```
 1: /** Modelo Java, BufferedReader, PrintWriter e EOF ..... */
 2: /** Fatorial simples c/ BigInteger ..... */
 3:
 4: /** Vector ..... */
 5: /** Algoritmos Java Collections ..... */
 6: /** Manipulacao de bits ..... */
 7: /** Stack/Queue do Java ..... */
 8: /** Map/Set do Java ..... */
 9: /** Priority Queue do Java ..... */
10: /** Grafos (implementacao) ..... */
11: /** Union-Find Disjoint Sets ..... */
12: /** Arvore de Segmentos ..... */
13: /** Fenwick Tree (Binary Indexed Tree, ou BIT) ..... */
14:
15: /** Backtracking (exemplo) ..... */
16: /** Programacao Dinamica (ex. 1: Top-Down) ..... */
17: /** Programacao Dinamica (ex. 2: Bottom-Up) ..... */
18: /** Max 1D Range Sum ..... */
19: /** Maximum Sum ..... */
20: /** Longest Increasing Subsequence (LIS) ..... */
21: /** Algoritmo da Mochila 0-1 ..... */
22: /** Coin Change (Problema do Troco) ..... */
23: /** Problema do Caixeiro Viajante ..... */
24: /** Qtd. de formas de obter um numero N somando K numeros ..... */
25: /** Cutting Sticks ..... */
26:
27: /** DFS (Busca em Profundidade) ..... */
28: /** Flood Fill / grafo implicito em matriz ..... */
29: /** Kruskal e Prim (Arvore Geradora Minima) ..... */
30: /** BFS (Busca em Largura/Amplitude) ..... */
31: /** Dijkstra ..... */
32: /** Bellman-Ford ..... */
33: /** Floyd-Warshall ..... */
34: /** Edmonds-Karp ..... */
35: /** Emparelhamento Maximo em Grafos Bipartidos ..... */
36: /** IntegerPair.java ..... */
37: /** IntegerTriple.java ..... */
38:
39: /** BigInteger (soma) ..... */
40: /** BigInteger (mod) ..... */
41: /** BigInteger (primos) ..... */
42: /** BigInteger (divisao) ..... */
43: /** BigInteger (exponenciacao) ..... */
44: /** Crivo de Eratostenes (descobre n's primos) ..... */
45: /** Floyd's Cycle-Finding Algorithm ..... */
46:
47: /** Strings (algoritmos basicos) ..... */
48: /** Knuth-Morris-Pratt (string matching) ..... */
49: /** Alinhamento de Strings (Needleman-Wunsch) ..... */
50: /** Array de Sufixos ..... */
51:
52: /** Pontos e Linhas ..... */
53: /** Circulos ..... */
54: /** Triangulos ..... */
55: /** Poligonos ..... */
56:
57: /** 15-Puzzle Problem with IDA* ..... */
58: /** Prog. Dinamica com Bitmask ..... */
59: /** Prog. Dinamica (outro exemplo) ..... */
60: /** Outras tecnicas ..... */
61:
62: /** Eliminacao Gaussiana ..... */
63: /** Lowest Common Ancestor (LCA) ..... */
64: /** Pollard Rho (fatoracao) ..... */
65: /** Range Minimum Query (RMQ) ..... */
66: /** Fibonacci Modular ..... */
67: /** Shortest Path Faster Algorithm ..... */
68: /** Algarismos Romanos ..... */
69: /** Distancia entre pontos em esfera + dist. euclidiana ..... */
70: /** Componentes Fortemente Conectadas ..... */
```

71:

```java
/*******************************************************************************/
/** Modelo Java, BufferedReader, PrintWriter e EOF ........................... */
/*******************************************************************************/

import java.io.*;
import java.util.*;
import java.math.*;

class Main{

  public static void main(String args[]) throws IOException{
    Locale.setDefault(new Locale("en", "US"));   // SEMPRE INCLUA ESTA LINHA
    // Scanner e System.out.print* demoram;
    // se possivel, use BufferedReader e PrintWriter
    // Leia o arquivo Java-EntradaESaida.pdf para referencia de ambos
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pr =
      new PrintWriter(new BufferedWriter(new OutputStreamWriter(System.out)));

    /* desenvolva sua solucao aqui */

    /* exemplo de uso de BufferedReader e PrintWriter */
    String s;
    int contador = 0;
    while((s = br.readLine()) != null){ // teste de EOF com BufferedReader
      pr.println(s); // aviso: printWriter imprime apenas no final da execucao
      contador++;
      pr.printf("%d linha(s) lida(s)\n", contador);
    }
    /* fim do exemplo */

    pr.close(); // SEMPRE INCLUA ESTA LINHA se usar PrintWriter
    // caso contrario, nenhum dado vai ser impresso
  }

}
```

```
 1: /*************************************************************************/
 2: /** Fatorial simples c/ BigInteger ..................................... */
 3: /*************************************************************************/
 4:
 5: import java.util.Scanner;
 6: import java.math.BigInteger;
 7:
 8: class Main {                          // standard Java class name in UVa OJ
 9:   public static void main(String[] args) {
10:     BigInteger fac = BigInteger.ONE;
11:     for (int i = 2; i <= 25; i++)
12:       fac = fac.multiply(BigInteger.valueOf(i));   // it is in the library!
13:     System.out.println(fac);
14:   }
15: }
16:
```

```
1: /*********************************************************************************/
2: /** Vector ..................................................................... */
3: /*********************************************************************************/
4:
5: import java.util.*;
6:
7: class ch2_01_array_vector {
8:   public static void main(String[] args) {
9:     // initial value {7,7,7,0,0} and thus initial size (5)
10:    int[] arr = new int[] {7,7,7,0,0};
11:    // initial size (5) and initial value {5,5,5,5,5}
12:    Vector<Integer> v = new Vector<Integer>(Collections.nCopies(5, 5));
13:
14:    // 7 and 5, for Java Vector, we must use 'get'
15:    System.out.println("arr[2] = " + arr[2] + " and v[2] = " + v.get(2));
16:
17:    for (int i = 0; i <= 4; i++) {
18:      arr[i] = i;
19:      v.set(i, i);                    // for Java Vector, we must use 'set'
20:    }
21:
22:    // 2 and 2
23:    System.out.println("arr[2] = " + arr[2] + " and v[2] = " + v.get(2));
24:
25:    // arr[5] = 5;   // static array will generate index out of bound error
26:    // uncomment the line above to see the error
27:
28:    v.add(5);  // vector will resize itself (use method add in Java Vector)
29:    System.out.println("v[5] = " + v.get(5));                          // 5
30:  }
31: }
```

```java
/*********************************************************************************/
/** Algoritmos Java Collections ..................................................*/
/*********************************************************************************/

import java.util.*;

// This source code is not as complete as ch2_02_algorithm_collections.cpp

class team implements Comparable<team> {
  private int id, solved, penalty;

  public team(int id, int solved, int penalty) {
    this.id = id;
    this.solved = solved;
    this.penalty = penalty;
  }

  public int compareTo(team o) {
    if (solved != o.solved) // can use this primary field to decide sorted order
      return o.solved - solved;   // ICPC rule: sort by number of problem solved
    else if (penalty != o.penalty)          // solved == o.solved, but we can use
                                          // secondary field to decide sorted order
      return penalty - o.penalty;        // ICPC rule: sort by descending penalty
    else                              // solved == o.solved AND penalty == o.penalty
      return id - o.id;                       // sort based on increasing team ID
  }

  public String toString() {
    return "id: " + id + ", solved: " + solved + ", penalty: " + penalty;
  }
}

class ch2_02_algorithm_collections {
  public static void main(String[] args) {
    Vector<Integer> v = new Vector<Integer>();

    v.add(10);
    v.add(7);
    v.add(2);
    v.add(15);
    v.add(4);

    // sort descending with vector
    Collections.sort(v);
    // if we want to modify comparison function, use the overloaded method:
    // Collections.sort(List list, Comparator c);
    Collections.reverse(v);

    System.out.println(v);
    System.out.printf("==================\n");

    // shuffle the content again
    Collections.shuffle(v);
    System.out.println(v);
    System.out.printf("==================\n");

    // sort ascending
    Collections.sort(v);
    System.out.println(v);
    System.out.printf("==================\n");

    Vector<team> nus = new Vector<team>();
    nus.add(new team(1, 1, 10));
    nus.add(new team(2, 3, 60));
    nus.add(new team(3, 1, 20));
    nus.add(new team(4, 3, 60));

    // without sorting, they will be ranked like this:
    for (int i = 0; i < 4; i++)
      System.out.println(nus.get(i));
```

```
  71:
  72:     Collections.sort(nus);              // sort using a comparison function
  73:     System.out.printf("==================\n");
  74:     // after sorting using ICPC rule, they will be ranked like this:
  75:     for (int i = 0; i < 4; i++)
  76:       System.out.println(nus.get(i));
  77:     System.out.printf("==================\n");
  78:
  79:     int pos = Collections.binarySearch(v, 7);
  80:     System.out.println("Trying to search for 7 in v, found at index = " + pos);
  81:
  82:     pos = Collections.binarySearch(v, 77);
  83:     System.out.println("Trying to search for 77 in v, found at index = " + pos);
  84:     // output is -5 (explanation below)
  85:
  86:     /*
  87:     binarySearch will return:
  88:       index of the search key, if it is contained in the list;
  89:       otherwise, (-(insertion point) - 1).
  90:       The insertion point is defined as the point at which the key would be
  91:       inserted into the list: the index of the first element greater than the key,
  92:       or list.size(), if all elements in the list are less than the specified key.
  93:       Note that this guarantees that the return value will be >= 0 if and only if
  94:       the key is found.
  95:     */
  96:
  97:     // sometimes these two useful simple macros are used
  98:     System.out.printf("min(10, 7) = %d\n", Math.min(10, 7));
  99:     System.out.printf("max(10, 7) = %d\n", Math.max(10, 7));
 100:   }
 101: }
```

```
/*****************************************************************************/
/** Manipulacao de bits ..................................................... */
/*****************************************************************************/

import java.util.*;

// note: for example usage of BitSet, see ch5_06_primes.java

class ch2_03_bit_manipulation {
  private static int setBit(int S, int j) { return S | (1 << j); }

  private static int isOn(int S, int j) { return S & (1 << j); }

  private static int clearBit(int S, int j) { return S & ~(1 << j); }

  private static int toggleBit(int S, int j) { return S ^ (1 << j); }

  private static int lowBit(int S) { return S & (-S); }

  private static int setAll(int n) { return (1 << n) - 1; }

  // returns S % N, where N is a power of 2
  private static int modulo(int S, int N) { return ((S) & (N - 1)); }

  private static int isPowerOfTwo(int S) { return (S & (S - 1)) == 0 ? 1 : 0; }

  private static int nearestPowerOfTwo(int S) { return
    ((int)Math.pow(2.0, (int)((Math.log((double)S) / Math.log(2.0)) + 0.5))); }

  private static int turnOffLastBit(int S) { return ((S) & (S - 1)); }

  private static int turnOnLastZero(int S) { return ((S) | (S + 1)); }

  private static int turnOffLastConsecutiveBits(int S){ return ((S) & (S + 1)); }

  private static int turnOnLastConsecutiveZeroes(int S){ return ((S) | (S - 1)); }

  private static void printSet(int vS) {              // in binary representation
    System.out.printf("S = %2d = ", vS);
    Stack<Integer> st = new Stack<Integer>();
    while (vS > 0) {
      st.push(vS % 2);
      vS /= 2;
    }
    while (!st.empty()) {                           // to reverse the print order
      System.out.printf("%d", st.peek());
      st.pop();
    }
    System.out.printf("\n");
  }

  public static void main(String[] args) {
    int S, T;

    System.out.printf("1. Representation (all indexing are 0-based and counted
from right)\n");
    S = 34; printSet(S);
    System.out.printf("\n");

    System.out.println("2. Multiply S by 2, then divide S by 4 (2x2), then by 2");
    S = 34; printSet(S);
    S = S << 1; printSet(S);
    S = S >> 2; printSet(S);
    S = S >> 1; printSet(S);
    System.out.printf("\n");

    System.out.printf("3. Set/turn on the 3-th item of the set\n");
    S = 34; printSet(S);
    S = setBit(S, 3); printSet(S);
    System.out.printf("\n");
```

```
   70:
   71:     System.out.printf("4. Check if the 3-th and then 2-nd item of the set is
on?\n");
   72:     S = 42; printSet(S);
   73:     T = isOn(S, 3); System.out.printf("T = %d, %s\n", T, T != 0 ? "ON" : "OFF");
   74:     T = isOn(S, 2); System.out.printf("T = %d, %s\n", T, T != 0 ? "ON" : "OFF");
   75:     System.out.printf("\n");
   76:
   77:     System.out.printf("5. Clear/turn off the 1-st item of the set\n");
   78:     S = 42; printSet(S);
   79:     S = clearBit(S, 1); printSet(S);
   80:     System.out.printf("\n");
   81:
   82:     System.out.printf("6. Toggle the 2-nd item and then 3-rd item of the set\n");
   83:     S = 40; printSet(S);
   84:     S = toggleBit(S, 2); printSet(S);
   85:     S = toggleBit(S, 3); printSet(S);
   86:     System.out.printf("\n");
   87:
   88:     System.out.printf("7. Check the first bit from right that is on\n");
   89:     S = 40; printSet(S);
   90:     T = lowBit(S); System.out.printf("T = %d (this is always a power of 2)\n", T);
   91:     S = 52; printSet(S);
   92:     T = lowBit(S); System.out.printf("T = %d (this is always a power of 2)\n", T);
   93:     System.out.printf("\n");
   94:
   95:     System.out.printf("8. Turn on all bits in a set of size n = 6\n");
   96:     S = setAll(6); printSet(S);
   97:     System.out.printf("\n");
   98:
   99:     System.out.printf("9. Other tricks (not shown in the book)\n");
  100:     System.out.printf("8 %c 4 = %d\n", '%', modulo(8, 4));
  101:     System.out.printf("7 %c 4 = %d\n", '%', modulo(7, 4));
  102:     System.out.printf("6 %c 4 = %d\n", '%', modulo(6, 4));
  103:     System.out.printf("5 %c 4 = %d\n", '%', modulo(5, 4));
  104:     System.out.printf("is %d power of two? %d\n", 9, isPowerOfTwo(9));
  105:     System.out.printf("is %d power of two? %d\n", 8, isPowerOfTwo(8));
  106:     System.out.printf("is %d power of two? %d\n", 7, isPowerOfTwo(7));
  107:     for (int i = 0; i <= 16; i++)
  108:       System.out.printf("Nearest power of two of %d is %d\n", i,
nearestPowerOfTwo(i));
  109:     System.out.printf("S = %d, turn off last bit in S, S = %d\n", 40,
turnOffLastBit(40));
  110:     System.out.printf("S = %d, turn on last zero in S, S = %d\n", 41,
turnOnLastZero(41));
  111:     System.out.printf("S = %d, turn off last consectuve bits in S, S = %d\n", 39,
turnOffLastConsecutiveBits(39));
  112:     System.out.printf("S = %d, turn on last consecutive zeroes in S, S = %d\n",
36, turnOnLastConsecutiveZeroes(36));
  113:   }
  114: }
```

```java
/*********************************************************************************/
/** Stack/Queue do Java ..................................................... */
/*********************************************************************************/

import java.util.*;

class ch2_04_stack_queue {
  public static void main(String[] args) {
    Stack<Character> s = new Stack<Character>();

    // Queue is abstract, must be instantiated with LinkedList
    // (special case for Java Queue)
    Queue<Character> q = new LinkedList<Character>();

    Deque<Character> d = new LinkedList<Character>();

    System.out.println(s.isEmpty());          // currently s is empty, true
    System.out.println("==================");
    s.push('a');
    s.push('b');
    s.push('c');
    // stack is LIFO, thus the content of s is currently like this:
    // c <- top
    // b
    // a
    System.out.println(s.peek());                                  // output 'c'
    s.pop();                                                      // pop topmost
    System.out.println(s.peek());                                  // output 'b'
    System.out.println(s.empty());       // currently s is not empty, false
    System.out.println("==================");

    System.out.println(q.isEmpty());          // currently q is empty, true
    System.out.println("==================");
    while (!s.isEmpty()) {                 // stack s still has 2 more items
      q.offer(s.peek()); // enqueue 'b', and then 'a'
      // (the method name in Java Queue for push/enqueue operation is 'offer')
      s.pop();
    }
    q.offer('z');                                          // add one more item
    System.out.println(q.peek());                                  // prints 'b'
    // in Java, it is harder to see the back of the queue...

    // output 'b', 'a', then 'z' (until queue is empty),
    // according to the insertion order above
    System.out.println("==================");
    while (!q.isEmpty()) {
      System.out.printf("%c\n", q.peek());         // take the front first
      q.poll();                           // before popping (dequeue-ing) it
    }

    System.out.println("==================");
    d.addLast('a');
    d.addLast('b');
    d.addLast('c');
    System.out.printf("%c - %c\n", d.getFirst(), d.getLast()); // prints 'a - c'
    d.addFirst('d');
    System.out.printf("%c - %c\n", d.getFirst(), d.getLast()); // prints 'd - c'
    d.pollLast();
    System.out.printf("%c - %c\n", d.getFirst(), d.getLast()); // prints 'd - b'
    d.pollFirst();
    System.out.printf("%c - %c\n", d.getFirst(), d.getLast()); // prints 'a - b'
  }
}
```

```java
/*********************************************************************************/
/** Map/Set do Java ..................................................... */
/*********************************************************************************/

import java.util.*;

class ch2_05_map_set {
  public static void main(String[] args) {
    // note: there are many clever usages of this set/map
    // that you can learn by looking at top coder's codes
    TreeSet<Integer> used_values = new TreeSet<Integer>();
    // must use TreeSet as Set is an abstract class
    used_values.clear();
    TreeMap<String, Integer> mapper = new TreeMap<String, Integer>();
    // must use TreeMap as Map is an abstract class
    mapper.clear();

    // suppose we enter these 7 name-score pairs below
    /*
    john 78
    billy 69
    andy 80
    steven 77
    felix 82
    grace 75
    martin 81
    */
    mapper.put("john", 78);   used_values.add(78);
    mapper.put("billy", 69);  used_values.add(69);
    mapper.put("andy", 80);   used_values.add(80);
    mapper.put("steven", 77); used_values.add(77);
    mapper.put("felix", 82);  used_values.add(82);
    mapper.put("grace", 75);  used_values.add(75);
    mapper.put("martin", 81); used_values.add(81);

    // then the internal content of mapper MAY be something like this:
    // re-read balanced BST concept if you do not understand this diagram
    // the keys are names (string)!
    //                          (grace,75)
    //           (billy,69)                 (martin,81)
    //     (andy,80)   (felix,82)    (john,78)  (steven,77)

    // iterating through the content of mapper will give a sorted output
    // based on keys (names)
    System.out.println(mapper.keySet());
    System.out.println(mapper.values());

    // map can also be used like this
    System.out.println("steven's score is " + mapper.get("steven") +
      ", grace's score is " + mapper.get("grace"));
    System.out.println("==================");

    // interesting usage of SubMap
    // display data between ["f".."m") ('felix' is included, martin' is excluded)
    SortedMap<String, Integer> res = mapper.subMap("f", "m");
    System.out.println(res.keySet());
    System.out.println(res.values());

    // the internal content of used_values MAY be something like this
    // the keys are values (integers)!
    //                  (78)
    //         (75)             (81)
    //     (69)    (77)    (80)    (82)

    // O(log n) search, found
    System.out.println(used_values.contains(77)); // returns true
    System.out.println(used_values.headSet(77)); // returns [69, 75]
    // (these two are before 77 in the inorder traversal of this BST)
    System.out.println(used_values.tailSet(77)); // returns [77, 78, 80, 81, 82]
    // (these five are equal or after 77 in the inorder traversal of this BST)
```

```
71:     // O(log n) search, not found
72:     if (!used_values.contains(79))
73:       System.out.println("79 not found");
74:   }
75: }
```

```
1: /*****************************************************************************/
2: /** Priority Queue do Java ..................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6:
7: class pair < X, Y > { // utilizing Java "Generics"
8:   X _first;
9:   Y _second;
10:
11:   public pair(X f, Y s) { _first = f; _second = s; }
12:
13:   X first() { return _first; }
14:   Y second() { return _second; }
15:
16:   void setFirst(X f) { _first = f; }
17:   void setSecond(Y s) { _second = s; }
18: }
19:
20: class ch2_06_priority_queue {
21:   public static void main(String[] args) {
22:     // introducing 'pair'
23:     PriorityQueue < pair < Integer, String > > pq =
24:      new PriorityQueue < pair < Integer, String > >(1,
25:       new Comparator< pair < Integer, String > >() {
26:                 // overriding the compare method
27:         public int compare(pair<Integer, String> i, pair<Integer, String> j){
28:           return j.first() - i.first(); // currently max heap,
29:           // reverse these two to try produce min-heap
30:         }
31:       }
32:     );
33:
34:     // suppose we enter these 7 money-name pairs below
35:     /*
36:     100 john
37:     10 billy
38:     20 andy
39:     100 steven
40:     70 felix
41:     2000 grace
42:     70 martin
43:     */
44:     // inserting a pair in O(log n)
45:     pq.offer(new pair < Integer, String > (100, "john"));
46:     pq.offer(new pair < Integer, String > (10, "billy"));
47:     pq.offer(new pair < Integer, String > (20, "andy"));
48:     pq.offer(new pair < Integer, String > (100, "steven"));
49:     pq.offer(new pair < Integer, String > (70, "felix"));
50:     pq.offer(new pair < Integer, String > (2000, "grace"));
51:     pq.offer(new pair < Integer, String > (70, "martin"));
52:     // this is how we use Java PriorityQueue
53:     // priority queue will arrange items in 'heap' based
54:     // on the first key in pair, which is money (integer), largest first
55:     // if first keys tied, use second key, which is name, largest first
56:
57:     // the internal content of pq heap MAY be something like this:
58:     // re-read (max) heap concept if you do not understand this diagram
59:     // the primary keys are money (integer), secondary keys are names (string)!
60:     //                          (2000,grace)
61:     //            (100,steven)                  (70,martin)
62:     //     (100,john)   (10,billy)     (20,andy)   (70,felix)
63:
64:     // let's print out the top 3 person with most money
65:     pair<Integer, String> result = pq.poll(); // O(1) to access the
66:     // top / max element + O(log n) removal of the top and repair the structure
67:     System.out.println(result.second() + " has " + result.first() + " $");
68:     result = pq.poll();
69:     System.out.println(result.second() + " has " + result.first() + " $");
70:     result = pq.poll();
```

```
71:     System.out.println(result.second() + " has " + result.first() + " $");
72:   }
73: }
```

```
/*****************************************************************************/
/** Grafos (implementacao) ..................................................... */
/*****************************************************************************/

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

class pair < X, Y > { // utilizing Java "Generics"
  X _first;
  Y _second;

  public pair(X f, Y s) { _first = f; _second = s; }

  X first() { return _first; }
  Y second() { return _second; }

  void setFirst(X f) { _first = f; }
  void setSecond(Y s) { _second = s; }
}

class ch2_07_graph_ds {
  public static void main(String[] args) throws Exception {
    int V, E, total_neighbors, id, weight, a, b;

    // Try this input for Adjacency Matrix/List/EdgeList
    // Adj Matrix
    //   for each line: |V| entries, 0 or the weight
    // Adj List
    //   for each line: num neighbors, list of neighbors + weight pairs
    // Edge List
    //   for each line: a-b of edge(a,b) and weight
    /*
    6
      0  10   0   0 100   0
     10   0   7   0   8   0
      0   7   0   9   0   0
      0   0   9   0  20   5
    100   8   0  20   0   0
      0   0   0   5   0   0
    6
    2 2 10 5 100
    3 1 10 3 7 5 8
    2 2 7 4 9
    3 3 9 5 20 6 5
    3 1 100 2 8 4 20
    1 4 5
    7
    1 2 10
    1 5 100
    2 3 7
    2 5 8
    3 4 9
    4 5 20
    4 6 5
    */

    File f = new File("in_07.txt");
    Scanner sc = new Scanner(f);
    V = sc.nextInt(); // we must know this size first!
