
 AdjList[7].add(6);  
 AdjList[8].add(6);  
 Arrays.fill(visited, -1);  
 int numComp = 0;  
 for (int i = 0; i < n; i++) {  
 if (visited[i] == -1) {  
 out.println("Component " + ++numComp + ":");  
 graphCheck(i, out);  
 }  
 }  
 out.close();  
 }  
}

5.6. FindStronglyConnectedComponents  
  
public class FindStronglyConnectedComponents {  
 public static int n;  
 private static int dfsRoot;  
 private static int rootChildren;  
 private static int dfsNumberCounter;  
 private static boolean[] visited = new boolean[100000];  
 private static int[] num = new int[100000];  
 private static int[] low = new int[100000];  
 private static int numSCC = 0;  
 private static boolean[] articulation\_vertex = new boolean[100000];  
 private static int[] parent = new int[100000];  
 private static ArrayList<Integer>[] AdjList = new ArrayList[100000];  
 private static Stack<Integer> S = new Stack<>();  
  
 private static void tarjanSCC(int u, PrintWriter out) {  
 low[u] = num[u] = dfsNumberCounter++;  
 S.push(u);  
 visited[u] = true;  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (num[v] == -1) {  
 tarjanSCC(v, out);  
 }  
 if (visited[v]) {  
 low[u] = Math.min(low[u], low[v]);  
 }  
 }  
 if (low[u] == num[u]) {  
 out.printf("SCC %d:", ++numSCC);  
 while (true) {  
 int v = S.pop();  
 visited[v] = false;  
 out.printf(" %d", v);  
 if (u == v) break;  
 }  
 out.println();  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 n = 8;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[1].add(3);  
 AdjList[2].add(1);  
 AdjList[3].add(4);  
 AdjList[3].add(2);  
 AdjList[4].add(5);  
 AdjList[5].add(7);  
 AdjList[7].add(6);  
 AdjList[6].add(4);  
 Arrays.fill(num, -1);  
 Arrays.fill(low, 0);  
 Arrays.fill(visited, false);  
 dfsNumberCounter = numSCC = 0;  
 for (int i = 0; i < n; i++) {  
 if (num[i] == -1) {  
 tarjanSCC(i, out);  
 }  
 }  
 out.close();  
 }  
}

5.7. FloodFill  
  
public class FloodFill {  
  
 private static int R;  
 private static int C;  
 private static char[][] grid = new char[1000][1000];  
 private static int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};  
 private static int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};  
  
 private static int floodFill(int r, int c, char c1, char c2) {  
 if (r < 0 || r >= R || c < 0 || c >= C) {  
 return 0;  
 }  
 if (grid[r][c] != c1) {  
 return 0;  
 }  
 int s = 1;  
 grid[r][c] = c2;  
 for (int i = 0; i < 8; i++) {  
 s += floodFill(r + dr[i], c + dc[i], c1, c2);  
 }  
 return s;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 out.close();  
 }  
}

5.8. mst

5.8.1. Kruskal  
  
import javafx.util.Pair;  
  
  
public class Kruskal {  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 ArrayList<Pair<Integer, Pair<Integer, Integer>>> EdgeList = new ArrayList<>();  
 int e = 7;  
 EdgeList.add(new Pair<>(4, new Pair<>(0, 1)));  
 EdgeList.add(new Pair<>(4, new Pair<>(0, 2)));  
 EdgeList.add(new Pair<>(6, new Pair<>(0, 3)));  
 EdgeList.add(new Pair<>(6, new Pair<>(0, 4)));  
 EdgeList.add(new Pair<>(2, new Pair<>(1, 2)));  
 EdgeList.add(new Pair<>(8, new Pair<>(2, 3)));  
 EdgeList.add(new Pair<>(9, new Pair<>(3, 4)));  
 //Minimum Spanning Tree  
 EdgeList.sort(Comparator.comparingInt(Pair::getKey));  
 //Maximum Spanning Tree  
// EdgeList.sort((o1, o2) -> o2.getKey() - o1.getKey());  
 int mst\_cost = 0;  
 UnionFind unionFind = new UnionFind(5);  
 for (int i = 0; i < e; i++) {  
 Pair<Integer, Pair<Integer, Integer>> f = EdgeList.get(i);  
 if (!unionFind.isSameSet(f.getValue().getKey(), f.getValue().getValue())) {  
 mst\_cost += f.getKey();  
 unionFind.unionSet(f.getValue().getKey(), f.getValue().getValue());  
 }  
 }  
 out.printf("MST cost = %d (Kruskal's)\n", mst\_cost);  
 out.close();  
 }  
  
