Shells:

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

/**
 * @author Autumn Tan
 */

public class Solution {

	public static void main(String[] args) throws IOException {
		Scanner in = new Scanner(System.in);

		int T = in.nextInt();
		for (int i = T; i > 0; i--) {
			int blue = in.nextInt(), green = in.nextInt();
			if (blue + green < 10) {System.out.println("CLOSED");
continue;}

			if (blue % 2 == 0 || green % 2 == 1)
System.out.println("CLOSED");
			else System.out.println("OPEN");
		}

		in.close();
	}

}
```

Cryptopwatty:

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

public class Solution {
    public static void main(String[] args) throws IOException {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        String key = scan.next();
        scan.nextLine();
        for(int i = 0; i < n; i++) {
            String s = scan.nextLine();
            String decoded = "";
            for(int j = 0; j < s.length(); j++) {
                if(s.charAt(j) >= 65 && s.charAt(j) < 91) {
                    char current = s.charAt(j);
                    decoded += (char)(key.indexOf(current + "") + 65);
                }
                else {
                    decoded += s.charAt(j);
                }
            }
            System.out.println(decoded);
        }
        scan.close();
    }
}
```

Oh, No, My Letters!

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

public class Solution {

	public static void main(String[] args) throws IOException {
		//Scanner in = new Scanner(System.in);
		Scanner in = new Scanner(System.in);

		int m = in.nextInt();
		String[] parts = new String[m];
		for (int p = 0; p < m; p++) parts[p] = in.next();
		in.nextLine();
		String recreated = in.nextLine();
		boolean valid = true;
		for (String p: parts)
		{
			if (recreated.contains(p)) recreated =
recreated.replaceAll(p, " ");
			else {valid = false; break;}
		}

		if (valid) System.out.println("Potentially!");
		else System.out.println("No, No, No!!!");

		in.close();
	}

}
```

Krabby Land:

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

/**
 * @author Sam Ziegelbein
 */
public class Solution {

    public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);

      int people = scan.nextInt();
      int minAge = scan.nextInt() * 365 + scan.nextInt();

      int[] ages = new int[people];
      for (int k = 0; k < people; k++)
      {
            scan.next();
            ages[k] = scan.nextInt() * 365 + scan.nextInt();
      }
      Arrays.sort(ages);

      int numAdultsRequired;
      if (people % 3 != 0)
            numAdultsRequired = people / 3 + 1;
      else
            numAdultsRequired = people / 3;

      int waitingTime = minAge - ages[people - numAdultsRequired];

      System.out.println(waitingTime / 365 + " year(s) and " + waitingTime
% 365 + " day(s)");
    }
}
```

Krusty Krab Orders:

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

/**
 * @author Sam Ziegelbein
 */
public class Solution {

	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);

		int days = scan.nextInt();

		for (int k = 0; k < days; k++)
		{
			int items = scan.nextInt();
			double sales = scan.nextDouble();
			int customers = scan.nextInt();

			int krabbyPatties, coralBits, kelpShakes;

			krabbyPatties = items - customers;
			kelpShakes = (int) (sales + krabbyPatties - 1.25 *
krabbyPatties - items);
			coralBits = items - krabbyPatties - kelpShakes;

			System.out.println(krabbyPatties + " krabby patties, " +
coralBits +
					" coral bits, " + kelpShakes + " kelp shakes");
		}
	}


}
```

Magic Jellyfish Field:

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

/**
 * @author Eric K. Zhang
 */
class Solution {
	public static void main(String[] args) throws IOException {
		Scanner scan = new Scanner(System.in);
		int r = scan.nextInt(), c = scan.nextInt();
		int ans = 0;
		for (int i = 0; i < r * c; i++)
			ans = Math.max(ans, scan.nextInt());
		System.out.println(ans);
	}
}
```

Plankton Chaser:

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

/**
 * @author Sam Ziegelbein
 */
public class Solution {
	static double startTime;
	static double spongeSpeed;
	static double plankSpeed;