                      // remember that if V is > 100, try NOT to use AdjMat!
    int[][] AdjMat = new int[V][];
    for (int i = 0; i < V; i++) {
      AdjMat[i] = new int[V];
      for (int j = 0; j < V; j++)
        AdjMat[i][j] = sc.nextInt();
    }

    System.out.println("Neighbors of vertex 0:");
    for (int j = 0; j < V; j++) // O(|V|)
      if (AdjMat[0][j] != 0)
```

```
71:         System.out.println("Edge 0-" + j + " (weight = " + AdjMat[0][j] + ")");
72:
73:     V = sc.nextInt();
74:     Vector< Vector< pair < Integer, Integer > > > AdjList =
75:       new Vector< Vector< pair < Integer, Integer > > >(V);
76:     for (int i = 0; i < V; i++) { // for each vertex
77:       Vector< pair < Integer, Integer > > Neighbor =
78:         new Vector < pair < Integer, Integer > >();
79:       AdjList.add(Neighbor); // add this empty neighbor list to Adjacency List
80:     }
81:
82:     for (int i = 0; i < V; i++) {
83:       total_neighbors = sc.nextInt();
84:       for (int j = 0; j < total_neighbors; j++) {
85:         id = sc.nextInt();
86:         weight = sc.nextInt();
87:         AdjList.get(i).add( new pair < Integer, Integer > (id - 1, weight));
88:         // some index adjustment
89:       }
90:     }
91:
92:     System.out.println("Neighbors of vertex 0:");
93:     Iterator it = AdjList.get(0).iterator(); // AdjList[0] contains required info.
94:     while (it.hasNext()) { // O(k), where k is the number of neighbors
95:       pair< Integer, Integer > val = (pair< Integer, Integer >)it.next();
96:       System.out.println("Edge 0-"+ val.first()+ " (weight = "+ val.second()+")");
97:     }
98:
99:     E = sc.nextInt();
100:    PriorityQueue < pair < Integer, pair < Integer, Integer > > > EdgeList =
101:     new PriorityQueue < pair < Integer, pair < Integer, Integer > > >(1,
102:      new Comparator< pair < Integer, pair < Integer, Integer > > >() {
103:        // overriding the compare method
104:        public int compare(pair < Integer, pair < Integer, Integer > > i,
105:         pair < Integer, pair < Integer, Integer > > j) {
106:          return i.first() - j.first(); // currently min heap based on cost
107:        }
108:      }
109:    );
110:
111:    for (int i = 0; i < E; i++) {
112:      a = sc.nextInt();
113:      b = sc.nextInt();
114:      pair < Integer, Integer > ab = new pair < Integer, Integer > (a, b);
115:      weight = sc.nextInt();
116:      EdgeList.offer(new pair < Integer, pair < Integer, Integer > >
117:                                (-weight,  ab) ); // trick to reverse sort order */
118:    }
119:
120:    // edges sorted by weight (smallest->largest)
121:    for (int i = 0; i < E; i++) {
122:      pair < Integer, pair < Integer, Integer > > edge = EdgeList.poll();
123:      // negate the weight again
124:      System.out.println("weight: " + (-edge.first()) + " (" +
125:        edge.second().first() + "-" + edge.second().second() + ")");
126:    }
127:  }
128: }
```

```
    1: /*****************************************************************************/
    2: /** Union-Find Disjoint Sets ................................................ */
    3: /*****************************************************************************/
    4:
    5: import java.util.*;
    6:
    7: // Union-Find Disjoint Sets Library written in OOP manner,
    8: // using both path compression and union by rank heuristics
    9: class UnionFind {                                              // OOP style
   10:   private Vector<Integer> p, rank, setSize;
   11:   private int numSets;
   12:
   13:   public UnionFind(int N) {
   14:     p = new Vector<Integer>(N);
   15:     rank = new Vector<Integer>(N);
   16:     setSize = new Vector<Integer>(N);
   17:     numSets = N;
   18:     for (int i = 0; i < N; i++) {
   19:       p.add(i);
   20:       rank.add(0);
   21:       setSize.add(1);
   22:     }
   23:   }
   24:
   25:   public int findSet(int i) {
   26:     if (p.get(i) == i) return i;
   27:     else {
   28:       int ret = findSet(p.get(i)); p.set(i, ret);
   29:       return ret; } }
   30:
   31:   public Boolean isSameSet(int i, int j) { return findSet(i) == findSet(j); }
   32:
   33:   public void unionSet(int i, int j) {
   34:     if (!isSameSet(i, j)) { numSets--;
   35:     int x = findSet(i), y = findSet(j);
   36:     // rank is used to keep the tree short
   37:     if (rank.get(x) > rank.get(y)) {
   38:                         p.set(y, x); setSize.set(x, setSize.get(x) +
setSize.get(y)); }
   39:     else      { p.set(x, y); setSize.set(y, setSize.get(y) + setSize.get(x));
   40:                 if (rank.get(x) == rank.get(y)) rank.set(y, rank.get(y) + 1); } } }
   41:   public int numDisjointSets() { return numSets; }
   42:   public int sizeOfSet(int i) { return setSize.get(findSet(i)); }
   43: }
   44:
   45: class ch2_08_unionfind_ds {
   46:   public static void main(String[] args) {
   47:     System.out.printf("Assume that there are 5 disjoint sets initially\n");
   48:     UnionFind UF = new UnionFind(5); // create 5 disjoint sets
   49:     System.out.printf("%d\n", UF.numDisjointSets()); // 5
   50:     UF.unionSet(0, 1);
   51:     System.out.printf("%d\n", UF.numDisjointSets()); // 4
   52:     UF.unionSet(2, 3);
   53:     System.out.printf("%d\n", UF.numDisjointSets()); // 3
   54:     UF.unionSet(4, 3);
   55:     System.out.printf("%d\n", UF.numDisjointSets()); // 2
   56:     System.out.printf("isSameSet(0, 3) = %b\n", UF.isSameSet(0, 3)); // false
   57:     System.out.printf("isSameSet(4, 3) = %b\n", UF.isSameSet(4, 3)); // true
   58:     for (int i = 0; i < 5; i++) // fS will return 1 for {0, 1} and 3 for {2, 3, 4}
   59:       System.out.printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i,
UF.findSet(i), i, UF.sizeOfSet(i));
   60:     UF.unionSet(0, 3);
   61:     System.out.printf("%d\n", UF.numDisjointSets()); // 1
   62:     for (int i = 0; i < 5; i++) // findSet will return 3 for {0, 1, 2, 3, 4}
   63:       System.out.printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i,
UF.findSet(i), i, UF.sizeOfSet(i));
   64:   }
   65: }
```

```
1: /*****************************************************************************/
2: /** Arvore de Segmentos ..................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6:
7: class SegmentTree {          // the segment tree is stored like a heap array
8:   private int[] st, A;
9:   private int n;
10:  private int left (int p) { return p << 1; } // same as binary heap operations
11:  private int right(int p) { return (p << 1) + 1; }
12:
13:  private void build(int p, int L, int R) {
14:    if (L == R)                               // as L == R, either one is fine
15:      st[p] = L;                                          // store the index
16:    else {                                  // recursively compute the values
17:      build(left(p) , L               , (L + R) / 2);
18:      build(right(p), (L + R) / 2 + 1, R          );
19:      int p1 = st[left(p)], p2 = st[right(p)];
20:      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
21:  } }
22:
23:  private int rmq(int p, int L, int R, int i, int j) {          // O(log n)
24:    if (i >  R || j <  L) return -1; // current segment outside query range
25:    if (L >= i && R <= j) return st[p];               // inside query range
26:
27:     // compute the min position in the left and right part of the interval
28:    int p1 = rmq(left(p) , L               , (L+R) / 2, i, j);
29:    int p2 = rmq(right(p), (L+R) / 2 + 1, R          , i, j);
30:
31:    if (p1 == -1) return p2;   // if we try to access segment outside query
32:    if (p2 == -1) return p1;                                // same as above
33:    return (A[p1] <= A[p2]) ? p1 : p2; }          // as as in build routine
34:
35:  private int update_point(int p, int L, int R, int idx, int new_value) {
36:    // this update code is still preliminary, i == j
37:    // must be able to update range in the future!
38:    int i = idx, j = idx;
39:
40:    // if the current interval does not intersect
41:    // the update interval, return this st node value!
42:    if (i > R || j < L)
43:      return st[p];
44:
45:    // if the current interval is included in the update range,
46:    // update that st[node]
47:    if (L == i && R == j) {
48:      A[i] = new_value; // update the underlying array
49:      return st[p] = L; // this index
50:    }
51:
52:    // compute the minimum position in the
53:    // left and right part of the interval
54:    int p1, p2;
55:    p1 = update_point(left(p) , L               , (L + R) / 2, idx, new_value);
56:    p2 = update_point(right(p), (L + R) / 2 + 1, R          , idx, new_value);
57:
58:    // return the position where the overall minimum is
59:    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
60:  }
61:
62:  public SegmentTree(int[] _A) {
63:    A = _A; n = A.length;                    // copy content for local usage
64:    st = new int[4 * n];
65:    for (int i = 0; i < 4 * n; i++)
66:          st[i] = 0; // create vector with length 'len' and fill it with zeroes
67:    build(1, 0, n - 1);                                   // recursive build
68:  }
69:
70:  public int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); } // overloading
```

```
  71:
  72:   public int update_point(int idx, int new_value) {
  73:     return update_point(1, 0, n - 1, idx, new_value); }
  74: }
  75:
  76: class ch2_09_segmenttree_ds {
  77:   public static void main(String[] args) {
  78:     int[] A = new int[] { 18, 17, 13, 19, 15, 11, 20 }; // the original array
  79:     SegmentTree st = new SegmentTree(A);
  80:
  81:     System.out.printf("              idx    0, 1, 2, 3, 4,  5, 6\n");
  82:     System.out.printf("              A is {18,17,13,19,15, 11,20}\n");
  83:     System.out.printf("RMQ(1, 3) = %d\n", st.rmq(1, 3)); // answer = index 2
  84:     System.out.printf("RMQ(4, 6) = %d\n", st.rmq(4, 6)); // answer = index 5
  85:     System.out.printf("RMQ(3, 4) = %d\n", st.rmq(3, 4)); // answer = index 4
  86:     System.out.printf("RMQ(0, 0) = %d\n", st.rmq(0, 0)); // answer = index 0
  87:     System.out.printf("RMQ(0, 1) = %d\n", st.rmq(0, 1)); // answer = index 1
  88:     System.out.printf("RMQ(0, 6) = %d\n", st.rmq(0, 6)); // answer = index 5
  89:
  90:     System.out.printf("              idx    0, 1, 2, 3, 4,  5, 6\n");
  91:     System.out.printf("Now, modify A into {18,17,13,19,15,100,20}\n");
  92:     st.update_point(5, 100);                  // update A[5] from 11 to 100
  93:     System.out.printf("These values do not change\n");
  94:     System.out.printf("RMQ(1, 3) = %d\n", st.rmq(1, 3));               // 2
  95:     System.out.printf("RMQ(3, 4) = %d\n", st.rmq(3, 4));               // 4
  96:     System.out.printf("RMQ(0, 0) = %d\n", st.rmq(0, 0));               // 0
  97:     System.out.printf("RMQ(0, 1) = %d\n", st.rmq(0, 1));               // 1
  98:     System.out.printf("These values change\n");
  99:     System.out.printf("RMQ(0, 6) = %d\n", st.rmq(0, 6));           // 5->2
 100:     System.out.printf("RMQ(4, 6) = %d\n", st.rmq(4, 6));           // 5->4
 101:     System.out.printf("RMQ(4, 5) = %d\n", st.rmq(4, 5));           // 5->4
 102:   }
 103: }
```

```java
/*****************************************************************************/
/** Fenwick Tree (Binary Indexed Tree, ou BIT) ............................. */
/*****************************************************************************/

import java.util.*;

class FenwickTree {
  private Vector<Integer> ft;

  private int LSOne(int S) { return (S & (-S)); }

  public FenwickTree() {}

  // initialization: n + 1 zeroes, ignore index 0
  public FenwickTree(int n) {
    ft = new Vector<Integer>();
    for (int i = 0; i <= n; i++) ft.add(0);
  }

  public int rsq(int b) {                                 // returns RSQ(1, b)
    int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
    return sum; }

  public int rsq(int a, int b) {                          // returns RSQ(a, b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }

  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {                       // note: n = ft.size() - 1
    for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v); }
};

class ch2_10_fenwicktree_ds {
  public static void main(String[] args) {

    FenwickTree ft =              // idx   0 1 2 3 4 5 6 7  8 9 10, no index 0!
      new FenwickTree(10);        // ft = {-,0,0,0,0,0,0,0,  0,0,0}
    ft.adjust(2, 1);              // ft = {-,0,1,0,1,0,0,0,  1,0,0}, idx 2,4,8 => +1
    ft.adjust(4, 1);              // ft = {-,0,1,0,2,0,0,0,  2,0,0}, idx 4,8 => +1
    ft.adjust(5, 2);              // ft = {-,0,1,0,2,2,2,0,  4,0,0}, idx 5,6,8 => +2
    ft.adjust(6, 3);              // ft = {-,0,1,0,2,2,5,0,  7,0,0}, idx 6,8 => +3
    ft.adjust(7, 2);              // ft = {-,0,1,0,2,2,5,2,  9,0,0}, idx 7,8 => +2
    ft.adjust(8, 1);              // ft = {-,0,1,0,2,2,5,2,10,0,0}, idx 8 => +1
    ft.adjust(9, 1);              // ft = {-,0,1,0,2,2,5,2,10,1,1}, idx 9,10 => +1
    System.out.printf("%d\n", ft.rsq(1, 1)); // 0 => ft[1] = 0
    System.out.printf("%d\n", ft.rsq(1, 2)); // 1 => ft[2] = 1
    System.out.printf("%d\n", ft.rsq(1, 6)); // 7 => ft[6] + ft[4] = 5 + 2 = 7
    System.out.printf("%d\n", ft.rsq(1, 10));// 11 => ft[10] + ft[8] = 1 + 10 = 11
    System.out.printf("%d\n", ft.rsq(3, 6)); // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1

    ft.adjust(5, 2); // update demo
    System.out.printf("%d\n", ft.rsq(1, 10)); // now 13
  }
}
```

```java
/*********************************************************************************/
/** Backtracking (exemplo) ..................................................... */
/*********************************************************************************/

import java.util.*;

class Main { /* 8 Queens Chess Problem */
  private static int[] row = new int[9];
  private static int TC, a, b, lineCounter;
  // it is ok to use global variables in competitive programming

  private static boolean place(int col, int tryrow) {
    for (int prev = 1; prev < col; prev++) // check previously placed queens
      if (row[prev] == tryrow ||
       (Math.abs(row[prev] - tryrow) == Math.abs(prev - col)))
        return false; // an infeasible solution if share same row or same diagonal
    return true;
  }

  private static void backtrack(int col) {
    for (int tryrow = 1; tryrow <= 8; tryrow++) // try all possible row
      if (place(col, tryrow)) { // if can place a queen at this col and row...
        row[col] = tryrow; // put this queen in this col and row
        if (col == 8 && row[b] == a){ // a candidate solution & (a, b) has 1 queen
          System.out.printf("%2d      %d", ++lineCounter, row[1]);
          for (int j = 2; j <= 8; j++) System.out.printf(" %d", row[j]);
          System.out.printf("\n");
        }
        else
          backtrack(col + 1); // recursively try next column
  }   }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    TC = sc.nextInt();
    while (TC-- > 0) {
      a = sc.nextInt();
      b = sc.nextInt();
      for (int i = 0; i < 9; i++) row[i] = 0;
      lineCounter = 0;
      System.out.printf("SOLN       COLUMN\n");
      System.out.printf(" #      1 2 3 4 5 6 7 8\n\n");
      backtrack(1); // generate all possible 8! candidate solutions
      if (TC > 0) System.out.printf("\n");
    }
  }
}
```

```java
/*********************************************************************************/
/** Programacao Dinamica (ex. 1: Top-Down) ..................................... */
/*********************************************************************************/

import java.util.*;

class Main { /* UVa 11450 - Wedding Shopping - Top Down */
  private static int M, C, K;
  private static int[][] price = new int[25][25],//price[g (<= 20)][model (<= 20)]
  private static int[][] memo = new int[210][25];//memo[money (<= 200)][g (<= 20)]

  private static int shop(int money, int g) {
    if(money < 0) return -1000000000; // fail, return a large negative number (1B)
    if(g == C)    return M - money;   // we have bought last garment, done
    if(memo[money][g] != -1) return memo[money][g]; // this state has been visited
    int ans = -1000000000;
    for (int model = 1; model <= price[g][0]; model++) // try all possible models
      ans = Math.max(ans, shop(money - price[g][model], g + 1));
    return memo[money][g] = ans; // assign ans to dp table + return it!
  }

  public static void main(String[] args) { // easy to code if you are already
    Scanner sc = new Scanner(System.in);   // familiar with it
    int i, j, TC, score;

    TC = sc.nextInt();
    while (TC-- > 0) {
      M = sc.nextInt(); C = sc.nextInt();
      for (i = 0; i < C; i++) {
        K = sc.nextInt();
        price[i][0] = K; // to simplify coding, we store K in price[i][0]
        for (j = 1; j <= K; j++)
          price[i][j] = sc.nextInt();
      }

      for (i = 0; i < 210; i++)
        for (j = 0; j < 25; j++)
          memo[i][j] = -1; // initialize DP memo table

      score = shop(M, 0); // start the top-down DP
      if (score < 0) System.out.printf("no solution\n");
      else           System.out.printf("%d\n", score);
    }
  }
}
```

```java
/*********************************************************************************/
/** Programacao Dinamica (ex. 2: Bottom-Up) ..................................... */
/*********************************************************************************/

import java.util.*;

class Main { /* UVa 11450 - Wedding Shopping - Bottom Up */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int i, j, l, TC, M, C, K;
    int[][] price = new int[25][25]; // price[g (<= 20)][model (<= 20)]
    Boolean[][] reachable = new Boolean[210][25];
                                     // reachable table[money (<= 200)][g (<= 20)]
    TC = sc.nextInt();
    while (TC-- > 0) {
      M = sc.nextInt(); C = sc.nextInt();
      for (i = 0; i < C; i++) {
        K = sc.nextInt();
        price[i][0] = K; // to simplify coding, we store K in price[i][0]
        for (j = 1; j <= K; j++)
          price[i][j] = sc.nextInt();
      }

      for (i = 0; i < 210; i++)
        for (j = 0; j < 25; j++)
          reachable[i][j] = false; // clear everything

      for (i = 1; i <= price[0][0]; i++) // initial values
        if (M - price[0][i] >= 0)
          reachable[M - price[0][i]][0] = true; // if only using 1st garment g = 0

      // for each remaining garment
      for (j = 1; j < C; j++) // (note: this is written in column major)
        for (i = 0; i < M; i++) if (reachable[i][j - 1]) //if can reach this state
          for (l = 1; l <= price[j][0]; l++) if (i - price[j][l] >= 0) // flag the
            reachable[i - price[j][l]][j] = true; //rest as long as it is feasible

      for (i = 0; i <= M && !reachable[i][C - 1]; i++); //answer is in last column

      if (i == M + 1)                          // nothing in this last column
        System.out.printf("no solution\n"); // has its bit turned on
      else
        System.out.printf("%d\n", M - i);
    }
  }
}
```

```
1: /*****************************************************************************/
2: /** Max 1D Range Sum ......................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6:
7: class ch3_04_Max1DRangeSum {
8:   public static void main(String[] args) {
9:     int n = 9, A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 };   // a sample array A
10:    int running_sum = 0, ans = 0;
11:    for (int i = 0; i < n; i++)                                    // O(n)
12:      if (running_sum + A[i] >= 0) {  // the overall running sum is still +ve
13:        running_sum += A[i];
14:        ans = Math.max(ans, running_sum);     // keep the largest RSQ overall
15:      }
16:      else        // the overall running sum is -ve, we greedily restart here
17:        running_sum = 0;      // because starting from 0 is better for future
18:                              // iterations than starting from -ve running sum
19:    System.out.printf("Max 1D Range Sum = %d\n", ans);         // should be 9
20: } }
```

```
1: /*********************************************************************************/
2: /** Maximum Sum ..................................................................... */
3: /*********************************************************************************/
4:
5: import java.util.*;
6:
7: class Main { /* Maximum Sum, 0.528s in UVa (C++ version runs in 0.008s) */
8:   public static void main(String[] args) {
9:     Scanner sc = new Scanner(System.in);
10:    int i, j, l, r, row, n, maxSubRect, subRect;
11:    int[][] A = new int[101][101];
12:
13:    n = sc.nextInt();
14:    for (i = 0; i < n; i++)
15:      for (j = 0; j < n; j++) {
16:        A[i][j] = sc.nextInt();
17:        if (j > 0) A[i][j] += A[i][j - 1]; // pre-processing
18:      }
19:
20:    maxSubRect = -127*100*100;    // the lowest possible value for this problem
21:    for (l = 0; l < n; l++) for (r = l; r < n; r++) {
22:      subRect = 0;
23:      for (row = 0; row < n; row++) {
24:        // Max 1D Range Sum on columns of this row i
25:        if (l > 0) subRect += A[row][r] - A[row][l - 1];
26:        else       subRect += A[row][r];
27:
28:        // Kadane's algorithm on rows
29:        if (subRect < 0) subRect = 0;
30:        maxSubRect = Math.max(maxSubRect, subRect);
31:    } }
32:
33:    System.out.printf("%d\n", maxSubRect);
34: } }
```

```java
/*****************************************************************************/
/** Longest Increasing Subsequence (LIS) ..................................... */
/*****************************************************************************/

import java.util.*;

class ch3_06_LIS {
  static void reconstruct_print(int end, int[] a, int[] p) {
    int x = end;
    Stack<Integer> s = new Stack();
    for (; p[x] >= 0; x = p[x]) s.push(a[x]);
    System.out.printf("[%d", a[x]);
    for (; !s.isEmpty(); s.pop()) System.out.printf(", %d", s.peek());
    System.out.printf("]\n");
  }

  public static void main(String[] args) {
    final int MAX_N = 100000;

    int n = 11;
    int[] A = new int[] {-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4};
    int[] L_id = new int[MAX_N], P = new int[MAX_N];
    Vector<Integer> L = new Vector<Integer>();

    int lis = 0, lis_end = 0;
    for (int i = 0; i < n; ++i) {
      int pos = Collections.binarySearch(L, A[i]);
      if (pos < 0) pos = -(pos + 1); // some adjustments are needed
      if (pos >= L.size()) L.add(A[i]);
      else                 L.set(pos, A[i]);
      L_id[pos] = i;
      P[i] = pos > 0 ? L_id[pos - 1] : -1;
      if (pos + 1 > lis) {
        lis = pos + 1;
        lis_end = i;
      }

      System.out.printf("Considering element A[%d] = %d\n", i, A[i]);
      System.out.printf("LIS ending at A[%d] is of length %d: ", i, pos + 1);
      reconstruct_print(i, A, P);
      System.out.println("L is now: " + L);
      System.out.printf("\n");
    }

    System.out.printf("Final LIS is of length %d: ", lis);
    reconstruct_print(lis_end, A, P);
  }
}
```

```
/*********************************************************************************/
/** Algoritmo da Mochila 0-1 ..................................................... */
/*********************************************************************************/

import java.util.*;

class Main { /* SuperSale */
  // 0-1 Knapsack DP (Top-Down) - faster as not all states are visited

  private static final int MAX_N = 1010;
  private static final int MAX_W = 40;
  private static int N, MW;
  private static int[] V = new int[MAX_N], W = new int[MAX_N];
  private static int[][] memo = new int[MAX_N][MAX_W];

  private static int value(int id, int w) {
    if (id == N || w == 0) return 0;
    if (memo[id][w] != -1) return memo[id][w];
    if (W[id] > w)         return memo[id][w] = value(id + 1, w);
    return memo[id][w] = Math.max(value(id + 1, w), V[id] + value(id + 1, w -
W[id]));
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int i, j, T, G, ans;

    T = sc.nextInt();
    while (T-- > 0) {
      for (i = 0; i < MAX_N; i++)
        for (j = 0; j < MAX_W; j++)
          memo[i][j] = -1;

      N = sc.nextInt();
      for (i = 0; i < N; i++) {
        V[i] = sc.nextInt();
        W[i] = sc.nextInt();
      }

      ans = 0;
      G = sc.nextInt();
      while (G-- > 0) {
        MW = sc.nextInt();
