algorithms apps bluemask.

```
1
2   public class Graduation {
3
4       private int N;
5       private int K;
6       private int M;
7       private int L;
8
9       private int[] pre_requisites;
10      private int[] subjects;
11
12      private static final int MAX = 999999;
13
14      public Graduation(int N, int K, int M, int L, int[] pre_requisites, int[]
        subjects) {
15
16          this.N = N;
17          this.K = K;
18          this.M = M;
19          this.L = L;
20
21          this.pre_requisites = pre_requisites;
22          this.subjects = subjects;
23
24  //      System.out.println(N + ", " + K + ", " + M + ", " + L);
25  //      for(int i=0; i<N; i++)
26  //          System.out.println(pre_requisites[i]);
27  //      for(int i=0; i<M; i++)
28  //          System.out.println(semesters[i]);
29      }
30
31      public int min() {
32
33          return min(0, 0, 0);
34      }
35
36      public int min(int present, int semester, int apply) {
37
38          if(Integer.bitCount(present) >= K)
39              return apply;
40          if(semester == M)
41              return MAX;
42
43          int min = MAX;
44
45          // not apply
46          min = min(min, min(present, semester+1, apply));
47
48          // apply
49          int canTake = subjects[semester]&(~present);
50          for(int i=0; i<N; i++) {
51              if((canTake&(1<<i)) == (1<<i) &&
52                      (pre_requisites[i]&present) != pre_requisites[i])
53                  canTake &= ~(1<<i);
54          }
55          for(int take=canTake; take>0; take=(take-1)&canTake) {
56
57              if(Integer.bitCount(take) > L)
58                  continue;
59
60              min = min(min, min(present|take, semester+1, apply+1));
61          }
62
63          return min;
64      }
65
66      private int min(int a, int b) {
67          if(a<b) return a;
68          else    return b;
69      }
70
71      public static void main(String[] args) throws NumberFormatException,
```

```
        IOException {
72
73          if(args.length == 0) {
74              System.out.println("args fails.");
75              return;
76          }
77
78          FastReader in = new FastReader(args);
79
80          int test = in.nextInt();
81          for(int t=0; t<test; t++) {
82
83              int N = in.nextInt();
84              int K = in.nextInt();
85              int M = in.nextInt();
86              int L = in.nextInt();
87
88              int[] pre_requisites = new int[N];
89              for(int i=0; i<N; i++) {
90                  int pre_req = 0;
91                  int r = in.nextInt();
92                  for(int j=0; j<r; j++)
93                      pre_req |= (1<<in.nextInt());
94                  pre_requisites[i] = pre_req;
95              }
96
97              int[] subjects = new int[M];
98              for(int i=0; i<M; i++) {
99                  int subject = 0;
100                 int c = in.nextInt();
101                 for(int j=0; j<c; j++)
102                     subject |= (1<<in.nextInt());
103                 subjects[i] = subject;
104             }
105
106             Graduation g = n                    M, L, pre_requisites, subjects);
107             System.out.println(g.min());
108         }
109     }
110 }
```

```
1   package algorithms.apss.combinatorial.search;
2
3   /**
4    *
5    * @author taehyeok.jang
6    *
7    * @problem Alergy
8    * @reference APSS 426p.
9    *
10   * @description
11   */
12  public class Alergy {
13
14  }
15
```

```java
package algorithms.apss.combinatorial.search;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

/**
 *
 * @author taehyeok.jang
 *
 * @problem BoardCover
 * @reference APSS 422p.
 *
 * @description
 * find a maximum number of blocks which are put on given board.
 */
public class BoardCover {

    private int h, w;
    private int[][] board;
    private int r, c;
    private int m;
    private int[][] block;
    private int block_size;

    private int best;

    private int[][] rotation;
    private int[][] aux;

    Vector<Vector<Coord>> rotations;

    public BoardCover(int h,                    :d, int r, int c, int[][] block) {

        this.h = h;
        this.w = w;
        this.board = board;
        this.r = r;
        this.c = c;
        this.m = max(r, c);
        this.block = block;

        best = 0;

        init();
    }

    public int max() {

        return search(0, 0, 0);
    }

    private int search(int x, int y, int placed) {

        int left = (h*w-(x*w+y));
        if(placed + left/block_size < best)
            return 0;

        if(x == h)
            return best = placed;
        if(y == w)
            return search(x+1, 0, placed);

        int max = 0;

        Iterator<Vector<Coord>> rotates = rotations.iterator();
        Vector<Coord> r;
        while(rotates.hasNext()) {

            r = rotates.next();
```

```
            if(isPossible(x, y, r)) {
                set(x, y, r, 1);
                max = max(max, search(x, y+1, placed+1));
                set(x, y, r, 0);
            }
        }

        max = max(max, search(x, y+1, placed));

        return max;
    }


    private boolean isPossible(int x, int y, Vector<Coord> rotation) {

        Iterator<Coord> iterator = rotation.iterator();
        while(iterator.hasNext()) {
            int cx, cy;
            Coord coord = iterator.next();
            cx = x + coord.x;
            cy = y + coord.y;
            if(cx<0 || cx>=h || cy<0 || cy>=w)
                return false;
            if(board[cx][cy] > 0)
                return false;
        }
        return true;
    }
    private boolean set(int x, int y, Vector<Coord> rotation, int s) {

        Iterator<Coord> coords = rotation.iterator();
        while(coords.hasNext()) {
            int cx, cy;
            Coord coord = co
            cx = x + coord.x
            cy = y + coord.y
            board[cx][cy] = s;
        }
        return true;
    }

    private void init() {

        rotation = new int[m][m];
        aux = new int[m][m];

        for(int i=0; i<m; i++)
            for(int j=0; j<m; j++)
                rotation[i][j] = 0;
        for(int i=0; i<r; i++)
            for(int j=0; j<c; j++) {
                if(block[i][j] == 1)
                    block_size++;
                rotation[i][j] = block[i][j];
            }

        rotations = new Vector<Vector<Coord>>();
        for(int i=0; i<4; i++)
            rotations.add(new Vector<Coord>());

        int v = 0;
        while(v < 4) {
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
                    if(rotation[i][j] == 1)
                        rotations.get(v).add(new Coord(i, j));
            rotate(m, rotation, aux);
            close(m, rotation);
            v++;
        }
    }
```

```
145
146     private void print(int[][] rectangle) {
147
148         for(int i=0; i<rectangle.length; i++) {
149             for(int j=0; j<rectangle[0].length; j++)
150                 System.out.print(rectangle[i][j] + " ");
151             System.out.println();
152         }
153     }
154
155     private void print(int n, int[][] sqr) {
156
157         for(int i=0; i<n; i++) {
158             for(int j=0; j<n; j++)
159                 System.out.print(sqr[i][j] + " ");
160             System.out.println();
161         }
162     }
163
164     /*
165     private void printRotations() {
166
167         Iterator<Vector<Coord>> i = rotations.iterator();
168         while(i.hasNext()) {
169             Iterator<Coord> j = i.next().iterator();
170             while(j.hasNext()) {
171                 Coord c = j.next();
172                 System.out.println(c.x + ", "+ c.y);
173             }
174             System.out.println();
175         }
176     }
177     */
178
179     // rotate clockwise.
180     private void rotate(int n, int[][] sqr, int[][] aux) {
181
182         for(int i=0; i<n; i++)
183             for(int j=0; j<n; j++)
184                 aux[i][j] = sqr[i][j];
185
186         for(int i=0; i<n; i++)
187             for(int j=0; j<n; j++)
188                 sqr[i][j] = aux[n-1-j][i];
189     }
190     private void close(int n, int[][] sqr) {
191
192         while(true) {
193             int count = 0;
194             for(int i=0; i<n; i++)
195                 if(sqr[0][i] == 0)
196                     count++;
197             if(count < n)
198                 break;
199             else {
200                 for(int i=0; i<n-1; i++)
201                     for(int j=0; j<n; j++)
202                         sqr[i][j] = sqr[i+1][j];
203                 for(int j=0; j<n; j++)
204                     sqr[n-1][j] = 0;
205             }
206         }
207
208         while(true) {
209             int count = 0;
210             for(int i=0; i<n; i++)
211                 if(sqr[i][0] == 0)
212                     count++;
213             if(count < n)
214                 break;
215             else {
216                 for(int i=0; i<n; i++)
```

```
                    for(int j=0; j<n-1; j++)
                        sqr[i][j] = sqr[i][j+1];
                for(int i=0; i<n; i++)
                    sqr[i][n-1] = 0;
            }
        }
    }

    private int max(int a, int b) {
        if(a>b) return a;
        else    return b;
    }

    private class Coord {

        private final int x;
        private final int y;

        public Coord(int x, int y) {

            this.x = x;
            this.y = y;
        }
    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
            return;
        }

        BufferedReader br = [illegible]ew FileReader(args[0]));

        int numOfTest;
        numOfTest = Integer.parseInt(br.readLine());

        String s;
        String[] sArr;

        for(int test=0; test<numOfTest; test++) {

            sArr = br.readLine().split(" ");

            int h = Integer.parseInt(sArr[0]);
            int w = Integer.parseInt(sArr[1]);
            int r = Integer.parseInt(sArr[2]);
            int c = Integer.parseInt(sArr[3]);

            int[][] board = new int[h][w];
            int[][] block = new int[r][c];

            for(int i=0; i<h; i++) {
                s = br.readLine();
                for(int j=0; j<w; j++)
                    if(s.charAt(j) == '#') board[i][j] = 1;
                    else                   board[i][j] = 0;
            }

            for(int i=0; i<r; i++) {
                s = br.readLine();
                for(int j=0; j<c; j++)
                    if(s.charAt(j) == '#') block[i][j] = 1;
                    else                   block[i][j] = 0;
            }
            double start = System.currentTimeMillis();

            BoardCover bc = new BoardCover(h, w, board, r, c, block);
            System.out.println(bc.max());

            double end = System.currentTimeMillis();
```

```
288                 System.out.println("time: " + (end-start));
289             }
290
291             br.close();
292         }
293
294     }
295
```

```
1   package algorithms.apss.combinatorial.search;
2
3   import java.io.FileReader;
4   import java.io.BufferedReader;
5   import java.io.IOException;
6   import java.util.Arrays;
7   import java.util.Random;
8
9
10  /**
11   *  error report
12   *
13   *  1. stack overflow. ...
14   *  2. cannot check basic rule, which is that line sum equals to bound. we can
     set at every put.
15   *  3. in case of previous solution, we cannot search all combinatorial cases. ->
     step search expected.
16   *  4. real-time synchronization of bound. can we overcome dfs with set, un-set?
     yes.
17   *  5. dfs forward each line, or every direction?
18   *      technique
19   *          there is no clear evidence that dfs should traverse line by line.
20   *          at each line, jump if filled. xxx (deprecated).
21   *          each dfs should return some signal to caller function.
22   *
23   *  7. dfs is prominent as a solution, but it is hard to implement.
24   */
25
26  public class Kakuro {
27
28      private int n;
29      private int[][] board;
30      private int total;
31      private int q;
32      private Hint[] hints;
33
34      private int[][] solution;
35
36      // direction 0...3: right, down, left, up.
37      private static int[] dx = {0, 1, 0, -1};
38      private static int[] dy = {1, 0, -1, 0};
39
40      public Kakuro(int n, int[][] board, int total, int q, Hint[] hints) {
41
42          this.n = n;
43          this.board = board;
44          this.q = q;
45          this.hints = hints;
46          this.total = total;
47
48          for(int i=0; i<q; i++)
49              board[hints[i].x][hints[i].y] = hints[i].sum;
50
51          solution = new int[n][n];
52          for(int i=0; i<n; i++)
53              for(int j=0; j<n; j++)
54                  solution[i][j] = 0;
55
56          print(n, board);
57
58  //      for(int i=0; i<q; i++)
59  //          System.out.println(hints[i].x + " " + hints[i].y + " " + hints[i].d +
     " "  + hints[i].sum + " " + hints[i].length);
60
61      }
62
63      public void solve() {
64
65          // TODO
66          // this pre-process is not needed except finding initial start point.
            maybe?
67          // not sure. pruning is no that efficient by just selecting start point.
```

```
68
69          Arrays.sort(hints);
70          dfs(hints[0].x, hints[0].y, 0);
71      }
72
73      private boolean dfs(int x, int y, int count) {
74
75  //      debugging.
76  //      if(count < 100) {
77  //          System.out.println("dfs " + x + ", " + y + "... " + count);
78  //          print(n, solution);
79  //      }
80
81          if(total == count)
82              print(n, solution);
83
84          int cx, cy;
85          boolean isPossible = false; // should handle marginal case carefully.
86          for(int d=0; d<4; d++) {
87
88              cx = x + dx[d];
89              cy = y + dy[d];
90
91              if(cx<1 || cx >= n || cy<1 || cy >= n)
92                  continue;
93              if(board[cx][cy] != 1 && solution[cx][cy] != 0)
94                  continue;
95
96              boolean isLast = false;
97              int direction = -1; // dummy.
98              for(int i=0; i<2; i++)
99                  if(isLast(cx, cy, i) && isLast(cx, cy, i+2)) {
100                     isLast = true;
101                     direction
102                 }
103
104             if(isLast) {
105
106                 // TODO !!!!!!
107                 // always set two-way bound
108                 int bound = bound(cx, cy, direction, -1);
109
110                 if(isSatisfy(cx, cy, bound)) {
111
112                     solution[cx][cy] = bound;
113                     bound(cx, cy, direction, 0);
114
115                     dfs(cx, cy, count+1);
116
117                     // un-set TODO is un-set really needed? because any block
                        must have number greater than 0.
118                     solution[cx][cy] = 0;
119                     bound(cx, cy, direction, bound);
120                 }
121                 else {
122
123                     continue;
124                 }
125
126             }
127             else {
128
129
130                 // TODO !!!!!!
131                 // always set two-way bound
132
133                 // TODO line-sum == bound
134                 int min = 9;
135                 min = min(min, bound(cx, cy, 0, -1));
136                 min = min(min, bound(cx, cy, 1, -1));
137
138                 int bound = bound(cx, cy, direction, -1);
```

```
139
140                 // TODO cannot always start from 1.
141                 for(int i=1; i<min; i++) {
142
143                     solution[cx][cy] = i;
144                     bound(cx, cy, direction, bound-i);
145
146                     dfs(cx, cy, count+1);
147
148                     // un-set TODO is un-set really needed? because any block
                        must have number greater than 0.
149                     solution[cx][cy] = 0;
150                     bound(cx, cy, direction, bound);
151                 }
152             }
153
154         }
155
156         return isPossible;
157     }
158
159     /**
160      * @param x, y  coordinate
161      * @param d     direction
162      * @param set   -1: get, else: set
163      */
164     private int bound(int x, int y, int d, int set) {
165
166         if(d<0 || d>1)
167             throw new IllegalArgumentException();
168
169         while(board[x][y] == 1) {
170             x -= dx[d];
171             y -= dy[d];
172         }
173         if(set == -1)   return board[x][y];         // get
174         else            return board[x][y] = set;   // set
175     }
176
177     private boolean isLast(int x, int y, int d) {
178
179         if(d<0 || d>3)
180             throw new IllegalArgumentException();
181
182         while(true) {
183             x += dx[d];
184             y += dy[d];
185             if((board[x][y] != 1) && (x<1 || x >= n || y<1 || y >= n))
186                 break;
187             if(solution[x][y] == 0)
188                 return false;
189         }
190         return true;
191     }
192
193     private boolean isSatisfy(int x, int y, int set) {
194
195         // range check
196         if(set<1 || set>9)
197             return false;
198
199         // uniqueness check
200         int cx, cy;
201         for(int d=0; d<4; d++) {
202             cx = x;
203             cy = y;
204             while(true) {
205                 cx += dx[d];
206                 cy += dy[d];
207                 if((board[cx][cy] != 1) && (cx<1 || cx >= n || cy<1 || cy >= n))
208                     break;
209                 if(solution[cx][cy] == set)
```

```
                    return false;
                }
            }

            return true;

        }


        private int min(int a, int b) {
            if(a<b) return a;
            else    return b;
        }
        private void print(int size, int[][] square) {
            for(int i=0; i<size; i++) {
                for(int j=0; j<size; j++)
                    System.out.print(square[i][j] + "\t");
                System.out.println();
            }
        }

        private static class Hint implements Comparable {

            private final int x;
            private final int y;
            private final int d; // 0: horizontal, 1: vertical.
            private int sum;
            private int length;

            public Hint(int x, int y, int d, int sum) {

                this.x = x-1;
                this.y = y-1;
                this.d = d;
                this.sum = sum;
                this.length = -1;
            }

            // must be called.
            private void length(int n, int[][] board) {

                int l = 0;
                int cx = x + dx[d]; int cy = y + dy[d];
                while((cx < n && cy < n) && board[cx][cy] == 1) {
                    l++;
                    cx += dx[d];
                    cy += dy[d];
                }

                length = l;
            }

            @Override
            public int compareTo(Object arg0) {

                Hint that = (Hint)arg0;
                if(this.length == -1 || that.length == -1)
                    throw new IllegalArgumentException();

                if(this.length > that.length)       return 1;
                else if(this.length < that.length)  return -1;
                else {
                    if(this.sum > that.sum)         return 1;
                    else if(this.sum < that.sum)    return -1;
                    else                            return 0;
                }
            }
        }

        public static void main(String[] args) throws NumberFormatException,
        IOException {
```

```
281         if(args.length == 0) {
282             System.out.println("args fails.");
283             return;
284         }
285
286         BufferedReader br = new BufferedReader(new FileReader(args[0]));
287
288         int test = Integer.parseInt(br.readLine());
289         String[] sArr;
290
291         for(int t=0; t<test; t++) {
292
293             int n = Integer.parseInt(br.readLine());
294             int[][] board = new int[n][n];
295             int total = 0;
296             for(int i=0; i<n; i++) {
297                 sArr = br.readLine().split(" ");
298                 for(int j=0; j<n; j++) {
299                     int s = Integer.parseInt(sArr[j]);
300                     board[i][j] = s;
301                     if(s == 1)
302                         total++;
303                 }
304             }
305             int q = Integer.parseInt(br.readLine());
306             Hint[] hints = new Hint[q];
307             for(int i=0; i<q; i++) {
308                 sArr = br.readLine().split(" ");
309                 hints[i] = new Hint(Integer.parseInt(sArr[0]),
                    Integer.parseInt(sArr[1]), Integer.parseInt(sArr[2]),
                    Integer.parseInt(sArr[3]));
310                 hints[i].length(n, board);
311             }
312
313             Kakuro k = new K                    al, q, hints);
314 //          k.solve();
315         }
316         br.close();
317     }
318
319 }
320
```

```
1   package algorithms.apss.combinatorial.search;
2
3   import java.io.FileReader;
4   import java.io.BufferedReader;
5   import java.io.IOException;
6   import java.util.Arrays;
7
8   public class Kakuro2 {
9
10      private int n;
11      private int[][] board;
12      private int total;
13      private int q;
14      private Hint[] hints;
15
16      private int[][][] candidates;
17
18      private int[][] solution;
19
20      // direction 0...3: right, down, left, up.
21      private static int[] dx = {0, 1, 0, -1};
22      private static int[] dy = {1, 0, -1, 0};
23
24
25      public Kakuro2(int n, int[][] board, int total, int q, Hint[] hints) {
26
27          this.n =
28          this.board = board;
29          this.q = q;
30          this.hints = hints;
31          this.total = total;
32
33          for(int i=0; i<q; i++)
34              board[hints[i].x][hints[i].y] = hints[i].sum;
35
36          candidates = new int[10][40][1024];