	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);

		int cases = scan.nextInt();
		for (int k = 0; k < cases; k++)
		{
			startTime = scan.nextDouble();
			plankSpeed = scan.nextDouble();
			spongeSpeed = scan.nextDouble();
			System.out.printf("%.7f%n", findTime());
		}
	}

	public static double findTime()
	{
		double min = 0;
		double max = 1000000000;
		double time = 0;
		for (int k = 0; k < 500; k++)
		{
			time = (min + max) / 2;
			if (getSpongeX(time, getPlankX(), getPlankY(time)) <
getPlankX())
				min = time;
			else
				max = time;
		}
		return time;
	}

	public static double getPlankX()
	{
```

```
            return plankSpeed * startTime;
      }

      public static double getPlankY(double t)
      {
            return plankSpeed * (t - startTime);
      }

      public static double getSpongeX(double t, double pX, double pY)
      {
            double angle = Math.atan2(pY, pX);
            return spongeSpeed * Math.cos(angle) * (t - startTime);
      }

}
```

Grid Arrangements:

```
import java.util.*;
import java.io.*;
public class Solution
{
    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);
        int m = in.nextInt();
        for (int i = 0; i < m; i ++)
        {
            int n = in.nextInt();
            int sum = 0;
            for (int k = 1; k <= n; k ++)
            {
                sum += (n/k);
            }
            System.out.println(sum);
        }
    }
}
```

You Snooze, You Loose

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

/**
 * @author Eric K. Zhang
 */
class Solution {
	static final int[] di = { 0, 0, 1, -1 };
	static final int[] dj = { 1, -1, 0, 0 };

	public static void main(String[] args) throws IOException {
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		String[] ar = new String[10];
		for (int i = 0; i < 10; i++)
			ar[i] = br.readLine();

		int[][] dist = new int[10][10];
		Queue<int[]> q = new ArrayDeque<>();
		int ei = -1, ej = -1;
		for (int i = 0; i < 10; i++) {
			for (int j = 0; j < 10; j++) {
				dist[i][j] = -1;
				if (ar[i].charAt(j) == 'S') {
					dist[i][j] = 0;
					q.add(new int[] { i, j });
				}
				else if (ar[i].charAt(j) == 'K') {
					ei = i;
					ej = j;
				}
			}
		}

		// Breadth-first search
		while (!q.isEmpty()) {
			int[] pos = q.remove();
			for (int k = 0; k < 4; k++) {
				int i = pos[0] + di[k], j = pos[1] + dj[k];
				if (i < 0 || i >= 10 || j < 0 || j >= 10 ||
ar[i].charAt(j) == '#' || dist[i][j] != -1)
					continue;
				dist[i][j] = dist[pos[0]][pos[1]] + 1;
```

```
                        q.add(new int[] { i, j });
                  }
            }

            int d = dist[ei][ej];
            if (d <= 15)
                  System.out.println("On time: " + d + " minutes");
            else
                  System.out.println("Late: " + (d - 15) + " minutes");
      }
}
```

Prime Establishment:

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

/**
 * @author Sam Ziegelbein
 */
public class Solution {
	static boolean[] sieve;
	static
	{
		sieve = new boolean[11000];
		Arrays.fill(sieve, true);
		sieve[0] = sieve[1] = false;
		for (int i = 2; i < sieve.length; i++)
		{
			if (sieve[i])
			{
				for (int k = i * 2; k < sieve.length; k += i)
					sieve[k] = false;
			}
		}
	}

	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);

		int days = scan.nextInt();

		for (int d = 0; d < days; d++)
		{
			int customers = scan.nextInt();
			int nextPrime = customers + 1;
			while (!sieve[nextPrime])
				nextPrime++;
			System.out.println(nextPrime);
		}
	}

}
```

Cheap Crabs:

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

/**
 * @author Eric K. Zhang
 */
class Solution {
      public static void main(String[] args) throws IOException {
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            int n = Integer.parseInt(br.readLine());
            StringTokenizer st = new StringTokenizer(br.readLine());

            int best = 0, cur = 0;
            int pre = Integer.parseInt(st.nextToken());
            for (int i = 1; i < n; i++) {
                  // Kadane's algorithm
                  int x = Integer.parseInt(st.nextToken());
                  cur = Math.max(0, cur + 1 - Math.abs(x - pre));
                  best = Math.max(best, cur);
                  pre = x;
            }

            System.out.println(best + 1);
      }
}
```

Primes Are Hard:

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

public class Solution {

    static int[] sieve;