        ans += value(0, MW);
      }

      System.out.printf("%d\n", ans);
    }
  }
}
```

```
1: /*****************************************************************************/
2: /** Coin Change (Problema do Troco) ..... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: class Main { /* Coin Change, 1.492s in Java, 0.038s in C++ */
9:   // O(NV) DP solution
10:
11:   // N and coinValue are fixed for this problem, max V is 7489
12:   private static int N = 5, V;
13:   private static int[] coinValue = new int[] {1, 5, 10, 25, 50};
14:   private static int[][] memo = new int[6][7500];
15:
16:   private static int ways(int type, int value) {
17:     if (value == 0)               return 1;
18:     if (value < 0 || type == N)   return 0;
19:     if (memo[type][value] != -1) return memo[type][value];
20:     return memo[type][value] = ways(type + 1, value) +
21:       ways(type, value - coinValue[type]);
22:   }
23:
24:   public static void main(String[] args) throws Exception {
25:     // This solution is TLE without using BufferedReader and PrintWriter
26:     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
27:     PrintWriter pr = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));
28:
29:     for (int i = 0; i < 6; i++)
30:       for (int j = 0; j < 7500; j++)
31:         memo[i][j] = -1; // we only need to initialize this once
32:
33:     while (true) {
34:       String line = br.readLine();
35:       if (line == null) break;
36:       V = Integer.parseInt(line);
37:       pr.printf("%d\n", ways(0, V));
38:     }
39:
40:     pr.close(); // do not forget to do this
41:   }
42: }
```

```java
/*********************************************************************************/
/** Problema do Caixeiro Viajante ..... */
/*********************************************************************************/

import java.util.*;

class Main { /* Collecting Beepers */
  // DP TSP

  private static int i, j, TC, xsize, ysize, n;
  private static int[] x = new int[11], y = new int[11]; // Karel + max 10 beepers
  private static int[][] dist = new int[11][11], memo = new int[11][1 << 11];

  private static int tsp(int pos, int bitmask) { // bitmask stores the visited
    if (bitmask == (1 << (n + 1)) - 1)                    // coordinates
      return dist[pos][0]; // return trip to close the loop
    if (memo[pos][bitmask] != -1)
      return memo[pos][bitmask];

    int ans = 2000000000;
    for (int nxt = 0; nxt <= n; nxt++) // O(n) here
      // if coordinate nxt is not visited yet
      if (nxt != pos && (bitmask & (1 << nxt)) == 0)
        ans = Math.min(ans, dist[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));
    return memo[pos][bitmask] = ans;
  }

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

    TC = sc.nextInt();
    while (TC-- > 0) {
      xsize = sc.nextInt(); ysize = sc.nextInt(); // these two values are not used
      x[0] = sc.nextInt(); y[0] = sc.nextInt();
      n = sc.nextInt();
      for (i = 1; i <= n; i++) { // karel's position is at index 0
        x[i] = sc.nextInt();
        y[i] = sc.nextInt();
      }

      for (i = 0; i <= n; i++) // build distance table
        for (j = 0; j <= n; j++)                                  // Manhattan
          dist[i][j] = Math.abs(x[i] - x[j]) + Math.abs(y[i] - y[j]); //  distance

      for (i = 0; i < 11; i++)
        for (j = 0; j < (1 << 11); j++)
          memo[i][j] = -1;

      System.out.printf("The shortest path has length %d\n", tsp(0, 1)); // DP-TSP
    }
  }
}
```

```java
/*********************************************************************************/
/** Qtd. de formas de obter um numero N somando K numeros ..................... */
/*********************************************************************************/

import java.util.*;

class Main { /* How do you add? */
  // top-down

  private static int N, K;
  private static int[][] memo = new int[110][110];

  private static int ways(int N, int K) {
    if (K == 1) // only can use 1 number to add up to N
      return 1; // the answer is definitely 1, that number itself
    else if (memo[N][K] != -1)
      return memo[N][K];

    // if K > 1, we can choose one number from [0..N] to be one of the number
    // and recursively compute the rest
    int total_ways = 0;
    for (int split = 0; split <= N; split++)                  // we just need the
      total_ways = (total_ways + ways(N - split, K - 1)) % 1000000; // modulo 1M
    return memo[N][K] = total_ways; // memoize them
  }

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

    for (int i = 0; i < 110; i++)
      for (int j = 0; j < 110; j++)
        memo[i][j] = -1;

    while (true) {
      N = sc.nextInt();
      K = sc.nextInt();
      if (N == 0 && K == 0)
        break;
      System.out.println(ways(N, K)); // some recursion formula + top down DP
    }
  }
}
```

```java
/*****************************************************************************/
/** Cutting Sticks ..... */
/*****************************************************************************/
import java.util.*;

class Main { /* Cutting Sticks, 1.762s in Java, 0.302s in C++ */
  // Top-Down DP

  private static int[] arr = new int[55];
  private static int[][] memo = new int[55][55];

  private static int cut(int left, int right) {
    if (left + 1 == right)       return 0;
    if (memo[left][right] != -1) return memo[left][right];

    int ans = 2000000000;
    for (int i = left + 1; i < right; i++)
     ans = Math.min(ans, cut(left, i) + cut(i, right) + (arr[right] - arr[left]));
    return memo[left][right] = ans;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int i, j, l, n;

    while (true) {
      l = sc.nextInt();
      if (l == 0)
        break;

      arr[0] = 0;
      n = sc.nextInt();
      for (i = 1; i <= n; i++)
        arr[i] = sc.nextInt();
      arr[n + 1] = l;

      for (i = 0; i < 55; i++)
        for (j = 0; j < 55; j++)
          memo[i][j] = -1;

      // start with left = 0 and right = n + 1
      System.out.printf("The minimum cutting is %d.\n", cut(0, n + 1));
    }
  }
}
```

```
1: /*****************************************************************************/
2: /** DFS (Busca em Profundidade) ..... */
3: /*****************************************************************************/
4:
5: import java.util.Collections;
6: import java.util.Iterator;
7: import java.util.Scanner;
8: import java.util.Stack;
9: import java.util.Vector;
10: import java.io.*;
11:
12: public class ch4_01_dfs {
13:   private static final int DFS_WHITE = -1; // normal DFS
14:   private static final int DFS_BLACK = 1;
15:   private static final int DFS_GRAY = 2;
16:   private static Vector < Vector < IntegerPair > > AdjList =
17:             new Vector < Vector < IntegerPair > >();
18:   private static Vector < Integer > dfs_num, dfs_low, dfs_parent;
19:   private static Vector < Boolean > articulation_vertex, visited;
20:   private static Stack < Integer > S; // additional information for SCC
21:   private static Vector<Integer> topologicalSort; // additional info. for toposort
22:   private static int numComp, dfsNumberCounter, dfsRoot, rootChildren;
23:
24:   private static void initDFS(int V) { // used in normal DFS
25:     dfs_num = new Vector < Integer > ();
26:     dfs_num.addAll(Collections.nCopies(V, DFS_WHITE));
27:     numComp = 0;
28:   }
29:
30:   private static void initGraphCheck(int V) {
31:     initDFS(V);
32:     dfs_parent = new Vector < Integer > ();
33:     dfs_parent.addAll(Collections.nCopies(V, 0));
34:     numComp = 0;
35:   }
36:
37:   private static void initArticulationPointBridge(int V) {
38:     initGraphCheck(V);
39:     dfs_low = new Vector < Integer > ();
40:     dfs_low.addAll(Collections.nCopies(V, 0));
41:     articulation_vertex = new Vector < Boolean > ();
42:     articulation_vertex.addAll(Collections.nCopies(V, false));
43:     dfsNumberCounter = 0;
44:   }
45:
46:   private static void initTarjanSCC(int V) {
47:     initGraphCheck(V);
48:     dfs_low = new Vector < Integer > ();
49:     dfs_low.addAll(Collections.nCopies(V, 0));
50:     dfsNumberCounter = 0;
51:     numComp = 0;
52:     S = new Stack < Integer > ();
53:     visited = new Vector < Boolean > ();
54:     visited.addAll(Collections.nCopies(V, false));
55:   }
56:
57:   private static void initTopologicalSort(int V) {
58:     initDFS(V);
59:     topologicalSort = new Vector < Integer > ();
60:   }
61:
62:   private static void dfs(int u) { // DFS for normal usage
63:     System.out.printf(" %d", u); // this vertex is visited
64:     dfs_num.set(u, DFS_BLACK); // mark as visited
65:     Iterator it = AdjList.get(u).iterator();
66:     while (it.hasNext()) { // try all neighbors v of vertex u
67:       IntegerPair v = (IntegerPair)it.next();
68:       if (dfs_num.get(v.first()) == DFS_WHITE) // avoid cycle
69:         dfs(v.first()); // v is a (neighbor, weight) pair
70:     }
```

```
71:   }
72:
73:   private static void floodfill(int u, int color) {
74:     dfs_num.set(u, color); // not just a generic DFS_BLACK
75:     Iterator it = AdjList.get(u).iterator();
76:     while (it.hasNext()) { // try all neighbors v of vertex u
77:       IntegerPair v = (IntegerPair)it.next();
78:       if (dfs_num.get(v.first()) == DFS_WHITE) // avoid cycle
79:         floodfill(v.first(), color); // v is a (edge, weight) pair
80:     }
81:   }
82:
83:   // DFS for checking graph edge properties...
84:   private static void graphCheck(int u) {
85:     dfs_num.set(u, DFS_GRAY); // color this as DFS_GRAY (temporary)
86:     Iterator it = AdjList.get(u).iterator();
87:     while (it.hasNext()) { // traverse this AdjList
88:       IntegerPair v = (IntegerPair)it.next();
89:       if (dfs_num.get(v.first()) == DFS_WHITE) { // DFS_GRAY to DFS_WHITE
90:         // System.out.printf("  Tree Edge (%d, %d)\n", u, v.first());
91:         dfs_parent.set(v.first(), u); // parent of this children is me
92:         graphCheck(v.first());
93:       }
94:       else if (dfs_num.get(v.first()) == DFS_GRAY) { // DFS_GRAY to DFS_GRAY
95:         if (v.first() == dfs_parent.get(u))
96:           System.out.printf(" Bidirectional Edge (%d, %d) - (%d, %d)\n", u,
97:             v.first(), v.first(), u);
98:         else
99:           System.out.printf(" Back Edge (%d, %d) (Cycle)\n", u, v.first());
100:      }
101:      else if (dfs_num.get(v.first()) == DFS_BLACK) // DFS_GRAY to DFS_BLACK
102:        System.out.printf(" Forward/Cross Edge (%d, %d)\n", u, v.first());
103:    }
104:    dfs_num.set(u, DFS_BLACK); // now color this as DFS_BLACK (DONE)
105:  }
106:
107:  private static void articulationPointAndBridge(int u) {
108:    dfs_low.set(u, dfsNumberCounter);
109:    dfs_num.set(u, dfsNumberCounter++); // dfs_low[u] <= dfs_num[u]
110:    Iterator it = AdjList.get(u).iterator();
111:    while (it.hasNext()) { // try all neighbors v of vertex u
112:      IntegerPair v = (IntegerPair)it.next();
113:      if (dfs_num.get(v.first()) == DFS_WHITE) { // a tree edge
114:        dfs_parent.set(v.first(), u); // parent of this children is me
115:        if (u == dfsRoot) // special case
116:          rootChildren++; // count children of root
117:        articulationPointAndBridge(v.first());
118:        if (dfs_low.get(v.first()) >= dfs_num.get(u)) // for articulation point
119:          articulation_vertex.set(u, true); // store this information first
120:        if (dfs_low.get(v.first()) > dfs_num.get(u)) // for bridge
121:          System.out.printf(" Edge (%d, %d) is a bridge\n", u, v.first());
122:        dfs_low.set(u, Math.min(dfs_low.get(u), dfs_low.get(v.first())));
123:        // update dfs_low[u]
124:      }
125:      else if (v.first() != dfs_parent.get(u)) // a back edge and not direct cycle
126:        dfs_low.set(u, Math.min(dfs_low.get(u), dfs_num.get(v.first())));
127:        // update dfs_low[u]
128:    }
129:  }
130:
131:  private static void tarjanSCC(int u) {
132:    dfs_num.set(u, dfsNumberCounter);
133:    dfs_low.set(u, dfsNumberCounter++); // dfs_low[u] <= dfs_num[u]
134:    S.push(u); // store u according to order of visitation
135:    visited.set(u, true);
136:
137:    Iterator it = AdjList.get(u).iterator();
138:    while (it.hasNext()) { // try all neighbors v of vertex u
139:      IntegerPair v = (IntegerPair)it.next();
140:      if (dfs_num.get(v.first()) == DFS_WHITE) // a tree edge
```

```
141:         tarjanSCC(v.first());
142:       if (visited.get(v.first())) // condition for update
143:         dfs_low.set(u, Math.min(dfs_low.get(u), dfs_low.get(v.first())));
144:     }
145:
146:     if (dfs_low.get(u) == dfs_num.get(u)) { // if this is a root (start) of an SCC
147:       System.out.printf("SCC %d: ", ++numComp);
148:       while (true) {
149:         int v = S.peek(); S.pop(); visited.set(v, false);
150:         System.out.printf(" %d", v);
151:         if (u == v) break;
152:       }
153:       System.out.printf("\n");
154:     }
155:   }
156:
157:   private static void topoVisit(int u) {
158:     dfs_num.set(u, DFS_BLACK);
159:     Iterator it = AdjList.get(u).iterator();
160:     while (it.hasNext()) {
161:       IntegerPair v = (IntegerPair)it.next();
162:       if (dfs_num.get(v.first()) == DFS_WHITE)
163:         topoVisit(v.first());
164:     }
165:     topologicalSort.add(u);
166:   }
167:
168:   private static void printThis(String message) {
169:     System.out.printf("==================================\n");
170:     System.out.printf("%s\n", message);
171:     System.out.printf("==================================\n");
172:   }
173:
174:   public static void main(String[] args) throws Exception {
175:     int i, j, V, total_neighbors, id, weight;
176:
177:     File f = new File("in_01.txt");
178:     Scanner sc = new Scanner(f);
179:
180:     V = sc.nextInt();
181:     AdjList.clear();
182:     for (i = 0; i < V; i++) {
183:       Vector < IntegerPair > Neighbor = new Vector < IntegerPair >();
184:       // create vector of pair<int, int>
185:       AdjList.add(Neighbor); // store blank vector first
186:     }
187:
188:     for (i = 0; i < V; i++) {
189:       total_neighbors = sc.nextInt();
190:       for (j = 0; j < total_neighbors; j++) {
191:         id = sc.nextInt();
192:         weight = sc.nextInt();
193:         AdjList.get(i).add(new IntegerPair(id, weight));
194:       }
195:     }
196:
197:     initDFS(V); // call this first before running DFS
198:     printThis("Standard DFS Demo (the input graph must be UNDIRECTED)");
199:     for (i = 0; i < V; i++) // for each vertex i in [0..V-1]
200:       if (dfs_num.get(i) == DFS_WHITE) { // if not visited yet
201:         System.out.printf("Component %d, visit:", ++numComp);
202:         dfs(i);
203:         System.out.printf("\n");
204:       }
205:     System.out.printf("There are %d connected components\n", numComp);
206:
207:     initDFS(V); // call this first before running DFS
208:     printThis("Flood Fill Demo (the input graph must be UNDIRECTED)");
209:     for (i = 0; i < V; i++) // for each vertex i in [0..V-1]
210:       if (dfs_num.get(i) == DFS_WHITE) // if not visited yet
```

```
211:         floodfill(i, ++numComp);
212:     for (i = 0; i < V; i++)
213:       System.out.printf("Vertex %d has color %d\n", i, dfs_num.get(i));
214:
215:     // make sure that the given graph is DAG
216:     initTopologicalSort(V);
217:     printThis("Topological Sort (the input graph must be DAG)");
218:     for (i = 0; i < V; i++)
219:       if (dfs_num.get(i) == DFS_WHITE)
220:         topoVisit(i);
221:     for (i = topologicalSort.size() - 1; i >= 0; i--) // access from back to front
222:       System.out.printf(" %d", topologicalSort.get(i));
223:     System.out.printf("\n");
224:
225:     initGraphCheck(V);
226:     printThis("Graph Edges Property Check");
227:     for (i = 0; i < V; i++)
228:       if (dfs_num.get(i) == DFS_WHITE) {
229:         System.out.printf("Component %d:\n", ++numComp);
230:         graphCheck(i);
231:       }
232:
233:     initArticulationPointBridge(V);
234:     printThis("Articulation Points & Bridges (the input graph must be UNDIRECTED)"
);
235:     System.out.printf("Bridges:\n");
236:     for (i = 0; i < V; i++)
237:       if (dfs_num.get(i) == DFS_WHITE) {
238:         dfsRoot = i; rootChildren = 0;
239:         articulationPointAndBridge(i);
240:         articulation_vertex.set(dfsRoot, (rootChildren > 1)); // special case
241:       }
242:
243:     System.out.printf("Articulation Points:\n");
244:     for (i = 0; i < V; i++)
245:       if (articulation_vertex.get(i))
246:         System.out.printf(" Vertex %d\n", i);
247:
248:     initTarjanSCC(V);
249:     printThis("Strongly Connected Components (the input graph must be DIRECTED)");
250:     for (i = 0; i < V; i++)
251:       if (dfs_num.get(i) == DFS_WHITE)
252:         tarjanSCC(i);
253:   }
254: }
```

```java
/*********************************************************************************/
/** Flood Fill / grafo implicito em matriz ..................................... */
/*********************************************************************************/

import java.util.*;
import java.text.*;

// classic DFS flood fill

class Main { /* UVa 469 - Wetlands of Florida, 0.659s in Java, 0.162s in C++ */
  private static String line;
  private static char[][] grid = new char[150][];
  private static int TC, R, C, row, col;

  private static int dr[] = {1,1,0,-1,-1,-1, 0, 1}; // S,SE,E,NE,N,NW,W,SW
  private static int dc[] = {0,1,1, 1, 0,-1,-1,-1}; // neighbors

  private static int floodfill(int r, int c, char c1, char c2) {
    if (r<0 || r>=R || c<0 || c>=C) return 0; // outside
    if (grid[r][c] != c1) return 0; // we want only c1
    grid[r][c] = c2; // important step to avoid cycling!
    int ans = 1; // coloring c1 -> c2, add 1 to answer
    for (int d = 0; d < 8; d++) // recurse to neighbors
      ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    return ans;
  }

  // inside the void main(String[] args) of the
  // solution for UVa 469 - Wetlands of Florida
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // read the implicit graph as global 2D array 'grid'/R/C
    // and (row, col) query coordinate
    TC = sc.nextInt(); sc.nextLine();
    sc.nextLine(); // remove dummy line

    while (TC-- > 0) {
      R = 0;
      while (true) {
        grid[R] = sc.nextLine().toCharArray();
        if (grid[R][0] != 'L' && grid[R][0] != 'W') // start of query
          break;
        R++;
      }
      C = grid[0].length;


      line = new String(grid[R]);
      while (true) {
        StringTokenizer st = new StringTokenizer(line);
        row = Integer.parseInt(st.nextToken()); row--; // index starts from 0!
        col = Integer.parseInt(st.nextToken()); col--;
        System.out.println(floodfill(row, col, 'W', '.'));
        // change water 'W' to '.'; count size of this lake
        floodfill(row, col, '.', 'W'); // restore for next query
        if (sc.hasNext()) line = sc.nextLine();
        else              break; // last test case
        if (line.length() == 0) break; // next test case
      }

      if (TC > 0)
        System.out.println();
    }
  }
}
```

```
1: /*****************************************************************************/
2: /** Kruskal e Prim (Arvore Geradora Minima) ..................................*/
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: // Union-Find Disjoint Sets Library written in OOP manner,
9: // using both path compression and union by rank heuristics
10: class UnionFind {                                               // OOP style
11:   private Vector<Integer> p, rank, setSize;
12:   private int numSets;
13:
14:   public UnionFind(int N) {
15:     p = new Vector<Integer>(N);
16:     rank = new Vector<Integer>(N);
17:     setSize = new Vector<Integer>(N);
18:     numSets = N;
19:     for (int i = 0; i < N; i++) {
20:       p.add(i);
21:       rank.add(0);
22:       setSize.add(1);
23:     }
24:   }
25:
26:   public int findSet(int i) {
27:     if (p.get(i) == i) return i;
28:     else {
29:       int ret = findSet(p.get(i)); p.set(i, ret);
30:       return ret; } }
31:
32:   public Boolean isSameSet(int i, int j) { return findSet(i) == findSet(j); }
33:
34:   public void unionSet(int i, int j) {
35:     if (!isSameSet(i, j)) { numSets--;
36:     int x = findSet(i), y = findSet(j);
37:     // rank is used to keep the tree short
38:     if (rank.get(x) > rank.get(y)) {
39:                 p.set(y, x); setSize.set(x, setSize.get(x) + setSize.get(y)); }
40:     else      { p.set(x, y); setSize.set(y, setSize.get(y) + setSize.get(x));
41:                 if (rank.get(x) == rank.get(y)) rank.set(y, rank.get(y) + 1); } } }
42:   public int numDisjointSets() { return numSets; }
43:   public int sizeOfSet(int i) { return setSize.get(findSet(i)); }
44: }
45:
46: public class ch4_03_kruskal_prim {
47:   static Vector<Vector<IntegerPair>> AdjList = new Vector<Vector<IntegerPair>>();
48:   static Vector<Boolean> taken = new Vector<Boolean>(); // global boolean flag
49:                                                         // to avoid cycle
50:   static PriorityQueue<IntegerPair> pq = new PriorityQueue<IntegerPair>();
51:                                       // priority queue to help choose shorter edges
52:
53:   static void process(int vtx) { //  we do not need to use negative sign to
54:     taken.set(vtx, true);        //  reverse the sort order
55:     for (int j = 0; j < (int)AdjList.get(vtx).size(); j++) {
56:       IntegerPair v = AdjList.get(vtx).get(j);
57:       if (!taken.get(v.first()))
58:         pq.offer(new IntegerPair(v.second(), v.first()));
59:   } }
60:
61:   public static void main(String[] args) throws Exception {
62:     int V, E, u, v, w;
63:
64:     /*
65:     // Graph in Figure 4.10 left, format: list of weighted edges
66:     // This example shows another form of reading graph input
67:     5 7
68:     0 1 4
69:     0 2 4
70:     0 3 6
```

```
71:     0 4 6
72:     1 2 2
73:     2 3 8
74:     3 4 9
75:     */
76:
77:     File f = new File("in_03.txt");
78:     Scanner sc = new Scanner(f);
79:
80:     V = sc.nextInt();
81:     E = sc.nextInt();
82:     // Kruskal's algorithm merged with Prim's algorithm
83:
84:     AdjList.clear();
85:     for (int i = 0; i < V; i++) {
86:       Vector < IntegerPair > Neighbor = new Vector < IntegerPair >();
87:       // create vector of pair<int, int>
88:       AdjList.add(Neighbor); // store blank vector first
89:     }
90:     Vector<IntegerTriple> EdgeList = new Vector<IntegerTriple>();
91:
92:     // sort by edge weight O(E log E)
93:     // PQ default: sort descending. Trick: use <(negative) weight(i, j), <i, j> >
94:     for (int i = 0; i < E; i++) {
95:       u = sc.nextInt();
96:       v = sc.nextInt();
97:       w = sc.nextInt();
98:       EdgeList.add(new IntegerTriple(w, u, v));                // (w, u, v)
99:       AdjList.get(u).add(new IntegerPair(v, w));
100:      AdjList.get(v).add(new IntegerPair(u, w));
101:    }
102:    Collections.sort(EdgeList);
103:
104:    int mst_cost = 0;           // all V are disjoint sets at the beginning
105:    UnionFind UF = new UnionFind(V);
106:    for (int i = 0; i < E; i++) {                    // for each edge, O(E)
107:      IntegerTriple front = EdgeList.get(i);
108:      if (!UF.isSameSet(front.second(), front.third())) {          // check
109:        mst_cost += front.first();            // add the weight of e to MST
110:        UF.unionSet(front.second(), front.third());            // link them
111:    } }
112:
113:    // note: the number of disjoint sets must eventually be 1 for a valid MST
114:    System.out.printf("MST cost = %d (Kruskal's)\n", mst_cost);
115:
116:
117:
118:  // inside int main() --- assume the graph is stored in AdjList, pq is empty
119:    for (int i = 0; i < V; i++)
120:      taken.add(false);                // no vertex is taken at the beginning
121:    process(0);   // take vertex 0 and process all edges incident to vertex 0
122:    mst_cost = 0;
123:    while (!pq.isEmpty()) { // repeat until V vertices (E=V-1 edges) are taken
124:      IntegerPair front = pq.peek(); pq.poll();
125:      u = front.second(); w = front.first();   // no need to negate id/weight
126:      if (!taken.get(u)) {           // we have not connected this vertex yet
127:        mst_cost += w;
128:        process(u); // take u, process all edges incident to u
129:      }
130:    }                                          // each edge is in pq only once!