37
38          solution = new int[n][n];
39          for(int i=0; i<n; i++)
40              for(int j=0; j<n; j++)
41                  solution[i][j] = 0;
42
43          print(n, board);
44  //
45  //          for(int i=0; i<q; i++)
46  //              System.out.println(hints[i].x + " " + hints[i].y + " " + hints[i].d +
    " "  + hints[i].sum + " " + hints[i].length);
47
48      }
49
50      public void solve() {
51
52          // TODO
53          // this pre-process is not needed except finding initial start point.
            maybe?
54          // not sure. pruning is no that efficient by just selecting start point.
55          Arrays.sort(hints);
56      }
57
58      private int min(int a, int b) {
59          if(a<b) return a;
60          else    return b;
61      }
62      private void print(int size, int[][] square) {
63          for(int i=0; i<size; i++) {
64              for(int j=0; j<size; j++)
65                  System.out.print(square[i][j] + "\t");
66              System.out.println();
67          }
68      }
69
70      private static class Hint implements Comparable {
```

```
71
72          private final int x;
73          private final int y;
74          private final int d; // 0: horizontal, 1: vertical.
75          private int sum;
76          private int length;
77
78          public Hint(int x, int y, int d, int sum) {
79
80              this.x = x-1;
81              this.y = y-1;
82              this.d = d;
83              this.sum = sum;
84              this.length = -1; // dummy
85          }
86
87          // must be called.
88          private void length(int n, int[][] board) {
89
90              int l = 0;
91              int cx = x + dx[d]; int cy = y + dy[d];
92              while((cx < n && cy < n) && board[cx][cy] == 1) {
93                  l++;
94                  cx += dx[d];
95                  cy += dy[d];
96              }
97
98              length = l;
99          }
100
101         @Override
102         public int compareTo(Object arg0) {
103
104             Hint that = (Hint)arg0;
105             if(this.length ==                    == -1)
106                 throw new IllegalArgumentException();
107
108             if(this.length > that.length)        return 1;
109             else if(this.length < that.length)   return -1;
110             else {
111                 if(this.sum > that.sum)          return 1;
112                 else if(this.sum < that.sum)     return -1;
113                 else                             return 0;
114             }
115         }
116     }
117
118     public static void main(String[] args) throws NumberFormatException,
        IOException (
119
120         if(args.length == 0) {
121             System.out.println("args fails.");
122             return;
123         }
124
125         BufferedReader br = new BufferedReader(new FileReader(args[0]));
126
127         int test = Integer.parseInt(br.readLine());
128         String[] sArr;
129
130         for(int t=0; t<test; t++) {
131
132             int n = Integer.parseInt(br.readLine());
133             int[][] board = new int[n][n];
134             int total = 0;
135             for(int i=0; i<n; i++) {
136                 sArr = br.readLine().split(" ");
137                 for(int j=0; j<n; j++) {
138                     int s = Integer.parseInt(sArr[j]);
139                     board[i][j] = s;
140                     if(s == 1)
141                         total++;
```

```
142                     }
143                 }
144                 int q = Integer.parseInt(br.readLine());
145                 Hint[] hints = new Hint[q];
146                 for(int i=0; i<q; i++) {
147                     sArr = br.readLine().split(" ");
148                     hints[i] = new Hint(Integer.parseInt(sArr[0]),
                        Integer.parseInt(sArr[1]), Integer.parseInt(sArr[2]),
                        Integer.parseInt(sArr[3]));
149                     hints[i].length(n, board);
150                 }
151
152                 Kakuro2 k = new Kakuro2(n, board, total, q, hints);
153 //              k.solve();
154             }
155             br.close();
156         }
157
158 }
159
```

```java
package algorithms.apss.combinatorial.search;

import java.util.Random;

import edu.princeton.cs.algs4.StdDraw;

/**
 *
 * @author taehyeok.jang
 *
 * @problem LTs
 * @reference APSS 422p.
 *
 * @description
 * linear transformation practice & visualization
 */
public class LT {

    /**
     * time complexity: Θ(n^2)
     * space complexity: Θ(n^2), no additional space.
     */
    public static void rotate(int[][] matrix, int n) {

        for(int x=0; x<n/2; x++) {

            for(int y=x; y<n-x-1; y++) {

                int temp = matrix[x][y];

                matrix[x][y]            = matrix[y][n-x-1];
                matrix[y][n-x-1]        = matrix[n-x-1][n-y-1];
                matrix[n-x-1][n-y-1]    = matrix[n-y-1][x];
                matrix[n-y-1][x]        = temp;
            }
        }
    }

    /**
     * @param coord 2-dimensional
     * @param theta theta
     * @return transformed coord
     */
    public static double[] LinearTransform(double[] coord, double theta) {

//      double[] c = new double[2];
//
//      c[0] = coord[0]*Math.cos(theta*Math.PI) + coord[1]*Math.sin(theta*Math.PI);
//      c[1] = -coord[0]*Math.sin(theta*Math.PI) +
coord[1]*Math.cos(theta*Math.PI);
//
//      return c;

        double x = coord[0]*Math.cos(theta*Math.PI) -
        coord[1]*Math.sin(theta*Math.PI);
        double y = coord[0]*Math.sin(theta*Math.PI) +
        coord[1]*Math.cos(theta*Math.PI);

        coord[0] = x; coord[1] = y;

        return coord;
    }

    public static void main(String[] args) {

//      StdDraw.setCanvasSize(1200, 1200);
//
        StdDraw.setXscale(0, 400);
        StdDraw.setYscale(0, 400);

        StdDraw.setPenRadius(0.002);
        StdDraw.setPenColor(StdDraw.BLACK);
```

```
70  //      StdDraw.line(50, 50, 250, 50); // x0, y0, x1, y1.
71
72  //      StdDraw.setPenRadius(0.02);
73  //      StdDraw.setPenColor(StdDraw.RED);
74  //      for(int j=0; j<k; j++)
75  //          StdDraw.point(50 + arr[j], 50);
76
77          double[] coord = new double[2];
78
79          final double ix = 100.0;
80          final double iy = 100.0;
81
82          final double offset = 200;
83
84          coord[0] = ix;
85          coord[1] = iy;
86
87          for(int i=0; i<1000; i++) {
88
89              StdDraw.point(offset + coord[0], offset + coord[1]);
90
91              coord = LinearTransform(coord, (double)i*(1.0/400000.0));
92          }
93
94
95
96  //      Random r = new Random();
97  //
98  //      int radius = 100;
99  //      double offset = 0.0;
100 //      for(int i=0; i<1000; i++) {
101 //
102 //          double x = (radius + offset) * Math.cos((double)i * (1/1000.0) *
    Math.PI) + 200;
103 //          double y = (radi                    sin((double)i * (1/1000.0) *
    Math.PI) + 200;
104 //
105 ////             offset += 0.01 * r.nextInt(5);
106 //
107 //          StdDraw.point(x, y);
108 //      }
109     }
110 }
111
```

```
1   package algorithms.apss.decision.problem;
2
3   import java.util.LinkedList;
4   import java.util.Queue;
5
6   /**
7    * @author taehyeok.jang
8    *
9    * @problem Arctic
10   * @reference APSS 450p.
11   *
12   * @description
13   * find a minimum range of telegraphic communication so that all basements are
     able to commnunicate.
14   */
15  public class Arctic {
16
17          private int n;
18          private Base[] bases;
19          private double[][] edge;
20
21          private double max;
22
23          private Queue<Integer> queue;
24          private boolean[] visited;
25
26          public Arctic(int n, Base[] bases) {
27
28              this.n = n ;
29              this.bases = bases;
30
31              edge = new double[n][n];
32              for(int i=0; i<n; i++)
33                  for(int j=0; j<n; j++)
34                      edge[i][                    [i], bases[j]);
35
36              max = -1;
37              for(int i=0; i<n; i++)
38                  for(int j=0; j<n; j++)
39                      max = max(max, edge[i][j]);
40
41              queue = new LinkedList<Integer>();
42              visited = new boolean[n];
43          }
44
45          private double optimize() {
46
47              double lo = 0.0;
48              double hi = max;
49              double mid;
50
51              int iteration = 0;
52              while(iteration++ < 100) {
53
54                  mid = (lo+hi)/2;
55
56                  if(decision(mid))   hi = mid;
57                  else                lo = mid;
58              }
59
60              return lo; // error term: (hi-lo)/2^101.
61          }
62
63          // can all basements be connected to others given distance or less.
64          private boolean decision(double distance) {
65
66              queue.clear();
67              for(int i=0; i<n; i++)
68                  visited[i] = false;
69
70              queue.add(0);
71
```

```
72              while(!queue.isEmpty()) {
73
74                  int index = queue.remove();
75                  visited[index] = true;
76
77                  for(int i=0; i<n; i++)
78                      if(edge[index][i] <= distance && visited[i] == false)
79                          queue.add(i);
80              }
81
82              for(int i=0; i<n; i++)
83                  if(visited[i] == false)
84                      return false;
85
86              return true;
87          }
88
89          private double max(double a, double b) {
90              if(Double.compare(a, b)>0)  return a;
91              else                        return b;
92          }
93
94          private double distance(Base a, Base b) {
95
96              return Math.sqrt((b.x-a.x)*(b.x-a.x) + (b.y-a.y)*(b.y-a.y));
97          }
98
99          private static class Base {
100
101             private double x;
102             private double y;
103
104             public Base(double x, double y) {
105                 this.x = x;
106                 this.y = y;
107             }
108         }
109
110         public static void main(String[] args) throws NumberFormatException,
            IOException {
111
112             if(args.length == 0) {
113                 System.out.println("args fails.");
114                 return;
115             }
116
117             BufferedReader br = new BufferedReader(new FileReader(args[0]));
118
119             int test = Integer.parseInt(br.readLine());
120             String[] sArr;
121
122             for(int t=0; t<test; t++) {
123
124                 int n = Integer.parseInt(br.readLine());
125                 Base[] bases = new Base[n];
126                 for(int i=0; i<n; i++) {
127                     sArr = br.readLine().split(" ");
128                     bases[i] = new Base(Double.parseDouble(sArr[0]),
                        Double.parseDouble(sArr[1]));
129                 }
130
131                 Arctic a = new Arctic(n, bases);
132                 System.out.println(a.optimize());
133             }
134
135             br.close();
136     }
137 }
138
```

```
1   package algorithms.apss.decision.problem;
2
3   import java.io.FileReader;
4   import java.io.BufferedReader;
5   import java.io.IOException;
6
7   /**
8    * @author taehyeok.jang
9    *
10   * @problem Arctic
11   * @reference APSS 450p.
12   *
13   * @description
14   * find a minimum range of telegraphic communication so that all basements are
     able to communicate.
15   */
16  public class CanadaTrip {
17
18
19      /**
20       * array implementation is impossible, because k<2^31 and length<10^7 would
         make a program very inefficient.
21       *
22       */
23
24      private int n;
25      private int[] L;
26      private int[] M;
27      private int[] G;
28
29      private int max;
30
31      public CanadaTrip(int n, int[] L, int[] M, int[] G) {
32
33          this.n = n ;
34          this.L = L;
35          this.M = M;
36          this.G = G;
37
38          max = 0;
39          for(int i=0; i<n; i++)
40              max = max(max, L[i]);
41      }
42
43      public int location(int k) {
44
45          return optimize(k);
46      }
47
48      private int optimize(int k) {
49
50  //      System.out.println("k: " + k);
51
52          int lo = 0;
53          int hi = max;
54          int mid;
55
56          int iteration = 0;
57          while(iteration++ < 30) {
58
59              mid = (lo+hi)/2;
60
61              if(decision(mid, k))    hi = mid;
62              else                    lo = mid;
63          }
64
65          return hi;
66      }
67
68      // given location, are there greater or equal signs than k?
69      private boolean decision(int location, int k) {
70
```

```
71          int sign = 0;
72          for(int i=0; i<n; i++) {
73
74              int start = L[i]-M[i];
75              for(int j=0; j<M[i]/G[i]+1; j++) { // able to calculate by O(1).
76                  if(start <= location)
77                      sign++;
78                  start += G[i];
79              }
80          }
81
82          return sign >= k;
83      }
84
85      private int max(int a, int b) {
86          if(a>b) return a;
87          else    return b;
88      }
89
90      public static void main(String[] args) throws NumberFormatException,
        IOException {
91
92          if(args.length == 0) {
93              System.out.println("args fails.");
94              return;
95          }
96
97          BufferedReader br = new BufferedReader(new FileReader(args[0]));
98
99          int test = Integer.parseInt(br.readLine());
100         String[] sArr;
101
102         for(int t=0; t<test; t++) {
103
104             sArr = br.readLi
105             int n = Integer.
106             int k = Integer.parseInt(sArr[1]);
107
108             int[] L = new int[n];
109             int[] M = new int[n];
110             int[] G = new int[n];
111
112             for(int i=0; i<n; i++) {
113                 sArr = br.readLine().split(" ");
114                 L[i] = Integer.parseInt(sArr[0]);
115                 M[i] = Integer.parseInt(sArr[1]);
116                 G[i] = Integer.parseInt(sArr[2]);
117             }
118
119             CanadaTrip ct = new CanadaTrip(n, L, M, G);
120             System.out.println(ct.location(k));
121         }
122
123         br.close();
124     }
125
126 }
127
```

```java
package algorithms.apss.decision.problem;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;


/**
 * @author taehyeok.jang
 *
 * @problem DARPA Grand Challenge
 * @reference APSS 446p.
 *
 * @description
 * find a maximum interval between adjacent cameras.
 */
public class Darpa {

    private int m, n;
    private int length;
    private int[] location;

    public Darpa(int m, int n, int length, int[] location) {

        this.m = m;
        this.n = n;
        this.length = length;
        this.location = location;
    }

    // select n cameras so that it maximize minimum interval.
    public int optimize() {

        int lo = 0;
        int hi = length;
        int mid;

        int iteration = 0;
        while(iteration++ < 100) {

            mid = (lo+hi)/2;

            if(decision(mid))   lo = mid;
            else                hi = mid;
        }

        return lo; // error term: (hi-lo)/2^101.
    }

    // can set more than n cameras given interval.
    private boolean decision(int interval) {

        int set = 0;
        int limit = -1;
        for(int i=0; i<m; i++) {

            if(limit <= location[i]) {
                set++;
                limit = location[i] + interval;
            }
        }

        return set >= n;
    }


    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
            return;
```

```
72          }
73
74          BufferedReader br = new BufferedReader(new FileReader(args[0]));
75
76          int numOfTest;
77          numOfTest = Integer.parseInt(br.readLine());
78
79          Object obj;
80
81          String s;
82          String[] sArr;
83
84          for(int test=0; test<numOfTest; test++) {
85
86              //System.out.println((test+1) + "-th test");
87
88              sArr = br.readLine().split(" ");
89              int m = Integer.parseInt(sArr[0]);
90              int n = Integer.parseInt(sArr[1]);
91              int length = Integer.parseInt(br.readLine());
92
93              int[] location = new int[m];
94              sArr = br.readLine().split(" ");
95              for(int i=0; i<m; i++)
96                  location[i] = Integer.parseInt(sArr[i]);
97
98              Darpa d = new Darpa(m, n, length, location);
99              System.out.println(d.optimize());
100
101         }
102
103         br.close();
104     }
105
106
107 }
108
```

```
1   package algorithms.apss.decision.problem;
2
3   import java.io.FileReader;
4   import java.io.BufferedReader;
5   import java.io.IOException;
6   import java.util.Collections;
7   import java.util.Iterator;
8   import java.util.PriorityQueue;
9   import java.util.Vector;
10
11  public class Withdrawl {
12
13      private int n;
14      private int k;
15      private Subject[] subjects;
16
17      private Vector<Double> v;
18
19      public Withdrawl(int n, int k, Subject[] subjects) {
20
21          this.n = n;
22          this.k = k;
23          this.subjects = subjects;
24
25          v = new Vector<Double>();
26      }
27
28      public double min() {
29
30          return optimize();
31      }
32
33      private double optimize() {
34
35          double lo = 0;
36          double hi = 1;
37          double mid;
38
39          int iteration = 0;
40          while(iteration++ < 100) {
41
42              mid = (lo+hi)/2;
43
44              if(decision(mid))   hi = mid;
45              else                lo = mid;
46          }
47
48          return lo;
49      }
50
51      // can we find cumulative rank less or equal than given cumulate?
52      private boolean decision(double cumulate) {
53
54          v.clear();
55
56          for(int i=0; i<n; i++)
57              v.add(cumulate*subjects[i].total-subjects[i].rank);
58
59          Collections.sort(v);
60
61          double sum = 0;
62          Iterator<Double> iterator = v.iterator();
63          for(int i=0; i<n-k; i++)
64              iterator.next();
65          while(iterator.hasNext())
66              sum += iterator.next();
67
68          return sum >= 0;
69      }
70
71      private double min(double a, double b) {
72          if(a<b) return a;
```

```
73          else    return b;
74      }
75
76      private static class Cummulate implements Comparable {
77
78          private double c;
79
80          public Cummulate(double c) {
81
82              this.c = c;
83          }
84
85          @Override
86          public int compareTo(Object arg0) {
87
88              return Double.compare(this.c, ((Cummulate)arg0).c);
89          }
90
91      }
92
93      private static class Subject implements Comparable {
94
95          private final int rank;
96          private final int total;
97
98          private final double ratio;
99
100         public Subject(int rank, int total) {
101
102             this.rank = rank;
103             this.total = total;
104
105             this.ratio = (double)rank/(double)total;
106         }
107
108         @Override
109         public int compareTo(Object arg0) {
110
111             return Double.compare(this.ratio, ((Subject)arg0).ratio);
112         }
113     }
114
115     public static void main(String[] args) throws NumberFormatException,
        IOException {
116
117         if(args.length == 0) {
118             System.out.println("args fails.");
119             return;
120         }
121
122         BufferedReader br = new BufferedReader(new FileReader(args[0]));
123
124         int test = Integer.parseInt(br.readLine());
125         String[] sArr;
126
127         for(int t=0; t<test; t++) {
128
129             sArr = br.readLine().split(" ");
130             int n = Integer.parseInt(sArr[0]);
131             int k = Integer.parseInt(sArr[1]);
132
133             Subject[] subjects = new Subject[n];
134             sArr = br.readLine().split(" ");
135             for(int i=0; i<n; i++) {
136                 subjects[i] = new Subject(Integer.parseInt(sArr[2*i]),
                    Integer.parseInt(sArr[2*i+1]));
137             }
138
139             Withdrawl w = new Withdrawl(n, k, subjects);
140             System.out.println(w.min());
141         }
142
```

```
143             br.close();
144         }
145
146
147     }
148
```

```java
package algorithms.apss.divide.and.conquer;

public class Fence {

    /**
     * @@@ Fence
     *
     *  - page
     *      195p
     *
     *  - description
     *      find a maximum fence area to be recycled.
     *
     */

    private int[] board;
    private int maxArea;

    public Fence(int[] board) {

        this.board = board;
        this.maxArea = -1;
    }

    public void findMaxArea() {

        this.maxArea = findMaxArea(this.board, 0, this.board.length-1);
    }

    // return size
    private int findMaxArea(int[] arr, int left, int right) {

//      StdOut.println("find " + left + ", " + right);

        if(left >= right)
            return arr[left];

        int l= findMaxArea(arr, left, (left+right)/2);
        int r = findMaxArea(arr, (left+right)/2 + 1, right);

        int area = max(l, r);

        int creep = creep(arr, (left+right)/2, left, right);

        return max(area, creep);
    }

    private int creep(int[] arr, int pivot, int left, int right) {

        int low = pivot;
        int hi = pivot + 1;

        int height = min(arr[low], arr[hi]);
        int area = height * 2;

        while(low > left || hi < right) {

            if(low > left && (hi == right || arr[low-1] > arr[hi+1])) {
                low--;
                height = min(arr[low], height);
            }
            else {
                hi++;
                height = min(height, arr[hi]);
            }

            area = max(area, height * (hi-low+1));
        }

        return area;
    }