}

5.8.2. Prim  
  
public class Prim {  
  
 private static boolean[] taken = new boolean[1000000];  
 private static ArrayList<Pair<Integer, Integer>>[] AdjList = new ArrayList[1000000];  
 private static PriorityQueue<Pair<Integer, Integer>> pq =  
 new PriorityQueue<>();  
  
 public static void process(int vtx) {  
 taken[vtx] = true;  
 for (int i = 0; i < AdjList[vtx].size(); i++) {  
 Pair<Integer, Integer> v = AdjList[vtx].get(i);  
 if (!taken[v.getFirst()]) {  
 pq.add(new Pair<>(v.getSecond(), v.getFirst()));  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 Arrays.fill(taken, false);  
 int n = 5;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(new Pair<>(1, 4));  
 AdjList[0].add(new Pair<>(2, 4));  
 AdjList[0].add(new Pair<>(3, 6));  
 AdjList[0].add(new Pair<>(4, 6));  
 AdjList[1].add(new Pair<>(2, 2));  
 AdjList[2].add(new Pair<>(3, 8));  
 AdjList[3].add(new Pair<>(4, 9));  
 AdjList[4].add(new Pair<>(3, 9));  
 process(0);  
 int mst\_cost = 0;  
 while (!pq.isEmpty()) {  
 Pair<Integer, Integer> f = pq.poll();  
 int u = f.getSecond();  
 int w = f.getFirst();  
 if (!taken[u]) {  
 mst\_cost += w;  
 process(u);  
 }  
 }  
 out.printf("MST cost = %d (Prim's)\n", mst\_cost);  
 out.close();  
 }  
  
 static class Pair<A, B> implements Comparable<Pair<A, B>> {  
 private A first;  
 private B second;  
  
 Pair(A first, B second) {  
 super();  
 this.first = first;  
 this.second = second;  
 }  
  
 public int hashCode() {  
 int hashFirst = first != null ? first.hashCode() : 0;  
 int hashSecond = second != null ? second.hashCode() : 0;  
  
 return (hashFirst + hashSecond) \* hashSecond + hashFirst;  
 }  
  
 public boolean equals(Object other) {  
 if (other instanceof Pair) {  
 Pair otherPair = (Pair) other;  
 return  
 ((this.first == otherPair.first ||  
 (this.first != null && otherPair.first != null &&  
 this.first.equals(otherPair.first))) &&  
 (this.second == otherPair.second ||  
 (this.second != null && otherPair.second != null &&  
 this.second.equals(otherPair.second))));  
 }  
  
 return false;  
 }  
  
 public String toString() {  
 return "(" + first + ", " + second + ")";  
 }  
  
 A getFirst() {  
 return first;  
 }  
  
 public void setFirst(A first) {  
 this.first = first;  
 }  
  
 B getSecond() {  
 return second;  
 }  
  
 public void setSecond(B second) {  
 this.second = second;  
 }  
  
 @Override  
 public int compareTo(Pair<A, B> o) {  
 return (Integer) this.getFirst() - (Integer) o.getFirst();  
 }  
 }  
}

5.9. TopologicalSort  
  
public class TopologicalSort {  
  
 public static int n;  
 private static boolean[] visited = new boolean[100000];  
 private static ArrayList<Integer>[] AdjList = new ArrayList[100000];  
 private static ArrayList<Integer> ts = new ArrayList();  
  
 private static String dfs(int u) {  
 StringBuilder ch = new StringBuilder(u + "");  
 visited[u] = true;  
 for (int i = 0; i < AdjList[u].size(); i++) {  
 int v = AdjList[u].get(i);  
 if (!visited[v]) {  
 ch.append(" ").append(dfs(v));  
 }  
 }  
 ts.add(u);  
 return ch.toString();  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 int n = 9;  
 for (int i = 0; i < n; i++) {  
 AdjList[i] = new ArrayList<>();  
 }  
 AdjList[0].add(1);  
 AdjList[1].add(0);  
 AdjList[1].add(2);  
 AdjList[1].add(3);  
 AdjList[2].add(1);  
 AdjList[2].add(3);  
 AdjList[3].add(1);  
 AdjList[3].add(2);  
 AdjList[3].add(4);  
 AdjList[4].add(3);  
 AdjList[6].add(8);  
 AdjList[6].add(7);  
 AdjList[7].add(6);  
 AdjList[8].add(6);  
 int numCC = 0;  
 for (int i = 0; i < n; i++) {  
 if (!visited[i]) {  
 out.println("CC " + ++numCC + ": " + dfs(i));  
 }  
 }  
 for (int i = 0; i < n; i++) {  
 out.print(ts.get(ts.size()-i-1)+" ");  
 }  
 out.println();  
 out.close();  
 }  
}