131:    System.out.printf("MST cost = %d (Prim's)\n", mst_cost);
132:  }
133: }
```

```
1: /*****************************************************************************/
2: /** BFS (Busca em Largura/Amplitude) ..................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: public class ch4_04_bfs {
9:   private static int V, E, a, b, s;
10:  private static Vector< Vector< IntegerPair > > AdjList =
11:            new Vector< Vector< IntegerPair > >();
12:  private static Vector < Integer > p = new Vector < Integer > ();
13:
14:  private static void printpath(int u) {
15:    if (u == s) { System.out.printf("%d", u); return; }
16:    printpath(p.get(u));
17:    System.out.printf(" %d", u);
18:  }
19:
20:  public static void main(String[] args) throws Exception {
21:    /*
22:    // Graph in Figure 4.3, format: list of unweighted edges
23:    // This example shows another form of reading graph input
24:    13 16
25:    0 1    1 2    2 3    0 4    1 5    2 6    3 7    5 6
26:    4 8    8 9    5 10   6 11   7 12   9 10   10 11  11 12
27:    */
28:
29:    File f = new File("in_04.txt");
30:    Scanner sc = new Scanner(f);
31:
32:    V = sc.nextInt();
33:    E = sc.nextInt();
34:
35:    AdjList.clear();
36:    for (int i = 0; i < V; i++) {
37:      Vector< IntegerPair > Neighbor = new Vector < IntegerPair >();
38:      AdjList.add(Neighbor); // add neighbor list to Adjacency List
39:    }
40:
41:    for (int i = 0; i < E; i++) {
42:      a = sc.nextInt();
43:      b = sc.nextInt();
44:      AdjList.get(a).add(new IntegerPair(b, 0));
45:      AdjList.get(b).add(new IntegerPair(a, 0));
46:    }
47:
48:    // as an example, we start from this source, see Figure 4.3
49:    s = 5;
50:
51:    // BFS routine
52:    // inside void main(String[] args) -- we do not use recursion,
53:    // thus we do not need to create separate function!
54:    Vector<Integer> dist = new Vector<Integer>();
55:    dist.addAll(Collections.nCopies(V, 1000000000));
56:    dist.set(s, 0); // start from source
57:    Queue<Integer> q = new LinkedList<Integer>(); q.offer(s);
58:    p.clear();
59:    p.addAll(Collections.nCopies(V, -1));
60:    // to store parent information (p must be a global variable!)
61:    int layer = -1; // for our output printing purpose
62:    Boolean isBipartite = true;
63:
64:    while (!q.isEmpty()) {
65:      int u = q.poll(); // queue: layer by layer!
66:      if (dist.get(u) != layer) System.out.printf("\nLayer %d:", dist.get(u));
67:      layer = dist.get(u);
68:      System.out.printf(", visit %d", u);
69:      Iterator it = AdjList.get(u).iterator();
70:      while (it.hasNext()) { // for each neighbours of u
```

```
71:         IntegerPair v = (IntegerPair)it.next();
72:         if (dist.get(v.first()) == 1000000000) { // if v not visited before
73:           dist.set(v.first(), dist.get(u) + 1); // then v is reachable from u
74:           q.offer(v.first()); // enqueue v for next steps
75:           p.set(v.first(), u); // parent of v is u
76:         }
77:         else if ((dist.get(v.first()) % 2) == (dist.get(u) % 2)) // same parity
78:           isBipartite = false;
79:       }
80:     }
81:
82:     System.out.printf("\nShortest path: ");
83:     printpath(7); System.out.printf("\n");
84:     System.out.printf("isBipartite? %d\n", isBipartite ? 1 : 0);
85:   }
86: }
```

```
1: /*****************************************************************************/
2: /** Dijkstra ................................................................ */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: public class ch4_05_dijkstra {
9:   public static final int INF = 1000000000; //10^9, not MAX_INT, to avoid overflow
10:   private static Vector< Vector< IntegerPair > > AdjList =
11:     new Vector< Vector< IntegerPair > >();
12:
13:   public static void main(String[] args) throws Exception {
14:     int V, E, s, u, v, w;
15:
16:     /*
17:     // Graph in Figure 4.17
18:     5 7 2
19:     2 1 2
20:     2 3 7
21:     2 0 6
22:     1 3 3
23:     1 4 6
24:     3 4 5
25:     0 4 1
26:     */
27:
28:     File f = new File("in_05.txt");
29:     Scanner sc = new Scanner(f);
30:     V = sc.nextInt();
31:     E = sc.nextInt();
32:     s = sc.nextInt();
33:
34:     AdjList.clear();
35:     for (int i = 0; i < V; i++) {
36:       Vector< IntegerPair > Neighbor =
37:         new Vector < IntegerPair >();
38:       AdjList.add(Neighbor); // add neighbor list to Adjacency List
39:     }
40:
41:     for (int i = 0; i < E; i++) {
42:       u = sc.nextInt();
43:       v = sc.nextInt();
44:       w = sc.nextInt();
45:       AdjList.get(u).add(new IntegerPair (v, w)); // first time using weight
46:     }
47:
48:     // Dijkstra routine
49:     Vector < Integer > dist = new Vector < Integer > ();
50:     dist.addAll(Collections.nCopies(V, INF)); // INF = 1*10^9,
51:     dist.set(s, 0);                           // not MAX_INT, to avoid overflow
52:     PriorityQueue < IntegerPair > pq = new PriorityQueue < IntegerPair >(1,
53:       new Comparator< IntegerPair >() { // overriding the compare method
54:         public int compare(IntegerPair i, IntegerPair j) {
55:           return i.first() - j.first();
56:         }
57:       }
58:     );
59:     pq.offer(new IntegerPair (0, s)); // sort based on increasing distance
60:
61:     while (!pq.isEmpty()) { // main loop
62:       IntegerPair top = pq.poll(); // greedy: pick shortest unvisited vertex
63:       int d = top.first(); u = top.second();
64:       if(d > dist.get(u)) continue; // This check is important! We want to process
65:       Iterator it = AdjList.get(u).iterator();    // vertex u only once but we can
66:       while (it.hasNext()) { // all outgoing edges from u
67:         IntegerPair p = (IntegerPair) it.next();
68:         v = p.first();
69:         int weight_u_v = p.second();
70:         // (note: Record SP spanning tree here if needed.)
```

```
71:         // (This is similar as printpath in BFS)
72:         if (dist.get(u) + weight_u_v < dist.get(v)) { // if can relax
73:           dist.set(v, dist.get(u) + weight_u_v); // relax
74:           pq.offer(new IntegerPair(dist.get(v), v));
75:           // enqueue this neighbor regardless whether it is already in pq or not

76:     } } }
77:
78:     for (int i = 0; i < V; i++) // index + 1 for final answer
79:       System.out.printf("SSSP(%d, %d) = %d\n", s + 1, i + 1, dist.get(i));
80:   }
81: }
```

```
1: /*****************************************************************************/
2: /** Bellman-Ford ..................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: public class ch4_06_bellman_ford {
9:   public static final int INF = 1000000000;
10:   private static Vector< Vector< IntegerPair > > AdjList =
11:             new Vector< Vector< IntegerPair > >();
12:
13:   public static void main(String[] args) throws Exception {
14:     int V, E, s, a, b, w;
15:
16:     /*
17:     // Graph in Figure 4.18, no negative cycle
18:     5 5 0
19:     0 1 1
20:     0 2 10
21:     1 3 2
22:     2 3 -10
23:     3 4 3
24:
25:     // Graph in Figure 4.19, negative cycle exists
26:     3 3 0
27:     0 1 1000
28:     1 2 15
29:     2 1 -42
30:     */
31:
32:     File f = new File("in_06.txt");
33:     Scanner sc = new Scanner(f);
34:
35:     V = sc.nextInt();
36:     E = sc.nextInt();
37:     s = sc.nextInt();
38:
39:     AdjList.clear();
40:     for (int i = 0; i < V; i++) {
41:       Vector< IntegerPair > Neighbor =
42:         new Vector < IntegerPair >();
43:       AdjList.add(Neighbor); // add neighbor list to Adjacency List
44:     }
45:
46:     for (int i = 0; i < E; i++) {
47:       a = sc.nextInt();
48:       b = sc.nextInt();
49:       w = sc.nextInt();
50:       AdjList.get(a).add(new IntegerPair(b, w)); // first time using weight
51:     }
52:
53:     // as an example, we start from this source (see Figure 1.15)
54:     Vector< Integer > dist = new Vector< Integer >();
55:     dist.addAll(Collections.nCopies(V, INF));
56:     dist.set(s, 0);
57:
58:     // Bellman Ford routine
59:     for (int i = 0; i < V - 1; i++) // relax all E edges V-1 times, O(V)
60:       for (int u = 0; u < V; u++) { // these two loops = O(E)
61:         Iterator it = AdjList.get(u).iterator();
62:         while (it.hasNext()) { // relax these edges
63:           IntegerPair v = (IntegerPair)it.next();
64:           dist.set(v.first(),
65:             Math.min(dist.get(v.first()), dist.get(u) + v.second()));
66:         }
67:       }
68:
69:     boolean negative_cycle_exist = false;
70:     for (int u = 0; u < V; u++) { // one more pass to check
```

```
   71:       Iterator it = AdjList.get(u).iterator();
   72:       while (it.hasNext()) { // relax these edges
   73:         IntegerPair v = (IntegerPair)it.next();
   74:         if(dist.get(v.first()) > dist.get(u) + v.second()) // should be false, but
   75:           negative_cycle_exist = true; // if possible, then negative cycle exists!
   76:       }
   77:     }
   78:     System.out.printf("Negative Cycle Exist? %s\n", negative_cycle_exist ? "Yes"
: "No");
   79:
   80:     if (!negative_cycle_exist)
   81:       for (int i = 0; i < V; i++)
   82:         System.out.printf("SSSP(%d, %d) = %d\n", s, i, dist.get(i));
   83:   }
   84: }
```

```
1: /*****************************************************************************/
2: /** Floyd-Warshall ..................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: public class ch4_07_floyd_warshall {
9:   public static void main(String[] args) throws Exception {
10:    int i, j, k, V, E, a, b, weight;
11:
12:    /*
13:    // Graph in Figure 4.20
14:    5 9
15:    0 1 2
16:    0 2 1
17:    0 4 3
18:    1 3 4
19:    2 1 1
20:    2 4 1
21:    3 0 1
22:    3 2 3
23:    3 4 5
24:    */
25:
26:    File f = new File("in_07.txt");
27:    Scanner sc = new Scanner(f);
28:
29:    V = sc.nextInt();
30:    E = sc.nextInt();
31:
32:    int[][] AdjMat = new int[V][];
33:    for (i = 0; i < V; i++) {
34:      AdjMat[i] = new int[V];
35:      for (j = 0; j < V; j++)
36:        AdjMat[i][j] = 1000000000; // use 1.10^9 to avoid overflow
37:      AdjMat[i][i] = 0;
38:    }
39:
40:    for (i = 0; i < E; i++) {
41:      a = sc.nextInt();
42:      b = sc.nextInt();
43:      weight = sc.nextInt();
44:      AdjMat[a][b] = weight; // directed graph
45:    }
46:
47:    for (k = 0; k < V; k++) // O(v^3) Floyd Warshall's code is here
48:      for (i = 0; i < V; i++)
49:        for (j = 0; j < V; j++)
50:          AdjMat[i][j] = Math.min(AdjMat[i][j],
51:                                  AdjMat[i][k] + AdjMat[k][j]);
52:
53:    for (i = 0; i < V; i++)
54:      for (j = 0; j < V; j++)
55:        System.out.printf("APSP(%d, %d) = %d\n", i, j, AdjMat[i][j]);
56:  }
57: }
```

```
1: /*****************************************************************************/
2: /** Edmonds-Karp ............................................................ */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: public class ch4_08_edmonds_karp {
9:   private static final int MAX_V = 40; // enough for sample graph in
10:  private static final int INF = 1000000000; // Figure 4.24/4.25/4.26
11:
12:  // we need these global variables
13:  private static int[][] res = new int[MAX_V][]; // define MAX_V appropriately
14:  private static int mf, f, s, t;
15:  private static Vector < Integer > p = new Vector < Integer > ();
16:
17:  // traverse the BFS spanning tree as in print_path (section 4.3)
18:  private static void augment(int v, int minEdge) {
19:    if (v == s) { // reach the source,
20:        f = minEdge; return; } // record minEdge in a global variable 'f'
21:    else if (p.get(v) != -1) {
22:        augment(p.get(v), Math.min(minEdge, res[p.get(v)][v])); // recursive call
23:        res[p.get(v)][v] -= f; res[v][p.get(v)] += f; // alter residual capacities
24:    }
25:  }
26:
27:  public static void main(String[] args) throws Exception {
28:    int V, k, vertex, weight;
29:
30:    /*
31:    // Graph in Figure 4.24
32:    4 0 1
33:    2 2 70 3 30
34:    2 2 25 3 70
35:    3 0 70 3 5 1 25
36:    3 0 30 2 5 1 70
37:
38:    // Graph in Figure 4.25
39:    4 0 3
40:    2 1 100 3 100
41:    2 2 1 3 100
42:    1 3 100
43:    0
44:
45:    // Graph in Figure 4.26.A
46:    5 1 0
47:    0
48:    2 2 100 3 50
49:    3 3 50 4 50 0 50
50:    1 4 100
51:    1 0 125
52:
53:    // Graph in Figure 4.26.B
54:    5 1 0
55:    0
56:    2 2 100 3 50
57:    3 3 50 4 50 0 50
58:    1 4 100
59:    1 0 75
60:
61:    // Graph in Figure 4.26.C
62:    5 1 0
63:    0
64:    2 2 100 3 50
65:    2 4 5 0 5
66:    1 4 100
67:    1 0 125
68:    */
69:
70:    File ff = new File("in_08.txt");
```

```
    Scanner sc = new Scanner(ff);

    V = sc.nextInt();
    s = sc.nextInt();
    t = sc.nextInt();

    for (int i = 0; i < V; i++) {
      res[i] = new int[MAX_V];
      k = sc.nextInt();
      for (int j = 0; j < k; j++) {
        vertex = sc.nextInt();
        weight = sc.nextInt();
        res[i][vertex] = weight;
      }
    }

    mf = 0;
    while (true) { // run O(VE^2) Edmonds Karp to solve the Max Flow problem
      f = 0;

      // run BFS, please examine parts of the BFS code that is different than
      Queue < Integer > q = new LinkedList < Integer > ();  // in Section 4.3
      Vector < Integer > dist = new Vector < Integer > ();
      dist.addAll(Collections.nCopies(V, INF)); // #define INF 2000000000
      q.offer(s);
      dist.set(s, 0);
      p.clear();
      p.addAll(Collections.nCopies(V, -1));
      // (we have to record the BFS spanning tree)
      while (!q.isEmpty()) {  // (we need the shortest path from s to t!)
        int u = q.poll();
        if (u == t) break; // immediately stop BFS if we already reach sink t
        for (int v = 0; v < MAX_V; v++)
        // note: enumerating neighbors with AdjMatrix is 'slow'
          if (res[u][v] > 0 && dist.get(v) == INF) { // res[u][v] can change!
            dist.set(v, dist.get(u) + 1);
            q.offer(v);
            p.set(v, u); // parent of vertex v is vertex u
          }
      }

      augment(t, INF); // find the min edge weight 'f' along this path, if any
      if(f == 0) break; // if we cannot send any more flow ('f' = 0), end the loop
      mf += f; // we can still send a flow, increase the max flow!
    }

    System.out.printf("%d\n", mf); //this is the max flow value of this flow graph
  }
}
```

```
1: /*****************************************************************************/
2: /** Emparelhamento Maximo em Grafos Bipartidos ............................. */
3: /*****************************************************************************/
4:
5: import java.util.*;
6: import java.text.*;
7:
8: class ch4_09_mcbm {
9:   private static Vector < Vector < Integer > > AdjList =
10:    new Vector < Vector < Integer > >();
11:  private static Vector < Integer > match, visited; // global variables
12:
13:  private static int Aug(int l) {
14:    if (visited.get(l) == 1) return 0;
15:    visited.set(l, 1);
16:
17:    Iterator it = AdjList.get(l).iterator();
18:    while (it.hasNext()) { // either greedy assignment or recurse
19:      Integer right = (Integer)it.next();
20:      if (match.get(right) == -1 || Aug(match.get(right)) == 1) {
21:        match.set(right, l);
22:        return 1; // we found one matching
23:      }
24:    }
25:
26:    return 0; // no matching
27:  }
28:
29:  private static Boolean isprime(int v) {
30:    int primes[] = new int[] {2,3,5,7,11,13,17,19,23,29};
31:    for (int i = 0; i < 10; i++)
32:      if (primes[i] == v)
33:        return true;
34:    return false;
35:  }
36:
37:  public static void main(String[] args) {
38:    int i, j;
39:
40: /*
41:    // Graph in Figure 4.40 can be built on the fly
42:    // we know there are 6 vertices in this bipartite graph,
43:    // left side are numbered 0,1,2, right side 3,4,5
44:    //int V = 6, V_l = 3;
45:    //int set1[] = new int[] {1,7,11}, set2[] = new int[] {4,10,12};
46:
47:    // Graph in Figure 4.41 can be built on the fly
48:    // we know there are 5 vertices in this bipartite graph,
49:    // left side are numbered 0,1, right side 3,4,5
50:    int V = 5, V_l = 2;
51:    int set1[] = new int[] {1,7}, set2[] = new int[] {4,10,12};
52:
53:    // build the bipartite graph, only directed edge from l to right is needed
54:    AdjList.clear();
55:    for (i = 0; i < V; i++) {
56:      Vector<Integer> Neighbor = new Vector<Integer>();
57:      AdjList.add(Neighbor); // store blank vector first
58:    }
59:
60:    for (i = 0; i < V_l; i++)
61:      for (j = 0; j < 3; j++)
62:        if (isprime(set1[i] + set2[j]))
63:          AdjList.get(i).add(3 + j);
64: */
65:
66:    // For bipartite graph in Figure 4.44, V = 5, Vleft = 3 (vertex 0 unused)
67:    // AdjList[0] = {} // dummy vertex, but you can choose to use this vertex
68:    // AdjList[1] = {3, 4}
69:    // AdjList[2] = {3}
70:    // AdjList[3] = {}   // we use directed edges from left to right set only
```

```
71:     // AdjList[4] = {}
72:
73:     int V = 5, V_l = 3;
74:     AdjList.clear();
75:     for (i = 0; i < V; i++) {
76:       Vector<Integer> Neighbor = new Vector<Integer>();
77:       AdjList.add(Neighbor); // store blank vector first
78:     }
79:     AdjList.get(1).add(3);
80:     AdjList.get(1).add(4);
81:     AdjList.get(2).add(3);
82:
83:     int MCBM = 0;
84:     match = new Vector < Integer > ();
85:     match.addAll(Collections.nCopies(V, -1));
86:     for (int l = 0; l < V_l; l++) {
87:       visited = new Vector < Integer > ();
88:       visited.addAll(Collections.nCopies(V_l, 0));
89:       MCBM += Aug(l);
90:     }
91:     System.out.printf("Found %d matchings\n", MCBM);
92:   }
93: }
```

```
 1: /*****************************************************************************/
 2: /** IntegerPair.java ........................................................ */
 3: /*****************************************************************************/
 4:
 5: class IntegerPair implements Comparable {
 6:   Integer _first, _second;
 7:
 8:   public IntegerPair(Integer f, Integer s) {
 9:     _first = f;
10:     _second = s;
11:   }
12:
13:   public int compareTo(Object o) {
14:     if (!this.first().equals((IntegerPair)o).first())
15:       return this.first() - ((IntegerPair)o).first();
16:     else
17:       return this.second() - ((IntegerPair)o).second();
18:   }
19:
20:   Integer first() { return _first; }
21:   Integer second() { return _second; }
22: }
```

```
1: /*****************************************************************************/
2: /** IntegerTriple.java ..................................................... */
3: /*****************************************************************************/
4:
5: class IntegerTriple implements Comparable {
6:   Integer _first, _second, _third;
7:
8:   public IntegerTriple(Integer f, Integer s, Integer t) {
9:     _first = f;
10:    _second = s;
11:    _third = t;
12:  }
13:
14:  public int compareTo(Object o) {
15:    if (!this.first().equals(((IntegerTriple)o).first()))
16:      return this.first() - ((IntegerTriple)o).first();
17:    else if (!this.second().equals(((IntegerTriple)o).second()))
18:      return this.second() - ((IntegerTriple)o).second();
19:    else
20:      return this.third() - ((IntegerTriple)o).third();
21:  }
22:
23:  Integer first() { return _first; }
24:  Integer second() { return _second; }
25:  Integer third() { return _third; }
26:
27:  public String toString() { return first() + " " + second() + " " + third(); }
28: }
```

```java
/*********************************************************************************/
/** BigInteger (soma) ..................................................... */
/*********************************************************************************/

import java.util.Scanner; // Scanner class is inside package java.util
import java.math.BigInteger; // BigInteger class is inside package java.math

class Main { /* UVa 10925 - Krakovia, 0.732s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int caseNo = 1;
    while (true) {
      int N = sc.nextInt(), F = sc.nextInt(); // N bills, F friends
      if (N == 0 && F == 0) break;
      BigInteger sum = BigInteger.ZERO; // BigInteger has this constant ZERO
      for (int i = 0; i < N; i++) { // sum the N large bills
        BigInteger V = sc.nextBigInteger(); // for reading next BigInteger!
        sum = sum.add(V); // this is BigInteger addition
      }
      System.out.println("Bill #" + (caseNo++) + " costs " +
        sum + ": each friend should pay " + sum.divide(BigInteger.valueOf(F)));
      System.out.println(); // the line above is BigInteger division
} } }                       // divide the large sum to F friends

```

```java
/*********************************************************************************/
/** BigInteger (mod) ..................................................... */
/*********************************************************************************/

import java.util.Scanner;
import java.math.BigInteger;

class Main { /* UVa 10551 - Basic Remains, 0.345s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (true) {
      int b = sc.nextInt();
      if (b == 0) break;
      String p_str = sc.next();
      BigInteger p = new BigInteger(p_str, b); // special constructor!