```

```
73      private int max(int a, int b) {
74
75          if(a>=b) return a;
76          else     return b;
77      }
78
79      private int min(int a, int b) {
80
81          if(a<=b) return a;
82          else     return b;
83      }
84
85      public int getMaxArea() (
86
87          return maxArea;
88      }
89
90      public static void main(String[] args) (
91
92          int numOfTest;
93          Fence fence;
94
95          int[] board;
96
97          numOfTest = StdIn.readInt();
98
99          for(int i=0; i<numOfTest; i++) {
100
101             int n = StdIn.readInt();
102             board = new int[n];
103
104             for(int j=0; j<n; j++)
105                 board[j] = StdIn.readInt();
106
107             fence = new Fenc
108             fence.findMaxArea...
109
110             StdOut.println(fence.getMaxArea());
111         }
112
113     }
114 }
115
```

```
1  package algorithms.apss.divide.and.conquer;
2
3  public class Karatsuba {
4
5      public Karatsuba() {
6
7      }
8  }
9
```

```java
package algorithms.apss.divide.and.conquer;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class QuadTree {

    /**
     * @@@ QuadTree
     *
     *  - page
     *      191p
     *
     *  - description
     *      quad-tree compress a given picture that was flop upside-down.
     *
     *      compress : in-order traversal (1, 2, 3, 4)
     *      flip-compress : in-order traversal (3, 4, 1, 2)
     *
     *  - related class
     *
     */
    private char[] arr;
    private char[] aux;

    public QuadTree(String compress) {

        this.arr = compress.toCharArray();
        this.aux = new char[this.arr.length];
    }

    public void decompress() {

        decompress(0);
    }

    private int[] decompress(int index) {

//      StdOut.println("decompress : " + arr[index] + " at " + index);

        if(index>this.arr.length-1)
            return null;

        int[] range = new int[2];

        if(this.arr[index] == 'w' || this.arr[index] == 'b') {

            range[0] = index;
            range[1] = index;

            return range;
        }

        int[][] partition = new int[4][2];

        partition[0] = decompress(index+1);
        partition[1] = decompress(partition[0][1]+1);
        partition[2] = decompress(partition[1][1]+1);
        partition[3] = decompress(partition[2][1]+1);

        range[0] = index;
        range[1] = partition[3][1];

//      StdOut.println("range " + range[0] + ", " + range[1]);

        flip(this.arr, partition);

        return range;
    }


    /**
```

```
73          * transform {a[], b[], c[], d[]} into {c[], d[], a[], b[]}
74          * @param partition range of input arrays.
75          */
76         private void flip(char[] arr, int[][] partition) {
77
78             int first = partition[0][0];
79             int pivot = partition[2][0];
80             int last = partition[3][1];
81
82             for(int i=0; i<last+1; i++)
83                 this.aux[i] = arr[i];
84
85             int index = first;
86             int j;
87
88             j = pivot;
89             while(j<last+1)
90                 arr[index++] = this.aux[j++];
91
92             j = first;
93             while(index<last+1)
94                 arr[index++] = this.aux[j++];
95         }
96
97
98     /*
99      *  limitation : implemented by recursive function call,
100     *               but code is not so intuitive (just well implemented as DFS
        traversal).
101     *               also base case and recurrence is mixed in a function,
102     *               it is not implemented by divide-conquer strategy.
103     *
104     * private int decompress(int index) {
105
106            int first = index;
107            int[] position = new
108
109            for(int i=0; i<4; i++) {
110
111                if(index > this.arr.length-1) // flip complete.
112                    return index;
113
114                if(arr[index] == 'x')
115                    index = decompress(index+1)-1;
116
117                position[i] = index++;
118            }
119
120            flip(arr, first, position);
121
122            return index; // return a last index of a decompressed area.
123        }*/
124
125        /**
126         * transform (a[], b[], c[], d[]) into (c[], d[], a[], b[])
127         * @param first a[0]
128         * @param last  a[last], b[last], ...
129         * @return
130         */
131    /*  private char[] flip(char[] arr, int first, int[] position) {
132
133            for(int i=first; i<position[3]+1; i++)
134                this.aux[i] = arr[i];
135
136            int pivot = position[1] + 1;
137            int last = position[3] + 1;
138
139            int index = first;
140            int j;
141
142            j = pivot;
143            while(j<last)
```

```
144                 arr[index++] = this.aux[j++];
145
146             j = first;
147             while(index<last)
148                 arr[index++] = this.aux[j++];
149
150             return arr;
151         }*/
152
153         public void print() {
154
155             for(int i=0; i<arr.length; i++)
156                 StdOut.print(arr[i]);
157             StdOut.println();
158         }
159
160         public static void main(String[] args) {
161
162             int numOfTest;
163             QuadTree quadTree;
164
165             numOfTest = StdIn.readInt();
166
167             for(int i=0; i<numOfTest; i++) {
168
169                 String compress = StdIn.readString();
170
171                 quadTree = new QuadTree(compress);
172                 quadTree.decompress();
173
174                 quadTree.print();
175             }
176         }
177     }
178
```

```
1  package algorithms.apss.dynamic.programming;
2
3  import java.io.BufferedReader;
4  import java.io.FileReader;
5  import java.io.IOException;
6
7  public class AsymTiling {
8
9      private int length;
10     private Tiling tile;
11
12     private final int MOD = 1000 * 1000 * 1000 + 7;
13
14     public AsymTiling(int n) {
15
16         this.length = n;
17         this.tile = new Tiling(n);
18     }
19
20     public int getNum() {
21
22         return asymmetric(this.length);
23     }
24
25     public void print() {
26
27         for(int i=0; i<this.length; i++)
28             System.out.println(i + " : " + tile.tiling(i));
29     }
30
31     private int asymmetric(int n) {
32
33         if(n % 2 == 1)
34             return (tile.tili                    (n/2) + MOD) % MOD;
35
36         int asym = tile.tili
37         asym = (asym - tile.tiling(n/2) + MOD) % MOD;
38         asym = (asym - tile.tiling(n/2-1) + MOD) % MOD;
39
40         return asym;
41     }
42
43
44     public static void main(String[] args) throws NumberFormatException,
       IOException {
45
46         if(args.length == 0) {
47             System.out.println("args fails.");
48             return;
49         }
50
51         BufferedReader br = new BufferedReader(new FileReader(args[0]));
52
53         int numOfTest;
54         AsymTiling at;
55
56         int n;
57
58         numOfTest = Integer.parseInt(br.readLine());
59
60         for(int i=0; i<numOfTest; i++) {
61
62             n = Integer.valueOf(br.readLine());
63
64             at = new AsymTiling(n);
65
66             System.out.println("@@@@ result : " + at.getNum());
67
68 //          at.print();
69         }
70         br.close();
71
```

```java
package algorithms.apss.dynamic.programming;

public class Binomial {

    /**
     * @@@ Binomial
     *
     *  - page
     *      209p
     *
     *  - description
     *      binomial is a representative function that accelerates a performance
     *      by dynamic programming.
     *
     *
     *      (n)combination(r) = (n-1)combination(r) + (n-1)combination(r-1).
     *
     */


    static final int CACHE_SIZE = 100;
    private static int[][] cache;

    // this is called before invoking any methods
    static {

        cache = new int[CACHE_SIZE][CACHE_SIZE];

        for(int i=0; i<CACHE_SIZE; i++)
            for(int j=0; j<CACHE_SIZE; j++)
                cache[i][j] = -1;
    }

    public static int combination(int n, int r) {

        if(r == 0 || n == r)
            return 1;
        if(cache[n][r] != -1) {
//          System.out.println("static caching ...");
            return cache[n][r];

        }

        return (cache[n][r] = combination(n-1, r) + combination(n-1, r-1));
    }


    public static void main(String[] args) {

        final int ITERATION = 100;

        int n = 0;
//      while(n++ < ITERATION) {
//          for(int i=0; i<25; i++)
//              for(int j=0; j<i; j++) {
//                  binomial.combination(i, j);
//              }
//      }

        System.out.println(combination(9, 5));
    }

}
```

```java
package algorithms.apss.dynamic.programming;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class JLIS {

    /**
     * @@@ JLIS
     *
     *  - page
     *      236p
     *
     *  - description
     *      find a longest (strictly) increasing joined sub-sequence.
     *      after getting LIS from each sequence, joined by increasing order.
     *
     */

    /**
     *
     * @@@ check list
     *
     * 1. how to check a duplicated between aSeq and bSeq ?
     * => choose a element of each sequence bigger than max(aSeq[a], bSeq[b]).
     *
     * 2. simple iteration looks not enough. but 2-depth iteration must be heavy to compute.
     * => fine enough :)
     *
     */

    private int[] aSeq;
    private int[] bSeq;

    private int[][] cache;

    public JLIS(String a, String b) {

        String[] sa = a.split(" ");
        String[] sb = b.split(" ");

        this.aSeq = new int[sa.length];
        this.bSeq = new int[sb.length];

        for(int i=0; i<sa.length; i++)
            this.aSeq[i] = Integer.parseInt(sa[i]);
        for(int i=0; i<sb.length; i++)
            this.bSeq[i] = Integer.parseInt(sb[i]);

        this.cache = new int[sa.length+1][sb.length+1];
        for(int i=0; i<sa.length+1; i++)
            for(int j=0; j<sb.length+1; j++)
                this.cache[i][j] = -1;
    }

    public int get() {

        return get(-1, -1) - 2;
    }


    // return a maximum length of joined sequence that starts from aSeq[a], bSeq[b]
    public int get(int a, int b) {

        if(cache[a+1][b+1] != -1)
            return cache[a+1][b+1];

        int aElem = (a == -1 ? -1 : aSeq[a]);
        int bElem = (b == -1 ? -1 : bSeq[b]);
```

```
72          int maxElement = max(aElem, bElem);
73          int max = 2;
74
75          for(int i=a+1; i<aSeq.length; i++)
76              if(aSeq[i] > maxElement)
77                  max = max(max, 1+get(i, b));
78          for(int i=b+1; i<bSeq.length; i++)
79              if(bSeq[i] > maxElement)
80                  max = max(max, 1+get(a, i));
81
82          return cache[a+1][b+1] = max;
83      }
84
85      private int max(int a, int b) {
86          if(a>b) return a;
87          else    return b;
88      }
89
90      public static void main(String[] args) throws IOException {
91
92          if(args.length == 0) {
93              System.out.println("args fails.");
94              return;
95          }
96
97          BufferedReader br = new BufferedReader(new FileReader(args[0]));
98
99          int numOfTest;
100         JLIS jlis;
101         String a, b;
102
103         numOfTest = Integer.parseInt(br.readLine());
104
105         for(int i=0; i<numOf
106
107             a = br.readLine();
108             b = br.readLine();
109
110             jlis = new JLIS(a, b);
111
112             System.out.println(jlis.get());
113         }
114         br.close();
115
116     }
117 }
118
```

```java
package algorithms.apss.dynamic.programming;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

import edu.princeton.cs.algs4.StdOut;

public class LIS {

    /***
     * @@@ LIS
     *
     *  - page
     *      230p
     *
     *  - description
     *      find a longest (strictly) increasing sub-sequence.
     *
     *
     */

    private int[] sequence;
    private int[] cache;

    public int[] output = {
        3, 7, 12, 8, 3, 7, 6
    };

    public LIS(String input) {

        String[] arr = input.split(" ");

        sequence = new int[a
        for(int i=0; i<arr.l
            sequence[i] = In                    ]);

        cache = new int[arr.length];
        for(int i=0; i<arr.length; i++)
            cache[i] = -1;
    }

    public int get() {

        return get(-1) - 1;
    }

    public int get(int start) {

        if(start == sequence.length-1)
            return 1;
        if(cache[start+1] != -1)
            return cache[start+1];

        int max = 1;
        for(int i=start+1; i<sequence.length; i++)
            if(start == -1 || sequence[start] < sequence[i])
                max = max(max, 1+get(i));

        return cache[start+1] = max;
    }

    private int max(int a, int b) {
        if(a>b) return a;
        else    return b;
    }


    public static void main(String[] args) throws IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
```

```
73              return;
74          }
75
76          BufferedReader br = new BufferedReader(new FileReader(args[0]));
77
78          int numOfTest;
79          LIS lis;
80          String sequence;
81
82          numOfTest = Integer.parseInt(br.readLine());
83
84          for(int i=0; i<numOfTest; i++) {
85
86              sequence = br.readLine();
87              lis = new LIS(sequence);
88
89              System.out.println("@@@@ result : " + lis.get());
90
91
92          }
93          br.close();
94
95      }
96  }
97
```

```java
package algorithms.apss.dynamic.programming;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class Numb3rs {

    /**
     *  @@@ Numb3rs
     *
     *  - page
     *      269p
     *
     *  - description
     *      find a probability for Dr.Dunival to hide in each village.
     *
     */


    private int numOfVillages;  // the number of villages
    private double[][] probabilityFromTo; // [from village][to village]
    private double[][] probabilityOfDays; // [days][village]
    /**
     *
     * @param n     the number of villages
     * @param d     days
     * @param p     index of a village which a prison is in
     * @param path  matrix of paths
     */
    public Numb3rs(int n, int d, int p, int[][] paths) {

        this.numOfVillages = n;
        probabilityFromTo = new double[n][n];
        probabilityOfDays = new double[d+1][n];

        int count;
        for(int i=0; i<n; i++){
            count = 0;
            for(int j=0; j<n; j++)
                if(paths[i][j] == 1)
                    count++;
            for(int j=0; j<n; j++) {
                if(paths[i][j] == 1)    probabilityFromTo[i][j] = 1/(double)count;
                else                    probabilityFromTo[i][j] = 0;
            }
        }

        for(int j=0; j<n; j++)
            probabilityOfDays[0][j] = 0;
        for(int i=1; i<d+1; i++)
            for(int j=0; j<n; j++)
                probabilityOfDays[i][j] = -1.0;
        probabilityOfDays[0][p] = 1;
    }

    /**
     * already solved sub-problems are called in O(1) by dynamic programming.
     * time complexity : T(n) = O(d*(n^2))
     * @param d days
     * @param q index of a village
     * @return  probability
     */
    private double probability(int d, int q) {

        if(probabilityOfDays[d][q] != -1.0)
            return probabilityOfDays[d][q];

        double probability = 0.0;
        for(int i=0; i<this.numOfVillages; i++)
            probability += probability(d-1, i) * probabilityFromTo[i][q];

        return probabilityOfDays[d][q] = probability;
```

```
73      }
74
75      /**
76       *
77       * @param q index of a village
78       * @return  probability
79       */
80      public double getProbability(int q) {
81
82          return probability(this.probabilityOfDays.length-1, q);
83      }
84
85      public static void main(String[] args) {
86
87          int numOfTest;
88          Numb3rs numb3rs;
89          int[][] paths;
90
91          numOfTest = StdIn.readInt();
92
93          for(int i=0; i<numOfTest; i++) {
94
95              int n = StdIn.readInt();
96              int d = StdIn.readInt();
97              int p = StdIn.readInt();
98
99              paths = new int[n][n];
100
101             for(int j=0; j<n; j++)
102                 for(int k=0; k<n; k++)
103                     paths[j][k] = StdIn.readInt();
104
105             numb3rs = new Numb3rs(n, d, p, paths);
106
107             int t = StdIn.re
108             int q;
109             for(int j=0; j<t; j++) {
110                 q = StdIn.readInt();
111                 StdOut.print(numb3rs.getProbability(q) + "\t");
112             }
113
114             StdOut.println();
115         }
116     }
117 }
118
```

```
1  package algorithms.apss.dynamic.programming;
2
3  import java.io.BufferedReader;
4  import java.io.FileReader;
5  import java.io.IOException;
6
7  public class PI {
8
9      /**
10      * @@@ PI
11      *
12      *  - page
13      *      239p
14      *
15      *  - description
16      *
17      *
18      */
19
20     private int[] seq;
21     private final int[] level = {1, 2, 4, 5, 10};
22
23     private int POS_INF = 987654321;
24     private int NEG_INF = -987654321;
25
26     public PI(String s) {
27
28         char[] c = s.toCharArray();
29
30         seq = new int[c.length+5];
31         for(int i=0; i<c.length; i++)
32             seq[i] = Integer.valueOf(c[i]);
33         for(int i=c.length; i<c.length+5; i++)
34             seq[i] = -1;
35     }
36
37     public int getMin() {
38
39         return getMin(0);
40     }
41     /**
42      * @@@ check list
43      *
44      * 1.
45      * we cannot decide levels of interval by simply seeing that interval.
46      * we should consider elements around interval.
47      *
48      * 2.
49      *
50      *
51      *
52      *
53      */
54     private int getMin(int start) {
55
56         System.out.println("start: " + start);
57
58         if(start > seq.length -4) {
59             System.out.println("boundary case");
60             return -1; // TODO should implement boundary case carefully.
61         }
62
63
64         int length = 0; // needed to be initialized only once.
65
66         /**
67          * each pattern check start~start+8
68          */
69
70         // pattern 1
71         for(int i=start; i<start+5; i++) {
72
```

```
            if(seq[i] == seq[i+1] && seq[i] == seq[i+2]) {

                System.out.println("case 1");
                length = 3;

                if(seq[i] == seq[i+3]) length++;
                if(seq[i] == seq[i+4]) length++;

                return level[1] + getMin(i+length);
            }
        }

        // pattern 2
        for(int i=start; i<start+5; i++) {

            int diff = seq[i]-seq[i+1];

            if(abs(diff) == 1 && diff == seq[i+1]-seq[i+2]) {

                System.out.println("case 2");
                length = 3;

                if(diff == seq[i+2]-seq[i+3]) length++;
                if(diff == seq[i+3]-seq[i+4]) length++;

                return level[2] + getMin(i+length);
            }
        }

        // pattern 3
        for(int i=start; i<start+5; i++) {

            int diff = seq[i]-seq[i+1];

            if(diff == -(seq

                System.out.println("case 3");
                length = 3;

                if(diff == (seq[i+2]-seq[i+3])) length++;
                if(diff == -(seq[i+3]-seq[i+4])) length++;

                return level[3] + getMin(i+length);
            }
        }

        // pattern 4
        for(int i=start; i<start+5; i++) {

            int diff = seq[i] - seq[i+1];

            if(diff == seq[i+1]-seq[i+2]) {

                System.out.println("case 4");
                length = 3;

                if(diff == seq[i+2]-seq[i+3]) length++;
                if(diff == seq[i+3]-seq[i+4]) length++;

                return level[4] + getMin(i+length);
            }
        }
        // others
        System.out.println("case 5");
        int min = POS_INF;
        for(int i=3; i<5; i++)
            min = min(min, level[5] + getMin(start+i));

        return min;
    }

    private int min(int a, int b) {
```

```
145             if(a<b) return a;
146             else    return b;
147         }
148         private int abs(int num) {
149             if(num>0) return num;
150             else      return -num;
151         }
152
153         public static void main(String[] args) throws IOException {
154
155             if(args.length == 0) {
156                 System.out.println("args fails.");
157                 return;
158             }
159
160             BufferedReader br = new BufferedReader(new FileReader(args[0]));
161
162             int numOfTest;
163             PI pi;
164             String s;
165
166             numOfTest = Integer.parseInt(br.readLine());
167
168             for(int i=0; i<numOfTest; i++) {
169
170                 s = br.readLine();
171                 pi = new PI(s);
172
173                 System.out.println("@@@@@@result " + pi.getMin());
174             }
175             br.close();
176
177         }
178     }
179
```

```
1   package algorithms.apss.dynamic.programming;
2
3   import java.io.BufferedReader;
4   import java.io.FileReader;
5   import java.io.IOException;
6
7   public class Poly {
8
9       /**
10       * @@@ Poly
11       *
12       *  - page
13       *      264p
14       *
15       *  - description
16       *      find the number of vertical monotone polyominos for given n squares.
17       *
18       */
19      private int length;
20      private int[][] cache;
21
22      private int MOD = 10 * 1000 * 1000;
23
24      public Poly(int n) {
25
26          this.length = n;
27
28          cache = new int[n+1][n+1];
29          for(int i=0; i<n+1; i++)
30              for(int j=0; j<n+1; j++)
31                  cache[i][j] = -1;
32      }
33
34      public int getNum() {
35
36          return poly(this.length, 0);
37      }
38
39      /**
40       * @param n : n pieces left
41       * @param prev : length of previous line
42       * @return : the number of vertical monotone polys
43       */
44      public int poly(int n, int prev) {
45
46  //      System.out.println(n + ", " + prev);
47
48          if(cache[n][prev] != -1)
49              return cache[n][prev];
50          if(n == 0)
51              return 1; // return 1 means that one poly has been completed given
                'n==0'.
52
53          int sum = 0;
54          for(int i=1; i<n+1; i++) {
55              if(n == this.length) sum += poly(n-i, i) % MOD;
56              else                 sum += (prev+i-1) * poly(n-i, i) % MOD;
57          }
58
59          return cache[n][prev] = sum;
60      }
61
62
63      public static void main(String[] args) throws NumberFormatException,
        IOException {
64
65
66          if(args.length == 0) {
67              System.out.println("args fails.");
68              return;
69          }
70
```

```
71          BufferedReader br = new BufferedReader(new FileReader(args[0]));
72
73          int numOfTest;
74          Poly poly;
75
76          int n;
77
78          numOfTest = Integer.parseInt(br.readLine());
79
80          for(int i=0; i<numOfTest; i++) {
81
82              n = Integer.valueOf(br.readLine());
83              poly = new Poly(n);
84
85              System.out.println("@@@@ result : " + poly.getNum());
86          }
87          br.close();
88
89      }
90
91  }
92
```

```
1
2   package algorithms.apss.dynamic.programming;
3
4   import java.io.BufferedReader;
5   import java.io.FileReader;
6
7   import java.io.IOException;
8
9   public class Quantization {
10
11      /**
12       * @@@ Quantization
13       *
14       *  - page
15       *      244p
16       *
17       *  - description
18       *      find a minimum StdDev with quantization.
19       *      looks like a clustering...?
20       *
21       *  - approach
22       *
23       *      1. exhaustive search
24       *          select quantized element between min(seq) and max(seq).
25       *          => cannot compute. ~ O(nPs)
26       *
27       *      2. sorting?
28       *
29       */
30      private int[] seq;
31      private int[] quantized;
32      private int[][] cache;
33
34      private final int POS_INF = 987654321;
35      private final int NEG_IN
36
37      public Quantization(int                    int s) {
38
39          String[] seq = sequence.split(" ");
40          this.seq = new int[n];
41          for(int i=0; i<n; i++)
42              this.seq[i] = Integer.valueOf(seq[i]);
43
44          this.quantized = new int[s];
45          for(int i=0; i<s; i++)
46              this.quantized[i] = -1;
47
48          this.cache = new int[n][s];
49          for(int i=0; i<n; i++)
50              for(int j=0; j<s; j++)
51                  this.cache[i][j] = -1;
52
53          sort(this.seq);
54      }
55
56      public int quantize() {
57
58          return quantize(0, 0);
59      }
60
61      private int quantize(int from, int quant) {
62
63  //      System.out.println("quantize " + quant);
64
65          if(cache[from][quant] != -1)
66              return cache[from][quant];
67
68          if(quant == this.quantized.length-1)
69              return cache[from][quant] = minError(this.seq, from,
                this.seq.length-1); // TODO
70
71          int min = POS_INF;
```

```
72          for(int i=from; i<this.seq.length-(this.quantized.length-quant)+1; i++)
73              min = min(min,  minError(this.seq, from, i) + quantize(i+1, quant+1));
74
75          return cache[from][quant] = min;
76      }
77
78      private int minError(int[] arr, int from, int to) {
79
80          int average = average(arr, from ,to);
81          int sum = 0;
82
83          for(int i=from; i<to+1; i++)
84              sum += (arr[i] - average) * (arr[i] - average);
85
86          return sum;
87      }
88
89      private int average(int[] arr, int from, int to) {
90
91          int sum = 0;
92          int avg;
93
94          for(int i=from; i<to+1; i++)
95              sum += arr[i];
96
97          avg = sum / (to - from + 1);
98
99          int half = (to - from + 1) / 2;
100         if(sum % (to - from + 1) > half) return avg + 1;
101         else                             return avg;
102     }
103
104     private int min(int a, int b) {
105         if(a<b) return a;
106         else    return b;
107     }
108
109     // insertion sort
110     private void sort(int[] arr) {
111
112         for(int i=0; i<arr.length; i++) {
113             for(int j=i; j>0 && arr[j] < arr[j-1]; j--) {
114                 int temp = arr[j];
115                 arr[j] = arr[j-1];
116                 arr[j-1] = temp;
117             }
118         }
119     }
120
121     public static void main(String[] args) throws NumberFormatException,
        IOException {
122
123         if(args.length == 0) {
124             System.out.println("args fails.");
125             return;
126         }
127
128         BufferedReader br = new BufferedReader(new FileReader(args[0]));
129
130         int numOfTest;
131         Quantization quant;
132
133         String ns;
134         int n; int s;
135         String sequence;
136
137         numOfTest = Integer.parseInt(br.readLine());
138
139         for(int i=0; i<numOfTest; i++) {
140
141             ns = br.readLine();
142             n = Integer.valueOf(ns.split(" ")[0]);
```

```
143                 s = Integer.valueOf(ns.split(" ")[1]);
144                 sequence = br.readLine();
145                 quant = new Quantization(n, sequence, s);
146 
147                 System.out.println("@@@@ result : " + quant.quantize());
148             }
149             br.close();
150 
151         }
152     }
153 
```

```
1   package algorithms.apss.dynamic.programming;
2
3   public class Tiling {
4
5       /**
6        * @@@ Tiling
7        *
8        *  - page
9        *      252p
10       *
11       *  - description
12       *      find the number of ways to cover 2 x n puzzles with 2 x 1 block.
13       *
14       *  - related class
15       *      AsymTiling
16       */
17      private int length;
18      private int[] cache;
19
20      private final int MOD = 1000 * 1000 * 1000 + 7;
21
22
23      public Tiling(int n) {
24
25          this.length = n;
26
27          cache = new int[n+1];
28          for(int i=0; i<n+1; i++)
29              this.cache[i] = -1;
30      }
31
32      public int tiling(int n) {
33
34          if(cache[n] != -1)
35              return cache[n];
36          if(n <= 1)
37              return 1;
38
39          return cache[n] = (tiling(n-1) + tiling(n-2)) % MOD;
40      }
41
42      public static void main(String[] args) {
43
44      }
45  }
46
```

```
1   package algorithms.apss.dynamic.programming;
2
3   import edu.princeton.cs.algs4.StdIn;
4   import edu.princeton.cs.algs4.StdOut;
5
6   public class TrianglePath {
7
8       /**
9        *
10       * @@@ TrianglePath
11       *
12       *  - page
13       *      226p
14       *
15       *  - description
16       *      this problem is one of the classical dynamic programming example.
17       *      find a path which maximize the sum of numbers on. we can go down or
         down-right side
18       *      at each place.
19       *
20       *  - time complexity
21       *      1. brute-force : O((h-1)*2^(h-1))
22       *      2. implementation : O(h^2)
23       *
24       */
25
26      private int[][] triangle;
27      private int height;
28
29      private int[][] maxSum;
30      private int max;
31
32      public TrianglePath(int height, int[] triangle) {
33
34          this.triangle = new                    t+2];
35          this.height = height,
36          this.maxSum = new int[height+2][height+2];
37          this.max = -1;
38
39          for(int i=0; i<height+2; i++)
40              for(int j=0; j<height+2; j++)
41                  this.triangle[i][j] = -1;
42
43          int index = 0;
44          for(int i=0; i<height; i++)
45              for(int j=0; j<i+1; j++)
46                  this.triangle[i+1][j+1] = triangle[index++];
47
48          for(int i=0; i<height+2; i++)
49              for(int j=0; j<height+2; j++)
50                  this.maxSum[i][j] = 0;
51
52  //      maxPath();  // iteration implementation.
53          this.max = maxPath(1, 1);   // recursively call implementation.
54
55  //      print();
56  //      printMaxSum();
57      }
58
59      /**
60       * top-bottom implementation.
61       * time complexity : T(n) = O(h^2)
62       */
63      private void maxPath() {
64
65          int index = 0;
66          for(int i=0; i<height; i++)
67              for(int j=0; j<i+1; j++) {
68                  if(this.maxSum[i][j+1] >= this.maxSum[i][j])
69                      this.maxSum[i+1][j+1] = this.maxSum[i][j+1] +
                        this.triangle[i+1][j+1];
70                  else
```

```
71                      this.maxSum[i+1][j+1] = this.maxSum[i][j] +
                        this.triangle[i+1][j+1];
72              }
73
74          for(int i=0; i<height;i++)
75              if(max < this.maxSum[height][i+1])
76                  max = this.maxSum[height][i+1];
77      }
78
79      /**
80       * bottom-up implementation. recursively function call.
81       * already solved sub-problems are called in O(1).
82       * time complexity : T(n) = O(h^2)
83       */
84      private int maxPath(int row, int col) {
85
86          if(this.maxSum[row][col] != 0)
87              return this.maxSum[row][col];
88          if(row == height)
89              return this.triangle[row][col];
90
91          return this.maxSum[row][col] = this.triangle[row][col] +
            max(maxPath(row+1, col), maxPath(row+1, col+1));
92      }
93
94      private int max(int a, int b) {
95
96          if(a>=b) return a;
97          else     return b;
98      }
99
100     private void print() {
101
102         StdOut.println("triangle");
103
104         for(int i=0; i<height
105             for(int j=0; j<height; j++)
106                 StdOut.print(this.triangle[i+1][j+1] + "\t");
107             StdOut.println();
108         }
109     }
110
111     private void printMaxSum() {
112
113         StdOut.println("max sum");
114
115         for(int i=0; i<height; i++) {
116             for(int j=0; j<height; j++)
117                 StdOut.print(this.maxSum[i+1][j+1] + "\t");
118             StdOut.println();
119         }
120     }
121
122     public int getMaxSum() {
123
124         return max;
125     }
126
127
128     public static void main(String[] args) {
129
130         int numOfTest;
131         int[] triangle;
132         TrianglePath trianglePath;
133
134         numOfTest = StdIn.readInt();
135
136         for(int i=0; i<numOfTest; i++) {
137
138             int h = StdIn.readInt();
139             int size = (h*(h+1))/2;
140
```

```
141                 triangle = new int[size];
142
143                 for(int j=0; j<size; j++)
144                     triangle[j] = StdIn.readInt();
145
146                 trianglePath = new TrianglePath(h, triangle);
147
148                 StdOut.println(trianglePath.getMaxSum());
149             }
150         }
151
152
153
154     }
155
```

```java
package algorithms.apss.dynamic.programming;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class WildCard {

    /**
     * @@@ WildCard
     *
     *  - page
     *
     *  - description
     *      when parsing, we call a expression like '?', '*' wild-card.
     *      given a expression including wild-card, find a name that fits the pattern.
     *
     *      the problem mainly handles a wild-card '*' that results a unknown length of a name.
     *
     *
     *  - error report
     *      1. cannot handle last cases precisely when pIndex = last or nIndex = last.
     *      1.1. when nIndex = last,
     *      2. recursive implementation works quite well. so we don't have to care
     *          more complicated inputs. all are similar as a subproblem.
     *      3. if * occurs, we should iterate by (n - left chars) -> super exponential.
     *
     *      => needed to optimize.
     */

    private static boolean[][] output = {
        {false, true, true, false, false, false},   // he?p
        {true, true, true, false, true},            // *p*
        {true, true, true, true, false},            // *bb*
        {false, true, false, true, true},           // *a??*svs*zl23*n??
        {true, false, false, false, false, true}    // ***a
    };

    private String sp;
    private char[] pattern;
    private char[] name;

    public WildCard(String pattern) {

        this.sp = pattern;
        this.pattern = pattern.toCharArray();
    }

    public boolean match(String name) {

        System.out.println(sp + ", " + name);

        this.name = name.toCharArray();
        return match(0, 0);
    }


    /**
     *
     * @param pIndex    index for pattern
     * @param nIndex    index for name
     * @return 1 : true, 0 : false
     */
    private boolean match(int pIndex, int nIndex) {
```

```
69
70  //      StdOut.println(pIndex + ", " + nIndex);
71
72          if(pIndex > pattern.length-1) {
73              if(nIndex < name.length)    return false;
74              else                        return true;
75          }
76          if(nIndex > name.length-1) {
77              if(pattern[pIndex] == '*')  return true;
78              if(pIndex < pattern.length) return false;
79              else                        return true;
80          }
81
82          if(pattern[pIndex] == '?') {
83              return match(pIndex+1, nIndex+1);
84          }
85          if(pattern[pIndex] == '*') { // TODO hard.
86
87              if(pIndex == pattern.length-1)
88                  return true;
89
90              for(int i=nIndex; i<name.length; i++)
91                  if(match(pIndex+1, i))
92                      return true;
93
94              return false;
95          }
96
97          // case characters, numbers.
98          if(pattern[pIndex] == name[nIndex])
99              return match(pIndex+1, nIndex+1);
100         else
101             return false;
102     }
103
104
105
106     public static void main(String[] args) throws NumberFormatException,
        IOException {
107
108         if(args.length == 0) {
109             System.out.println("args fails.");
110             return;
111         }
112
113
114         BufferedReader br = new BufferedReader(new FileReader(args[0]));
115
116         int numOfTest;
117         WildCard wildCard;
118
119         numOfTest = Integer.parseInt(br.readLine());
120         boolean isSucceed = true;
121
122         for(int i=0; i<numOfTest; i++) {
123
124             String pattern = br.readLine();
125             int n = Integer.parseInt(br.readLine());
126
127             wildCard = new WildCard(pattern);
128
129             for(int j=0; j<n; j++) {
130                 String s = br.readLine();
131                 if(wildCard.match(s) != output[i][j]) {
132                     isSucceed = false;
133                     StdOut.println("error occurs\n" +
134                                    pattern + ", " + s);
135                 }
136             }
137         }
138
139         br.close();
```

```
140
141             if(isSuccess)
142                 StdOut.println("test succeeded.");
143         }
144
145
146
147     }
148
```

```java
package algorithms.apss.dynamic.programming;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class WildCardDP {

    /**
     * @@@ WildCard
     *
     *  - page
     *
     *  - description
     *      when parsing, we call a expression like '?', '*' wild-card.
     *      given a expression including wild-card, find a name that fits the pattern.
     *
     *      the problem mainly handles a wild-card '*' that results a unknown length of a name.
     *
     *
     */

    private static boolean[][] output = {
        {false, true, true, false, false, false},    // he?p
        {true, true, true, false, true},             // *p*
        {true, true, true, true, false},             // *bb*
        {false, true, false, true, true},            // *a??*svs*z123*n??
        {true, false, false,                    }    // ***a
    };

    private char[] pattern;
    private char[] name;

    private int CACHE_SIZE = 100;
    private int[][] cache;

    public WildCardDP(String pattern) {

        this.pattern = pattern.toCharArray();
        this.cache = new int[CACHE_SIZE][CACHE_SIZE];
    }

    public boolean match(String name) {

        initCache();

        this.name = name.toCharArray();
        return match(0, 0) == 1;
    }

    /**
     *
     * @param pIndex    index for pattern
     * @param nIndex    index for name
     * @return 1 : true, 0 : false
     */
    private int match(int pIndex, int nIndex) {

        if(cache[pIndex][nIndex] != -1)
            return cache[pIndex][nIndex];

        if(pIndex > pattern.length-1) {
            if(nIndex < name.length)    return 0;
            else                        return 1;
        }
        if(nIndex > name.length-1) {
```

```
71              if(pattern[pIndex] == '*')  return 1;
72              if(pIndex < pattern.length) return 0;
73              else                        return 1;
74          }
75
76          if(pattern[pIndex] == '?') {
77              return match(pIndex+1, nIndex+1);
78          }
79          if(pattern[pIndex] == '*') { // TODO hard.
80
81              if(pIndex == pattern.length-1)
82                  return 1;
83
84              for(int i=nIndex; i<name.length; i++)
85                  if(match(pIndex+1, i) == 1)
86                      return 1;
87
88              return cache[pIndex][nIndex] = 0;
89          }
90
91          // case characters, numbers.
92          if(pattern[pIndex] == name[nIndex])
93              return match(pIndex+1, nIndex+1);
94          else
95              return 0;
96      }
97
98      private void initCache() {
99
100         for(int i=0; i<CACHE_SIZE; i++)
101             for(int j=0; j<CACHE_SIZE; j++)
102                 cache[i][j] = -1;
103     }
104
105
106     public static void main(S                    ws NumberFormatException,
        IOException {
107
108         if(args.length == 0) {
109             System.out.println("args fails.");
110             return;
111         }
112
113         BufferedReader br = new BufferedReader(new FileReader(args[0]));
114
115
116         int numOfTest;
117         WildCardDP wildCard;
118
119         numOfTest = Integer.parseInt(br.readLine());
120         boolean isSucceed = true;
121
122
123         for(int i=0; i<numOfTest; i++) {
124
125             String pattern = br.readLine();
126             int n = Integer.parseInt(br.readLine());
127
128             wildCard = new WildCardDP(pattern);
129
130             for(int j=0; j<n; j++) {
131                 String s = br.readLine();
132                 if(wildCard.match(s) != output[i][j]) {
133                     isSucceed = false;
134                     StdOut.println("error occurs\n" +
135                                    pattern + ", " + s);
136                 }
137             }
138         }
139
140         br.close();
141
```

```
142             if(isSucceed)
143                 StdOut.println("test succeeded.");
144         }
145
146
147
148     }
149
```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Random;
import java.util.Vector;