6. Main  
  
public class Main {  
  
 //BufferedReader le plus rapide dans la lecture des données.  
 //PrintWriter le plus rapide dans l'affichage des données.  
 public static void main(String[] args) {  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  
 PrintWriter out = new PrintWriter(System.out);  
 out.println(GoodMethods.bisection(0.1, 1000, 2));  
 out.close();  
 }  
  
}

7. SegmentTree

7.1. SegmentTreeMin  
  
public class SegmentTreeMin {  
  
 private int n;  
 private int[] st;  
 private int[] A;  
  
 public SegmentTreeMin(int[] A) {  
 this.A = A;  
 n = A.length;  
 st = new int[4 \* n];  
 build(1, 0, n - 1);  
 }  
  
 public int rmq(int i, int j) {  
 return rmq(1, 0, n - 1, i, j);  
 }  
  
 private int left(int p) {  
 return p << 1;  
 }  
  
 private int right(int p) {  
 return (p << 1) + 1;  
 }  
  
 private void build(int p, int L, int R) {  
 if (L == R) {  
 st[p] = L;  
 } else {  
 build(left(p), L, (L + R) / 2);  
 build(right(p), (L + R) / 2 + 1, R);  
 int p1 = st[left(p)];  
 int p2 = st[right(p)];  
 st[p] = (A[p1] <= A[p2]) ? p1 : p2;  
 }  
 }  
  
 private int rmq(int p, int L, int R, int i, int j) {  
 if (i > R || j < L) {  
 return -1;  
 }  
 if (L >= i && R <= j) {  
 return st[p];  
 }  
 int p1 = rmq(left(p), L, (L + R) / 2, i, j);  
 int p2 = rmq(right(p), (L + R) / 2 + 1, R, i, j);  
 if (p1 == -1) {  
 return p2;  
 }  
 if (p2 == -1) {  
 return p1;  
 }  
 return (A[p1] <= A[p2]) ? p1 : p2;  
 }  
}

7.2. SegmentTreeOrXor  
  
public class SegmentTreeOrXor {  
  
 private int n;  
 private int[] st;  
 private int[] A;  
  
 public SegmentTreeOrXor(int[] A, int op) {  
 this.A = A;  
 n = A.length;  
 st = new int[4 \* n];  
 build(1, 0, n - 1, op);  
 }  
  
 public void update(int p, int L, int R, int i, int val, int op) {  
 if (L == R) {  
 st[p] = A[i] = val;  
 return;  
 }  
 if (i >= L && i <= (L + R) / 2) {  
 update(left(p), L, (L + R) / 2, i, val, op ^ 1);  
 } else if (i > (L + R) / 2 && i <= R) {  
 update(right(p), (L + R) / 2 + 1, R, i, val, op ^ 1);  
 }  
 int p1 = st[left(p)];  
 int p2 = st[right(p)];  
 st[p] = merge(p1, op, p2);  
 }  
  
 public int getRoot() {  
 return st[1];  
 }  
  
 private int left(int p) {  
 return p << 1;  
 }  
  
 private int right(int p) {  
 return (p << 1) + 1;  
 }  
  
 private int merge(int x1, int op, int x2) {  
 if (op == 1) {  
 return x1 ^ x2;  
 }  
 return x1 | x2;  
 }  
  
 private void build(int p, int L, int R, int op) {  
 if (L == R) {  
 st[p] = A[L];  
 } else {  
 build(left(p), L, (L + R) / 2, op ^ 1);  
 build(right(p), (L + R) / 2 + 1, R, op ^ 1);  
 int p1 = st[left(p)];  
 int p2 = st[right(p)];  
 st[p] = merge(p1, op, p2);  
 }  
 }  
  