      String m_str = sc.next();
      BigInteger m = new BigInteger(m_str, b); // 2nd parameter is the radix/base
      System.out.println((p.mod(m)).toString(b)); // can output in any radix/base
} } }
```

```java
/*********************************************************************************/
/** BigInteger (primos) ..................................................... */
/*********************************************************************************/

import java.util.Scanner;
import java.math.BigInteger;

class Main { /* Simply Emirp, 2.788s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (sc.hasNext()) {
      int N = sc.nextInt();
      BigInteger BN = BigInteger.valueOf(N);
      String R = new StringBuffer(BN.toString()).reverse().toString();
      int RN = Integer.parseInt(R);
      BigInteger BRN = BigInteger.valueOf(RN);
      System.out.printf("%d is ", N);
      if (!BN.isProbablePrime(10)) // certainty 10 is enough for most cases
        System.out.println("not prime.");
      else if (N != RN && BRN.isProbablePrime(10))
        System.out.println("emirp.");
      else
        System.out.println("prime.");
} } }
```

```java
/*********************************************************************************/
/** BigInteger (divisao) ........................................................ */
/*********************************************************************************/

import java.util.Scanner;
import java.math.BigInteger;

class Main { /* UVa 10814 - Simplifying Fractions, 0.212s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int N = sc.nextInt();
    while (N-- > 0) { // unlike in C/C++, we have to supply > 0 in (N-- > 0)
      BigInteger p = sc.nextBigInteger();
      String ch = sc.next(); // we ignore the division sign in input
      BigInteger q = sc.nextBigInteger();
      BigInteger gcd_pq = p.gcd(q); // wow :)
      System.out.println(p.divide(gcd_pq) + " / " + q.divide(gcd_pq));
} } }
```

```java
/*******************************************************************************/
/** BigInteger (exponenciacao) ................................................ */
/*******************************************************************************/

import java.util.Scanner;
import java.math.BigInteger;

class Main { /* UVa 1230 - LA 4104 - MODEX, 0.189s in Java */
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int c = sc.nextInt();
    while (c-- > 0) {
      BigInteger x = BigInteger.valueOf(sc.nextInt()); // valueOf converts
      BigInteger y = BigInteger.valueOf(sc.nextInt()); // simple integer
      BigInteger n = BigInteger.valueOf(sc.nextInt()); // into BigInteger
      System.out.println(x.modPow(y, n)); // look ma, it's in the library ;)
} } }
```

```java
/*******************************************************************************/
/** Crivo de Eratostenes (descobre n's primos) ..............................  */
/*******************************************************************************/

import java.util.*;

public class ch5_06_primes {
  int _sieve_size;
  boolean[] bs;   // 10^7 should be enough for most cases
  List<Integer> primes = new ArrayList<Integer>();
  // compact list of primes in form of vector<int>

  // first part

  void sieve(int upperbound) {         // create list of primes in [0..upperbound]
    _sieve_size = upperbound + 1;                  // add 1 to include upperbound
    bs = new boolean[_sieve_size];
    Arrays.fill(bs,true);                                    // set all bits to 1
    bs[0] = bs[1] = false;                                 // except index 0 and 1
    for (long i = 2; i < _sieve_size; i++) if (bs[(int)i]) {
      // cross out multiples of i starting from i * i!
      for (long j = i * i; j < _sieve_size; j += i) bs[(int)j] = false;
      primes.add((int)i);  // also add this vector containing list of primes
  } }                                                // call this method in main method

  boolean isPrime(long N) {            // a good enough deterministic prime tester
    if (N < _sieve_size) return bs[(int)N];               // O(1) for small primes
    for (int i = 0; i < primes.size(); i++)
      if (N % primes.get(i) == 0) return false;
    return true;                      // it takes longer time if N is a large prime!
  }                      // note: only work for N <= (last prime in vi "primes")^2


  // second part

  List<Integer> primeFactors(long N) {
    // List<Integer> 'primes' (generated by sieve) is optional
    List<Integer> factors = new ArrayList<Integer>();
    int PF_idx = 0;
    long PF = primes.get(PF_idx);           // using PF = 2, 3, 4, ..., is also ok
    while (N != 1 && (PF * PF <= N)) {   // stop at sqrt(N), but N can get smaller
      while (N % PF == 0) { N /= PF; factors.add((int)PF); }     // remove this PF
      PF = primes.get(++PF_idx);                              // only consider primes!
    }
    if (N != 1) factors.add((int)N);     // special case if N is actually a prime
    return factors;         // if pf exceeds 32-bit integer, you have to change vi
  }

  // third part

  long numPF(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = 0;
    while (N != 1 && (PF * PF <= N)) {
      while (N % PF == 0) { N /= PF; ans++; }
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans++;
    return ans;
  }

  long numDiffPF(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = 0;
    while (N != 1 && (PF * PF <= N)) {
      if (N % PF == 0) ans++;                                 // count this pf only once
      while (N % PF == 0) N /= PF;
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans++;
```

```
    return ans;
  }

  long sumPF(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = 0;
    while (N != 1 && (PF * PF <= N)) {
      while (N % PF == 0) { N /= PF; ans += PF; }
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans += N;
    return ans;
  }

  long numDiv(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = 1;              // start from ans = 1
    while (N != 1 && (PF * PF <= N)) {
      long power = 0;                                           // count the power
      while (N % PF == 0) { N /= PF; power++; }
      ans *= (power + 1);                               // according to the formula
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans *= 2;             // (last factor has pow = 1, we add 1 to it)
    return ans;
  }

  long sumDiv(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = 1;              // start from ans = 1
    while (N != 1 && (PF * PF <= N)) {
      long power = 0;
      while (N % PF == 0) { N /= PF; power++; }
      ans *= ((long)Math.pow((double)PF, power + 1.0) - 1) / (PF - 1);  // formula
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans *= ((long)Math.pow((double)N, 2.0) - 1) / (N - 1); // last one
    return ans;
  }

  long EulerPhi(long N) {
    int PF_idx = 0;
    long PF = primes.get(PF_idx), ans = N;              // start from ans = N
    while (N != 1 && (PF * PF <= N)) {
      if (N % PF == 0) ans -= ans / PF;                 // only count unique factor
      while (N % PF == 0) N /= PF;
      PF = primes.get(++PF_idx);
    }
    if (N != 1) ans -= ans / N;                                     // last factor
    return ans;
  }

  void run(){
    // first part: the Sieve of Eratosthenes
    sieve(10000000);                        // can go up to 10^7 (need few seconds)
    System.out.printf("%b\n", isPrime(2147483647));             // 10-digits prime
    System.out.printf("%b\n", isPrime(136117223861L)); // no prime, 104729*1299709

    // second part: prime factors
    List<Integer> res = primeFactors(2147483647);  // slowest, 2147483647 is prime
    for (int i : res) System.out.printf("> %d\n", i);

    res = primeFactors(136117223861L);   // slow, 2 large pfactors 104729*1299709
    for (int i : res) System.out.printf("# %d\n", i);

    res = primeFactors(142391208960L);   // faster, 2^10*3^4*5*7^4*11*13
    for (int i : res) System.out.printf("! %d\n", i);

    //res = primeFactors((long)(1010189899 * 1010189899)); // "error"

```

```
141:     // third part: prime factors variants
142:     System.out.printf("numPF(%d) = %d\n", 50, numPF(50));         // 2^1 * 5^2 => 3
143:     System.out.printf("numDiffPF(%d) = %d\n", 50, numDiffPF(50));// 2^1 * 5^2 => 2
144:     System.out.printf("sumPF(%d) = %d\n", 50, sumPF(50));         // 2^1 * 5^2 =>
145:                                                                   // 2 + 5 + 5 = 12
146:     System.out.printf("numDiv(%d) = %d\n", 50, numDiv(50)); // 1, 2, 5, 10, 25, 50
147:                                                             // 6 divisors
148:     System.out.printf("sumDiv(%d) = %d\n", 50, sumDiv(50));
149:                                                    // 1 + 2 + 5 + 10 + 25 + 50 = 93
150:     System.out.printf("EulerPhi(%d) = %d\n", 50, EulerPhi(50));
151:                                  // 20 integers < 50 are relatively prime with 50
152:   }
153:
154:   public static void main(String[] args){
155:     new ch5_06_primes().run();
156:   }
157: }
```

```java
/*********************************************************************************/
/** Floyd's Cycle-Finding Algorithm ..... */
/*********************************************************************************/

// Pseudo-Random Numbers, 0.288s in Java, 0.022s in C++

import java.util.*;

class Main {
  static int Z, I, M, L, mu, lambda;

  static int f(int x) { return (Z * x + I) % M; }

  // function "int f(int x)" must be defined earlier
  static void floydCycleFinding(int x0) {
    // 1st part: finding k*mu, hare's speed is 2x tortoise's
    int tortoise = f(x0), hare = f(f(x0)); // f(x0) is the node next to x0
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(f(hare)); }
    // 2nd part: finding mu, hare and tortoise move at the same speed
    mu = 0; hare = x0;
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare); mu++; }
    // 3rd part: finding lambda, hare moves, tortoise stays
    lambda = 1; hare = f(tortoise);
    while (tortoise != hare) { hare = f(hare); lambda++; }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    for (int caseNo = 1; ; caseNo++) {
      Z = sc.nextInt();
      I = sc.nextInt();
      M = sc.nextInt();
      L = sc.nextInt();
      if (Z == 0 && I == 0 && M == 0 && L == 0) break;
      floydCycleFinding(L);
      System.out.printf("Case %d: %d\n", caseNo, lambda);
    }
  }
}
```

```
1: /*********************************************************************************/
2: /** Strings (algoritmos basicos) ..................................................*/
3: /*********************************************************************************/
4:
5: import java.util.*;
6: import java.io.*;
7:
8: class ch6_01_basic_string {
9:   static int isvowel(char ch) { // make sure ch is in lowercase
10:    String vowel = "aeiou";
11:    for (int j = 0; j < 5; j++)
12:      if (vowel.charAt(j) == ch)
13:        return 1;
14:    return 0;
15:  }
16:
17:  public static void main(String[] args) throws Exception {
18:    int i, pos, digits, alphas, vowels, consonants;
19:    Boolean first, prev_dash, this_dash;
20:    String str = "";
21:    first = true; // technique to differentiate first line with the other lines
22:    prev_dash = this_dash = false; // to differentiate whether the previous line
23:                                   // contains a dash or not
24:    File f = new File("ch6.txt");
25:    Scanner sc = new Scanner(f);
26:    while (sc.hasNext()) {
27:      String line = sc.nextLine();
28:      if (line.equals(".......")) break;
29:      if (line.charAt(line.length() - 1) == '-') { // if the last character
30:        line = line.substring(0, line.length() - 1);   // is '-', delete it
31:        this_dash = true;
32:      }
33:      else
34:        this_dash = false;
35:      if (!first && !prev_dash)
36:        str = str + " "; // only append " " if this line is the second one onwards
37:      first = false;
38:      str = str + line;
39:      prev_dash = this_dash;
40:    }
41:
42:    char[] temp = str.toCharArray();
43:    for (i = digits = alphas = vowels = consonants = 0; i < str.length(); i++) {
44:      temp[i] = Character.toLowerCase(temp[i]); // make each character lower case
45:      digits += Character.isDigit(temp[i]) ? 1 : 0;
46:      alphas += Character.isLetter(temp[i]) ? 1 : 0;
47:      vowels += isvowel(temp[i]); // already returns 1 or 0
48:    }
49:    consonants = alphas - vowels;
50:    str = new String(temp);
51:    System.out.println(str);
52:    System.out.printf("%d %d %d\n", digits, vowels, consonants);
53:    int hascs3233 = (str.indexOf("cs3233") != -1) ? 1 : 0;
54:
55:    Vector<String> tokens = new Vector<String>();
56:    TreeMap<String, Integer> freq = new TreeMap<String, Integer>();
57:    StringTokenizer st = new StringTokenizer(str, " .");
58:    while (st.hasMoreTokens()) {
59:      String p = st.nextToken();
60:      tokens.add(p);
61:      if (!freq.containsKey(p)) freq.put(p, 1);
62:      else                      freq.put(p, freq.get(p) + 1);
63:    }
64:
65:    Collections.sort(tokens);
66:    System.out.println(tokens.get(0) + " " + tokens.get(tokens.size() - 1));
67:    System.out.printf("%d\n", hascs3233);
68:
69:    int ans_s = 0, ans_h = 0, ans_7 = 0;
70:    String lastline = sc.nextLine();
```

```
71:     for (i = 0; i < lastline.length(); i++) {
72:       char ch = lastline.charAt(i);
73:            if (ch == 's') ans_s++;
74:       else if (ch == 'h') ans_h++;
75:       else if (ch == '7') ans_7++;
76:     }
77:     System.out.printf("%d %d %d\n", ans_s, ans_h, ans_7);
78:   }
79: }
```

```java
/*****************************************************************************/
/** Knuth-Morris-Pratt (string matching) ..................................... */
/*****************************************************************************/

import java.util.*;

class ch6_02_kmp {
  char[] T, P; // T = text, P = pattern
  int n, m; // n = length of T, m = length of P
  int [] b; // b = back table

  void naiveMatching() {
    for (int i = 0; i < n; i++) { // try all potential starting indices
      Boolean found = true;
      for (int j = 0; j < m && found; j++) // use boolean flag 'found'
        if (i + j >= n || P[j] != T[i + j]) // if mismatch found
          found = false; // abort this, shift starting index i by +1
      if (found) // if P[0 .. m - 1] == T[i .. i + m - 1]
        System.out.printf("P is found at index %d in T\n", i);
  } }

  void kmpPreprocess() { // call this before calling kmpSearch()
    int i = 0, j = -1; b[0] = -1; // starting values
    while (i < m) { // pre-process the pattern string P
      while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
      i++; j++; // if same, advance both pointers
      b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
  } }           // in the example of P = "SEVENTY SEVEN" above

  void kmpSearch() { // this is similar as kmpPreprocess(), but on string T
    int i = 0, j = 0; // starting values
    while (i < n) { // search through string T
      while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
      i++; j++; // if same, advance both pointers
      if (j == m) { // a match found when j == m
        System.out.printf("P is found at index %d in T\n", i - j);
        j = b[j]; // prepare j for the next possible match
  } } }

  void run() {
    String Tstr = "I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN";
    String Pstr = "SEVENTY SEVEN";
    T = new String(Tstr).toCharArray();
    P = new String(Pstr).toCharArray();
    n = T.length;
    m = P.length;

    System.out.println(T);
    System.out.println(P);
    System.out.println();

    System.out.printf("Naive Mathing\n");
    naiveMatching();
    System.out.println();

    System.out.printf("KMP\n");
    b = new int[100010];
    kmpPreprocess();
    kmpSearch();
    System.out.println();

    System.out.printf("String Library\n");
    int pos = Tstr.indexOf(Pstr);
    while (pos != -1) {
      System.out.printf("P is found at index %d in T\n", pos);
      pos = Tstr.indexOf(Pstr, pos + 1);
    }
    System.out.println();
  }

```

```
71:   public static void main(String[] args){
72:     new ch6_02_kmp().run();
73:   }
74: }
```

```java
/*****************************************************************************/
/** Alinhamento de Strings (Needleman-Wunsch) ................................ */
/*****************************************************************************/

import java.util.*;

class ch6_03_str_align {
  public static void main(String[] args){
    char[] A = "ACAATCC".toCharArray(), B = "AGCATGC".toCharArray();
    int[][] table = new int[20][20]; // Needleman Wunsnch's algorithm
    int i, j, n = A.length, m = B.length;

    // insert/delete = -1 point
    for (i = 1; i <= n; i++)
      table[i][0] = i * -1;
    for (j = 1; j <= m; j++)
      table[0][j] = j * -1;

    for (i = 1; i <= n; i++)
      for (j = 1; j <= m; j++) {
        // match = 2 points, mismatch = -1 point
        table[i][j] = table[i - 1][j - 1] + (A[i - 1] == B[j - 1] ? 2 : -1);
        // insert/delete = -1 point
        table[i][j] = Math.max(table[i][j], table[i - 1][j] - 1); // delete
        table[i][j] = Math.max(table[i][j], table[i][j - 1] - 1); // insert
      }

    System.out.printf("DP table:\n");
    for (i = 0; i <= n; i++) {
      for (j = 0; j <= m; j++)
        System.out.printf("%3d", table[i][j]);
      System.out.printf("\n");
    }
    System.out.printf("Maximum Alignment Score: %d\n", table[n][m]);
  }
}
```

```
1: /*****************************************************************************/
2: /** Array de Sufixos ..................................................... */
3: /*****************************************************************************/
4:
5: import java.util.*;
6:
7: class ch6_04_sa {
8:   Scanner scan;
9:   char T[];                          // the input string, up to 100K characters
10:  int n;                                          // the length of input string
11:
12:  int[] RA, tempRA;              // rank array and temporary rank array
13:  Integer[] SA, tempSA;          // suffix array and temporary suffix array
14:  int[] c;                                      // for counting/radix sort
15:
16:  char P[];      // the pattern string (for string matching)
17:  int m;              // the length of pattern string
18:
19:  int[] Phi;     // for computing longest common prefix
20:  int[] PLCP;
21:  int[] LCP;     // LCP[i] stores the LCP between
22:                 // previous suffix "T + SA[i-1]" and current suffix "T + SA[i]"
23:
24:  void countingSort(int k) {
25:    int i, sum, maxi = Math.max(300, n);   // up to 255 ASCII chars or length of n
26:    for (i = 0; i < 100010; i++) c[i] = 0;                // clear frequency table
27:    for (i = 0; i < n; i++)                     // count the frequency of each rank
28:      c[i + k < n ? RA[i + k] : 0]++;
29:    for (i = sum = 0; i < maxi; i++) {
30:      int t = c[i]; c[i] = sum; sum += t;
31:    }
32:    for (i = 0; i < n; i++)               // shuffle the suffix array if necessary
33:      tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
34:    for (i = 0; i < n; i++)                          // update the suffix array SA
35:      SA[i] = tempSA[i];
36:  }
37:
38:  void constructSA() {              // this version can go up to 100000 characters
39:    int i, k, r;
40:    for (i = 0; i < n; i++) RA[i] = T[i];                      // initial rankings
41:    for (i = 0; i < n; i++) SA[i] = i;          // initial SA: {0, 1, 2, ..., n-1}
42:    for (k = 1; k < n; k <<= 1) {            // repeat sorting process log n times
43:      countingSort(k);       // actually radix sort: sort based on the second item
44:      countingSort(0);               // then (stable) sort based on the first item
45:      tempRA[SA[0]] = r = 0;                   // re-ranking; start from rank r = 0
46:      for (i = 1; i < n; i++)                          // compare adjacent suffices
47:        tempRA[SA[i]] =      // if same pair => same rank r; otherwise, increase r
48:          (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
49:      for (i = 0; i < n; i++)                           // update the rank array RA
50:        RA[i] = tempRA[i];
51:  } }
52:
53:  void computeLCP() {
54:    int i, L;
55:    Phi[SA[0]] = -1;                                                // default value
56:    for (i = 1; i < n; i++)                                 // compute Phi in O(n)
57:      Phi[SA[i]] = SA[i-1];         // remember which suffix is behind this suffix
58:    for (i = L = 0; i < n; i++) {                   // compute Permuted LCP in O(n)
59:      if (Phi[i] == -1) { PLCP[i] = 0; continue; }                 // special case
60:      while (i + L < T.length && Phi[i] + L < T.length &&
61:                  T[i + L] == T[Phi[i] + L]) L++; // L will be increased max n times
62:      PLCP[i] = L;
63:      L = Math.max(L-1, 0);                       // L will be decreased max n times
64:    }
65:    for (i = 1; i < n; i++)                                 // compute LCP in O(n)
66:      LCP[i] = PLCP[SA[i]];   // put the permuted LCP back to the correct position
67:  }
68:
69:  int strncmp(char[] a, int i, char[] b, int j, int n){
70:    for (int k=0; i+k < a.length && j+k < b.length; k++){
```

```
    71:       if (a[i+k] != b[j+k]) return a[i+k] - b[j+k];
    72:     }
    73:     return 0;
    74:   }
    75:
    76:   int[] stringMatching() {                          // string matching in O(m log n)
    77:     int lo = 0, hi = n-1, mid = lo;                   // valid matching = [0 .. n-1]
    78:     while (lo < hi) {                                                 // find lower bound
    79:       mid = (lo + hi) / 2;                                          // this is round down
    80:       int res = strncmp(T, SA[mid], P, 0, m);       // try to find P in suffix
'mid'
    81:       if (res >= 0) hi = mid;             // prune upper half (notice the >= sign)
    82:       else          lo = mid + 1;                 // prune lower half including mid
    83:     }                                              // observe '=' in "res >= 0" above
    84:     if (strncmp(T,SA[lo], P,0, m) != 0) return new int[]{-1, -1};  // if not found
    85:     int[] ans = new int[]{ lo, 0} ;
    86:
    87:     lo = 0; hi = n - 1; mid = lo;
    88:     while (lo < hi) {                 // if lower bound is found, find upper bound
    89:       mid = (lo + hi) / 2;
    90:       int res = strncmp(T, SA[mid], P,0, m);
    91:       if (res > 0) hi = mid;                                        // prune upper half
    92:       else         lo = mid + 1;                 // prune lower half including mid
    93:     }                                  // (notice the selected branch when res == 0)
    94:     if (strncmp(T, SA[hi], P,0, m) != 0) hi--;                        // special case
    95:     ans[1] = hi;
    96:     return ans;
    97:   } // return lower/upper bound as the first/second item of the pair, respectively
    98:
    99:   void LRS() {                            // print out the length and the actual LRS
   100:     int i, idx = 0, maxLCP = 0;
   101:
   102:     for (i = 1; i < n; i++)                                                  // O(n)
   103:       if (LCP[i] > maxLCP) {
   104:         maxLCP = LCP[i];
   105:         idx = i;
   106:       }
   107:
   108:     System.out.printf("\nThe LRS is '%s' with length = %d\n\n",
   109:       new String(T).substring(SA[idx], maxLCP), maxLCP);
   110:   }
   111:
   112:   int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }
   113:
   114:   void LCS() {                            // print out the length and the actual LCS
   115:     int i, j, maxLCP = 0, idx = 0;
   116:     // not used in Java version
   117:     // char ans[MAX_N];
   118:     // strcpy(ans, "");
   119:
   120:     //System.out.printf("\nRemember, T = '%s'\nNow, enter another string P:\n",
   121:                                                                  // new String(T));
   122:     // T already has '.' at the back
   123:     P = new String("CATA").toCharArray();
   124:     m = P.length;
   125:     T = (new String(T) + new String(P) + "#").toCharArray();   // append P and '#'
   126:     n = T.length;                                                    // update n
   127:     constructSA();                                                // O(n log n)
   128:     computeLCP();                                                    // O(n)
   129:     System.out.printf("\nThe LCP information of 'T+P' = '%s':\n", new String(T));
   130:     System.out.printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
   131:     for (i = 0; i < n; i++)
   132:       System.out.printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i],
owner(SA[i]),
   133:         new String(T, SA[i], T.length - SA[i]));
   134:
   135:     for (i = 1, maxLCP = -1; i < n; i++)
   136:       if (LCP[i] > maxLCP && owner(SA[i]) != owner(SA[i-1])) {  // different owner
   137:         maxLCP = LCP[i];
   138:         idx = i;
```

```
139:         // not used in Java version
140:         // strncpy(ans, T + SA[i], maxLCP);
141:         // ans[maxLCP] = 0;
142:       }
143:
144:     System.out.printf("\nThe LCS is '%s' with length = %d\n",
145:       new String(T).substring(SA[idx], SA[idx] + maxLCP),
146:       maxLCP);
147:   }
148:
149:   void run() {
150:     int MAX_N = 100010;
151:     c = new int[MAX_N];
152:     RA = new int[MAX_N];
153:     tempRA = new int[MAX_N];
154:     SA = new Integer[MAX_N];
155:     tempSA = new Integer[MAX_N];
156:     Phi = new int[MAX_N];
157:     PLCP = new int[MAX_N];
158:     LCP = new int[MAX_N];
159:
160: //System.out.println("Enter a string T below, we will compute its Suffix Array:");
161:     T = new String("GATAGACA$").toCharArray();
162:     n = T.length;
163:
164:     constructSA();                                                    // O(n log n)
165:     System.out.printf("The Suffix Array of string T = '%s' is shown below " +
166:       "(O(n log n) version):\n", new String(T));
167:     System.out.printf("i\tSA[i]\tSuffix\n");
168:     for (int i = 0; i < n; i++)
169:      System.out.printf("%2d\t%2d\t%s\n", i, SA[i],
170:        new String(T, SA[i], T.length - SA[i]));
171:
172:     computeLCP();                                                       // O(n)