public class BlockGame {

    /**
     * @@@ BlockGame
     *
     *  - page
     *      344p
     *
     *  - description
     *      print a result of block game given board.
     *
     *  - todo
     *      1. is brute-force approach best? O(n^2 * 2^n^2)
     *      2. find how many puzzle states in game.
     */

    private int board;
    /*
     * I-shaped blocks = (n-1) * n * 2.
     * L-shaped blocks = 4 * (n-1).
     */
    private Vector<Integer> I_block;
    private Vector<Integer> L_block;

    private int[] cache;

    public BlockGame(int boa

        this.board = board;

        cache = new int[(1<<25)+1];
        for(int i=0; i<(1<<25); i++)
            cache[i] = -2;

        initBlocks();

//      printBlocks();
    }

    private void initBlocks() {

        I_block = new Vector<Integer>();
        L_block = new Vector<Integer>();

        for(int i=0; i<5; i++) {
            for(int j=0; j<5; j++) {
                if(j != 4) I_block.add(cell(j+1, i) + cell(j, i));
                if(i != 4) I_block.add(cell(j, i) + cell(j, i+1));
            }
        }

        for(int i=0; i<4; i++) {
            for(int j=0; j<4; j++) {
                L_block.add(cell(j, i) + cell(j+1, i) + cell(j, i+1));
                L_block.add(cell(j, i) + cell(j+1, i) + cell(j+1, i+1));
                L_block.add(cell(j+1, i) + cell(j+1, i+1) + cell(j, i+1));
                L_block.add(cell(j, i) + cell(j, i+1) + cell(j+1, i+1));
            }
        }
    }

    public void printBlocks() {
```

```
73          Iterator<Integer> iterator;
74
75          iterator= I_block.iterator();
76          while(iterator.hasNext())
77              print((int)iterator.next());
78
79          iterator= L_block.iterator();
80          while(iterator.hasNext())
81              print((int)iterator.next());
82      }
83
84      /*
85       * 0 1 2 3 4
86       * 5 6 7 8 9
87       * ....
88       */
89      private int cell(int y, int x) {
90
91          return (1<<(y + 5*x));
92      }
93
94
95      public boolean canWin() {
96
97          if(canWin(board) == 1) return true;
98          else                   return false;
99      }
100
101     // win: 1, lose: -1. draw doesn't exist.
102     private int canWin(int board) {
103
104 //      System.out.println(board);
105
106         if(cache[board] != -
107             return cache[boa
108
109         Iterator<Integer> iterator;
110
111         int block;
112
113         iterator = I_block.iterator();
114         while(iterator.hasNext()){
115             block = (int) iterator.next();
116             if((board & block) == 0)
117                 if(canWin(board | block) == -1)
118                     return cache[board] = 1;
119         }
120
121         iterator = L_block.iterator();
122         while(iterator.hasNext()){
123             block = (int) iterator.next();
124             if((board & block) == 0)
125                 if(canWin(board | block) == -1)
126                     return cache[board] = 1;
127         }
128
129         return cache[board] = -1;
130     }
131
132     public void print(int board) {
133
134         for(int i=0; i<5; i++) {
135             for(int j=0; j<5; j++) {
136                 if((board & (1<<(5*i + j))) != 0) System.out.print("# ");
137                 else                              System.out.print(". ");
138             }
139             System.out.println();
140         }
141     }
142
143     public void result() {
144
```