}

7.3. SegmentTreePGCD  
  
public class SegmentTreePGCD {  
  
 private int n;  
 private int[] st;  
 private int[] A;  
  
 public SegmentTreePGCD(int[] A) {  
 this.A = A;  
 n = A.length;  
 st = new int[5 \* n];  
 build(1, 0, n - 1);  
 }  
  
 public void update(int p, int L, int R, int i, int val) {  
 if (L == R) {  
 st[p] = A[i] = val;  
 return;  
 }  
 if (i >= L && i <= (L + R) / 2) {  
 update(left(p), L, (L + R) / 2, i, val);  
 } else if (i > (L + R) / 2 && i <= R) {  
 update(right(p), (L + R) / 2 + 1, R, i, val);  
 }  
 int p1 = st[left(p)];  
 int p2 = st[right(p)];  
 st[p] = pgcd(p1, p2);  
 }  
  
 public int rmq(int i, int j) {  
 return rmq(1, 0, n - 1, i, j);  
 }  
  
 private int left(int p) {  
 return p << 1;  
 }  
  
 private int right(int p) {  
 return (p << 1) + 1;  
 }  
  
 private int pgcd(int a, int b) {  
 if (b == 0) {  
 return a;  
 }  
 return pgcd(b, a % b);  
 }  
  
 private void build(int p, int L, int R) {  
 if (L == R) {  
 st[p] = A[L];  
 } else {  
 build(left(p), L, (L + R) / 2);  
 build(right(p), (L + R) / 2 + 1, R);  
 int p1 = st[left(p)];  
 int p2 = st[right(p)];  
 st[p] = pgcd(p1, p2);  
 }  
 }  
  
 private int rmq(int p, int L, int R, int i, int j) {  
 if (i > R || j < L) {  
 return -1;  
 }  
 if (L >= i && R <= j) {  
 return st[p];  
 }  
 int p1 = rmq(left(p), L, (L + R) / 2, i, j);  
 int p2 = rmq(right(p), (L + R) / 2 + 1, R, i, j);  
 if (p1 == -1) {  
 return p2;  
 }  
 if (p2 == -1) {  
 return p1;  
 }  
 return pgcd(p1, p2);  
 }  
  
 public static int s = 0;  
  
 public void verif(int p, int L, int R, int i, int j, int x) {  
 if (s > 1) {  
 return;  
 }  
 if (i > R || j < L) {  
 return;  
 }  
 if (L == R) {  
 if (st[p] % x != 0) {  
 s++;  
 }  
 return;  
 }  
 if (st[left(p)] % x != 0) {  
 verif(left(p), L, (L + R) / 2, i, j, x);  
 }  
 if (st[right(p)] % x != 0) {  
 verif(right(p), (L + R) / 2 + 1, R, i, j, x);  
 }  
 }  
  
}

7.4. SegmentTreeString  
  
public class SegmentTreeString {  
  
 private int n;  
 private String[] st;  
 private char[] A;  
  
 public SegmentTreeString(char[] A) {  
 this.A = A;  
 n = A.length;  
 st = new String[5 \* n];  
 build(1, 0, n - 1);  
 }  
  
 public void update(int p, int L, int R, int i, char val) {  
 if (L == R) {  
 A[i] = val;  
 st[p] = val + "";  
 return;  
 }  
 if (i >= L && i <= (L + R) / 2) {  
 update(left(p), L, (L + R) / 2, i, val);  
 } else if (i > (L + R) / 2 && i <= R) {  
 update(right(p), (L + R) / 2 + 1, R, i, val);  
 }  
 String p1 = st[left(p)];  
 String p2 = st[right(p)];  
 st[p] = p1 + p2;  
 }  
  
 public String rmq(int i, int j) {  
 return rmq(1, 0, n - 1, i, j);  
 }  
  
 private int left(int p) {  
 return p << 1;  
 }  
  
 private int right(int p) {  
 return (p << 1) + 1;  
 }  
  
 private int pgcd(int a, int b) {  
 if (b == 0) {  
 return a;  
 }  
 return pgcd(b, a % b);  
 }  
  
 private void build(int p, int L, int R) {  
 if (L == R) {  
 st[p] = A[L] + "";  
 } else {  
 build(left(p), L, (L + R) / 2);  
 build(right(p), (L + R) / 2 + 1, R);  
 String p1 = st[left(p)];  
 String p2 = st[right(p)];  
 st[p] = p1 + p2;  
 }  
 }  
  
 private String rmq(int p, int L, int R, int i, int j) {  
 if (i > R || j < L) {  
 return "";  
 }  
 if (L >= i && R <= j) {  
 return st[p];  
 }  
 String p1 = rmq(left(p), L, (L + R) / 2, i, j);  
 String p2 = rmq(right(p), (L + R) / 2 + 1, R, i, j);  
 if (p1.equals("")) {  
 return p2;  
 }  
 if (p2.equals("")) {  
 return p1;  
 }  
 return p1 + p2;  
 }  
  