173:
174:     // LRS demo
175:     LRS();        // find the longest repeated substring of the first input string
176:
177:     // stringMatching demo
178: //System.out.println("\nNow enter a string P below, we will try to find P in T:");
179:     P = new String("A").toCharArray();
180:     int[] pos = stringMatching();
181:     if (pos[0] != -1 && pos[1] != -1) {
182:       System.out.printf("%s is found SA [%d .. %d] of %s\n",
183:         new String(P), pos[0], pos[1], new String(T));
184:       System.out.printf("They are:\n");
185:       for (int i = pos[0]; i <= pos[1]; i++)
186:         System.out.printf("  %s\n", new String(T, SA[i], T.length - SA[i]));
187:     } else
188:       System.out.printf("%s is not found in %s\n", new String(P), new String(T));
189:
190:     // LCS demo
191:     LCS();                    // find the longest common substring between T and P
192:
193:     // note that the LRS and LCS demo are slightly different in Java version
194:   }
195:
196:   public static void main(String[] args){
197:     new ch6_04_sa().run();
198:   }
199: }
```

```java
/*********************************************************************************/
/** Pontos e Linhas ..................................................... */
/*********************************************************************************/

import java.util.*;

class ch7_01_points_lines {
  final double INF = 1e9;
  final double EPS = 1e-9;
  // we will use constant Math.PI in Java

  double DEG_to_RAD(double d) { return d * Math.PI / 180.0; }

  double RAD_to_DEG(double r) { return r * 180.0 / Math.PI; }

  //struct point_i { int x, y; };        // basic raw form, minimalist mode
  class point_i { int x, y;          // whenever possible, work with point_i
    point_i() { x = y = 0; }                          // default constructor
    point_i(int _x, int _y) { x = _x; y = _y; } };          // user-defined

  class point implements Comparable<point>{
    double x, y;                   // only used if more precision is needed
    point() { x = y = 0.0; }                          // default constructor
    point(double _x, double _y) { x = _x; y = _y; }         // user-defined
    // use EPS (1e-9) when testing equality of two floating points
    public int compareTo(point other) {      // override less than operator
      if (Math.abs(x - other.x) > EPS)                 // useful for sorting
        return (int)Math.ceil(x - other.x);       // first: by x-coordinate
      else if (Math.abs(y - other.y) > EPS)
        return (int)Math.ceil(y - other.y);      // second: by y-coordinate
      else
        return 0; } };                                      // they are equal

  double dist(point p1, point p2) {                   // Euclidean distance
                      // Math.hypot(dx, dy) returns sqrt(dx * dx + dy * dy)
    return Math.hypot(p1.x - p2.x, p1.y - p2.y); }         // return double

  // rotate p by theta degrees CCW w.r.t origin (0, 0)
  point rotate(point p, double theta) {
    double rad = DEG_to_RAD(theta);       // multiply theta with PI / 180.0
    return new point(p.x * Math.cos(rad) - p.y * Math.sin(rad),
                     p.x * Math.sin(rad) + p.y * Math.cos(rad)); }

  class line { double a, b, c; };              // a way to represent a line

  // the answer is stored in the third parameter
  void pointsToLine(point p1, point p2, line l) {
    if (Math.abs(p1.x - p2.x) < EPS) {             // vertical line is fine
      l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;
    } else {
      l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
      l.b = 1.0;                 // IMPORTANT: we fix the value of b to 1.0
      l.c = -(double)(l.a * p1.x) - p1.y;
  } }

  //not needed since we will use the more robust form: ax + by + c = 0 (see above)
  class line2 { double m, c; };          // another way to represent a line

  int pointsToLine2(point p1, point p2, line2 l) {
   if (Math.abs(p1.x - p2.x) < EPS) {        // special case: vertical line
     l.m = INF;                     // l contains m = INF and c = x_value
     l.c = p1.x;                   // to denote vertical line x = x_value
     return 0;      // we need this return variable to differentiate result
   }
   else {
     l.m = (double)(p1.y - p2.y) / (p1.x - p2.x);
     l.c = p1.y - l.m * p1.x;
     return 1;        // l contains m and c of the line equation y = mx + c
  } }

```

```
71:   boolean areParallel(line l1, line l2) {       // check coefficients a & b
72:     return (Math.abs(l1.a-l2.a) < EPS) && (Math.abs(l1.b-l2.b) < EPS); }
73:
74:   boolean areSame(line l1, line l2) {           // also check coefficient c
75:     return areParallel(l1 ,l2) && (Math.abs(l1.c - l2.c) < EPS); }
76:
77:   // returns true (+ intersection point) if two lines are intersect
78:   boolean areIntersect(line l1, line l2, point p) {
79:     if (areParallel(l1, l2)) return false;              // no intersection
80:     // solve system of 2 linear algebraic equations with 2 unknowns
81:     p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
82:     // special case: test for vertical line to avoid division by zero
83:     if (Math.abs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
84:     else                      p.y = -(l2.a * p.x + l2.c);
85:     return true; }
86:
87:   class vec { double x, y;     // name: 'vec' is different from Java Vector
88:     vec(double _x, double _y) { x = _x; y = _y; } };
89:
90:   vec toVec(point a, point b) {               // convert 2 points to vector
91:     return new vec(b.x - a.x, b.y - a.y); }
92:
93:   vec scale(vec v, double s) {          // nonnegative s = [<1 .. 1 .. >1]
94:     return new vec(v.x * s, v.y * s); }             // shorter.same.longer
95:
96:   point translate(point p, vec v) {           // translate p according to v
97:     return new point(p.x + v.x , p.y + v.y); }
98:
99:   // convert point and gradient/slope to line
100:  void pointSlopeToLine(point p, double m, line l) {
101:    l.a = -m;                                                  // always -m
102:    l.b = 1;                                                    // always 1
103:    l.c = -((l.a * p.x) + (l.b * p.y)); }                  // compute this
104:
105:  void closestPoint(line l, point p, point ans) {
106:    line perpendicular = new line(); // perpendicular to l and pass through p
107:    if (Math.abs(l.b) < EPS) {             // special case 1: vertical line
108:      ans.x = -(l.c);   ans.y = p.y;      return; }
109:
110:    if (Math.abs(l.a) < EPS) {           // special case 2: horizontal line
111:      ans.x = p.x;      ans.y = -(l.c);   return; }
112:
113:    pointSlopeToLine(p, 1 / l.a, perpendicular);             // normal line
114:    // intersect line l with this perpendicular line
115:    // the intersection point is the closest point
116:    areIntersect(l, perpendicular, ans); }
117:
118:  // returns the reflection of point on a line
119:  void reflectionPoint(line l, point p, point ans) {
120:    point b = new point();
121:    closestPoint(l, p, b);                          // similar to distToLine
122:    vec v = toVec(p, b);                                 // create a vector
123:    ans = translate(translate(p, v), v); }            // translate p twice
124:
125:  double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }
126:
127:  double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }
128:
129:  // returns the distance from p to the line defined by
130:  // two points a and b (a and b must be different)
131:  // the closest point is stored in the 4th parameter
132:  double distToLine(point p, point a, point b, point c) {
133:    // formula: c = a + u * ab
134:    vec ap = toVec(a, p), ab = toVec(a, b);
135:    double u = dot(ap, ab) / norm_sq(ab);
136:    c = translate(a, scale(ab, u));                      // translate a to c
137:    return dist(p, c); }              // Euclidean distance between p and c
138:
139:  // returns the distance from p to the line segment ab defined by
140:  // two points a and b (still OK if a == b)
```

```
141:   // the closest point is stored in the 4th parameter
142:   double distToLineSegment(point p, point a, point b, point c) {
143:     vec ap = toVec(a, p), ab = toVec(a, b);
144:     double u = dot(ap, ab) / norm_sq(ab);
145:     if (u < 0.0) { c = new point(a.x, a.y);                    // closer to a
146:       return dist(p, a); }             // Euclidean distance between p and a
147:     if (u > 1.0) { c = new point(b.x, b.y);                    // closer to b
148:       return dist(p, b); }             // Euclidean distance between p and b
149:     return distToLine(p, a, b, c); }              // run distToLine as above
150:
151:   double angle(point a, point o, point b) {     // returns angle aob in rad
152:     vec oa = toVec(o, a), ob = toVec(o, b);
153:     return Math.acos(dot(oa, ob) / Math.sqrt(norm_sq(oa) * norm_sq(ob))); }
154:
155:   double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }
156:
157:   //// another variant: returns 'twice' the area of this triangle A-B-c
158:   //int area2(point p, point q, point r) {
159:   //  return p.x * q.y - p.y * q.x +
160:   //         q.x * r.y - q.y * r.x +
161:   //         r.x * p.y - r.y * p.x;
162:   //}
163:
164:   // note: to accept collinear points, we have to change the '> 0'
165:   // returns true if point r is on the left side of line pq
166:   boolean ccw(point p, point q, point r) {
167:     return cross(toVec(p, q), toVec(p, r)) > 0; }
168:
169:   // returns true if point r is on the same line as the line pq
170:   boolean collinear(point p, point q, point r) {
171:     return Math.abs(cross(toVec(p, q), toVec(p, r))) < EPS; }
172:
173:   void run() {
174:     // note that both P1 and P2 are (0.00, 0.00)
175:     point P1 = new point(), P2 = new point(), P3 = new point(0, 1);
176:     System.out.println(P1.compareTo(P2) == 0);                 // true
177:     System.out.println(P1.compareTo(P3) == 0);                // false
178:
179:     point[] P = new point[6];
180:     P[0] = new point(2, 2);
181:     P[1] = new point(4, 3);
182:     P[2] = new point(2, 4);
183:     P[3] = new point(6, 6);
184:     P[4] = new point(2, 6);
185:     P[5] = new point(6, 5);
186:
187:     // sorting points demo
188:     Arrays.sort(P);
189:     for (int i = 0; i < P.length; i++)
190:       System.out.printf("(%.2f, %.2f)\n", P[i].x, P[i].y);
191:
192:     // rearrange the points as shown in the diagram below
193:     P = new point[7];
194:     P[0] = new point(2, 2);
195:     P[1] = new point(4, 3);
196:     P[2] = new point(2, 4);
197:     P[3] = new point(6, 6);
198:     P[4] = new point(2, 6);
199:     P[5] = new point(6, 5);
200:     P[6] = new point(8, 6);
201:
202:     /*
203:     // the positions of these 7 points (0-based indexing)
204:     6   P4      P3  P6
205:     5           P5
206:     4   P2
207:     3       P1
208:     2   P0
209:     1
210:     0 1 2 3 4 5 6 7 8
```

```
  211:     */
  212:
  213:     double d = dist(P[0], P[5]);
  214:     System.out.printf("Euclidean distance between P[0] and P[5] = %.2f\n", d);
  215:                                                                 // should be 5.000
  216:     // line equations
  217:     line l1 = new line(), l2 = new line(), l3 = new line(), l4 = new line();
  218:
  219:     pointsToLine(P[0], P[1], l1);
  220:     System.out.printf("%.2f * x + %.2f * y + %.2f = 0.00\n", l1.a, l1.b, l1.c);
  221:     // should be -0.50 * x + 1.00 * y - 1.00 = 0.00
  222:
  223:     pointsToLine(P[0], P[2], l2);
  224:     // a vertical line, not a problem in "ax + by + c = 0" representation
  225:     System.out.printf("%.2f * x + %.2f * y + %.2f = 0.00\n", l2.a, l2.b, l2.c);
  226:     // should be 1.00 * x + 0.00 * y - 2.00 = 0.00
  227:
  228:     // parallel, same, and line intersection tests
  229:     pointsToLine(P[2], P[3], l3);
  230:     System.out.printf("l1 & l2 are parallel? %b\n", areParallel(l1, l2)); // no
  231:     System.out.printf("l1 & l3 are parallel? %b\n", areParallel(l1, l3)); // yes,
  232:                                     // l1 (P[0]-P[1]) and l3 (P[2]-P[3]) are parallel
  233:     pointsToLine(P[2], P[4], l4);
  234:     System.out.printf("l1 & l2 are the same? %b\n", areSame(l1, l2)); // no
  235:     System.out.printf("l2 & l4 are the same? %b\n", areSame(l2, l4)); // yes,
  236:                                 // l2 (P[0]-P[2]) and l4 (P[2]-P[4]) are the same line
  237:                         // (note, they are two different line segments, but same line)
  238:     point p12 = new point();
  239:     boolean res = areIntersect(l1, l2, p12);
  240:     // yes, l1 (P[0]-P[1]) and l2 (P[0]-P[2]) are intersect at (2.0, 2.0)
  241:     System.out.printf("l1 & l2 are intersect? %b, at (%.2f, %.2f)\n",
  242:       res, p12.x, p12.y);
  243:
  244:     // other distances
  245:     point ans = new point();
  246:     d = distToLine(P[0], P[2], P[3], ans);
  247:     System.out.printf("Closest point from P[0] to line         (P[2]-P[3]):
(%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d);
  248:     closestPoint(l3, P[0], ans);
  249:     System.out.printf("Closest point from P[0] to line V2      (P[2]-P[3]):
(%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, dist(P[0], ans));
  250:
  251:     d = distToLineSegment(P[0], P[2], P[3], ans);
  252:     System.out.printf("Closest point from P[0] to line SEGMENT (P[2]-P[3]):
(%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d); // closer to A (or P[2]) = (2.00, 4.00)
  253:     d = distToLineSegment(P[1], P[2], P[3], ans);
  254:     System.out.printf("Closest point from P[1] to line SEGMENT (P[2]-P[3]):
(%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d); // closer to midway between AB = (3.20,
4.60)
  255:     d = distToLineSegment(P[6], P[2], P[3], ans);
  256:     System.out.printf("Closest point from P[6] to line SEGMENT (P[2]-P[3]):
(%.2f, %.2f), dist = %.2f\n", ans.x, ans.y, d); // closer to B (or P[3]) = (6.00, 6.00)
  257:
  258:     reflectionPoint(l4, P[1], ans);
  259:     System.out.printf("Reflection point from P[1] to line      (P[2]-P[4]):
(%.2f, %.2f)\n", ans.x, ans.y); // should be (0.00, 3.00)
  260:
  261:     System.out.printf("Angle P[0]-P[4]-P[3] = %.2f\n", RAD_to_DEG(angle(P[0],
P[4], P[3]))); // 90 degrees
  262:     System.out.printf("Angle P[0]-P[2]-P[1] = %.2f\n", RAD_to_DEG(angle(P[0],
P[2], P[1]))); // 63.43 degrees
  263:     System.out.printf("Angle P[4]-P[3]-P[6] = %.2f\n", RAD_to_DEG(angle(P[4],
P[3], P[6]))); // 180 degrees
  264:
  265:     System.out.printf("P[0], P[2], P[3] form A left turn? %b\n", ccw(P[0], P[2],
P[3])); // no
  266:     System.out.printf("P[0], P[3], P[2] form A left turn? %b\n", ccw(P[0], P[3],
P[2])); // yes
  267:
  268:     System.out.printf("P[0], P[2], P[3] are collinear? %b\n", collinear(P[0],
```

```
P[2], P[3])); // no
  269:     System.out.printf("P[0], P[2], P[4] are collinear? %b\n", collinear(P[0],
P[2], P[4])); // yes
  270:
  271:     point p = new point(3, 7), q = new point(11, 13), r = new point(35, 30); //
collinear if r(35, 31)
  272:     System.out.printf("r is on the %s of line p-r\n", ccw(p, q, r) ? "left" :
"right"); // right
  273:
  274:     /*
  275:     // the positions of these 6 points
  276:        E<--  4
  277:              3       B D<--
  278:              2   A C
  279:              1
  280:     -4-3-2-1 0 1 2 3 4 5 6
  281:             -1
  282:             -2
  283:      F<--   -3
  284:     */
  285:
  286:     // translation
  287:     point A = new point(2.0, 2.0);
  288:     point B = new point(4.0, 3.0);
  289:     vec v = toVec(A, B); // imagine there is an arrow from A to B
  290:     point C = new point(3.0, 2.0);     // (see the diagram above)
  291:     point D = translate(C, v);
  292:     // D will be located in coordinate (3.0 + 2.0, 2.0 + 1.0) = (5.0, 3.0)
  293:     System.out.printf("D = (%.2f, %.2f)\n", D.x, D.y);
  294:     point E = translate(C, scale(v, 0.5));
  295:     // E will be located in coordinate (3.0 + 1/2 * 2.0, 2.0 + 1/2 * 1.0) =
  296:     System.out.printf("E = (%.2f, %.2f)\n", E.x, E.y); //        (4.0, 2.5)
  297:
  298:     // rotation
  299:     System.out.printf("B = (%.2f, %.2f)\n", B.x, B.y); // B = (4.0, 3.0)
  300:     point F = rotate(B, 90); // rotate B by 90 degrees COUNTER clockwise,
  301:     System.out.printf("F = (%.2f, %.2f)\n", F.x, F.y); // F = (-3.0, 4.0)
  302:     point G = rotate(B, 180); // rotate B by 180 degrees COUNTER clockwise,
  303:     System.out.printf("G = (%.2f, %.2f)\n", G.x, G.y); // G = (-4.0, -3.0)
  304:   }
  305:
  306:   public static void main(String[] args){
  307:     new ch7_01_points_lines().run();
  308:   }
  309: }
```

```
1: /*****************************************************************************/
2: /** Circulos ..................................................................... */
3: /*****************************************************************************/
4:
5: class ch7_02_circles {
6:   double DEG_to_RAD(double d) { return d * Math.PI / 180.0; }
7:   double RAD_to_DEG(double r) { return r * 180.0 / Math.PI; }
8:
9:   class point_i {
10:    int x, y;                    // whenever possible, work with point_i
11:    point_i() { x = y = 0; }                    // default constructor
12:    point_i(int _x, int _y) { x = _x; y = _y; }         // constructor
13:  };
14:
15:  class point {
16:    double x, y;               // only used if more precision is needed
17:    point() { x = y = 0.0; }                    // default constructor
18:    point(double _x, double _y) { x = _x; y = _y; }     // constructor
19:  };
20:
21:  int insideCircle(point_i p, point_i c, int r) { // all integer version
22:    int dx = p.x - c.x, dy = p.y - c.y;
23:    int Euc = dx * dx + dy * dy, rSq = r * r;           // all integer
24:    return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } // inside/border/outside
25:
26:  boolean circle2PtsRad(point p1, point p2, double r, point c) {
27:    double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
28:                (p1.y - p2.y) * (p1.y - p2.y);
29:    double det = r * r / d2 - 0.25;
30:    if (det < 0.0) return false;
31:    double h = Math.sqrt(det);
32:    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
33:    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
34:    return true; }
35:
36:  void run() {
37:    // circle equation, inside, border, outside
38:    point_i pt = new point_i(2, 2);
39:    int r = 7;
40:    point_i inside = new point_i(8, 2);
41:    System.out.printf("%d\n", insideCircle(inside, pt, r)); // 0-inside
42:    point_i border = new point_i(9, 2);
43:    System.out.printf("%d\n", insideCircle(border, pt, r)); // 1-at border
44:    point_i outside = new point_i(10, 2);
45:    System.out.printf("%d\n", insideCircle(outside, pt, r)); // 2-outside
46:
47:    double d = 2 * r;
48:    System.out.printf("Diameter = %.2f\n", d);
49:    double c = Math.PI * d;
50:    System.out.printf("Circumference / Perimeter = %.2f\n", c);
51:    double A = Math.PI * r * r;
52:    System.out.printf("Area of circle = %.2f\n", A);
53:
54:    System.out.printf("Length of arc (central angle = 60 degrees) = %.2f\n",
55:      60.0 / 360.0 * c);
56:    System.out.printf("Length of chord (central angle = 60 degrees) = %.2f\n",
57:      Math.sqrt((2 * r * r) * (1 - Math.cos(DEG_to_RAD(60.0)))));
58:    System.out.printf("Area of sector (central angle = 60 degrees) = %.2f\n",
59:      60.0 / 360.0 * A);
60:
61:    point p1 = new point();
62:    point p2 = new point(0.0, -1.0);
63:    point ans = new point();
64:    circle2PtsRad(p1, p2, 2.0, ans);
65:    System.out.printf("One of the center is (%.2f, %.2f)\n", ans.x, ans.y);
66:    circle2PtsRad(p2, p1, 2.0, ans);
67:    System.out.printf("The other center  is (%.2f, %.2f)\n", ans.x, ans.y);
68:  }
69:
70:  public static void main(String[] args){
```

```
71:      new ch7_02_circles().run();
72:    }
73: }
```

```
1: /*******************************************************************************/
2: /** Triangulos ..................................................................*/
3: /*******************************************************************************/
4:
5: import java.util.*;
6:
7: class ch7_03_triangles {
8:   final double EPS = 1e-9;
9:
10:   double DEG_to_RAD(double d) { return d * Math.PI / 180.0; }
11:   double RAD_to_DEG(double r) { return r * 180.0 / Math.PI; }
12:
13:   class point_i {
14:     int x, y;                    // whenever possible, work with point_i
15:     point_i() { x = y = 0; }                     // default constructor
16:     point_i(int _x, int _y) { x = _x; y = _y; }          // constructor
17:   };
18:
19:   class point {
20:     double x, y;               // only used if more precision is needed
21:     point() { x = y = 0.0; }                     // default constructor
22:     point(double _x, double _y) { x = _x; y = _y; }      // constructor
23:   };
24:
25:   double dist(point p1, point p2) {
26:     return Math.hypot(p1.x - p2.x, p1.y - p2.y); }
27:
28:   double perimeter(double ab, double bc, double ca) {
29:     return ab + bc + ca; }
30:
31:   double perimeter(point a, point b, point c) {
32:     return dist(a, b) + dist(b, c) + dist(c, a); }
33:
34:   double area(double ab, double bc, double ca) {
35:     //Heron's formula, split sqrt(a * b) into sqrt(a) * sqrt(b); in implementation
36:     double s = 0.5 * perimeter(ab, bc, ca);
37:     return
38:       Math.sqrt(s) * Math.sqrt(s - ab) * Math.sqrt(s - bc) * Math.sqrt(s - ca); }
39:
40:   double area(point a, point b, point c) {
41:     return area(dist(a, b), dist(b, c), dist(c, a)); }
42:
43:   //=====================================================================
44:   // from ch7_01_points_lines
45:   class line { double a, b, c; };              // a way to represent a line
46:
47:   // the answer is stored in the third parameter
48:   void pointsToLine(point p1, point p2, line l) {
49:     if (Math.abs(p1.x - p2.x) < EPS) {             // vertical line is fine
50:       l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;
51:     } else {
52:       l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
53:       l.b = 1.0;                 // IMPORTANT: we fix the value of b to 1.0
54:       l.c = -(double)(l.a * p1.x) - p1.y;
55:   } }
56:
57:   boolean areParallel(line l1, line l2) {       // check coefficients a & b
58:     return (Math.abs(l1.a-l2.a) < EPS) && (Math.abs(l1.b-l2.b) < EPS); }
59:
60:   // returns true (+ intersection point) if two lines are intersect
61:   boolean areIntersect(line l1, line l2, point p) {
62:     if (areParallel(l1, l2)) return false;              // no intersection
63:     // solve system of 2 linear algebraic equations with 2 unknowns
64:     p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
65:     // special case: test for vertical line to avoid division by zero
66:     if (Math.abs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
67:     else                      p.y = -(l2.a * p.x + l2.c);
68:     return true; }
69:
70:   class vec { double x, y;     // name: 'vec' is different from Java Vector
```

```
   71:     vec(double _x, double _y) { x = _x; y = _y; } };
   72:
   73:   vec toVec(point a, point b) {               // convert 2 points to vector
   74:     return new vec(b.x - a.x, b.y - a.y); }
   75:
   76:   vec scale(vec v, double s) {           // nonnegative s = [<1 .. 1 .. >1]
   77:     return new vec(v.x * s, v.y * s); }              // shorter.same.longer
   78:
   79:   point translate(point p, vec v) {           // translate p according to v
   80:     return new point(p.x + v.x , p.y + v.y); }
   81:   //=====================================================================
   82:
   83:   double rInCircle(double ab, double bc, double ca) {
   84:     return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca)); }
   85:
   86:   double rInCircle(point a, point b, point c) {
   87:     return rInCircle(dist(a, b), dist(b, c), dist(c, a)); }
   88:
   89:   // assumption: the required points/lines functions have been written
   90:   // returns 1 if there is an inCircle center, returns 0 otherwise
   91:   // if this function returns 1, ctr will be the inCircle center
   92:   // and r is the same as rInCircle
   93:   int inCircle(point p1, point p2, point p3, point ctr, double r) {
   94:     r = rInCircle(p1, p2, p3);
   95:     if (Math.abs(r) < EPS) return 0;                // no inCircle center
   96:
   97:     line l1 = new line(), l2 = new line(); // compute these two angle bisectors
   98:     double ratio = dist(p1, p2) / dist(p1, p3);
   99:     point p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));
  100:     pointsToLine(p1, p, l1);
  101:
  102:     ratio = dist(p2, p1) / dist(p2, p3);
  103:     p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));
  104:     pointsToLine(p2, p, l2);
  105:
  106:     areIntersect(l1, l2, ctr);           // get their intersection point
  107:     return 1; }
  108:
  109:   double rCircumCircle(double ab, double bc, double ca) {
  110:     return ab * bc * ca / (4.0 * area(ab, bc, ca)); }
  111:
  112:   double rCircumCircle(point a, point b, point c) {
  113:     return rCircumCircle(dist(a, b), dist(b, c), dist(c, a)); }
  114:
  115:   // assumption: the required points/lines functions have been written
  116:   // returns r, the radius of the circumCircle if there is a circumCenter center,
  117:   // and set ctr to be the circumCircle center
  118:   // returns 0 otherwise
  119:   double circumCircle(point p1, point p2, point p3, point ctr) {
  120:     double a = p2.x - p1.x, b = p2.y - p1.y;
  121:     double c = p3.x - p1.x, d = p3.y - p1.y;
  122:     double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
  123:     double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
  124:     double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
  125:     if (Math.abs(g) < EPS) return 0;
  126:
  127:     ctr.x = (d*e - b*f) / g;
  128:     ctr.y = (a*f - c*e) / g;
  129:     return dist(p1, ctr); } // distance from center to 1 of the 3 points
  130:
  131:   // returns true if point d is inside the circumCircle defined by a,b,c
  132:   boolean inCircumCircle(point a, point b, point c, point d) {
  133:     return ((a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y)
* (c.y - d.y)) +
  134:             (a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y))