```
145             int count = 0;
146             for(int i=0; i<(1<<25); i++)
147                 if(cache[i] != -2)
148                     count++;
149
150             System.out.println("state count: " + count);
151         }
152
153         public static void main(String[] args) throws NumberFormatException,
            IOException {
154
155             if(args.length == 0) {
156                 System.out.println("args fails.");
157                 return;
158             }
159
160             BufferedReader br = new BufferedReader(new FileReader(args[0]));
161
162             int numOfTest;
163             numOfTest = Integer.parseInt(br.readLine());
164
165             BlockGame game;
166
167             for(int test=0; test<numOfTest; test++) {
168
169                 //System.out.println((test+1) + "-th test");
170
171                 game = new BlockGame(Integer.valueOf(br.readLine()));
172
173                 if(game.canWin()) System.out.println("WINNING");
174                 else              System.out.println("LOSING");
175
176                 game.result();
177             }
178
179             br.close();
180         }
181
182
183
184     }
185
```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Dragon {

    /**
     * @@@ Dragon
     *
     *  - page
     *      306p
     *
     *  - description
     *      print a sequence from p to p+l of n-th generation dragon curve
     sequence.
     */

    /*
     * check list
     *
     * how to cache?
     *
     * to implement efficient cache, we need to efficient merge.
     *
     */

    private final char[] initial = {'+', 'F', 'X'};
    private final String[] parent = {"+FX", "-FX", "+YF", "-YF"};
    private final String[] child = {"+FX+YF", "-FX-YF", "+FX-YF", "-FX+YF"};

    private int n;

    private int p;
    private int l;

    private char[][] cache;

    public Dragon(int n, int p, int l) {

        this.n = n;

        this.p = p;
        this.l = l;

        cache = new char[n+1][3*(int)Math.pow(2, n)];
        for(int i=0; i<n+1; i++) {
            for(int j=0; j<3*(int)Math.pow(2, n); j++) {
                // ? as dummy value.
                cache[i][j] = '?';
            }
        }
    }

    public void curve() {

        // TODO
        // too bad... String concatenation takes total O(n^2) time.
        for(int i=p; i<p+l; i++) {
            System.out.print(curve(n, i));
        }
        System.out.println();
    }

    // TODO time complexity
    private char curve(int n, int index) {

//      System.out.println(n + ", " + index);

        if(n == 0)
            return initial[index];
```

```
72
73          //System.out.println(cache[n][index]);
74  //      System.out.println(cache[0][0]);
75          if(cache[n][index] != '?') {
76
77  //          System.out.println("caching...");
78              return cache[n][index];
79          }
80
81          int mod = index % 6;
82          int l = index - mod;
83          int r = index + (5-mod);
84
85          // TODO best?
86          String prev = "";
87          // Θ(3)
88          for(int i=l; i<r; i+=2)
89              prev += curve(n-1, i/2);
90
91  //      System.out.println("prev: " + prev);
92
93          String now = "";
94          // Θ(4)
95          for(int i=0; i<4; i++)
96              if(prev.equals(parent[i]))
97                  now = child[i];
98
99          // TODO best?
100         return cache[n][index] = now.charAt(index-1);
101     }
102
103     public static void main(String[] args) throws NumberFormatException,
        IOException {
104
105         if(args.length == 0)
106             System.out.println("args failed");
107             return;
108         }
109
110         BufferedReader br = new BufferedReader(new FileReader(args[0]));
111
112         int numOfTest;
113         numOfTest = Integer.parseInt(br.readLine());
114
115         Dragon dragon;
116
117         for(int test=0; test<numOfTest; test++) {
118
119             System.out.println((test+1) + "-th test");
120
121             String[] sArr = br.readLine().split(" ");
122             int n = Integer.parseInt(sArr[0]);
123             int p = Integer.parseInt(sArr[1]);
124             int l = Integer.parseInt(sArr[2]);
125
126             double s = System.currentTimeMillis();
127             dragon = new Dragon(n, p, l);
128
129             dragon.curve();
130             double e = System.currentTimeMillis();
131
132             System.out.println("time: " + (e-s));
133         }
134
135         br.close();
136     }
137 }
138
```

```java
package algorithms.apss.dynamic.programming.advanced;

public class Genius {

    /**
     * @@@ NumberGame
     *
     * - page
     *  359p
     *
     * solved on cloud!
     *
     */
}

```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

public class KLIS {

    /**
     * @@@ KLIS
     *
     *  - page
     *      299p
     *
     *  - description
     *      print k-th lis by dictionary order from given sequence.
     *      it can be assumed that we have more than k lis.
     *
     */

    private int[] sequence;
    private int[] cache;

    private Vector<Integer> vector;

    public KLIS(int[] sequence) {

        this.sequence = sequence;

        cache = new int[sequence.length+1];
        for(int i=0; i<sequence.length+1; i++)
            cache[i] = -1;

        // set lis
        lis(-1);
        cache[0] -= 1;
        cache[sequence.length] = 1;

        vector = new Vector<Integer>();
    }

    /*
     *  1. watch out!
     *      given results look weird, cause input is given so that following
     results are sorted by dict-order.
     *      but we should choose next element by dict-order manually.-> how?
     *
     *      actually, we can prove this.
     *      given next elements which have same depth, these are sorted by
     dict-order.
     *      -> if not sorted, some next elements must have different depth.
     *
     *  2. jump
     *      like problem 'morse', it looks unnecessary to traverse every lis.
     *      we can jump unnecessary paths.
     *      -> time complexity turns out O(n^2*log2n). log2n means nothing but
     binary search(jump).
     */


    public void get(int k) {

        get(-1, cache[0]+1);
    }

    /**
     * time complexity. O(n^2 * depth) = O(n^2*log2n). O(n^2) is
     possible.(advanced)
     *
     * @param n. n-th element included + @
```

```
69          * @param depth. lis length from sequence[n]
70          */
71         private void get(int n, int depth) {
72
73  //          System.out.println("n: " + n + ", depth: " + depth);
74
75             if(depth == 1) {
76                 print();
77                 return;
78             }
79
80             // if there is count[] array, we can call only one recursion.
81             for(int i=n+1; i<sequence.length; i++) {
82                 if((n == -1) || (sequence[n] < sequence[i] && cache[i+1] == depth-1)) {
83                     vector.add(sequence[i]);
84                     get(i, depth-1);
85                     vector.remove(vector.size()-1);
86                 }
87             }
88         }
89
90         private void print() {
91             Iterator<Integer> iterator = vector.iterator();
92             while(iterator.hasNext())
93                 System.out.print(iterator.next() + " ");
94             System.out.println();
95         }
96
97         // time complexity. O(n^2)
98         // return max length of increasing sequence from sequence[start].
99         private int lis(int start) {
100
101            if(start == this.sequence.length)
102                return 0;
103            if(cache[start+1] !=
104                return cache[sta
105
106            int max = 1;
107            for(int i=start+1; i<sequence.length; i++)
108                if(start == -1 || sequence[start] < sequence[i])
109                    max = max(max, 1+lis(i));
110
111            return cache[start+1] = max;
112        }
113
114        private int max(int a, int b) {
115            if(a>b) return a;
116            else    return b;
117        }
118
119        public static void main(String[] args) throws NumberFormatException,
           IOException {
120
121            if(args.length == 0) {
122                System.out.println("args fails.");
123                return;
124            }
125
126            BufferedReader br = new BufferedReader(new FileReader(args[0]));
127
128            int numOfTest;
129            numOfTest = Integer.parseInt(br.readLine());
130
131            KLIS klis;
132
133            int[] iArr;
134            String[] sArr;
135
136            for(int test=0; test<numOfTest; test++) {
137
138                sArr = br.readLine().split(" ");
139                int n = Integer.parseInt(sArr[0]);
```

```
140                 int k = Integer.parseInt(sArr[1]);
141
142                 sArr = br.readLine().split(" ");
143                 iArr = new int[n];
144                 for(int i=0; i<sArr.length; i++)
145                     iArr[i] = Integer.parseInt(sArr[i]);
146
147                 //
148                 klis = new KLIS(iArr);
149                 klis.get(k);
150             }
151
152             br.close();
153         }
154     }
155
```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

import algorithms.apss.dynamic.programming.Binomial;

public class Morse {

    /**
     * @@@ Morse
     *
     *  - page
     *      293p
     *
     *  - description
     *      print k-th morse signals by dict-order.
     *
     */

    private int n;
    private int m;

    private Vector<Character> vector;

    private int skip;

    public Morse(int n, int m) {

        this.n = n;
        this.m = m;

        this.vector = new Vector<Character>();
    }

    public void generate(int k) {

        generate(n, m, k);

        print();
        vector.clear();
    }

    private void generate(int n, int m, int k) {
/*
        if(n == 0 && m == 0) {
            System.out.println("last case");
            return;
        }
        if(n == 0) {
            vec.add('o');
            generate(0, m - 1, k);
            return;
        }
        if(m == 0) {
            vec.add('-');
            generate(n - 1, 0, k);
            return;
        } */

        // actually this condition is enough.
        if(n == 0) {
            for(int i=0; i<m; i++)
                vector.add('o');
            return;
        }

        if(k > combination(n+m-1, n-1)) {
            vector.add('o');
```

```
73              generate(n, m-1, k-combination(n+m-1, n-1));
74          }
75          else {
76              vector.add('-');
77              generate(n-1, m, k);
78          }
79
80      }
81
82      private void print() {
83          Iterator<Character> iterator = vector.iterator();
84          while(iterator.hasNext())
85              System.out.print(iterator.next());
86          System.out.println();
87      }
88
89      private int combination(int n, int r) {
90
91          return Binomial.combination(n, r);
92      }
93
94      public void generate3(int k) {
95
96          this.skip = k-1;
97          generate3(n, m, "");
98      }
99
100     private void generate3(int n, int m, String s) {
101
102         // asynchronous approach
103         // cause skip is global, so if search is finished we don't have to
            traversal anymore.
104         if(skip < 0)
105             return;
106
107         if(n == 0 && m == 0
108             if(skip == 0)
109                 System.out.println(s);
110             skip--;
111             return;
112         }
113         // examine given m, n sequence is needed to search or not.
114         if(combination(n+m, n) <= skip) {
115             skip -= combination(n+m, n);
116             return;
117         }
118         // search
119         if(n>0)
120             generate3(n-1, m, s+'-');
121         if(m>0)
122             generate3(n, m-1, s+'o');
123     }
124
125
126     public void generateAll(int n, int m, String s) {
127
128         if(n == 0 && m == 0) {
129             System.out.println(s);
130             return;
131         }
132
133         if(n>0)
134             generateAll(n-1, m, s + '-');
135         if(m>0)
136             generateAll(n, m-1, s + 'o');
137     }
138
139     public static void main(String[] args) throws NumberFormatException,
        IOException {
140
141         if(args.length == 0) {
142             System.out.println("args fails.");
```

```
143                 return;
144             }
145
146             BufferedReader br = new BufferedReader(new FileReader(args[0]));
147
148             int numOfTest;
149             numOfTest = Integer.parseInt(br.readLine());
150
151             Morse morse;
152
153             String s;
154             String[] sArr;
155
156             Binomial b = new Binomial();
157
158             for(int test=0; test<numOfTest; test++) {
159
160                 sArr = br.readLine().split(" ");
161                 int n = Integer.parseInt(sArr[0]);
162                 int m = Integer.parseInt(sArr[1]);
163
164                 int k = Integer.parseInt(br.readLine());
165
166                 morse = new Morse(n, m);
167
168 //              morse.generateAll(n, m, "");
169
170 //              for(int i=1; i<Binomial.combination(n+m, m)+1; i++)
171 //                  morse.generate(i);
172
173                 morse.generate3(4);
174
175             }
176
177             br.close();
178         }
179 }
180
```

```java
package algorithms.apss.dynamic.programming.advanced;

public class NumberGame {

    /**
     * @@@ NumberGame
     *
     * - page
     *  340p
     *
     * solved on cloud!
     *
     */
}

```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;

public class OCR {

    /**
     * @@@ OCR
     *
     *  - page
     *      285p
     *
     *  - description
     *      find the original which maximize conditional probability.
     */

    // m: the number of words. (1<=m<=500)
    private int m;

    private String[] dictionary;
    private HashMap<String, Integer> index;

    // B: probability that given word occurs at first.
    // T: probability that word[i] -> word[j].
    // M: probability that word[i] is classified as word[j].
    private double[] B;
    private double[][] T;
    private double[][] M;

    private int state;
    private String[] classif

    private double[][] cache
    private int[][] choose;

    public OCR(int m, String[] dictionary, double[] B, double[][] T, double[][]
    M) {

        this.m = m;

        this.dictionary = dictionary;
        this.index = new HashMap<String, Integer>();
        for(int i=0; i<m; i++)
            this.index.put(dictionary[i], i);

        this.B = log(B);
        this.T = log(T);
        this.M = log(M);

        cache = new double[100][m];
        for(int i=0; i<100; i++)
            for(int j=0; j<m; j++)
                cache[i][j] = -1.0;

        choose = new int[100][m];
        for(int i=0; i<100; i++)
            for(int j=0; j<m; j++)
                choose[i][j] = -1;
    }

    private double maxClassify(int state, String[] classified) {

        this.state = state;
        this.classified = classified;

        return maxClassify(0, 0);
    }
    /**
     * @param n: n-th state.
```

```
72          * @param word: index of previous state's word.
73          */
74         private double maxClassify(int n, int word) {
75
76             if(n == state)
77                 return 0;
78
79             int index = -1;
80             double max = -Double.MAX_VALUE;
81
82             double p_oc; // P(O|C)
83             double p_co, p_o, p_c; // P(C|O), P(O), P(C)
84
85             for(int i=0; i<m; i++) {
86
87                 p_co = M[i][index(classified[n])];
88                 p_o = (n == 0)? B[i] : T[word][i];
89                 p_c = (n == 0)? B[index(classified[n])] :
                    T[index(classified[n-1])][index(classified[n])];
90
91                 p_oc = (p_co + p_o - p_c) + maxClassify(n+1, i);
92
93                 if(max < p_oc) {
94                     max = p_oc;
95                     index = i;
96                 }
97             }
98
99             choose[n][word] = index;
100
101            return cache[n][word] = max;
102        }
103
104        public String reconstru
105
106            return reconstruct(
107        }
108        private String reconstruct(int n, int word) {
109
110            if(n == state)
111                return "";
112
113            int c = choose[n][word];
114            String s = dictionary[c];
115
116            return s + " " + reconstruct(n+1, c);
117        }
118
119        public void print() {
120
121            for(int i=0; i<10; i++){
122                for(int j=0; j<m; j++)
123                    System.out.print(choose[i][j] + "\t");
124                System.out.println();
125            }
126        }
127
128        private double[] log(double[] array) {
129
130            for(int i=0; i<array.length; i++)
131                array[i] = Math.log(array[i]);
132
133            return array;
134        }
135        private double[][] log(double[][] matrix) {
136
137            for(int i=0; i<matrix.length; i++)
138                for(int j=0; j<matrix[0].length; j++)
139                    matrix[i][j] = Math.log(matrix[i][j]);
140
141            return matrix;
142        }
```

ouble[][]

```
143
144         private int index(String word) {
145             // hashing.
146             return index.get(word);
147         }
148
149
150         public static void main(String[] args) throws NumberFormatException,
            IOException {
151
152             if(args.length == 0) {
153                 System.out.println("args fails.");
154                 return;
155             }
156
157
158             BufferedReader br = new BufferedReader(new FileReader(args[0]));
159
160             String[] sArr;
161
162             sArr = br.readLine().split(" ");
163
164             int m = Integer.parseInt(sArr[0]);
165             int q = Integer.parseInt(sArr[1]);
166
167             String[] dictionary = br.readLine().split(" ");
168
169             sArr = br.readLine().split(" ");
170
171             double[] B = new double[sArr.length];
172             for(int i=0; i<sArr.length; i++)
173                 B[i] = Double.parseDouble(sArr[i]);
174
175             double[][] T = new d
176             for(int i=0; i<m; i+
177                 sArr = br.readLi
178                 for(int j=0; j<m; j++)
179                     T[i][j] = Double.parseDouble(sArr[j]);
180             }
181
182             double[][] M = new double[m][m];
183             for(int i=0; i<m; i++) {
184                 sArr = br.readLine().split(" ");
185                 for(int j=0; j<m; j++)
186                     M[i][j] = Double.parseDouble(sArr[j]);
187             }
188
189             OCR ocr = new OCR(m, dictionary, B, T, M);
190
191
192             int test = 0;
193             while(test++ < q) {
194
195                 sArr = br.readLine().split(" ");
196
197                 int state = Integer.parseInt(sArr[0]);
198
199                 String[] classified = new String[sArr.length-1];
200                 for(int i=0; i<sArr.length-1; i++)
201                     classified[i] = sArr[i+1];
202
203                 ocr.maxClassify(state, classified);
204                 System.out.println(ocr.reconstruct());
205             }
206
207             br.close();
208         }
209
210     }
211
```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Vector;

public class Packing {

    /**
     * @@@ Packing
     *
     *  - page
     *      281p
     *
     *  - description
     *      get a list which maximizes a need of packing.
     *
     */

    private int capacity;
    private Item[] items;

    private int[][] cache;

    private Vector<String> choices;

    public Packing(int capacity, Item[] items) {

        this.capacity = capacity;
        this.items = items;

        cache = new int[item                    +2];
        for(int i=0; i<items
            for(int j=0; j<c
                cache[i][j] = -1;

        choices = new Vector<String>();
    }

    public int pack() {

        return pack(0, this.capacity);
    }

    public void reconstruct() {

        reconstruct(0, this.capacity);
    }

    // return max need with n-th ~ last items.
    private int pack(int n, int left) {

        if(n == this.items.length || left == 0)
            return 0;
        if(cache[n+1][left+1] != -1)
            return cache[n+1][left+1];

        int max = 0;

        // n-th item jump.
        max = pack(n+1, left);
        // n-th item include.
        if(items[n].volume <= left)
            max = max(max, items[n].need + pack(n+1, left-items[n].volume));

        return cache[n+1][left+1] = max;
    }

    private int max(int a, int b) {
        if(a>b) return a;
```

```
73              else    return b;
74          }
75
76          private void reconstruct(int n, int left) {
77
78              if(n == items.length)
79                  return;
80
81              if(pack(n, left) == pack(n+1, left)) {
82                  reconstruct(n+1, left);
83              }
84              else {
85                  choices.add(items[n].name);
86                  reconstruct(n+1, left-items[n].volume);
87              }
88
89          }
90
91          public void print() {
92
93      //      for(int i=0; i<items.length+1; i++){
94      //          for(int j=0; j<capacity+1; j++)
95      //              System.out.print(cache[i][j] + "\t");
96      //          System.out.println();
97      //      }
98
99              Iterator<String> iter = choices.iterator();
100             while(iter.hasNext())
101                 System.out.println(iter.next());
102             System.out.println();
103         }
104
105         public static void main(String[] args) throws NumberFormatException,
            IOException {
106
107             if(args.length == 0
108                 System.out.println("args fails.");
109                 return;
110             }
111
112             BufferedReader br = new BufferedReader(new FileReader(args[0]));
113
114             int numOfTest;
115             numOfTest = Integer.parseInt(br.readLine());
116
117             Packing p;
118             Item[] items;
119
120             String s;
121             String[] sArr;
122
123
124             for(int test=0; test<numOfTest; test++) {
125
126                 s = br.readLine();
127
128                 int numOfItems = Integer.valueOf(s.split(" ")[0]);
129                 int capacity = Integer.valueOf(s.split(" ")[1]);
130
131                 items = new Item[numOfItems];
132
133                 for(int i=0; i<numOfItems; i++) {
134
135                     sArr = br.readLine().split(" ");
136
137                     String name = sArr[0];
138                     int volume = Integer.valueOf(sArr[1]);
139                     int need = Integer.valueOf(sArr[2]);
140
141                     items[i] = new Item(name, volume, need);
142                 }
143
```

```
144                 p = new Packing(capacity, items);
145
146                 System.out.println("@@@@ result : " + p.pack());
147                 p.reconstruct();
148
149                 p.print();
150
151             }
152             br.close();
153         }
154
155         private static class Item {
156
157             String name;
158             int volume;
159             int need;
160
161             public Item(String name, int volume, int need) {
162
163                 this.name = name;
164                 this.volume = volume;
165                 this.need = need;
166             }
167         }
168
169     }
170
```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Sushi {

    private int budget;
    private int numOfMenus;

    private int[] price;
    private int[] priority;

    private int[] maxPriority;


    public Sushi(int budget, int numOfMenus, int[] price, int[] priority) {

        this.budget = budget/100;
        this.numOfMenus = numOfMenus;

        this.price = price;
        this.priority = priority;

        for(int i=0; i<numOfMenus; i++)
            this.price[i] /= 100;

        // scaled array
        int max = max(budget/100+1, 20000);
        maxPriority = new int[max];
        for(int i=0; i<max; i++)
            maxPriority[i] = -1;

        // initialize
        for(int i=0; i<numOf
            maxPriority[price[i]] = priority[i];
    }

    public int maxPriority() {

        return maxPriority(budget);
    }

    /*
     * TODO
     * implement sliding window. (size: LCM)
     */
    public int maxPriority(int budget) {

        /*
         * time complexity: Θ((budget/100) * numOfMenus).
         * space complexity: ~ Θ((budget/100). too heavy.
         */
        for(int i=0; i<budget; i++) {

            if(maxPriority[i] == -1)
                continue;

            for(int j=0; j<numOfMenus; j++)  {

                if(i+price[j] > budget)
                    continue;

                int prev = maxPriority[i+price[j]];
                int update = maxPriority[i] + priority[j];
                maxPriority[i+price[j]] = max(prev, update);
            }
        }

        // TODO inefficient... but acceptable
        int max = -1;
```

```
        for(int i=0; i<budget+1; i++)
            max = max(max, maxPriority[i]);

        return max;
    }

    private int max(int a, int b) {
        if(a>b) return a;
        else    return b;
    }


    public static void main(String[] args) throws NumberFormatException,
    IOException {

    if(args.length == 0) {
        System.out.println("args fails.");
        return;
    }

    BufferedReader br = new BufferedReader(new FileReader(args[0]));

    int numOfTest;
    numOfTest = Integer.parseInt(br.readLine());

    Sushi sushi;