    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);

        sieve = new int[1000001];
        findPrimes();

        int T = in.nextInt();
        in.nextLine();
        for (int idx = 0; idx < T; idx++) {
            String eq = in.nextLine();
            String[] nums = eq.split("=|x");
            ArrayList<Integer> actual = new ArrayList<>();

            boolean wrong = false;
            TreeSet<Integer> notPrime = new TreeSet<>();

            for (int i = nums.length - 1; i > 0; i--) {
                int x = Integer.parseInt(nums[i].trim());
                if (sieve[x] == x)
                    actual.add(x);
                else {
                    wrong = true;
                    notPrime.add(x);
                    while (sieve[x] != 0) {
                        actual.add(sieve[x]);
                        x /= sieve[x];
                    }
                }
            }

            if (wrong) {
                System.out.println("WRONG!");
                for (int x: notPrime)
                    System.out.println(x + " is not a prime number.");
                Collections.sort(actual);
                printEq(nums[0].trim() + " = ", actual);
```

```
            }
            else {
                System.out.println("correct!\n" + eq);
            }
        }

        in.close();
    }

    static void findPrimes() {
        for (int i = 2; i < sieve.length; i++) {
            if (sieve[i] == 0) {
                for (int j = i; j < sieve.length; j += i)
                    sieve[j] = i;
            }
        }
    }

    static void printEq(String start, ArrayList<Integer> arr) {
        for (int i: arr) start += i + " x ";
        System.out.println(start.substring(0, start.length() - 3));
    }

}
```

Angry Fish:

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

public class Solution {
    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        long[] days = new long[2 * n];
        for (int i = 0; i < n; i++) {
            long a = in.nextInt();
            long b = in.nextInt();
            long c = in.nextInt();
            days[2 * i] = 20 * b + a;
            days[2 * i + 1] = 20 * c + a;
        }
        Arrays.sort(days);
        int ans = 0;
        int last = 0;
        int val = 0;
        for (int i = 0; i < 2 * n; i++) {
            int species = (int) (days[i] % 20);
            int day = (int) (days[i] / 20);
            if (val != 0)
                ans += day - last;

            val ^= (1 << species);
            last = day;
        }
        System.out.println(ans);
    }
}
```

Tree Labels:

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

/**
 * Created by ezhang on 5/17/18.
 */
public class Solution {

    static int[] A;
    static List<Integer>[] adj;
    static int[] ans;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int N = Integer.parseInt(br.readLine());

        // Initialize variables
        A = new int[N];
        adj = new List[N];
        for (int i = 0; i < N; i++)
            adj[i] = new ArrayList<>();
        ans = new int[N];

        // Read input
        StringTokenizer st = new StringTokenizer(br.readLine());
        for (int i = 0; i < N; i++) {
            A[i] = Integer.parseInt(st.nextToken()) - 1;
        }
        for (int i = 1; i < N; i++) {
            st = new StringTokenizer(br.readLine());
            int u = Integer.parseInt(st.nextToken());
            int v = Integer.parseInt(st.nextToken());
            --u; --v;
            adj[u].add(v);
            adj[v].add(u);
        }

        // Solve problem
        // Iteration version of DFS to avoid possible stack overflow
        List<Integer> stack = new ArrayList<Integer>();
        stack.add(-1);
        stack.add(0);
```

```
        int[] indices = new int[N];
        int[] cnt = new int[N];
        int unique = 0;
        while (stack.size() >= 2) {
            int n = stack.get(stack.size() - 1);
            int p = stack.get(stack.size() - 2);
            if (indices[n] == 0) {
                // first time through
                if (cnt[A[n]]++ == 0)
                    ++unique;
                ans[n] = unique;
            }

            while (indices[n] < adj[n].size() && adj[n].get(indices[n]) ==
p)
                ++indices[n];
            if (indices[n] < adj[n].size())
                stack.add(adj[n].get(indices[n]++));
            else {
                stack.remove(stack.size() - 1);
                if (--cnt[A[n]] == 0)
                    --unique;
            }
        }

        // Output
        for (int i = 0; i < N; i++)
            System.out.println(ans[i]);
    }

    // static void dfs(int n, int p) {
    //     if (cnt[A[n]]++ == 0)
    //         ++unique;
    //     ans[n] = unique;

    //     for (int v : adj[n]) {
    //         if (v != p)
    //             dfs(v, n);
    //     }