}

7.5. SegmentTreeSum  
  
public class SegmentTreeSum {  
  
 private int n;  
 private int[] st;  
 private int[] A;  
  
 public SegmentTreeSum(int[] A) {  
 this.A = A;  
 n = A.length;  
 st = new int[4 \* n];  
 build(1, 0, n - 1);  
 }  
  
 public int rsq(int i, int j) {  
 return rsq(1, 0, n - 1, i, j);  
 }  
  
 public void updateValue(int i, int newVal) {  
 if (i < 0 || i > n - 1) {  
 System.out.println("Invalid Input!");  
 return;  
 }  
 int val = newVal - A[i];  
 A[i] = newVal;  
 updateValue(1, 0, n-1, i, val);  
 }  
  
 private int left(int p) {  
 return p << 1;  
 }  
  
 private int right(int p) {  
 return (p << 1) + 1;  
 }  
  
 private int build(int p, int L, int R) {  
 if (L == R) {  
 st[p] = A[L];  
 } else {  
 st[p] = build(left(p), L, (L + R) / 2)  
 + build(right(p), (L + R) / 2 + 1, R);  
 }  
 return st[p];  
 }  
  
 private int rsq(int p, int L, int R, int i, int j) {  
 if (i > R || j < L) {  
 return 0;  
 }  
 if (L >= i && R <= j) {  
 return st[p];  
 }  
 return rsq(left(p), L, (L + R) / 2, i, j)  
 + rsq(right(p), (L + R) / 2 + 1, R, i, j);  
 }  
  
 private void updateValue(int p, int L, int R, int i, int val) {  
 if (i < L || i > R) {  
 return;  
 }  
 st[p] = st[p] + val;  
 if (L != R) {  
 updateValue(left(p), L, (L + R) / 2, i, val);  
 updateValue(right(p), (L + R) / 2 + 1, R, i, val);  
 }  
 }  
}

8. UnionFind  
  
public class UnionFind {  
  
 private int[] p;  
 private int[] rank;  
  
 public UnionFind(int n) {  
 p = new int[n];  
 rank = new int[n];  
 for (int i = 0; i < p.length; i++) {  
 p[i] = i;  
 }  
 }  
  
 public int findSet(int i) {  
 return (p[i] == i) ? i : (p[i] = findSet(p[i]));  
 }  
  
 public boolean isSameSet(int i, int j) {  
 return findSet(i) == findSet(j);  
 }  
  
 public void unionSet(int i, int j) {  
 if (!isSameSet(i, j)) {  
 int x = findSet(i);  
 int y = findSet(j);  
 if (rank[x] > rank[y]) {  
 p[y] = x;  
 } else {  
 p[x] = y;  
 if (rank[x] == rank[y]) {  
 rank[y]++;  
 }  
 }  
 }  
 }  
  
}

9. UVa10003  
  
public class UVa10003 {  
  
 public static int[][] memo = new int[101][101];  
 public static int[] A = new int[1001];  
  
 public static int minCost(int left, int right) {  
 if (right==left+1) {  
 return 0;  
 }  
 if (memo[left][right] != -1) {  
 return memo[left][right];  
 }  
 int min = Integer.MAX\_VALUE;  
 for (int i = left+1; i < right; i++) {  
 min= Integer.min(min, minCost(left, i)  
 +minCost(i, right)+A[right]-A[left]);  
 }  
 return memo[left][right] = min;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 for (int i = 0; i < 101; i++) {  
 Arrays.fill(memo[i], -1);  
 }  
 int n = 3;  
 A = Arrays.copyOf(new int[]{0,25,50,75,100}, 5);  
 out.println(minCost(0, n+1));  
 out.close();  
 }  
}

10. UVa10943  
  
public class UVa10943 {  
  
 public static long[][] memo = new long[101][101];  
  
 public static long nbrWays(int n, int k) {  
 if (k == 1) {  
 return 1L;  
 }  
 if (memo[n][k] != -1) {  
 return memo[n][k];  
 }  
 long s = 0;  
 for (int i = 0; i < n + 1; i++) {  
 s += nbrWays(n - i, k - 1);  
 }  
 return memo[n][k] = s;  
 }  
  
 public static void main(String[] args) {  
 PrintWriter out = new PrintWriter(System.out);  
 for (int i = 0; i < 101; i++) {  
 Arrays.fill(memo[i], -1);  
 }  
 out.println(nbrWays(50, 3));  
 out.close();  
 }  
}