    String s;
    String[] sArr;

    int[] price;
    int[] priority;

    for(int test=0; test<n

        //System.out.print                    t");

        sArr = br.readLine().split(" ");

        int numOfMenus = Integer.valueOf(sArr[0]);
        int budget = Integer.valueOf(sArr[1]);

        price = new int[numOfMenus];
        priority = new int[numOfMenus];

        for(int i=0; i<numOfMenus; i++) {
            sArr = br.readLine().split(" ");
            price[i] = Integer.valueOf(sArr[0]);
            priority[i] = Integer.valueOf(sArr[1]);
        }

        sushi = new Sushi(budget, numOfMenus, price, priority);
        System.out.println(sushi.maxPriority());
    }

    br.close();
}


}
```

```java
package algorithms.apss.dynamic.programming.advanced;

public class TicTacToe {

    /**
     * @@@ TicTacToe
     *
     * - page
     *  337p
     *
     * solved on cloud!
     *
     */
}

```

```java
package algorithms.apss.dynamic.programming.advanced;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ZimBabwe {

    /**
     * @@@ ZimBabwe
     *
     *  - page
     *      320p
     *
     *  - description
     *      given price price and division number, find the number of likely previous price.
     *
     *  solved on cloud!
     */

}
```

```java
package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdOut;

public class Board {

    private final int[][] initialBoard;
    private int height;
    private int width;

    private int initLine;

    private int numOfWhiteBlocks;

    private boolean isSolved;
    private int numOfSolutions;

    public Board(int h, int w) {

        initialBoard = new int[h+2][w+2];
        for(int i=0; i<h+2; i++) {
            initialBoard[i][0] = 999;
            initialBoard[i][w+1] = 999;
        }
        for(int j=0; j<w+2; j++) {
            initialBoard[0][j] = 999;
            initialBoard[h+1][j] = 999;
        }

        height = h;
        width = w;

        initLine = 0;

        numOfWhiteBlocks = 0;

        isSolved = false;
        numOfSolutions = 0;
    }

    public void initPerLine(String line) throws Exception {

        if(initLine++ >= height)
            throw new Exception();

        char[] charArray = line.toCharArray();

        for(int i=0; i<width; i++) {
            if(charArray[i] == '#') {
                initialBoard[initLine][i+1] = 999;
            }
            else {
                initialBoard[initLine][i+1] = 0;
                numOfWhiteBlocks++;
            }
        }
    }

    public void solve() {

        if(numOfWhiteBlocks%3 == 0 && numOfWhiteBlocks > 0)
            solve(initialBoard, 0);

        isSolved = true;
    }

    public int getNumOfSolutions() {
        if(isSolved)    return numOfSolutions;
        else            return -1;
    }

    private void solve(int[][] board, int n) {
```

```
73
74  //      StdOut.println("solve depth at" + n);
75
76          int[] cover = findCover(board);
77
78          int x = cover[0];
79          int y = cover[1];
80          boolean w1  = (cover[2] == 1);
81          boolean w2  = (cover[3] == 1);
82          boolean w3  = (cover[4] == 1);
83          boolean w4  = (cover[5] == 1);
84
85          if(x == -1 && y == -1) {
86              if(isSolution(board))
87                  numOfSolutions++;
88          }
89          else {
90
91              n++;
92
93              if(w1) {
94
95                  int[][] copy = arrayCopy(board);
96
97                  copy[x][y] = n;
98                  copy[x+1][y] = n;
99                  copy[x][y+1] = n;
100
101                 solve(copy, n);
102
103             }
104             if(w2) {
105
106                 int[][] copy
107
108                 copy[x][y] =
109                 copy[x][y+1]   ...
110                 copy[x+1][y+1] = n;
111
112                 solve(copy, n);
113             }
114             if(w3) {
115
116                 int[][] copy = arrayCopy(board);
117
118                 copy[x][y] = n;
119                 copy[x+1][y-1] = n;
120                 copy[x+1][y] = n;
121
122                 solve(copy, n);
123             }
124             if(w4) {
125
126                 int[][] copy = arrayCopy(board);
127
128                 copy[x][y] = n;
129                 copy[x+1][y] = n;
130                 copy[x+1][y+1] = n;
131
132                 solve(copy, n);
133             }
134         }
135     }
136
137     private int[] findCover(int[][] board) {
138
139         // (x, y), (w1, w2, w3, w4)
140         int[] cover = new int[6];
141         for(int i=0; i<6; i++)
142             cover[i] = -1;
143
144         /**
```

```
         * w1 : W0   w2 : W0      w3: W    w4 : W
         *       0          0          ∞          ∞
         */

        for(int i=0; i<height; i++)
            for(int j=0; j<width; j++)
                if(board[i+1][j+1] == 0) {

                    boolean hasCover = false;

                    if(board[i+2][j+1] == 0 && board[i+1][j+2] == 0) {
                        hasCover = true;
                        cover[2] = 1;
                    }
                    if(board[i+1][j+2] == 0 && board[i+2][j+2] == 0) {
                        hasCover = true;
                        cover[3] = 1;
                    }
                    if(board[i+2][j] == 0 && board[i+2][j+1] == 0) {
                        hasCover = true;
                        cover[4] = 1;
                    }
                    if(board[i+2][j+1] == 0 && board[i+2][j+2] == 0) {
                        hasCover = true;
                        cover[5] = 1;
                    }

                    if(hasCover) {
                        cover[0] = i+1;
                        cover[1] = j+1;

                        return cover;
                    }
                }

        return cover;
    }


    private void printBoard(int[][] board) {

        for(int i=0; i<height+2; i++) {
            for(int j=0; j<width+2; j++)
                StdOut.print(board[i][j] + "\t");
            StdOut.println();
        }
        StdOut.println();
    }

    private int[][] arrayCopy(int [][] copyee)  {

        if(copyee.length == 0 || copyee[0].length == 0) {
            StdOut.println("Wrong copyee array size.");
        }

        int[][] copyer = new int[copyee.length][copyee[0].length];

        for(int i=0; i<copyer.length; i++)
            for(int j=0; j<copyer[0].length; j++)
                copyer[i][j] = copyee[i][j];

        return copyer;
    }

    private boolean isSolution(int[][] board) {

        for(int i=0; i<board.length; i++)
            for(int j=0; j<board[0].length; j++)
                if(board[i][j] == 0)
                    return false;

        return true;
```

```java
package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class BoardCover {

    /**
     *  @@@@ BoradCover
     *
     *  - page
     *      159p
     *
     *  - description
     *      given H x W area board, we should cover the all white lattices with L
     shaped block (3 block size).
     *      print the number of ways to cover all white spaces for given test
     cases.
     *
     *  - related class
     *      Board
     *
     *
  */

    // do not instantiate.
    private BoardCover() {

    }

    public static void main(String[] args) throws Exception {

        int numOfTest;
        Board board;

        numOfTest = StdIn.r

        for(int i=0; i<numOfTest; i++) {

            int h = StdIn.readInt();
            int w = StdIn.readInt();

            StdOut.println();
            StdOut.println("board : " + h + ", " + w);

            board = new Board(h, w);
            for(int j=0; j<h; j++) {
                String s = StdIn.readString();
                board.initPerLine(s);
            }

            board.solve();

            StdOut.println(board.getNumOfSolutions());
        }

    }

}
```

```java
package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdOut;

public abstract class Clocks {

    protected int[] clocks;
    protected int min;

    protected int[][] switches = {
            {0, 1, 2}, {3, 7, 9, 11}, {4, 10, 14, 15}, {0, 4, 5, 6, 7}, {6, 7, 8, 10, 12},
            {0, 2, 14, 15}, {3, 14, 15}, {4, 5, 7, 14, 15}, {1, 2, 3, 4, 5}, {3, 4, 5, 9, 13}
                        };

    public Clocks(int[] init) {

        this.clocks = init;
        this.min = 999; // default, max 30
    }

    abstract void sync();

    public int getMin() {

        return min;
    }

    public void print() {

        for(int i=0; i<clocks.length; i++)
            StdOut.print(clocks[i] + "\t");
        StdOut.println();
    }

    protected void turn(int switchNum/*, int iteration */) {
/*
        if(iteration == 0)
            return;
        if(iteration<0 || iteration>3)
            throw new IndexOutOfBoundsException();
*/
        for(int i=0; i<switches[switchNum].length; i++) {
            int num = switches[switchNum][i];
            if(clocks[num] == 12)   clocks[num] = 3;
            else                    clocks[num] += 3;
        }
    }

    protected boolean isSolved() {

        for(int i=0; i<clocks.length; i++)
            if(clocks[i] != 12)
                return false;
        return true;
    }

}
```

```java
package algorithms.apss.exhaustive.search;


public class ClocksBottomUp extends Clocks {

    public ClocksBottomUp(int[] init) {

        super(init);
    }

    @Override
    public void sync() {

        this.min = sync(-1);
    }

    /**
     * time complexity : T(n) = 4 * T(n-1) = O(4^n)
     * but it is enough computing power to calculate 4^10 operations.
     */
    private int sync(int switchNum) {

        switchNum++;

        if(switchNum > 9) {
            if(isSolved())  return 0;
            else            return 999;
        }

        int min = 999;
        int sync;

        for(int i=0; i<4; i++) {
            sync = sync(swit
            if(min > sync)
                min = sync +
            turn(switchNum);
        }

        return min;
    }
}

package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdOut;

public class ClocksTopDown extends Clocks {

    public ClocksTopDown(int[] init) {

        super(init);
    }

    @Override
    public void sync() {

        sync(-1, 0);
    }

    /**
     * time complexity : T(n) = 4 * T(n-1) = O(4^n)
     * but it is enough computing power to calculate 4^10 operations.
     */
    private void sync(int switchNum, int turn) {

        switchNum++;

        if(switchNum > 9) {
            if(isSolved()) {
                if(this.min > turn)
                    this.min = turn;
```

```
73                  }
74                  return;
75              }
76
77              for(int i=0; i<4; i++) {
78                  sync(switchNum, turn + i);
79                  turn(switchNum);
80              }
81          }
82      }
83
```

```java
package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class ClockSync {

    /**
     * @@@ ClockSync Problem
     *
     *  - page
     *      168p
     *
     *  - description
     *      16 clocks will have to be synced with each other, and there are switches linked to some clocks.
     *      when we push a switch clocks that are linked to it go by 3 hours. Find minimum trial for
     *      synchronizing all given clocks.
     *
     *  - related class
     *      Clocks
     *      ClocksBottomUp
     *      ClocksTopDown
     *
     *
     * TODO.
     * 0. any other implementation for product events, except recursive function call?
     *
     *
     */

    // do not instantiate
    private ClockSync() {

    }

    public static void main(String[] args) {

        int numOfTest;
        Clocks clocks;

        numOfTest = StdIn.readInt();

        for(int i=0; i<numOfTest; i++) {

            int[] init = new int[16];
            for(int j=0; j<16; j++)
                init[j] = StdIn.readInt();

            clocks = new ClocksTopDown(init);
            //clocks = new ClocksBottomUp(init);

            clocks.sync();
            StdOut.println(clocks.getMin());

        }
    }
}
```

```
}


package algorithms.apss.exhaustive.search;

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class Picnic {

    /**
     * @@@ Picnic
     *
     *  - page
     *      155p
     *
     *  - description
     *      find the number of different pairs set.
     *
     *  - related class
     *      null
     */


    private boolean[] students; // 0: not in pair, 1: in pair.
    private int[][] pairs;
    private int pairSet;

    /**
     * @param n      the number of students
     * @param m      the number of pairs
     * @param pairs m*2 array of pairs
     */
    public Picnic(int n, i                    [

        this.students = ne
        this.pairs = pairs;

        for(int i=0; i<n; i++)
            this.students[i] = false;

        this.pairSet = pairSet(0);
    }

    /**
     *
     *  time complexity : T(n) = O(numOfPairSet * (n/2)).
     */
    private int pairSet(int n) {

        if(n == this.students.length) // one pair set made.
            return 1;
        if(isInPair(n))
            return pairSet(n+1);

        int set = 0;
        for(int i=0; i<this.pairs.length; i++) {

            int a = this.pairs[i][0];
            int b = this.pairs[i][1];

            if(a == n || b == n) {
                if(!isInPair(a) && !isInPair(b)) {
                    setPair(a, b, true);
                    set += pairSet(n+1);
                    setPair(a, b, false);
                }
            }
        }

        return set;
    }
```

```
142
143         private boolean isInPair(int n) {
144
145             return this.students[n];
146         }
147
148         /**
149          * @param delta true: set, false: unset.
150          */
151         private void setPair(int i, int j, boolean delta) {
152
153             this.students[i] = delta;
154             this.students[j] = delta;
155         }
156
157         public int getNumOfPairSet() {
158
159             return pairSet;
160         }
161
162         public static void main(String[] args) {
163
164             int numOfTest;
165
166             Picnic picnic;
167             int[][] pairs;
168
169             numOfTest = StdIn.readInt();
170
171             for(int i=0; i<numOfTest; i++) {
172
173                 int n = StdIn.readInt();
174                 int m = StdIn.readInt();
175
176                 pairs = new int[
177                 for(int j=0; j<m
178                     pairs[j][0] = StdIn.readInt();
179                     pairs[j][1] = StdIn.readInt();
180                 }
181
182                 picnic = new Picnic(n, m, pairs);
183
184                 StdOut.println(picnic.getNumOfPairSet());
185             }
186
187
188     }
189 }
190
191
```

```java
package algorithms.apss.greedy.method;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem ActivitySelection
 * @reference APSS 366p.
 *
 * @description
 * find a maximum number of available team reserved.
 */
public class ActivitySelection {

    public ActivitySelection() {

    }

    /**
     * sort, and select a team which ends first.
     */
    public void select() {

    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.print
            return;
        }

        BufferedReader br = new BufferedReader(new FileReader(args[0]));

        int numOfTest;
        numOfTest = Integer.parseInt(br.readLine());

        Object obj;

        String s;
        String[] sArr;

        for(int test=0; test<numOfTest; test++) {

            //System.out.println((test+1) + "-th test");

            obj = new Object();
        }

       br.close();
}

}
```

```java
package algorithms.apss.greedy.method;


/**
 * @author taehyeok.jang
 *
 * @problem LunchBox
 * @reference APSS 376p.
 *
 * @description
 * sub-class of LunchBox
 */

public class Lunch implements Comparable {

    public int heat;
    public int eat;

    public Lunch(int heat, int eat) {

        this.heat = heat;
        this.eat = eat;
    }

    @Override
    public int compareTo(Object obj) {

        if(this.eat < ((Lunch)obj).eat)       return 1;
        else if(this.eat > ((Lunch)obj).eat) return -1;
        else                                  return 0;
    }

}
```

```
144
145
146
147 package algorithms.apss.greedy.method;
148
149 import java.io.FileReader;
150 import java.io.BufferedReader;
151 import java.io.IOException;
152 import java.util.Arrays;
153
154 /**
155  * @author taehyeok.jang
156  *
157  * @problem LunchBox
158  * @reference APSS 376p.
159  *
160  * @description
161  * find a minimum lunch time.
162  */
163 public class LunchBox {
164
165     private Lunch[] lunches;
166
167     public LunchBox(Lunch[] lunches) {
168
169         this.lunches = lunches;
170     }
171
172     // O(nlogn) + O(n) = O(nlogn).
173     public int minPeriod() {
174
175         Arrays.sort(this.lunches);
176
177         int heat = 0;
178         int end = 0;
179
180         for(int i=0; i<lunches.length; i++) {
181
182             heat += lunches[i].heat;
183
184             if(end < heat + lunches[i].eat)
185                 end = heat + lunches[i].eat;
186         }
187
188         return end;
189     }
190
191
192     public static void main(String[] args) throws NumberFormatException,
        IOException {
193
194         if(args.length == 0) {
195             System.out.println("args fails.");
196             return;
197         }
198
199         BufferedReader br = new BufferedReader(new FileReader(args[0]));
200
201         int numOfTest;
202         numOfTest = Integer.parseInt(br.readLine());
203
204         String[] sArr;
205
206         for(int test=0; test<numOfTest; test++) {
207
208             int n = Integer.parseInt(br.readLine());
209
210             int[] heat = new int[n];
211             int[] eat = new int[n];
212
213             sArr = br.readLine().split(" ");
214             for(int i=0; i<n; i++)
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232 }
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
```

```
                heat[i] = Integer.parseInt(sArr[i]);
            sArr = br.readLine().split(" ");
            for(int i=0; i<n; i++)
                eat[i] = Integer.parseInt(sArr[i]);

            Lunch[] lunches = new Lunch[n];
            for(int i=0; i<n; i++)
                lunches[i] = new Lunch(heat[i], eat[i]);

            //
            LunchBox box = new LunchBox(lunches);
            System.out.println(box.minPeriod());
        }

        br.close();
    }
}
```

```
287
288
289
290
291
292
293  package algorithms.apss.greedy.method;
294
295  import java.io.FileReader;
296  import java.io.BufferedReader;
297  import java.io.IOException;
298
299  public class MatchOrder {
300
301      /**
302       * @@@ MatchOrder
303       *
304       *  - page
305       *      371p.
306       *
307       *  - description
308       *      find a maximum number of winnings in coding contest.
309       *
310       */
311
312      public MatchOrder() {
313
314      }
315
316      /**
317       * case win: match a least higher ranked player.
318       * case lose: match a least ranked player.
319       */
320      public void match() {
321
322      }
323
324      public static void main(String[] args) throws NumberFormatException,
         IOException {
325
326          if(args.length == 0) {
327              System.out.println("args fails.");
328              return;
329          }
330
331          BufferedReader br = new BufferedReader(new FileReader(args[0]));
332
333          int numOfTest;
334          numOfTest = Integer.parseInt(br.readLine());
335
336          Object obj;
337
338          String s;
339          String[] sArr;
340
341          for(int test=0; test<numOfTest; test++) {
342
343              //System.out.println((test+1) + "-th test");
344
345              obj = new Object();
346          }
347
348          br.close();
349  }
350
351
352  }
353
354
355
356
357
358
359
360
```

```java
package algorithms.apss.greedy.method;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Arrays;

import util.BasicMath;


/**
 * @author taehyeok.jang
 *
 * @problem MinasTirith
 * @reference apss 388p
 *
 * @description
 * find a minimum guards on minas-tirith circle.
 */
public class MinasTirith extends BasicMath {

    private int n;

    private double[] y;
    private double[] x;
    private double[] r;

    private Range[] ranges;

    public MinasTirith(int n, double[] y, double[] x, double[] r) {

        this.n = n;
        this.y = y;
        this.x = x;
        this.r = r;

        ranges = new Range[n];

        // pre-processing.
        convergeToRange();
        Arrays.sort(ranges);
    }

    private void convergeToRange() {

        for(int i=0; i<n; i++) {

            double loc = fmod(Math.atan2(y[i], x[i]), 2*Math.PI);
            double range  = 2.0 * Math.asin(r[i]/16);

            ranges[i] = new Range(loc-range, loc+range);
        }
    }

    public int solveCircular() {

        int min = Integer.MAX_VALUE;

        for(int i=0; i<n; i++) {

            if(ranges[i].l <=0 || ranges[i].r >= 2*Math.PI) {

                double begin = fmod(ranges[i].r, 2*Math.PI);
                double end = fmod(ranges[i].l, 2*Math.PI);

                min = min(min, 1 + solveLinear(begin, end));
            }
        }

        return min;
    }

```

```
73      Private int solveLinear(double begin, double end) {
74
75          int used = 0;
76          int i = 0;
77
78          while(begin<end) {
79
80              double cover = -1;
81
82              while(i < n && ranges[i].l <= begin) {
83
84                  cover = max(cover, ranges[i].r);
85                  i++;
86              }
87
88              if(cover <= begin)
89                  return Integer.MAX_VALUE;
90
91              begin = cover;
92              used++;
93          }
94
95          return used;
96      }
97
98      private static class Range implements Comparable {
99
100         private double l;
101         private double r;
102
103         public Range(double l, double r) {
104
105             this.l = l;
106             this.r = r;
107         }
108
109         @Override
110         public int compareTo(Object arg0) {
111
112             Range that = (Range)arg0;
113
114             if(this.l > that.l) return 1;
115             else                return -1;
116         }
117     }
118
119
120
121     public static void main(String[] args) throws NumberFormatException,
        IOException {
122
123         if(args.length == 0) {
124             System.out.println("args fails.");
125             return;
126         }
127
128         BufferedReader br = new BufferedReader(new FileReader(args[0]));
129
130         int numOfTest;
131         numOfTest = Integer.parseInt(br.readLine());
132
133         Object obj;
134
135         String s;
136         String[] sArr;
137
138         for(int test=0; test<numOfTest; test++) {
139
140             System.out.println(test+1 + " th test...");
141
142             int n = Integer.parseInt(br.readLine());
143
```