    //     if (--cnt[A[n]] == 0)
    //         --unique;
    // }
}
```

Pool Table:

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

/*
 * @author Wuyou Xie, Eric Zhang
 */
public class Solution {

    public static void main(String[] args) throws IOException {
        Scanner scan = new Scanner(System.in);

        int vertices = scan.nextInt();
        int[][] ver = new int[vertices][2];
        for (int i = 0; i < vertices; i++) {
            for (int c = 0; c < 2; c++) {
                ver[i][c] = scan.nextInt();
            }
        }

        int[] start = new int[2];
        // We will be doing reflections that necessitate floating-point
arithmetic, hence double[]
        double[] target = new double[2];

        target[0] = scan.nextInt();
        target[1] = scan.nextInt();
        start[0] = scan.nextInt();
        start[1] = scan.nextInt();

        for (int i = vertices - 1; i >= 0; i--) {
            int j = (i + 1) % vertices;
            mirror(target, ver[i], ver[j]);
        }

        // Find angle part
        double diffX = target[0] - start[0];
        double diffY = target[1] - start[1];
        double angle = Math.toDegrees(Math.atan2(diffY, diffX));
        System.out.printf("%.8f\n", (angle + 360) % 360);
    }

    static void mirror(double[] target, int[] p, int[] q) {
        // Find unit normal vector <a, b> to PQ
```

```
    double a = p[1] - q[1];
    double b = q[0] - p[0];

    // Normalize to a unit normal vector
    double len = Math.sqrt(a * a + b * b);
    a /= len;
    b /= len;

    // Write the equation of the line as ax + by = c
    double c = a * p[0] + b * p[1];

    // Compute the signed distance from point to line
    double dist = c - (a * target[0] + b * target[1]);

    // Displace target by twice its distance
    target[0] += 2 * dist * a;
    target[1] += 2 * dist * b;
}
```