* (c.x - d.x) +
  135:             ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.x - d.x)
* (c.y - d.y) -
  136:             ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.y - d.y)
* (c.x - d.x) -
```

```
  137:             (a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y)
* (c.y - d.y)) -
  138:             (a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y))
* (c.y - d.y)) > 0.0;
  139:   }
  140:
  141:   boolean canFormTriangle(double a, double b, double c) {
  142:     return (a + b > c) && (a + c > b) && (b + c > a); }
  143:
  144:   void run() {
  145:     double base = 4.0, h = 3.0;
  146:     double A = 0.5 * base * h;
  147:     System.out.printf("Area = %.2f\n", A);
  148:
  149:     point a = new point();                          // a right triangle
  150:     point b = new point(4.0, 0.0);
  151:     point c = new point(4.0, 3.0);
  152:
  153:     double p = perimeter(a, b, c);
  154:     double s = 0.5 * p;
  155:     A = area(a, b, c);
  156:     System.out.printf("Area = %.2f\n", A); // must be the same as above
  157:
  158:     double r = rInCircle(a, b, c);
  159:     System.out.printf("R1 (radius of incircle) = %.2f\n", r); // 1.00
  160:     point ctr = new point();
  161:     int res = inCircle(a, b, c, ctr, r);
  162:     System.out.printf("R1 (radius of incircle) = %.2f\n", r); // same, 1.00
  163:     System.out.printf("Center = (%.2f, %.2f)\n", ctr.x, ctr.y); // (3.00, 1.00)
  164:
  165:     System.out.printf("R2 (radius of circumcircle) = %.2f\n", rCircumCircle(a, b,
c)); // 2.50
  166:     r = circumCircle(a, b, c, ctr);
  167:     System.out.printf("R2 (radius of circumcircle) = %.2f\n", r); // same, 2.50
  168:     System.out.printf("Center = (%.2f, %.2f)\n", ctr.x, ctr.y); // (2.00, 1.50)
  169:
  170:     point d = new point(2.0, 1.0); // inside triangle and circumCircle
  171:     System.out.printf("d inside circumCircle (a, b, c) ? %b\n", inCircumCircle(a,
b, c, d));
  172:     point e = new point(2.0, 3.9); // outside the triangle but inside circumCircle
  173:     System.out.printf("e inside circumCircle (a, b, c) ? %b\n", inCircumCircle(a,
b, c, e));
  174:     point f = new point(2.0, -1.1); // slightly outside
  175:     System.out.printf("f inside circumCircle (a, b, c) ? %b\n", inCircumCircle(a,
b, c, f));
  176:
  177:     // Law of Cosines
  178:     double ab = dist(a, b);
  179:     double bc = dist(b, c);
  180:     double ca = dist(c, a);
  181:     double alpha = RAD_to_DEG(Math.acos((ca * ca + ab * ab - bc * bc) / (2.0 * ca
* ab)));
  182:     System.out.printf("alpha = %.2f\n", alpha);
  183:     double beta  = RAD_to_DEG(Math.acos((ab * ab + bc * bc - ca * ca) / (2.0 * ab
* bc)));
  184:     System.out.printf("beta  = %.2f\n", beta);
  185:     double gamma = RAD_to_DEG(Math.acos((bc * bc + ca * ca - ab * ab) / (2.0 * bc
* ca)));
  186:     System.out.printf("gamma = %.2f\n", gamma);
  187:
  188:     // Law of Sines
  189:     System.out.printf("%.2f == %.2f == %.2f\n",
  190:       bc / Math.sin(DEG_to_RAD(alpha)),
  191:       ca / Math.sin(DEG_to_RAD(beta)),
  192:       ab / Math.sin(DEG_to_RAD(gamma)));
  193:
  194:     // Phytagorean Theorem
  195:     System.out.printf("%.2f^2 == %.2f^2 + %.2f^2\n", ca, ab, bc);
  196:
  197:     // Triangle Inequality
```

```
  198:     System.out.printf("(%d, %d, %d) => can form triangle? %b\n", 3, 4, 5,
canFormTriangle(3, 4, 5)); // yes
  199:     System.out.printf("(%d, %d, %d) => can form triangle? %b\n", 3, 4, 7,
canFormTriangle(3, 4, 7)); // no, actually straight line
  200:     System.out.printf("(%d, %d, %d) => can form triangle? %b\n", 3, 4, 8,
canFormTriangle(3, 4, 8)); // no
  201:   }
  202:
  203:   public static void main(String[] args){
  204:     new ch7_03_triangles().run();
  205:   }
  206: }
```

```java
/*******************************************************************************/
/** Poligonos ..................................................................*/
/*******************************************************************************/

import java.util.*;

class ch7_04_polygon {
  final double EPS = 1e-9;
  // In Java, we can use Math.PI instead of using Math.acos(-1.0)

  double DEG_to_RAD(double d) { return d * Math.PI / 180.0; }
  double RAD_to_DEG(double r) { return r * 180.0 / Math.PI; }

  class point implements Comparable<point>{
    double x, y;                    // only used if more precision is needed
    point() { x = y = 0.0; }                            // default constructor
    point(double _x, double _y) { x = _x; y = _y; }         // user-defined
    // use EPS (1e-9) when testing equality of two floating points
    public int compareTo(point other) {      // override less than operator
      if (Math.abs(x - other.x) > EPS)                  // useful for sorting
        return (int)Math.ceil(x - other.x);       // first: by x-coordinate
      else if (Math.abs(y - other.y) > EPS)
        return (int)Math.ceil(y - other.y);      // second: by y-coordinate
      else
        return 0; } };                                       // they are equal

  class vec { double x, y;      // name: 'vec' is different from Java Vector
    vec(double _x, double _y) { x = _x; y = _y; } };

  vec toVec(point a, point b) {                // convert 2 points to vector
    return new vec(b.x - a.x, b.y - a.y); }

  double dist(point p1, point p2) {                    // Euclidean distance
    return Math.hypot(p1.x - p2.x, p1.y - p2.y); }         // return double

  // returns the perimeter, which is the sum of Euclidian distances
  // of consecutive line segments (polygon edges)
  double perimeter(List<point> P) {
    double result = 0.0;
    for (int i = 0; i < (int)P.size()-1; i++)  // remember that P[0] = P[n-1]
      result += dist(P.get(i), P.get(i+1));
    return result; }

  // returns the area, which is half the determinant
  // works for both convex and concave polygons
  double area(List<point> P) {
    double result = 0.0, x1, y1, x2, y2;
    for (int i = 0; i < (int)P.size()-1; i++) {
      x1 = P.get(i).x; x2 = P.get(i+1).x;
      y1 = P.get(i).y; y2 = P.get(i+1).y;
      result += (x1 * y2 - x2 * y1);
    }
    return Math.abs(result) / 2.0; }

  double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

  double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

  double angle(point a, point o, point b) {     // returns angle aob in rad
    vec oa = toVec(o, a), ob = toVec(o, b);
    return Math.acos(dot(oa, ob) / Math.sqrt(norm_sq(oa) * norm_sq(ob))); }

  double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

  // note: to accept collinear points, we have to change the '> 0'
  // returns true if point r is on the left side of line pq
  boolean ccw(point p, point q, point r) {
    return cross(toVec(p, q), toVec(p, r)) > 0; }

  // returns true if point r is on the same line as the line pq
```

```
  71:   boolean collinear(point p, point q, point r) {
  72:     return Math.abs(cross(toVec(p, q), toVec(p, r))) < EPS; }
  73:
  74:   // returns true if we always make the same turn while examining
  75:   // all the edges of the polygon one by one
  76:   boolean isConvex(List<point> P) {
  77:     int sz = (int)P.size();
  78:     if (sz <= 3) return false;   // a point/sz=2 or a line/sz=3 is not convex
  79:     boolean isLeft = ccw(P.get(0), P.get(1), P.get(2)); // remember one result
  80:     for (int i = 1; i < sz-1; i++)            // then compare with the others
  81:       if (ccw(P.get(i), P.get(i+1), P.get((i+2) == sz ? 1 : i+2)) != isLeft)
  82:         return false;           // different sign -> this polygon is concave
  83:     return true; }                                  // this polygon is convex
  84:
  85:   // returns true if point p is in either convex/concave polygon P
  86:   boolean inPolygon(point pt, List<point> P) {
  87:     if ((int)P.size() == 0) return false;
  88:     double sum = 0; // assume first vertex = last vertex
  89:     for (int i = 0; i < (int)P.size()-1; i++) {
  90:       if (ccw(pt, P.get(i), P.get(i+1)))
  91:           sum += angle(P.get(i), pt, P.get(i+1));   // left turn/ccw
  92:       else sum -= angle(P.get(i), pt, P.get(i+1)); } // right turn/cw
  93:     return Math.abs(Math.abs(sum) - 2*Math.PI) < EPS; }
  94:
  95:   // line segment p-q intersect with line A-B.
  96:   point lineIntersectSeg(point p, point q, point A, point B) {
  97:     double a = B.y - A.y;
  98:     double b = A.x - B.x;
  99:     double c = B.x * A.y - A.x * B.y;
 100:     double u = Math.abs(a * p.x + b * p.y + c);
 101:     double v = Math.abs(a * q.x + b * q.y + c);
 102:     return new point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }
 103:
 104:   // cuts polygon Q along the line formed by point a -> point b
 105:   // (note: the last point must be the same as the first point)
 106:   List<point> cutPolygon(point a, point b, List<point> Q) {
 107:     List<point> P = new ArrayList<point>();
 108:     for (int i = 0; i < (int)Q.size(); i++) {
 109:       double left1 = cross(toVec(a, b), toVec(a, Q.get(i))), left2 = 0;
 110:       if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q.get(i+1)));
 111:       if (left1 > -EPS) P.add(Q.get(i)); // Q[i] is on the left of ab
 112:       if (left1 * left2 < -EPS) // edge (Q[i], Q[i+1]) crosses line ab
 113:         P.add(lineIntersectSeg(Q.get(i), Q.get(i+1), a, b));
 114:     }
 115:     if (!P.isEmpty() && P.get(P.size()-1).compareTo(P.get(0)) != 0)
 116:       P.add(P.get(0)); // make P's first point = P's last point
 117:     return P; }
 118:
 119:   /* Fecho Convexo */
 120:   point pivot = new point();
 121:   List<point> CH(List<point> P) {
 122:     int i, j, n = (int)P.size();
 123:     if (n <= 3) {
 124:       if (P.get(0).compareTo(P.get(n-1)) != 0) P.add(P.get(0));
 125:       // safeguard from corner case
 126:       return P; // special case, the CH is P itself
 127:     }
 128:
 129:     // first, find P0 = point with lowest Y and if tie: rightmost X
 130:     int P0 = 0;
 131:     for (i = 1; i < n; i++)
 132:       if (P.get(i).y  < P.get(P0).y ||
 133:          (P.get(i).y == P.get(P0).y && P.get(i).x > P.get(P0).x))
 134:         P0 = i;
 135:
 136:     point temp = P.get(0); P.set(0, P.get(P0)); P.set(P0 ,temp);
 137:     // swap P[P0] with P[0]
 138:
 139:     // second, sort points by angle w.r.t. P0
 140:     pivot = P.get(0); // use this global variable as reference
```

```
141:     Collections.sort(P, new Comparator<point>(){
142:       public int compare(point a, point b) { // angle-sorting function
143:         if (collinear(pivot, a, b))
144:           return dist(pivot, a) < dist(pivot, b) ? -1 : 1; // which one is closer?
145:         double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
146:         double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
147:         return (Math.atan2(d1y, d1x) - Math.atan2(d2y, d2x)) < 0 ? -1 : 1;
148:        }
149:     });
150:
151:     // third, the ccw tests
152:     List<point> S = new ArrayList<point>();
153:     S.add(P.get(n-1)); S.add(P.get(0)); S.add(P.get(1)); // initial S
154:     i = 2; // then, we check the rest
155:     while (i < n) { // note: n must be >= 3 for this method to work
156:       j = S.size() - 1;
157:       if (ccw(S.get(j-1), S.get(j), P.get(i))) // left turn, accept
158:         S.add(P.get(i++));
159:       else
160:         S.remove(S.size() - 1); // or pop the top of S until we have a left turn
161:     }
162:     return S; } // return the result
163:
164:   void run() {
165:     // 6 points, entered in counter clockwise order, 0-based indexing
166:     List<point> P = new ArrayList<point>();
167:     P.add(new point(1, 1));
168:     P.add(new point(3, 3));
169:     P.add(new point(9, 1));
170:     P.add(new point(12, 4));
171:     P.add(new point(9, 7));
172:     P.add(new point(1, 7));
173:     P.add(P.get(0)); // loop back
174:
175:     System.out.printf("Perimeter of polygon = %.2f\n", perimeter(P)); // 31.64
176:     System.out.printf("Area of polygon = %.2f\n", area(P)); // 49.00
177:     System.out.printf("Is convex = %b\n", isConvex(P)); // false (due to P1)
178:
179:     //// the positions of P6 and P7 w.r.t the polygon
180:     //7 P5--------------P4
181:     //6 |                   \
182:     //5 |                     \
183:     //4 |   P7                 P3
184:     //3 |   P1___             /
185:     //2 | / P6    \ ___     /
186:     //1 P0              P2
187:     //0 1 2 3 4 5 6 7 8 9 101112
188:
189:     point P6 = new point(3, 2); // outside this (concave) polygon
190:     System.out.printf("Point P6 is inside this polygon = %b\n", inPolygon(P6, P));
191:                                                                    // false
192:     point P7 = new point(3, 4); // inside this (concave) polygon
193:     System.out.printf("Point P7 is inside this polygon = %b\n", inPolygon(P7, P));
194:                                                                     // true
195:
196:     // cutting the original polygon based on line P[2] -> P[4] (get the left side)
197:     //7 P5--------------P4
198:     //6 |               |  \
199:     //5 |               |    \
200:     //4 |               |     P3
201:     //3 |   P1___       |    /
202:     //2 | /       \ ___ |  /
203:     //1 P0              P2
204:     //0 1 2 3 4 5 6 7 8 9 101112
205:     // new polygon (notice the index are different now):
206:     //7 P4--------------P3
207:     //6 |               |
208:     //5 |               |
209:     //4 |               |
210:     //3 |   P1___       |
```

```
211:     //2 | /       \ ___ |
212:     //1 P0             P2
213:     //0 1 2 3 4 5 6 7 8 9
214:
215:     P = cutPolygon(P.get(2), P.get(4), P);
216:     System.out.printf("Perimeter of polygon = %.2f\n", perimeter(P)); // now 29.15
217:     System.out.printf("Area of polygon = %.2f\n", area(P)); // 40.00
218:
219:     // running convex hull of the resulting polygon (index changes again)
220:     //7 P3-------------P2
221:     //6 |              |
222:     //5 |              |
223:     //4 |   P7         |
224:     //3 |              |
225:     //2 |              |
226:     //1 P0-------------P1
227:     //0 1 2 3 4 5 6 7 8 9
228:
229:     P = CH(P); // now this is a rectangle
230:     System.out.printf("Perimeter of polygon = %.2f\n", perimeter(P)); // 28.00
231:     System.out.printf("Area of polygon = %.2f\n", area(P)); // precisely 48.00
232:     System.out.printf("Is convex = %b\n", isConvex(P)); // true
233:     System.out.printf("Point P6 is inside this polygon = %b\n", inPolygon(P6, P));
234:                                                                 // true
235:     System.out.printf("Point P7 is inside this polygon = %b\n", inPolygon(P7, P));
236:                                                                 // true
237:   }
238:
239:   public static void main(String[] args){
240:     new ch7_04_polygon().run();
241:   }
242: }
```

```java
/*****************************************************************************/
/** 15-Puzzle Problem with IDA* ..................................................*/
/*****************************************************************************/
// 15-Puzzle Problem with IDA*, 2.975s in Java, 1.758s in C++

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

class ch8_01_UVa10181 {
  static final int INF = 1000000000;
  static final int ROW_SIZE = 4; // ROW_SIZE is a matrix of 4 x 4
  static final int PUZZLE = (ROW_SIZE*ROW_SIZE);
  static final int X = 15;

  static int[] p = new int[PUZZLE];
  static int lim, nlim;
  static int[] dr = new int[] { 0,-1, 0, 1}; // E,N,W,S
  static int[] dc = new int[] { 1, 0,-1, 0}; // R,U,L,D
  static TreeMap<Integer, Integer> pred = new TreeMap<Integer, Integer>();
  static TreeMap<Long, Integer> vis = new TreeMap<Long, Integer>();
  static char[] ans = new char[] {'R', 'U', 'L', 'D'};

  static int h1() { // heuristic: sum of Manhattan distances (compute all)
    int ans = 0;
    for (int i = 0; i < PUZZLE; i++) {
      int tgt_i = p[i] / 4, tgt_j = p[i] % 4;
      if (p[i] != X)
        ans += Math.abs(i / 4 - tgt_i) + Math.abs(i % 4 - tgt_j); // Manhattan
    }                                                             // distance
    return ans;
  }

  // heuristic: sum of manhattan distances (compute delta)
  static int h2(int i1, int j1, int i2, int j2) {
    int tgt_i = p[i2 * 4 + j2] / 4, tgt_j = p[i2 * 4 + j2] % 4;
    return -(Math.abs(i2 - tgt_i) + Math.abs(j2 - tgt_j)) +
      (Math.abs(i1 - tgt_i) + Math.abs(j1 - tgt_j));
  }

  static boolean goal() {
    for (int i = 0; i < PUZZLE; i++)
      if (p[i] != X && p[i] != i)
        return false;
    return true;
  }

  static boolean valid(int r, int c) {
    return 0 <= r && r < 4 && 0 <= c && c < 4;
  }

  static void swap(int i, int j, int new_i, int new_j) {
    int temp = p[i * 4 + j];
    p[i * 4 + j] = p[new_i * 4 + new_j];
    p[new_i * 4 + new_j] = temp;
  }

  static boolean DFS(int g, int h) {
    if (g + h > lim) {
      nlim = Math.min(nlim, g + h);
      return false;
    }

    if (goal())
      return true;

    long state = 0;
    // transform 16 numbers into 64 bits, exactly into ULL
    for (int i = 0; i < PUZZLE; i++) {
      state <<= 4; // move left 4 bits
      state += p[i]; // add this digit (max 15 or 1111)
```

```
    }

    // not pure backtracking... this is to prevent cycling
    if (vis.containsKey(state) && vis.get(state) <= g)
      return false; // not good
    vis.put(state, g); // mark this as visited

    int i, j, d, new_i, new_j;
    for (i = 0; i < PUZZLE; i++)
      if (p[i] == X)
        break;
    j = i % 4;
    i /= 4;

    for (d = 0; d < 4; d++) {
      new_i = i + dr[d]; new_j = j + dc[d];
      if (valid(new_i, new_j)) {
        int dh = h2(i, j, new_i, new_j);
        swap(i, j, new_i, new_j); // swap first
        pred.put(g + 1, d);
        if (DFS(g + 1, h + dh)) // if ok, no need to restore, just go ahead
          return true;
        swap(i, j, new_i, new_j); // restore
      }
    }

    return false;
  }

  static int IDA_Star() {
    lim = h1();
    while (true) {
      nlim = INF; // next limit
      pred.clear();
      vis.clear();
      if (DFS(0, h1()))
        return lim;
      if (nlim == INF)
        return -1;
      lim = nlim; // nlim > lim
      if (lim > 45) // pruning condition in the problem
        return -1;
    }
  }

  static void output(int d) {
    if (d == 0)
      return;
    output(d - 1);
    System.out.printf("%c", ans[pred.get(d)]);
  }

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

    int N = sc.nextInt();
    while (N-- > 0) {
      int i, j, blank = 0, sum = 0, ans = 0;
      for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++) {
          p[i * 4 + j] = sc.nextInt();
          if (p[i * 4 + j] == 0) {
            p[i * 4 + j] = X; // change to X (15)
            blank = i * 4 + j; // remember the index
          }
          else
            p[i * 4 + j]--; // use 0-based indexing
        }

      for (i = 0; i < PUZZLE; i++)
```

```
141:         for (j = 0; j < i; j++)
142:           if (p[i] != X && p[j] != X && p[j] > p[i])
143:             sum++;
144:       sum += blank / ROW_SIZE;
145:
146:       if (sum % 2 != 0 && ((ans = IDA_Star()) != -1)) {
147:         output(ans);
148:         System.out.printf("\n");
149:       }
150:       else
151:         System.out.printf("This puzzle is not solvable.\n");
152:     }
153:   }
154: }
```

```java
/*****************************************************************************/
/** Prog. Dinamica com Bitmask ..................................................*/
/*****************************************************************************/
import java.util.*;

class Main { /* UVa 10911 - Forming Quiz Teams, 0.462s in Java, 0.032s in C++ */
  private static int N, target;
  private static double dist[][] = new double[20][], memo[] = new double[65536];
                                                          // this is 2^16, max N = 8
  private static double matching(int bitmask) {          // DP state = bitmask
                         // we initialize 'memo' with -1 in the main function
    if (memo[bitmask] > -0.5)           // this state has been computed before
      return memo[bitmask];                    // simply lookup the memo table
    if (bitmask == target)                // all students are already matched
      return memo[bitmask] = 0;                                 // the cost is 0

    double ans = 2000000000.0;                // initialize with a large value
    int p1, p2;
    for (p1 = 0; p1 < 2 * N; p1++)
      if ((bitmask & (1 << p1)) == 0)
        break;                                  // find the first bit that is off
    for (p2 = p1 + 1; p2 < 2 * N; p2++)                // then, try to match p1
      if ((bitmask & (1 << p2)) == 0) // with another bit p2 that is also off
        ans = Math.min(ans,                                  // pick the minimum
                  dist[p1][p2] + matching(bitmask | (1 << p1) | (1 << p2)));

    return memo[bitmask] = ans;    // store result in a memo table and return
  }

  public static void main(String[] args) {
    int i, j, caseNo = 1;
    int[] x = new int[20], y = new int[20];

    Scanner sc = new Scanner(System.in);
    while (true) {
      N = sc.nextInt();
      if (N == 0)
        break;

      for (i = 0; i < 2 * N; i++) {
        String name = sc.next(); // dummy
        x[i] = sc.nextInt();
        y[i] = sc.nextInt();
      }

      for (i = 0; i < 2 * N - 1; i++) {
        dist[i] = new double[20];
        for (j = 0; j < 2 * N; j++)
          dist[i][j] = Math.hypot(x[i] - x[j], y[i] - y[j]);
      }

      // use DP to solve min weighted perfect matching on small general graph
      for (i = 0; i < 65536; i++)
        memo[i] = -1.0;
      target = (1 << (2 * N)) - 1;
      System.out.printf("Case %d: %.2f\n", caseNo++, matching(0));
    }
  }
}
```

```java
/*******************************************************************************/
/** Prog. Dinamica (outro exemplo) ..................................... */
/*******************************************************************************/
// ACORN, UVa 1231, LA 4106, 0.???s in Java, 0.???s in C++

import java.util.*;

class Main {
  public static void main(String[] args) {
    int i, j, c, t, h, f, a, n;
    int[][] acorn = new int[2010][2010];
    int[] dp = new int[2010];
    Scanner sc = new Scanner(System.in);

    c = sc.nextInt();
    while (c-- > 0) {
      t = sc.nextInt(); h = sc.nextInt(); f = sc.nextInt();
      for (i = 0; i < 2010; i++)
        for (j = 0; j < 2010; j++)
          acorn[i][j] = 0;
      for (i = 0; i < t; i++) {
        a = sc.nextInt();
        for (j = 0; j < a; j++) {
          n = sc.nextInt();
          acorn[i][n]++; // there is an acorn here
        }
      }

      for (int tree = 0; tree < t; tree++) // initialization
        dp[h] = Math.max(dp[h], acorn[tree][h]);
      for (int height = h - 1; height >= 0; height--)
        for (int tree = 0; tree < t; tree++) {
          acorn[tree][height] +=
            Math.max(acorn[tree][height + 1], // from this tree, +1 above
            ((height + f <= h) ? dp[height + f] : 0));
            // best from tree at height + f
          dp[height] =
            Math.max(dp[height], acorn[tree][height]); // update this too
        }
      System.out.printf("%d\n", dp[0]); // solution will be here
    }
    // ignore the last number 0
  }
}
```

```java
/*********************************************************************************/
/** Outras tecnicas ..................................................... */
/*********************************************************************************/
// World Finals Stockholm 2009, A - A Careful Approach,
// UVa 1079, LA 4445, 0.???s in Java, 0.578s in C++

import java.util.*;

class Main {
  static double[] a = new double[8], b = new double[8];
  static int[] order = new int[8], arr = new int[8];
  static int i, n, caseNo = 1;
  static double L, maxL;

  static double greedyLanding() {  // with certain landing order, and certain L,
          // try landing those planes and see what is the gap to b[order[n - 1]]
    double lastLanding = a[order[0]];      // greedy, 1st aircraft lands ASAP
    for (i = 1; i < n; i++) {                       // for the other aircrafts
      double targetLandingTime = lastLanding + L;
      if (targetLandingTime <= b[order[i]])
        // can land: greedily choose max of a[order[i]] or targetLandingTime
        lastLanding = Math.max(a[order[i]], targetLandingTime);
      else
        return 1;
    }
    // return +ve value to force binary search to reduce L
    // return -ve value to force binary search to increase L
    return lastLanding - b[order[n - 1]];
  }

  static int LSOne(int v) { return v & (-v); }

  // Java does not have next_permutation like C++ <algorithm>
  static void permutate(int cur, int unused) {
    if (cur == n) {
      // do things to curPermute
      double lo = 0, hi = 86400;              // min 0s, max 1 day = 86400s
      L = -1;                              // start with an infeasible solution
      while (Math.abs(lo - hi) >= 1e-3) {    // binary search L, EPS = 1e-3
        L = (lo + hi) / 2.0;   // we want the answer rounded to nearest int
        double retVal = greedyLanding();                // round down first
        if (retVal <= 1e-2) lo = L;                      // must increase L
        else                hi = L;          // infeasible, must decrease L
      }
      maxL = Math.max(maxL, L);        // get the max over all permutations
      return;
    }

    int p = unused;
    while (p > 0) {
      int c = LSOne(p);
      p -= c;
      int i = (int)(Math.log(c) / Math.log(2));
      order[cur] = arr[i];
      permutate(cur + 1, unused - c);
    }
  }

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    while (true) {
      n = sc.nextInt();                                          // 2 <= n <= 8
      if (n == 0) break;

      for (i = 0; i < n; i++) {   // plane i land safely at interval [ai, bi]
        a[i] = sc.nextDouble(); b[i] = sc.nextDouble();
        a[i] *= 60; b[i] *= 60;  // originally in minutes, convert to seconds
        order[i] = i;
        arr[i] = i;
      }
```

```
71:
72:       maxL = -1.0;                              // variable to be searched for
73:       permutate(0, (1 << n) - 1);     // permute plane landing order, up to 8!