```
144
145             double[] y = new double[n];
146             double[] x = new double[n];
147             double[] r = new double[n];
148
149             for(int i=0; i<n; i++) {
150
151                 sArr = br.readLine().split(" ");
152
153                 y[i] = Double.parseDouble(sArr[0]);
154                 x[i] = Double.parseDouble(sArr[1]);
155                 r[i] = Double.parseDouble(sArr[2]);
156             }
157
158             //
159             MinasTirith mt = new MinasTirith(n, y, x, r);
160             System.out.println(mt.solveCircular());
161         }
162
163         br.close();
164     }
165
166
167 }
168
169 package algorithms.apss.greedy.method;
170
171 import java.io.FileReader;
172 import java.io.BufferedReader;
173 import java.io.IOException;
174
175 import util.PriorityQueue;
176 /**
177  * @author taehyeok.jang
178  *
179  * @problem StrJoin
180  * @reference APSS 380p.
181  *
182  * @description
183  * find a minimum cost of string concatenation.
184  */
185
186 public class StrJoin {
187
188     private PriorityQueue<Integer> pq;
189
190     public StrJoin(int[] L) {
191
192         pq = new PriorityQueue<Integer>();
193         for(int i=0; i<L.length; i++)
194             pq.insert((Integer)L[i]);
195     }
196
197     public void print(int[] arr) {
198
199         for(int i=0; i<arr.length; i++)
200             System.out.print(arr[i] + " ");
201     }
202
203     public int minCost() {
204
205         int cost = 0;
206
207         while(pq.size() > 1) {
208
209             int a = pq.deleteMin();
210             int b = pq.deleteMin();
211
212             cost += (a+b);
213
214             pq.insert(a+b);
215         }
```

```
        return cost;
    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
            return;
        }

        BufferedReader br = new BufferedReader(new FileReader(args[0]));

        int numOfTest;
        numOfTest = Integer.parseInt(br.readLine());

        String[] sArr;

        for(int test=0; test<numOfTest; test++) {

            //System.out.println((test+1) + "-th test");

            int n = Integer.parseInt(br.readLine());
            int[] L = new int[n];

            sArr = br.readLine().split(" ");
            for(int i=0; i<n; i++)
                L[i] = Integer.parseInt(sArr[i]);

            //
            StrJoin sj = new StrJoin(L);
            System.out.println(sj.minCost());
        }

        br.close();
    }

}
```

```java
package algorithms.apss.linear.data.structure;

public class Brackets {

    /**
     * solved using stack.
     */
}


package algorithms.apss.linear.data.structure;

import java.util.LinkedList;
import java.util.Queue;
import java.util.StringTokenizer;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Ites {

    private int K;
    private int N;

    Sequence seq;

    public Ites(int K, int N) {

        this.K = K;
        this.N = N;

        this.seq = new Seque
    }

    public int count() {

        Queue<Integer> queue = new LinkedList<Integer>();

        int count = 0;
        int sum = 0;
        for(int i=0; i<N; i++) {

            int signal = seq.next();
            sum += signal;
            queue.add(signal);

            while(sum > K)
                sum -= queue.remove();

            if(sum == K)
                count++;
        }

        return count;
    }

    private static class Sequence {

        private final long MODULUS = (long)Math.pow(2, 32);
        private final long INITIAL_VALUE = 1983;

        private final long M = 214013;
        private final long P = 2531011;

        private long sequence;

        private int count;

        public Sequence() {
```

```java
    }

    publ
    IOEx

    }

    priv

        }
}

packag

import
import
import
import
```

```
73              this.sequence = (int)INITIAL_VALUE;
74              this.count = 0;
75          }
76
77          public int next() {
78
79              if(count++ > 0)
80                  sequence = (sequence*M+P)%MODULUS;
81
82              return (int)(sequence%10000+1);
83          }
84      }
85
86      public static void main(String[] args) throws NumberFormatException,
        IOException {
87
88          if(args.length == 0) {
89              System.out.println("args fails.");
90              return;
91          }
92
93          FastReader in = new FastReader(args);
94
95          int test = in.nextInt();
96          for(int t=0; t<test; t++) {
97
98              int K = in.nextInt();
99              int N = in.nextInt();
100
101             Ites ites = new Ites(K, N);
102             System.out.println(ites.count());
103         }
104     }
105
106     private static class Fa
107
108         private BufferedReader br;
109         private StringTokenizer st;
110
111         public FastReader(String[] args) throws FileNotFoundException {
112             //      br = new BufferedReader(new InputStreamReader(System.in));
113             br = new BufferedReader(new FileReader(args[0]));
114         }
115         public String next() {
116             while(st == null || !st.hasMoreElements()) {
117                 try                    { st = new
                    StringTokenizer(br.readLine()); }
118                 catch(IOException e)   { e.printStackTrace(); }
119             }
120             return st.nextToken();
121         }
122         public int nextInt() {
123             return Integer.parseInt(next());
124         }
125         public double nextDouble() {
126             return Double.parseDouble(next());
127         }
128         public String nextLine() {
129             String str = "";
130             try                    { str = br.readLine(); }
131             catch(IOException e)   { e.printStackTrace(); }
132             return str;
133         }
134     }
135 }
136
137 package algorithms.apss.linear.data.structure;
138
139 import java.util.Iterator;
140 import java.util.LinkedList;
141 import java.util.List;
142 import java.util.StringTokenizer;
```

```
143
144  import java.io.FileReader;
145  import java.io.BufferedReader;
146  import java.io.FileNotFoundException;
147  import java.io.IOException;
148
149  public class Josephus {
150
151      private int N;
152      private int K;
153
154      private List<Integer> soldiers;
155
156      public Josephus(int N, int K) {
157
158          this.N = N;
159          this.K = K;
160
161          soldiers = new LinkedList<Integer>();
162          for(int i=0; i<N; i++)
163              soldiers.add(i+1);
164      }
165
166      public void survivor() {
167
168          Iterator<Integer> iterator;
169
170          iterator = soldiers.iterator();
171          iterator.next();
172          iterator.remove();
173
174          while(soldiers.size() > 2) {
175
176              for(int i=0; i<K
177
178                  if(!iterator
179                      iterator = soldiers.iterator();
180
181                  iterator.next();
182              }
183
184              iterator.remove();
185          }
186
187          iterator = soldiers.iterator();
188          while(iterator.hasNext())
189              System.out.print(iterator.next() + " ");
190          System.out.println();
191      }
192
193      public static void main(String[] args) throws NumberFormatException,
         IOException {
194
195          if(args.length == 0) {
196              System.out.println("args fails.");
197              return;
198          }
199
200          FastReader in = new FastReader(args);
201
202          int test = in.nextInt();
203          for(int t=0; t<test; t++) {
204
205              int N = in.nextInt();
206              int K = in.nextInt();
207
208              Josephus j = new Josephus(N, K);
209              j.survivor();
210          }
211      }
212
213      private static class FastReader {
```

```
214
215         private BufferedReader br;
216         private StringTokenizer st;
217
218         public FastReader(String[] args) throws FileNotFoundException {
219             //      br = new BufferedReader(new InputStreamReader(System.in));
220             br = new BufferedReader(new FileReader(args[0]));
221         }
222         public String next() {
223             while(st == null || !st.hasMoreElements()) {
224                 try                                    { st = new
                    StringTokenizer(br.readLine()); }
                    catch(IOException e)    { e.printStackTrace(); }
225
226             }
227             return st.nextToken();
228         }
229         public int nextInt() {
230             return Integer.parseInt(next());
231         }
232         public double nextDouble() {
233             return Double.parseDouble(next());
234         }
235         public String nextLine() {
236             String str = "";
237             try                             { str = br.readLine(); }
238             catch(IOException e)    { e.printStackTrace(); }
239             return str;
240         }
241     }
242
243 }
244
245
```

```java
package algorithms.apss.partial.sum;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
import java.util.Vector;

import algorithms.FastReader;

public class Christmas {

    private int N;
    private int K;
    private int[] dolls;

    private int[] pSum; // partial sum

    public Christmas(int N, int K, int[] dolls) {

        this.N = N;
        this.K = K;
        this.dolls = dolls;

        pSum = new int[N];
        pSum[0] = dolls[0];
        for(int i=1; i<N; i++)
            pSum[i] = pSum[i-1] + dolls[i];
    }

    /**
     * time complexity: O(N+K
     */
    public int waysToBuy() {

        List<Vector<Integer>> modulus = new ArrayList<Vector<Integer>>();
        for(int i=0; i<K; i++)
            modulus.add(new Vector<Integer>());

        for(int i=0; i<N; i++) {
            int mod = pSum[i] % K;
            modulus.get(mod).add(i);
        }

        int ways = 0;
        for(int i=0; i<K; i++) {

            Vector<Integer> v = modulus.get(i);

            if(i == 0)
                ways += v.size();

            ways += (v.size()*(v.size()-1))/2;
        }

        return ways;
    }

    /**
     * greedy approach
     * time complexity: O(N+K)
     */
    public int maxBuys() {

        List<Vector<Integer>> modulus = new ArrayList<Vector<Integer>>();
        for(int i=0; i<K; i++)
            modulus.add(new Vector<Integer>());

        int max = 0;
```

```
        int start = -1;
        for(int i=0; i<N; i++) {

            int mod = pSum[i] % K;
            if(mod == 0) {
                max++;
                start = i+1;
            }
            else {

                if(modulus.get(mod).size() !=0 ) {

                    int index = modulus.get(mod).lastElement();
                    if(index >= start) {
                        max++;
                        start = i+1;
                    }
                }

                modulus.get(mod).add(i);
            }
        }

        return max;
    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
            return;
        }

        BufferedReader br =                          w FileReader(args[0]));

        int test = Integer.parseInt(br.readLine());
        String[] sArr;

        for(int t=0; t<test; t++) {

            sArr = br.readLine().split(" ");
            int N = Integer.parseInt(sArr[0]);
            int K = Integer.parseInt(sArr[1]);

            int[] dolls = new int[N];

            sArr = br.readLine().split(" ");
            for(int i=0; i<N; i++)
                dolls[i] = Integer.parseInt(sArr[i]);

            Christmas cm = new Christmas(N, K, dolls);
            System.out.println(cm.waysToBuy() + " " + cm.maxBuys());
        }

        br.close();
    }


}
```

eger>>();

teger>>();

```
1   package algorithms.apss.string.matching;
2
3   import java.util.StringTokenizer;
4
5   import java.io.FileReader;
6   import java.io.BufferedReader;
7   import java.io.FileNotFoundException;
8   import java.io.IOException;
9
10  public class KMP {
11
12      private final int M;
13      private final char[] pattern;
14
15      private final int[] pi;
16
17      private final int N;
18      private final char[] text;
19
20      public KMP(char[] pattern) {
21
22          this.M = pattern.length;
23          this.pattern = pattern;
24
25          this.pi = new int[M+1];
26
27          preprocess();
28
29          this.N = 987654321;
30          this.text = null;
31
32  //      for(int i=0; i<M; i++)
33  //          System.out.print(pattern[i] + "\t");
34  //      System.out.println()
35  //
36  //      for(int i=0; i<M+1;
37  //          System.out.print(pi[i] + "\t");
38  //      System.out.println();
39      }
40
41      public void match() {
42
43          int i = 0;
44          int j = -1;
45
46          while(i < N) {
47              if(j == -1 || text[i] == pattern[j]) {
48                  i++; j++;
49              }
50              else {
51                  j = pi[j];
52              }
53
54              if(j == M) {
55                  System.out.println("matched at " + i);
56                  j = pi[j];
57              }
58          }
59      }
60
61      private void preprocess() {
62
63          int j = 0;
64          int k = -1;
65
66          pi[0] = -1;
67
68          while(j < M) {
69              if(k == -1 || pattern[j] == pattern[k]) {
70                  j++; k++;
71                  pi[j] = k; // update pi only here.
72              }
73
74
75
76          }
77      }
78
79      publi
        IOExc
80
81          f
82
83          i
84
85
86          }
87
88          P
89
90          i
91          f
92
93
94  //
95
96
97
98
99
100
101
102
103
104
105     }
106
107 }
108
```

```
73              else {
74                  k = pi[k];
75              }
76          }
77      }
78
79      public static void main(String[] args) throws NumberFormatException,
        IOException {
80
81          final int a = 1;
82
83          if(args.length == 0) {
84              System.out.println("args fails.");
85              return;
86          }
87
88          FastReader in = new FastReader(args);
89
90          int test = in.nextInt();
91          for(int t=0; t<test; t++) {
92
93
94  //              StringTokenizer ss = new StringTokenizer(null);
95              String stream = "";
96
97              while((stream = in.next()) != null)
98                  System.out.println(stream);
99
100             String s = in.nextLine();
101             char[] pattern = s.toCharArray();
102
103             KMP kmp = new KMP(pattern);
104         }
105     }
106
107 }
108
```

```java
package algorithms.apss.string.matching;

import java.util.Arrays;
import java.util.Comparator;
import java.util.StringTokenizer;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class SuffixArray {

    private String str;
    private int N;
    private char[] s;

    private Integer[] suffixArray;

    public SuffixArray(String str) {

        this.str = str;
        this.N = str.length();
        this.s = str.toCharArray();
    }

    private void initialize() {

        int[] group = new int[N+1];
        for(int i=0; i<N; i++)
            group[i] = s[i]; // hmm...
        group[N] = -1;

        int[] aux = new int[t

        suffixArray = new Int
        for(int i=0; i<N; i++)
            suffixArray[i] = i;

        SuffixComparator comparator;

        int t = 1;
        while(t < N) {

            comparator = new SuffixComparator(group, t);
            Arrays.sort(suffixArray, comparator);

            t *= 2;
            if(t >= N)
                break;

            aux[N] = -1;
            aux[suffixArray[0]] = 0;

            for(int i=1; i<N; i++) {
                if(comparator.compare(suffixArray[i-1], suffixArray[i]) == -1)
                    aux[suffixArray[i]] = aux[suffixArray[i-1]]+1;
                else
                    aux[suffixArray[i]] = aux[suffixArray[i-1]];
            }

            copy(group, aux);
        }
    }

    public Integer[] get() {

        return suffixArray;
    }
    private void copy(int[] copyer, int[] copyee) {

        for(int i=0; i<N; i++)
```

```java

    }

    private

        for

        Syst
    }
    // print
    private

        for

        Syst
    }

    // longe
    // arr[i
    private

        int
        whil

        }
        retu
    }
    private

        priv
        priv

        pub

        }

        @Ov
        pub

//
//
//

        }
    }

    public
    IOExce

        if

        }

        Fa

        in
        fo
```

```
73              copyer[i] = copyee[i];
74      }
75
76      private void print() {
77
78          for(int i=0; i<N; i++)
79              print(s, suffixArray[i]);
80          System.out.println();
81      }
82      // print str from index.
83      private void print(char[] s, int index) {
84
85          for(int i=index; i<N; i++)
86              System.out.print(s[i]);
87          System.out.println();
88      }
89
90      // longest common prefix
91      // arr[i...], arr[j...]
92      private int LCP(char[] arr, int i, int j) {
93
94          int l = 0;
95          while(arr[i] == arr[j] && i < arr.length && j < arr.length) {
96              i++; j++;
97              l++;
98          }
99          return l;
100     }
101     private class SuffixComparator implements Comparator<Integer> {
102
103         private int[] group;
104         private int t;
105
106         public SuffixCompar                    t) {
107
108             this.group = gr
109             this.t = t;
110         }
111
112         @Override
113         public int compare(Integer i, Integer j) {
114
115             if      (group[i] < group[j]) return -1;
116             else if (group[i] > group[j]) return 1;
117             else {
118 //              System.out.println("given " + i + ", " + j);
119 //              print(s, i);
120 //              print(s, j);
121
122                 if(group[i+t] < group[j+t]) return -1;
123                 else                       return 1;
124             }
125         }
126     }
127
-1)  128     public static void main(String[] args) throws NumberFormatException,
         IOException {
129
130         if(args.length == 0) {
131             System.out.println("args fails.");
132             return;
133         }
134
135         FastReader in = new FastReader(args);
136
137         int test = in.nextInt();
138         for(int t=0; t<test; t++) {
139
140             String s = in.nextLine();
141
142             SuffixArray sa = new SuffixArray(s);
143             sa.initialize();
```

```
144
145                 sa.print();
146             }
147         }
148
149         private static class FastReader {
150
151         private BufferedReader br;
152         private StringTokenizer st;
153
154         public FastReader(String[] args) throws FileNotFoundException {
155     //          br = new BufferedReader(new InputStreamReader(System.in));
156                 br = new BufferedReader(new FileReader(args[0]));
157             }
158             public String next() {
159                 while(st == null || !st.hasMoreElements()) {
160                     String str = null;
161                     try                         { str = br.readLine(); if(str == null)
                        return null;
162                     st = new StringTokenizer(str); }
163                     catch(IOException e)        { e.printStackTrace(); }
164                 }
165                 return st.nextToken();
166             }
167             public int nextInt() {
168                 return Integer.parseInt(next());
169             }
170             public double nextDouble() {
171                 return Double.parseDouble(next());
172             }
173             public String nextLine() {
174                 String str = "";
175                 try                         { str = br.readLine(); }
176                 catch(IOException e)        { e.printStackTrace(); }
177                 return str;
178             }
179         }
180     }
181
```

```java
package algorithms.apss.tree;

import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Map.Entry;
import java.util.NoSuchElementException;
import java.util.Queue;
import java.util.StringTokenizer;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class BST<Key extends Comparable<Key>, Value> {

    private Node root;

    private class Node {

        private Key key;           // sorted by key
        private Value val;         // associated data
        private Node left, right;  // left and right subtrees
        private int size;          // number of nodes in subtree

        public Node(Key key, Value val, int size) {
            this.key = key;
            this.val = val;
            this.size = size;
        }
    }

    /**
     * Initializes an empty s
     */
    public BST() {

    }

    public boolean isEmpty() {
        return size() == 0;
    }

    public int size() {
        return size(root);
    }
    private int size(Node x) {
        if(x == null)   return 0;
        else            return x.size;
    }

    public Value get(Key key) {
        return get(root, key);
    }
    private Value get(Node x, Key key) {
        if(key == null) throw new IllegalArgumentException("called get() with a
        null key");
        if(x == null)   return null;
        int cmp = key.compareTo(x.key);
        if      (cmp < 0)   return get(x.left, key);
        else if (cmp > 0)   return get(x.right, key);
        else                return x.val;
    }

    public boolean contains(Key key) {
        if (key == null) throw new IllegalArgumentException("argument to
        contains() is null");
        return get(key) != null;
    }

    public void put(Key key, Value val) {
```

```
71          if(key == null) throw new IllegalArgumentException("called put() with a
            null key");
72          if(val == null) {
73              delete(key);
74              return;
75          }
76          root = put(root, key, val);
77          assert check();
78      }
79      private Node put(Node x, Key key, Value val) {
80          if(x == null) return new Node(key, val, 1);
81          int cmp = key.compareTo(x.key);
82          if      (cmp < 0)   x.left = put(x.left, key, val);
83          else if (cmp > 0)   x.right = put(x.right, key, val);
84          else                x.val = val;
85          x.size = 1 + size(x.left) + size(x.right);
86          return x;
87      }
88
89      public void deleteMin() {
90          if(isEmpty()) throw new NoSuchElementException("symbol table underflow");
91          root = deleteMin(root);
92          assert check();
93      }
94      private Node deleteMin(Node x) {
95          if(x.left == null) return x.right;
96          x.left = deleteMin(x.left);
97          x.size = 1 + size(x.left) + size(x.right);
98          return x;
99      }
100
101     public void deleteMax() {
102         if(isEmpty()) throw new NoSuchElementException("symbol table underflow");
103         root = deleteMax(r
104         assert check();
105     }
106     private Node deleteMax(Node x) {
107         if(x.right == null) return x.left;
108         x.right = deleteMax(x.right);
109         x.size = 1 + size(x.left) + size(x.right);
110         return x;
111     }
112
113     public void delete(Key key) {
114         if(key == null) throw new IllegalArgumentException("called delete() with
            a null key");
115         root = delete(root, key);
116 //      assert check();
117     }
118     private Node delete(Node x, Key key) {
119         if(x == null) return null;
120         int cmp = key.compareTo(x.key);
121         if      (cmp < 0) x.left = delete(x.left, key);
122         else if (cmp > 0) x.right = delete(x.right, key);
123         else {
124             if(x.right == null) return x.left;
125             if(x.left == null)  return x.right;
126             Node t = x;
127             x = min(t.right);
128             x.right = deleteMin(t.right);
129             x.left = t.left;
130         }
131         x.size = 1 + size(x.left) + size(x.right);
132         return x;
133     }
134
135     public Key min() {
136         if(isEmpty()) throw new NoSuchElementException("called min() with empty
            symbol table");
137         return min(root).key;
138     }
139     private Node min(Node x) {
```