Lucky Hands:

```
import java.util.*;

public class Solution {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner in = new Scanner(System.in);

		int T = in.nextInt();
		in.nextLine();
		double sf = 0, four = 0, full = 0, str = 0, three = 0, pair =
0;

		for (int idx = T; idx > 0; idx--)
		{
			String[] hand = in.nextLine().split(",");
			Hand x = new Hand();
			for (int i = 0; i < 10; i++)
				x.add(new Card(hand[i], i + 1));

			x.sumCol();
			String[] temp = x.sf();
			int four1 = x.four(), str1 = x.str() -
Integer.parseInt(temp[1]),
					three1 = x.three(), pair1 = x.two();
			int full1 = x.fh(pair1, three1, four1);
			System.out.println("Hand #" + (T - idx + 1));
			System.out.print("Straight Flush: " + temp[1] + "\n" +
temp[0]);
			System.out.println("Four of a Kind: " + four1);
			System.out.println("Full House: " + full1);
			System.out.println("Straight: " + str1);
			System.out.println("Three of a Kind: " + three1 +
"\nPair: " + pair1 + "\n");

			sf += Integer.parseInt(temp[1]);
			four += four1;
			full += full1;
			str += str1;
			three += three1;
			pair += pair1;
		}
```

```
            System.out.printf("Average\nStraight Flush: %.2f\nFour of a
Kind: %.2f\n"
                        + "Full House: %.2f\nStraight: %.2f\nThree of a
Kind: %.2f\n"
                        + "Pair: %.2f\n", sf / T, four / T, full / T, str /
T, three / T, pair / T);

            in.close();
      }

      static class Card implements Comparable<Card> {
            public int rank, pos;
            public String suit;
            static String convert = "0A234567890JQK";

            public Card(String card, int position)
            {
                  pos = position;
                  if (card.charAt(1) == '0') rank = 10;
                  else rank = convert.indexOf(card.substring(0 , 1));
                  suit = card.substring(card.length() - 1).trim();
            }

            @Override
            public String toString()
            {
                  if (rank == 10) return "10" + suit;
                  return "" + convert.substring(rank, rank + 1) + suit;
            }

            public int compareTo(Card b) {
                  return this.pos - b.pos;
            }

      }

      static class Hand {
            int[][] arr;
            Card[] all;
            int[] col;

            public Hand() {
                  arr = new int[4][13];
                  all = new Card[11];
                  col = new int[13];
```

```
	}

	public void add(Card x) {
		all[x.pos] = x;
		int r = 0;
		switch (x.suit)
		{
		case "C": r = 0; break;
		case "D": r = 1; break;
		case "H": r = 2; break;
		case "S": r = 3; break;
		}
		arr[r][x.rank - 1] = x.pos;
	}

	public String[] sf() {
		String result = "";
		TreeMap<ArrayList<Card>, String> map = new TreeMap<>(new
CardComp());
		int ct = 0;
		for (int r = 0; r < 4; r++) {
			for (int c = 0; c + 4 < 13; c++)
			{
				ArrayList<Card> potential = new
ArrayList<>();
				if (arr[r][c] != 0 && arr[r][c + 1] != 0 &&
arr[r][c + 2] != 0 &&
						arr[r][c + 3] != 0 && arr[r][c +
4] != 0)
				{
					for (int i = 0; i < 5; i++)
potential.add(all[arr[r][c + i]]);
					ct++;
				}
				else continue;
				ArrayList<Card> tmp = potential;
				Collections.sort(potential);
				map.put(tmp, print(potential));
			}
		}
		for (ArrayList<Card> key: map.keySet()) result +=
map.get(key) + "\n";
		return new String[]{result, Integer.toString(ct)};
	}
```

```
public int four() {
      int ct = 0;
      for (int c = 0; c < 13; c++)
            if (col[c] == 4) ct++;
      return ct;
}

public int three() {
      int ct = 0;
      for (int c = 0; c < 13; c++)
      {
            if (col[c] == 3) ct++;
            else if (col[c] == 4) ct += 4;
      }
      return ct;
}

public int two() {
      int ct = 0;
      for (int c = 0; c < 13; c++)
      {
            if (col[c] == 2) ct++;
            else if (col[c] == 3) ct += 3;
            else if (col[c] == 4) ct += 6;
      }
      return ct;
}

public int fh(int two, int three, int four) {
      int result = 0;
      result += (4 * four) * (two - 6);
      result += (three - 4 * four) * (two - 3);
      return result;
}

public int str() {
      int ct = 0;
      for (int c = 0; c + 4 < 13; c++)
      {
            int factor = 1;
            for (int i = 0; i < 5 && factor > 0; i++)
            {
                  if (col[c + i] < 1) factor = 0;
                  if (col[c + i] > 1) factor *= col[c + i];
            }
```

```
                ct += factor;
            }
            return ct;
        }

        public void sumCol() {
            for (int c = 0; c < 13; c++)
            {
                int sum = 0;
                for (int i = 0; i < 4; i++)
                    if (arr[i][c] != 0) sum++;
                col[c] = sum;
            }
        }

        public String print(ArrayList<Card> all) {
            String result = "";
            for (Card x: all) result += x.toString() + ",";
            return result.substring(0, result.length() - 1);
        }
    }

    static class CardComp implements  Comparator<ArrayList<Card>> {

        @Override
        public int compare(ArrayList<Card> a, ArrayList<Card> b)
        {
            if (a.equals(b)) return 0;
            Card one, two;
            int idx = 0;
            while (a.get(idx).equals(b.get(idx)))
                idx++;
            one = a.get(idx);
            two = b.get(idx);
            if (one.rank != two.rank) return one.rank - two.rank;
            return one.suit.compareTo(two.suit);
        }
    }
}
```

Line Drawing:

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

/**
 * Created by ezhang on 5/11/18.
 */
public class Solution {

    static int[][] grid, gridT;
    static int even = 0, odd = -1;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());
        int R = Integer.parseInt(st.nextToken());
        int C = Integer.parseInt(st.nextToken());
        int Q = Integer.parseInt(st.nextToken());
        grid = new int[R][C];
        gridT = new int[C][R];
        for (int i = 0; i < R; i++) {
            st = new StringTokenizer(br.readLine());
            for (int j = 0; j < C; j++) {
                grid[i][j] = gridT[j][i] =
Integer.parseInt(st.nextToken());
            }
        }

        for (int[] row : grid)
            solve(row);
        for (int[] col: gridT)
            solve(col);

        while (Q-- != 0) {
            int P = Integer.parseInt(br.readLine());
            System.out.println((P % 2 == 0 ? P <= even : P <= odd) ? "YES"
: "NO");
        }
    }

    static void solve(int[] ar) {
        int firstEven = 0, lastEven = 0;
        int firstOdd = -1, lastOdd = -1;
```