74:
75:       // other way for rounding is to use printf format string: %.0lf:%0.2lf
76:       maxL = (int)(maxL + 0.5);                      // round to nearest second
77:       System.out.printf("Case %d: %d:", caseNo++, (int)(maxL/60));
78:       if ((int)maxL%60 < 10) System.out.printf("0"); // one digit?
79:       System.out.printf("%d\n", (int)maxL%60);
80:     }
81:   }
82: }
```

```java
/*****************************************************************************/
/** Eliminacao Gaussiana ..................................................... */
/*****************************************************************************/

import java.util.*;

class AugmentedMatrix {
  public double[][] mat = new double[3][4];  // adjust this value as needed
  public AugmentedMatrix() {};
}

class ColumnVector {
  public double[] vec = new double[3];        // adjust this value as needed
  public ColumnVector() {};
}

class GaussianElimination {
  public static ColumnVector GE(int N, AugmentedMatrix Aug) {
    // input: N, Augmented Matrix Aug, output: Column vector X, the answer
    int i, j, k, l; double t;

    for (i = 0; i < N - 1; i++) {          // the forward elimination phase
      l = i;
      for (j = i + 1; j < N; j++)     // which row has largest column value
        if (Math.abs(Aug.mat[j][i]) > Math.abs(Aug.mat[l][i]))
          l = j;                                    // remember this row l
      // swap this pivot row, reason: minimize floating point error
      for (k = i; k <= N; k++) {        // t is a temporary double variable
        t = Aug.mat[i][k];
        Aug.mat[i][k] = Aug.mat[l][k];
        Aug.mat[l][k] = t;
      }
      for (j = i + 1; j < N; j++)   // the actual forward elimination phase
        for (k = N; k >= i; k--)
          Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];
    }

    ColumnVector Ans = new ColumnVector();   // the back substitution phase
    for (j = N - 1; j >= 0; j--) {                        // start from back
      for (t = 0.0, k = j + 1; k < N; k++) t += Aug.mat[j][k] * Ans.vec[k];
      Ans.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j]; // the answer is here
    }
    return Ans;
  }

  public static void main(String[] args) {
    AugmentedMatrix Aug = new AugmentedMatrix();
    Aug.mat[0][0] = 1; Aug.mat[0][1] = 1; Aug.mat[0][2] = 2; Aug.mat[0][3] = 9;
    Aug.mat[1][0] = 2; Aug.mat[1][1] = 4; Aug.mat[1][2] = -3; Aug.mat[1][3] = 1;
    Aug.mat[2][0] = 3; Aug.mat[2][1] = 6; Aug.mat[2][2] = -5; Aug.mat[2][3] = 0;

    ColumnVector X = GE(3, Aug);
    System.out.printf("X = %.1f, Y = %.1f, Z = %.1f\n",
      X.vec[0], X.vec[1], X.vec[2]);
  }
}
```

```java
/*****************************************************************************/
/** Lowest Common Ancestor (LCA) ..... */
/*****************************************************************************/

import java.util.*;

class LCA {
  public static Vector< Vector < Integer > > children =
    new Vector < Vector < Integer > > ();
  public static int[] L = new int[2000], E = new int[2000], H = new int[2000];
  public static int idx;

  public static void dfs(int cur, int depth) {
    H[cur] = idx;
    E[idx] = cur;
    L[idx++] = depth;
    for (int i = 0; i < children.get(cur).size(); i++) {
      dfs(children.get(cur).get(i), depth+1);
      E[idx] = cur;                              // backtrack to current node
      L[idx++] = depth;
    }
  }

  public static void buildRMQ() {
    idx = 0;
    for (int i = 0; i < 2000; i++)
      H[i] = -1;
    dfs(0, 0);                      // we assume that the root is at index 0
  }

  public static void main(String[] args) {
    for (int i = 0; i < 10; i++)
      children.add(new Vector < Integer > ());

    children.get(0).add(1); children.get(0).add(7);
    children.get(1).add(2); children.get(1).add(3); children.get(1).add(6);
    children.get(3).add(4); children.get(3).add(5);
    children.get(7).add(8); children.get(7).add(9);

    buildRMQ();
    for (int i = 0; i < 2*10-1; i++) System.out.printf("%d ", H[i]);
    System.out.printf("\n");
    for (int i = 0; i < 2*10-1; i++) System.out.printf("%d ", E[i]);
    System.out.printf("\n");
    for (int i = 0; i < 2*10-1; i++) System.out.printf("%d ", L[i]);
    System.out.printf("\n");
  }
}
```

```java
/*****************************************************************************/
/** Pollard Rho (fatoracao) ..................................................... */
/*****************************************************************************/

import java.util.*;

class Pollardsrho {
  // returns (a * b) % c, and minimize overflow
  public static long mulmod(long a, long b, long c) {
    long x = 0, y = a % c;
    while (b > 0) {
      if (b % 2 == 1) x = (x + y) % c;
      y = (y * 2) % c;
      b /= 2;
    }
    return x % c;
  }

  public static long abs_val(long a) { return a >= 0 ? a : -a; }

  public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b); } // standard gcd

  public static long pollard_rho(long n) {
    int i = 0, k = 2;
    long x = 3, y = 3;            // random seed = 3, other values possible
    while (true) {
      i++;
      x = (mulmod(x, x, n) + n - 1) % n;             // generating function
      long d = gcd(abs_val(y - x), n);                  // the key insight
      if (d != 1 && d != n) return d;       // found one non-trivial factor
      if (i == k) { y = x; k *= 2; }
    }
  }

  public static void main(String[] args) {
    long n = 2063512844981574047L; // we assume that n is not a large prime
    long ans = pollard_rho(n);      // break n into two non trivial factors
    if (ans > n / ans) ans = n / ans;         // make ans the smaller factor
    System.out.println(ans + " " + (n / ans)); // should be: 1112041493 1855607779
  }
}
```

```
1: /*****************************************************************************/
2: /** Range Minimum Query (RMQ) ................................................ */
3: /*****************************************************************************/
4: import java.util.*;
5:
6: class RMQ {                                          // Range Minimum Query
7:   private int[] _A = new int[1000];           // adjust this value as needed
8:   private int[][] SpT = new int[1000][10];
9:
10:  public RMQ(int n, int[] A) { // constructor as well as pre-processing routine
11:    for (int i = 0; i < n; i++) {
12:      _A[i] = A[i];
13:      SpT[i][0] = i; // RMQ of sub array starting at index i + length 2^0=1
14:    }
15:    // the two nested loops below have overall time complexity = O(n log n)
16:    for (int j = 1; (1<<j) <= n; j++) // for each j s.t. 2^j <= n, O(log n)
17:      for (int i = 0; i + (1<<j) - 1 < n; i++)    // for each valid i, O(n)
18:        if (_A[SpT[i][j-1]] < _A[SpT[i+(1<<(j-1))][j-1]])            // RMQ
19:          SpT[i][j] = SpT[i][j-1];    // start at index i of length 2^(j-1)
20:        else                    // start at index i+2^(j-1) of length 2^(j-1)
21:          SpT[i][j] = SpT[i+(1<<(j-1))][j-1];
22:  }
23:
24:  public int query(int i, int j) {
25:    int k =
26:      (int)Math.floor(Math.log((double)j-i+1) / Math.log(2.0)); // 2^k <= (j-i+1)
27:    if (_A[SpT[i][k]] <= _A[SpT[j-(1<<k)+1][k]]) return SpT[i][k];
28:    else                                         return SpT[j-(1<<k)+1][k];
29: } };
30:
31: class SparseTable {
32:   public static void main(String[] args) {
33:     // same example as in chapter 2: segment tree
34:     int n = 7;
35:     int[] A = new int[] {18, 17, 13, 19, 15, 11, 20};
36:     RMQ rmq = new RMQ(n, A);
37:     for (int i = 0; i < n; i++)
38:       for (int j = i; j < n; j++)
39:         System.out.printf("RMQ(%d, %d) = %d\n", i, j, rmq.query(i, j));
40:   }
41: }
```

```java
/*********************************************************************************/
/** Fibonacci Modular ..................................................... */
/*********************************************************************************/
// Modular Fibonacci, 0.282s in Java, 0.019s in C++

import java.util.*;

class Main {
  static int i, n, m, MAX_N = 2;
  static long MOD;

  static long[][] matMul(long[][] a, long[][] b) {    // O(n^3 ~> 1) as n=2
    long[][] ans = new long[2][2]; int i, j, k;
    for (i = 0; i < MAX_N; i++)
      for (j = 0; j < MAX_N; j++)
        for (ans[i][j] = k = 0; k < MAX_N; k++) {
          ans[i][j] += (a[i][k] % MOD) * (b[k][j] % MOD);
          ans[i][j] %= MOD;               // modulo arithmetic is used here
        }
    return ans;
  }

  static long[][] matPow(long[][] base, int p) {   // O(n^3 log p ~> log p)
    long[][] ans = new long[MAX_N][MAX_N]; int i, j;
    for (i = 0; i < MAX_N; i++)
      for (j = 0; j < MAX_N; j++)
        ans[i][j] = (i == j ? 1 : 0);            // prepare identity matrix
    while (p > 0) { // iterative version of Divide & Conquer exponentiation
      if ((p & 1) == 1)           // check if p is odd (the last bit is on)
        ans = matMul(ans, base);                          // update ans
      base = matMul(base, base);                        // square the base
      p >>= 1;                                            // divide p by 2
    }
    return ans;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (sc.hasNext()) {
      n = sc.nextInt(); m = sc.nextInt();
      long[][] ans = new long[MAX_N][MAX_N];   // special Fibonaccci matrix
      ans[0][0] = 1;  ans[0][1] = 1;
      ans[1][0] = 1;  ans[1][1] = 0;
      for (MOD = 1, i = 0; i < m; i++)                     // set MOD = 2^m
        MOD *= 2;
      ans = matPow(ans, n);                                   // O(log n)
      System.out.println(ans[0][1]);                     // this if fib(n)
    }
  }
}
```

```java
/*********************************************************************************/
/** Shortest Path Faster Algorithm ..................................... */
/*********************************************************************************/
// Sending email
// standard SSSP problem
// demo using SPFA only
// 2.442s in Java, 0.185s in C++

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

class IntegerPair implements Comparable {
  Integer _first, _second;

  public IntegerPair(Integer f, Integer s) {
    _first = f;
    _second = s;
  }

  public int compareTo(Object o) {
    if (this.first() != ((IntegerPair )o).first())
      return this.first() - ((IntegerPair )o).first();
    else
      return this.second() - ((IntegerPair )o).second();
  }

  Integer first() { return _first; }
  Integer second() { return _second; }
}

class Main {
  public static int i, j, t, n, m, S, T, a, b, w, caseNo = 1;
  public static Vector < Vector < IntegerPair> > AdjList;
  public static final int INF = 2000000000;

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

    t = Integer.parseInt(br.readLine());
    while (t-- > 0) {
      st = new StringTokenizer(br.readLine());
      n = Integer.parseInt(st.nextToken()); m = Integer.parseInt(st.nextToken());
      S = Integer.parseInt(st.nextToken()); T = Integer.parseInt(st.nextToken());

      // build graph
      AdjList = new Vector < Vector < IntegerPair > > ();
      for (i = 0; i < n; i++)
        AdjList.add(new Vector < IntegerPair > ());

      while (m-- > 0) {
        st = new StringTokenizer(br.readLine());
        a = Integer.parseInt(st.nextToken());
        b = Integer.parseInt(st.nextToken());
        w = Integer.parseInt(st.nextToken());
        AdjList.get(a).add(new IntegerPair(b, w)); // bidirectional
        AdjList.get(b).add(new IntegerPair(a, w));
      }

      // SPFA from source S
      // initially, only S has dist = 0 and in the queue
      Vector < Integer > dist = new Vector < Integer >();
      for (i = 0; i < n; i++)
        dist.add(INF);
      dist.set(S, 0);
      Queue < Integer > q = new LinkedList < Integer >();
      q.offer(S);
      Vector < Boolean > in_queue = new Vector < Boolean > ();
```

```
71:       for (i = 0; i < n; i++)
72:         in_queue.add(false);
73:       in_queue.set(S, true);
74:
75:
76:       while (!q.isEmpty()) {
77:         int u = q.peek(); q.poll(); in_queue.set(u, false);
78:         for (j = 0; j < AdjList.get(u).size(); j++) { // all outgoing edges from u
79:           int v = AdjList.get(u).get(j).first(),
80:             weight_u_v = AdjList.get(u).get(j).second();
81:           if (dist.get(u) + weight_u_v < dist.get(v)) { // if can relax
82:             dist.set(v, dist.get(u) + weight_u_v); // relax
83:             if (!in_queue.get(v)) {
84:               q.offer(v); // add to the queue only if it's not in the queue
85:               in_queue.set(v, true);
86:             }
87:           }
88:         }
89:       }
90:
91:       pr.printf("Case #%d: ", caseNo++);
92:       if (dist.get(T) != INF) pr.printf("%d\n", dist.get(T));
93:       else                    pr.printf("unreachable\n");
94:     }
95:
96:   pr.close();
97:   }
98: }
```

```java
/*********************************************************************************/
/** Algarismos Romanos ..................................................... */
/*********************************************************************************/
// Roman Numerals, 0.986s in Java (almost TLE), only 0.032s in C++

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

class Main {
  public static PrintWriter pr;

  public static void AtoR(int A) {
    // process from larger values to smaller values
    TreeMap<Integer, String> cvt =
      new TreeMap<Integer, String>(Collections.reverseOrder());
    cvt.put(1000, "M"); cvt.put(900, "CM"); cvt.put(500, "D"); cvt.put(400, "CD");
    cvt.put(100,  "C"); cvt.put(90,  "XC"); cvt.put(50,  "L"); cvt.put(40,  "XL");
    cvt.put(10,   "X"); cvt.put(9,   "IX"); cvt.put(5,   "V"); cvt.put(4,   "IV");
    cvt.put(1,    "I");

    Set keys = cvt.keySet();
    for (Iterator i = keys.iterator(); i.hasNext();) {
      Integer key = (Integer) i.next();
      String value = (String) cvt.get(key);
      while (A >= key) {
        pr.print(value);
        A -= key;
      }
    }
    pr.printf("\n");
  }

  public static void RtoA(String R) {
    TreeMap<Character, Integer> RtoA = new TreeMap<Character, Integer>();
    RtoA.put('I', 1);   RtoA.put('V', 5);   RtoA.put('X', 10);  RtoA.put('L', 50);
    RtoA.put('C', 100); RtoA.put('D', 500); RtoA.put('M', 1000);

    int value = 0;
    for (int i = 0; i < R.length(); i++)                    // check next char first
      if (i+1 < R.length() && RtoA.get(R.charAt(i)) < RtoA.get(R.charAt(i+1))) {
        value += RtoA.get(R.charAt(i+1)) - RtoA.get(R.charAt(i)); // by definition
        i++; }                                              // skip this char
      else value += RtoA.get(R.charAt(i));
    pr.printf("%d\n", value);
  }

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

    while (true) {
      String str = br.readLine();
      if (str == null) break;
      if (Character.isDigit(str.charAt(0)))
        AtoR(Integer.parseInt(str)); // Arabic to Roman Numerals
      else RtoA(str); // Roman to Arabic Numerals
    }

    pr.close();
  }
}
```

```java
/*******************************************************************************/
/** Distancia entre pontos em esfera + dist. euclidiana ..................... */
/*******************************************************************************/
// Tunnelling the Earth
// Great Circle distance + Euclidean distance
// 0.852s in Java, 0.019s in C++

import java.util.*;

class Main {
  static final int EARTH_RAD = 6371009; // in meters

  static double gcDistance(double pLat, double pLong,
                           double qLat, double qLong, double radius) {
    pLat *= Math.PI / 180; pLong *= Math.PI / 180;
    qLat *= Math.PI / 180; qLong *= Math.PI / 180;
    return radius *
      Math.acos(Math.cos(pLat)*Math.cos(pLong)*Math.cos(qLat)*Math.cos(qLong) +
           Math.cos(pLat)*Math.sin(pLong)*Math.cos(qLat)*Math.sin(qLong) +
           Math.sin(pLat)*Math.sin(qLat));
  }

  static double EucledianDistance(double pLat, double pLong, // 3D version
                                  double qLat, double qLong, double radius) {
    double phi1 = (90 - pLat) * Math.PI / 180;
    double theta1 = (360 - pLong) * Math.PI / 180;
    double x1 = radius * Math.sin(phi1) * Math.cos(theta1);
    double y1 = radius * Math.sin(phi1) * Math.sin(theta1);
    double z1 = radius * Math.cos(phi1);

    double phi2 = (90 - qLat) * Math.PI / 180;
    double theta2 = (360 - qLong) * Math.PI / 180;
    double x2 = radius * Math.sin(phi2) * Math.cos(theta2);
    double y2 = radius * Math.sin(phi2) * Math.sin(theta2);
    double z2 = radius * Math.cos(phi2);

    double dx = x1 - x2, dy = y1 - y2, dz = z1 - z2;
    return Math.sqrt(dx * dx + dy * dy + dz * dz);
  }

  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int TC = scan.nextInt();
    while (TC-- > 0) {
      double lat1 = scan.nextDouble();
      double lon1 = scan.nextDouble();
      double lat2 = scan.nextDouble();
      double lon2 = scan.nextDouble();
      System.out.printf("%.0f\n", gcDistance(lat1, lon1, lat2, lon2, EARTH_RAD) -
                                  EucledianDistance(lat1, lon1, lat2, lon2, EARTH_RAD));
    }
  }
}
```

```java
/*******************************************************************************/
/** Componentes Fortemente Conectadas ..................................... */
/*******************************************************************************/
// Come and Go
// check if the graph is strongly connected,
// i.e. the SCC of the graph is the graph itself (only 1 SCC)
// 0.835s in Java, 0.092s in C++

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

class IntegerPair implements Comparable {
  Integer _first, _second;

  public IntegerPair(Integer f, Integer s) {
    _first = f;
    _second = s;
  }

  public int compareTo(Object o) {
    if (this.first() != ((IntegerPair )o).first())
      return this.first() - ((IntegerPair )o).first();
    else
      return this.second() - ((IntegerPair )o).second();
  }

  Integer first() { return _first; }
  Integer second() { return _second; }
}

class Main {
  public static final int DFS_WHITE = -1;

  public static int i, j, N, M, V, W, P, numSCC;
  public static Vector < Vector < IntegerPair > > AdjList, AdjListT;
  public static Vector < Integer > dfs_num, S; // global variables

  public static void Kosaraju(int u, int pass) { // pass = 1 (original),
    dfs_num.set(u, 1);                           //        2 (transpose)
    Vector < IntegerPair > neighbor;
    if (pass == 1) neighbor = AdjList.get(u); else neighbor = AdjListT.get(u);
    for (int j = 0; j < neighbor.size(); j++) {
      IntegerPair v = neighbor.get(j);
      if (dfs_num.get(v.first()) == DFS_WHITE)
        Kosaraju(v.first(), pass);
    }
    S.add(u); // as in finding topological order in Section 4.2.5
  }

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

    while (true) {
      st = new StringTokenizer(br.readLine());
      N = Integer.parseInt(st.nextToken()); M = Integer.parseInt(st.nextToken());
      if (N == 0 && M == 0) break;

      AdjList = new Vector < Vector < IntegerPair > > ();
      AdjListT = new Vector < Vector < IntegerPair > > (); // the transposed graph
      for (i = 0; i < N; i++) {
        AdjList.add(new Vector < IntegerPair >());
        AdjListT.add(new Vector < IntegerPair >());
      }

      for (i = 0; i < M; i++) {
        st = new StringTokenizer(br.readLine());
        V = Integer.parseInt(st.nextToken());
```

```
71:         W = Integer.parseInt(st.nextToken());
72:         P = Integer.parseInt(st.nextToken());
73:         V--; W--;
74:         AdjList.get(V).add(new IntegerPair(W, 1)); // always
75:         AdjListT.get(W).add(new IntegerPair(V, 1));
76:         if (P == 2) { // if this is two way, add the reverse direction
77:           AdjList.get(W).add(new IntegerPair(V, 1));
78:           AdjListT.get(V).add(new IntegerPair(W, 1));
79:         }
80:       }
81:
82:       // run Kosaraju's SCC code here
83:       S = new Vector < Integer > (); // first pass is to record the 'post-order'
84:       dfs_num = new Vector < Integer > ();                  // of original graph
85:
86:       for (i = 0; i < N; i++)
87:         dfs_num.add(DFS_WHITE);
88:       for (i = 0; i < N; i++)
89:         if (dfs_num.get(i) == DFS_WHITE)
90:           Kosaraju(i, 1);
91:
92:       numSCC = 0; // second pass: explore the SCCs based on first pass result
93:       dfs_num = new Vector < Integer > ();
94:       for (i = 0; i < N; i++)
95:         dfs_num.add(DFS_WHITE);
96:       for (i = N-1; i >= 0; i--)
97:         if (dfs_num.get(S.get(i)) == DFS_WHITE) {
98:           numSCC++;
99:           Kosaraju(S.get(i), 2);
100:        }
101:
102:      // if SCC is only 1, print 1, otherwise, print 0
103:      pr.printf("%d\n", numSCC == 1 ? 1 : 0);
104:    }
105:
106:    pr.close();
107:  }
108: }
```