```
        int total = 0;
        for (int x : ar) {
            total += x;
            if (total % 2 == 0)
                lastEven = total;
            else {
                if (firstOdd == -1)
                    firstOdd = total;
                lastOdd = total;
            }
        }
        even = Math.max(even, Math.max(lastEven - firstEven, lastOdd -
firstOdd));
        if (firstOdd != -1)
            odd = Math.max(odd, Math.max(lastOdd - firstEven, lastEven -
firstOdd));
    }
}
```

Substitution Sort:

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

/**
 * @author Eric K. Zhang
 */
public class Solution {
	static boolean[][] adj;

	static boolean isAcyclic(int n, boolean[] vis, boolean[] cur) {
		if (cur[n]) return false; // Cycle found
		if (vis[n]) return true;
		vis[n] = cur[n] = true;
		for (int v = 0; v < 26; v++) {
			if (adj[n][v] && !isAcyclic(v, vis, cur))
				return false;
		}
		cur[n] = false;
		return true;
	}

	static void getPred(int vertex, List<Integer> li) {
		for (int n = 0; n < 26; n++) {
			if (adj[n][vertex] && !li.contains(n)) {
				li.add(n);
				getPred(n, li);
			}
		}
	}

	static int lexTopoSort(List<Integer> vertices, int[] rank, int[]
firstIndex, int t) {
		Collections.sort(vertices, (Integer a, Integer b) ->
firstIndex[a] - firstIndex[b]);
		for (int n : vertices) {
			if (rank[n] == -1) {
				List<Integer> pred = new ArrayList<>();
				getPred(n, pred);
				t = lexTopoSort(pred, rank, firstIndex, t);
				rank[n] = t++;
			}
		}
		return t;
```

```
	}

	static int[] substitutionSort(String[] words) {
		int[] firstIndex = new int[26];
		Arrays.fill(firstIndex, Integer.MAX_VALUE);
		int t = 0;
		for (int i = 0; i < words.length; i++) {
			for (int j = 0;  j < words[i].length(); j++) {
				int c = words[i].charAt(j) - 'A';
				if (firstIndex[c] == Integer.MAX_VALUE)
					firstIndex[c] = t;
				++t;
			}
		}

		adj = new boolean[26][26];
		for (int i = 0; i + 1 < words.length; i++) {
			String w1 = words[i], w2 = words[i + 1];
			int index = 0;
			while (index < w1.length() && index < w2.length()) {
				if (w1.charAt(index) != w2.charAt(index))
					break; // Found first difference
				++index;
			}
			if (index >= w1.length() || index >= w2.length()) {
				if (w1.length() > w2.length())
					return null; // Failed, w2 is a prefix of w1
				continue; // If w1 is a prefix of w2, nothing to
check
			}
			char c1 = w1.charAt(index), c2 = w2.charAt(index);
			adj[c1 - 'A'][c2 - 'A'] = true;
		}

		// Cycle detection
		boolean[] vis = new boolean[26];
		boolean[] cur = new boolean[26];
		for (int i = 0; i < 26; i++)
			if (!isAcyclic(i, vis, cur))
				return null; // Cycle detected, no topological
ordering

		// Topological sort
		int[] rank = new int[26];
		Arrays.fill(rank, -1);
```

```
            List<Integer> vertices = new ArrayList<>(26);
            for (int i = 0; i < 26; i++)
                  vertices.add(i);
            lexTopoSort(vertices, rank, firstIndex, 0);

            return rank;
      }

      public static void main(String[] args) throws IOException {
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            int n = Integer.parseInt(br.readLine());
            String[] words = new String[n];
            for (int i = 0; i < n; i++)
                  words[i] = br.readLine();

            int[] order = substitutionSort(words);
            if (order == null) {
                  System.out.println("No Solution");
                  return;
            }
            for (String word : words) {
                  StringBuilder sb = new StringBuilder(word.length());
                  for (int i = 0; i < word.length(); i++)
                        sb.append((char) ('A' + order[word.charAt(i) -
'A']));
                  System.out.println(sb.toString());
            }
      }
}
```

Balanced Tokens:

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

/**
 * @author Eric K. Zhang
 */
public class Solution {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));

        StringTokenizer st = new StringTokenizer(br.readLine());
        int N = Integer.parseInt(st.nextToken());
        int Q = Integer.parseInt(st.nextToken());

        Node[] verts = new Node[N];
        int[] vals = new int[N];
        String S = br.readLine();
        for (int i = 0; i < N; i++) {
            vals[i] = (S.charAt(i) == '(' ? 1 : -1);
            verts[i] = new Node();
            verts[i].set(vals[i]);
        }

        while (Q-- != 0) {
            st = new StringTokenizer(br.readLine());
            String command = st.nextToken();
            if (command.equals("connect")) {
                Node A = verts[Integer.parseInt(st.nextToken()) - 1];
                Node B = verts[Integer.parseInt(st.nextToken()) - 1];
                if (Node.connected(A, B))
                    out.println("no");
                else {
                    Node.link(A, B);
                    out.println("yes");
                }
            }
            else if (command.equals("disconnect")) {
                Node A = verts[Integer.parseInt(st.nextToken()) - 1];
                Node B = verts[Integer.parseInt(st.nextToken()) - 1];
```

```
                if (Node.cut(A, B))
                    out.println("yes");
                else
                    out.println("no");
            }
            else if (command.equals("toggle")) {
                int A = Integer.parseInt(st.nextToken()) - 1;
                vals[A] *= -1;
                verts[A].set(vals[A]);
            }
            else if (command.equals("balance")) {
                Node A = verts[Integer.parseInt(st.nextToken()) - 1];
                Node B = verts[Integer.parseInt(st.nextToken()) - 1];
                if (!Node.connected(A, B))
                    out.println("invalid");
                else {
                    int ans = Node.pathAggregate(A, B);
                    if (ans == 0)
                        out.println("balanced");
                    else
                        out.println(ans);
                }
            }
            else {
                throw new IllegalArgumentException("Invalid command: " +
command);
            }
        }

        out.flush();
    }

}

/**
 * Link-Cut Tree
 * http://www.cs.cmu.edu/~sleator/papers/self-adjusting.pdf
 */
class Node {

    void set(int x) {
        splay();
        this.x = x;
        update();
    }
```

```
static boolean connected(Node x, Node y) {
    if (x == y)
        return true;
    x.expose();
    y.expose();
    return x.p != null;
}

static void link(Node x, Node y) {
    x.makeRoot();
    x.p = y;
}

static boolean cut(Node x, Node y) {
    x.makeRoot();
    y.expose();
    if (y.l != x || x.l != null || x.r != null)
        return false;
    y.l.p = null;
    y.l = null;
    return true;
}

static int pathAggregate(Node x, Node y) {
    x.makeRoot();
    y.expose();
    return y.sum - 2 * y.min;
}

private Node p, l, r;
private int x, sum, min, max;
private boolean flip;

private boolean isRoot() {
    return p == null || (p.l != this && p.r != this);
}

private int dir() {
    if (isRoot())
        return -1;
    return p.r == this ? 1 : 0;
}

private void update() {
```

```
        if (l != null)
            l.push();
        if (r != null)
            r.push();
        int ls = (l != null ? l.sum : 0);
        int rs = (r != null ? r.sum : 0);
        sum = ls + x + rs;
        min = max = 0;
        if (l != null) {
            min = Math.min(min, l.min);
            max = Math.max(max, l.max);
        }
        min = Math.min(min, ls + x);
        max = Math.max(max, ls + x);
        if (r != null) {
            min = Math.min(min, ls + x + r.min);
            max = Math.max(max, ls + x + r.max);
        }
    }

    private void push() {
        if (flip) {
            Node temp = l;
            l = r;
            r = temp;

            int temp2 = min;
            min = sum - max;
            max = sum - temp2;

            if (l != null)
                l.flip = !l.flip;
            if (r != null)
                r.flip = !r.flip;
            flip = false;
        }
    }

    private static void connect(Node pa, Node ch, int dir) {
        if (ch != null)
            ch.p = pa;
        if (dir == 0)
            pa.l = ch;
        if (dir == 1)
            pa.r = ch;
```

```
}

private void rot() {
    if (isRoot())
        throw new IllegalStateException();

    int x = dir();
    Node pa = p;

    connect(pa.p, this, pa.dir());
    connect(pa, x == 0 ? r : l, x);
    connect(this, pa, 1 - x);

    pa.update();
    update();
}

private void splay() {
    while (!isRoot() && !p.isRoot()) {
        p.p.push();
        p.push();
        push();
        if (dir() == p.dir())
            p.rot();
        else
            rot();
        rot();
    }
    if (!isRoot()) {
        p.push();
        push();
        rot();
    }
    push();
}

private void expose() {
    Node pre = null;
    for (Node v = this; v != null; v = v.p) {
        v.splay();
        v.r = pre;
        v.update();
        pre = v;
    }
    splay();
```

```
    }

    private void makeRoot() {
        expose();
        flip = !flip;
    }

}
```