```
    if(x.left == null)  return x;
    else                return min(x.left);
}
public Key max() {
    if(isEmpty()) throw new NoSuchElementException("called max() with empty
    symbol table");
    return max(root).key;
}
private Node max(Node x) {
    if(x.right == null) return x;
    else                return max(x.right);
}

public int height() {
    return height(root);
}
private int height(Node x) {
    if(x == null) return -1;
    return 1 + Math.max(height(x.left), height(x.right));
}

public Key floor(Key key) {
    if(key == null) throw new IllegalArgumentException("argument to floor()
    is null");
    if(isEmpty())   throw new NoSuchElementException("called floor() with
    empty symbol table");
    Node x = floor(root, key);
    if(x == null)   return null;
    else            return x.key;
}
private Node floor(Node x, Key key) {
    if(x == null) return null;
    int cmp = key.compareTo(x.key);
    if  (cmp == 0)  retur
    if  (cmp < 0)   retur                    ;
    Node t = floor(x.righ
    if(t != null)   return t;
    else            return x;
}

public Key ceiling(Key key) {
    if(key == null) throw new IllegalArgumentException("argument to floor()
    is null");
    if(isEmpty())   throw new NoSuchElementException("called floor() with
    empty symbol table");
    Node x = ceiling(root, key);
    if(x == null)   return null;
    else            return x.key;
}
private Node ceiling(Node x, Key key) {
    if(x == null) return null;
    int cmp = key.compareTo(x.key);
    if  (cmp == 0)  return x;
    if  (cmp > 0)   return ceiling(x.right, key);
    Node t = ceiling(x.left, key);
    if(t != null)   return t;
    else            return x;
}

public int rank(Key key) {
    if(key == null) throw new IllegalArgumentException("argument to rank() is
    null");
    return rank(root, key);
}
private int rank(Node x, Key key) {
    if(x == null) return 0;
    int cmp = key.compareTo(x.key);
    if      (cmp < 0)   return rank(x.left, key);
    else if (cmp > 0)   return 1 + size(x.left) + rank(x.right, key);
    else                return size(x.left);
}
```

```
206     // select k-th ranked node.
207     public Key select(int k) {
208         if(k < 0 || k>= size())
209             throw new IllegalArgumentException("called select() with invalid
                argument");
210         Node x = select(root, k);
211         return x.key;
212     }
213     private Node select(Node x, int k) {
214         if(x == null) return null;
215         int t = size(x.left);
216         if      (t > k) return select(x.left, k);
217         else if (t < k) return select(x.right, k-t-1);
218         else            return x;
219     }
220
221     public Iterable<Key> keys() {
222         return keys(min(), max());
223     }
224     public Iterable<Key> keys(Key lo, Key hi) {
225         if(lo == null) throw new IllegalArgumentException("first argument to
                keys() is null");
226         if(hi == null) throw new IllegalArgumentException("second argument to
                keys() is null");
227         Queue<Key> queue = new LinkedList<Key>();
228         keys(root, queue, lo, hi);
229         return queue;
230     }
231     private void keys(Node x, Queue<Key> queue, Key lo, Key hi) {
232         if(x == null) return;
233         int cmplo = lo.compareTo(x.key);
234         int cmphi = hi.compareTo(x.key);
235         if(cmplo < 0) keys(
236         if(cmplo <= 0 && cm                    .key);
237         if(cmphi > 0) keys(                    i);
238     }
239
240     /**
241      * both bounds are inclusive.
242      */
243     public int size(Key lo, Key hi) {
244         if(lo == null) throw new IllegalArgumentException("first argument to
                size() is null");
245         if(hi == null) throw new IllegalArgumentException("second argument to
                size() is null");
246         if(lo.compareTo(hi) > 0) return 0;
247         if(contains(hi))     return rank(hi) - rank(lo) + 1;
248         else                 return rank(hi) - rank(lo);
249     }
250
251   /*************************************************************************
252     *  Check integrity of BST data structure.
253     *************************************************************************/
254     private boolean check() {
255         if (!isBST())            System.out.println("Not in symmetric order");
256         if (!isSizeConsistent()) System.out.println("Subtree counts not
                consistent");
257         if (!isRankConsistent()) System.out.println("Ranks not consistent");
258         return isBST() && isSizeConsistent() && isRankConsistent();
259     }
260
261     // does this binary tree satisfy symmetric order?
262     // Note: this test also ensures that data structure is a binary tree since
        order is strict
263     private boolean isBST() {
264         return isBST(root, null, null);
265     }
266
267     // is the tree rooted at x a BST with all keys strictly between min and max
268     // (if min or max is null, treat as empty constraint)
269     // Credit: Bob Dondero's elegant solution
270     private boolean isBST(Node x, Key min, Key max) {
```

```
271             if (x == null) return true;
272             if (min != null && x.key.compareTo(min) <= 0) return false;
273             if (max != null && x.key.compareTo(max) >= 0) return false;
274             return isBST(x.left, min, x.key) && isBST(x.right, x.key, max);
275         }
276
277         // are the size fields correct?
278         private boolean isSizeConsistent() { return isSizeConsistent(root); }
279         private boolean isSizeConsistent(Node x) {
280             if (x == null) return true;
281             if (x.size != size(x.left) + size(x.right) + 1) return false;
282             return isSizeConsistent(x.left) && isSizeConsistent(x.right);
283         }
284
285         // check that ranks are consistent
286         private boolean isRankConsistent() {
287             for (int i = 0; i < size(); i++)
288                 if (i != rank(select(i))) return false;
289             for (Key key : keys())
290                 if (key.compareTo(select(rank(key))) != 0) return false;
291             return true;
292         }
293
294
295         public static void main(String[] args) throws NumberFormatException,
         IOException {
296
297             Map<Integer, Integer> map = new HashMap<Integer, Integer>();
298             map.put(4, 8);
299             map.put(2, 4);
300             map.put(10, 2);
301             map.put(1, 6);
302
303             Iterator<Entry<Integ                          ap.entrySet().iterator();
304             while(it1.hasNext())
305                 System.out.print
306
307             Iterator<Integer> it2 = map.keySet().iterator();
308             while(it2.hasNext())
309                 System.out.println(map.get(it2.next()));
310
311             if(args.length == 0) {
312                 System.out.println("args fails.");
313                 return;
314             }
315
316             FastReader in = new FastReader(args);
317
318             int test = in.nextInt();
319             for(int t=0; t<test; t++) {
320
321             }
322         }
323
324     }
325
```

```java
package algorithms.apss.tree;

import java.util.StringTokenizer;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem EditorWars
 * @reference apss - 772p
 *
 * @description
 *
 */
public class EditorWars {

    private BipartiteUnionFind buf;

    private int n, m;
    private char[] comment;
    private int[] a, b;

    public EditorWars(int n, int m, char[] comment, int[] a, int[] b) {

        this.n = n;
        this.m = m;
        this.comment = comment;
        this.a = a;
        this.b = b;

        this.buf = new Bipar
    }

    public void solve() {

        for(int i=0; i<m; i++) {

            if(comment[i] == 'a') {
                if(!buf.ack(a[i], b[i])) {
                    System.out.println("CONTRADICTION AT " + (i+1));
                    return;
                }
            }
            else {
                if(!buf.dis(a[i], b[i])) {
                    System.out.println("CONTRADICTION AT " + (i+1));
                    return;
                }
            }
        }

        int max = 0;
        for(int i=0; i<n; i++) {

            if(buf.root(i) == i) {

                int j = buf.enemy[i];
                if(j > i) // prevent duplicate.
                    continue;

                int ally = buf.size[i];
                int enemy = (j==-1)? -1:buf.size[j];

                max += max(ally, enemy);
            }
        }

        System.out.println("MAX PARTY SIZE IS " + max);
    }

    }

    private

        retu
    }

    private

        pri

        pub

        )

        pu

        }

        pr
```

```
73      }
74
75      private int max(int a, int b) {
76
77          return a>b? a:b;
78      }
79
80      private class BipartiteUnionFind extends UnionFind{
81
82          private int[] enemy;
83
84          public BipartiteUnionFind(int n) {
85
86              super(n);
87
88              enemy = new int[n];
89              for(int i=0; i<n; i++)  enemy[i] = -1;
90          }
91
92          public boolean ack(int p, int q) {
93
94              int i = root(p);
95              int j = root(q);
96
97              if(enemy[i] == j || enemy[j] == i)
98                  return false;
99
100             int a = union(i, j);
101             int b = union(enemy[i], enemy[j]);
102             enemy[a] = b;
103             if(b != -1)
104                 enemy[b] = a;
105
106             return true;
107         }
108
109         public boolean dis(int p, int q) {
110
111             int i = root(p);
112             int j = root(q);
113
114             if(i == j)
115                 return false;
116
117             int a = union(i, enemy[j]);
118             int b = union(j, enemy[i]);
119             enemy[a] = b;
120             enemy[b] = a;
121
122             return true;
123         }
124     }
125
126     private class UnionFind {
127
128         protected int[] id;
129         protected int[] size;
130
131         public UnionFind(int n) {
132
133             id = new int[n];
134             size = new int[n];
135
136             for(int i=0; i<n; i++)  id[i] = i;
137             for(int i=0; i<n; i++)  size[i] = 1;
138         }
139
140         protected int root(int i) {
141
142             while(i!=id[i])
143                 i = id[i];
144             return i;
```

```
145         }
146
147
148         public int union(int p, int q) {
149
150             if(p == -1) return q;
151             if(q == -1) return p;
152
153             int i = root(p);
154             int j = root(q);
155
156             if(i == j)
157                 return i;
158
159             if(size[i]<size[j]) {
160                 id[i] = j;
161                 size[j] += size[i];
162                 return j;
163             }
164             else {
165                 id[j] = i;
166                 size[i] += size[j];
167                 return i;
168             }
169         }
170
171         public boolean connected(int p, int q) {
172
173             return root(p) == root(q);
174         }
175     }
176
    public static void main(String[] args) throws NumberFormatException,
    IOException {
177
178         if(args.length == 0)
179             System.out.print
180             return;
181         }
182
183         FastReader in = new FastReader(args);
184
185         int test = in.nextInt();
186         for(int t=0; t<test; t++) {
187
188             int n = in.nextInt();
189             int m = in.nextInt();
190
191             char[] comment = new char[m];
192             int[] a = new int[m];
193             int[] b = new int[m];
194             for(int i=0; i<m; i++) {
195                 if(in.next().equals("ACK")) comment[i] = 'a';
196                 else                        comment[i] = 'd';
197                 a[i] = in.nextInt();
198                 b[i] = in.nextInt();
199             }
200
201             EditorWars ew = new EditorWars(n, m, comment, a, b);
202             ew.solve();
203         }
204     }
205
206 }
207
```

```java
package algorithms.apss.tree;

import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.Vector;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem FamilyTree
 * @reference apss - 747p
 *
 * @description
 *
 */
public class FamilyTree {

    private int n;
    private Node[] member;

    // save serial number of each node, not member number.
    private int[] traversal;
    private static int INDEX = 0;
    private static int SERIAL = 0;

    // mapping array
    private int[] serial2member;
    private int[] member2ind

    private RMQ rmq;

    public FamilyTree(int n, int[] parent) {

        this.n = n;
        this.member = new Node[n];
        for(int i=0; i<n; i++)
            this.member[i] = new Node(i);
        for(int i=0; i<n-1; i++)
            member[parent[i]].children.add(i+1);

        serial2member = new int[n];
        member2index = new int[n];
        traversal = new int[2*n-1];
        initialize();

        rmq = new RMQ(traversal, 1);
    }

    private void initialize() {

        traverse(0, 0);
    }

    private void traverse(int k, int d) {

        int s = SERIAL++;

        member[k].serial = s;
        serial2member[s] = k;
        member[k].depth = d;

        member2index[k] = INDEX;

        Iterator<Integer> it = member[k].children.iterator();
        while(it.hasNext()) {

            traversal[INDEX++] = s;
```

```
73              traverse(it.next(), d+1);
74          }
75
76          traversal[INDEX++] = s;
77      }
78
79      public int distance(int a, int b) {
80
81          // return serial number of lca
82          int lca;
83          if(a<b) lca = rmq.query(member2index[a], member2index[b]);
84          else    lca = rmq.query(member2index[b], member2index[a]);
85
86          return member[a].depth + member[b].depth -
            2*member[serial2member[lca]].depth;
87      }
88
89      private class Node {
90
91          private int key;
92          private int serial;
93          private int depth;
94          private Vector<Integer> children;
95
96          public Node(int key) {
97
98              this.key = key;
99              this.children = new Vector<Integer>();
100         }
101     }
102
103     public static void main(String[] args) throws NumberFormatException,
        IOException {
104
105         if(args.length ==
106             System.out.pri
107             return;
108         }
109
110         FastReader in = new FastReader(args);
111
112         int test = in.nextInt();
113         for(int t=0; t<test; t++) {
114
115             int n = in.nextInt();
116             int q = in.nextInt();
117
118             int[] parent = new int[n-1];
119             for(int i=0; i<n-1; i++)
120                 parent[i] = in.nextInt();
121
122             FamilyTree ft = new FamilyTree(n, parent);
123             for(int i=0; i<q; i++)
124                 System.out.println(ft.distance(in.nextInt(),
                    in.nextInt()));
125         }
126     }
127
128 }
129
```

```java
package algorithms.apss.tree;

public class FenwickTree {

    private int size;
    private int[] tree;

    public FenwickTree(int size) {

        this.size = size;
        tree = new int[size+1];
        for(int i=0; i<size+1; i++)
            tree[i] = 0;
    }

    /**
     * return sum of [0...i].
     */
    public int sum(int i) {

        i++;
        int sum = 0;
        while(i > 0) {
            sum += tree[i];
            i &= (i-1);
        }
        return sum;
    }

    /**
     * update FenWick tree's node value recursively.
     */
    public void add(int i, int value) {

        i++;
        while(i < size+1) {
            tree[i] += value;
            i += (i&-i);
        }
    }

}
```

```
73  package algorithms.apss.tree;
74
75  import java.util.Arrays;
76  import java.util.Collections;
77  import java.util.Iterator;
78  import java.util.StringTokenizer;
79  import java.util.Vector;
80
81  import java.io.FileReader;
82  import java.io.BufferedReader;
83  import java.io.FileNotFoundException;
84  import java.io.IOException;
85
86  /**
87   *
88   * @author taehyeok.jang
89   *
90   * @problem Fortress
91   * @reference apss - 689p
92   *
93   * @description
94   *
95   */
96  public class Fortress {
97
98      private int n;
99      private Guard[] guards;
100
101     private Guard root;
102
103     public Fortress(int n, int[] x, int[] y, int[] r) {
104
105         this.n = n;
106         this.guards = new Gu
107         for(int i=0; i<n; i
108             this.guards[i] =                    , r[i]);
109
110         root = this.guards[0];
111         Arrays.sort(this.guards);
112
113         parseTree();
114     }
115
116     private void parseTree() {
117         // for sorted node i, find its parent.
118         for(int child=0; child<n; child++) {
119             for(int p=child+1; p<n; p++) {
120                 if(guards[child].isChildOf(guards[p])) {
121                     guards[p].addChild(guards[child]);
122                     break;
123                 }
124             }
125         }
126     }
127
128     /**
129      * call each tree's node once, entire time complexity is O(n).
130      */
131     public int maxPath() {
132
133         Vector<Integer> heights = new Vector<Integer>();
134
135         int h = 0;
136         Iterator<Guard> it = root.children.iterator();
137         while(it.hasNext()) {
138
139             int subtree = 1+height(it.next());
140             h = max(h, subtree);
141             heights.add(subtree);
142         }
143
144         if(heights.size()<2) {
```

```
            return h;
        }
        else {
            Collections.sort(heights);
            return heights.get(heights.size()-1)+heights.get(heights.size()-2);
        }
    }

    private int height(Guard g) {

        int h = 0;
        Iterator<Guard> it = g.children.iterator();
        while(it.hasNext())
            h = max(h, 1+height(it.next()));

        return h;
    }

    private int max(int a, int b) {

        return a>b? a:b;
    };

    /*
    private void traverse(Guard g) {

        // pre-order
        if(g == null)
            return ;

        System.out.println(g.r);

        Iterator<Guard> it = g.children.iterator();
        while(it.hasNext())
            traverse(it.next
    }
    */

    private class Guard implements Comparable<Guard> {

        private int x, y;
        private int r;

        private Vector<Guard> children;

        public Guard(int x, int y, int r) {
            this.x = x;
            this.y = y;
            this.r = r;
            children = new Vector<Guard>();
        }
        @Override
        public int compareTo(Guard that) {
            if(this.r >= that.r)     return 1;
            else                     return -1;
        }
        public boolean isChildOf(Guard parent) {
            if(parent.r <= this.r)
                return false;
            int dist = (parent.x-this.x)*(parent.x-this.x) +
            (parent.y-this.y)*(parent.y-this.y);
            return dist < (parent.r-this.r)*(parent.r-this.r);
        }
        public void addChild(Guard c) {
            children.add(c);
        }
    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
```

```
            System.out.println("args fails.");
            return;
        }

        FastReader in = new FastReader(args);

        int test = in.nextInt();
        for(int t=0; t<test; t++) {

            int n = in.nextInt();
            int[] x = new int[n];
            int[] y = new int[n];
            int[] r = new int[n];

            for(int i=0; i<n; i++) {
                x[i] = in.nextInt();
                y[i] = in.nextInt();
                r[i] = in.nextInt();
            }

            Fortress f = new Fortress(n, x, y, r);
            System.out.println(f.maxPath());
        }
    }

}
```

```java
package algorithms.apss.tree;

import java.util.StringTokenizer;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem MeasureTime
 * @reference apss - 758p
 *
 * @description
 *
 */
public class MeasureTime {

    private int n;
    private int[] array;

    private static final int MAX = 1000000;

    private int[] aux;

    public MeasureTime(int n, int[] array) {

        this.n = n;
        this.array = array;

        this.aux = new int[n]
    }

    /**
     *  a solution using FenWick tree. (cumulative approach)
     */
/*  public int solve() {

        FenwickTree ft = new FenwickTree(MAX);
        int measure = 0;
        for(int i=0; i<n; i++) {

            measure += ft.sum(MAX-1)-ft.sum(array[i]);
            ft.add(array[i], 1);
        }

        return measure;
    }*/

    public int solve() {

        return solve(array, 0, n-1);
    }

    /**
     * check the number of inversion while merging in merge sort.
     */
    public int solve(int[] array, int left, int right) {

        if(left == right)
            return 0;

        int mid = (left+right)/2;
        int inversion = solve(array, left, mid)+solve(array, mid+1, right);

        int temp = left;
        int l = left;
        int r = mid+1;
        while(l<=mid || r<=right) {
```

```
}
```

```
73
74                  if(l<=mid && (r>right || array[l]<=array[r])) {
75                      aux[temp++] = array[l++];
76                  }
77                  else {
78                      inversion += mid-l+1;
79                      aux[temp++] = array[r++];
80                  }
81              }
82
83              for(int i=left; i<right+1; i++)
84                  array[i] = aux[i];
85
86              return inversion;
87          }
88
89          public static void main(String[] args) throws NumberFormatException,
            IOException {
90
91              if(args.length == 0) {
92                  System.out.println("args fails.");
93                  return;
94              }
95
96              FastReader in = new FastReader(args);
97
98              int test = in.nextInt();
99              for(int t=0; t<test; t++) {
100
101                 int n = in.nextInt();
102                 int[] array = new int[n];
103                 for(int i=0; i<n; i++)
104                     array[i] = in.nextInt();
105
106                 MeasureTime mt                          array);
107                 System.out.prin
108             }
109         }
110
111 }
112
```

```
, right);
```

```java
package algorithms.apss.tree;

import java.util.StringTokenizer;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem Mordor
 * @reference apss - 746p
 *
 * @description
 *
 */
public class Mordor {

    private int q;
    private int[][] index;

    private RMQ rMinQ, rMaxQ;

    public Mordor(int n, int q, int[] height, int[][] index) {

        this.q = q;
        this.index = index;

        rMinQ = new RMQ(height, 1);
        rMaxQ = new RMQ(height, -1);
    }

    public void solve() {

        for(int i=0; i<q; i++) {

            int min = rMinQ.query(index[i][0], index[i][1]);
            int max = -rMaxQ.query(index[i][0], index[i][1]);

            System.out.println(max-min);
        }
    }

    public static void main(String[] args) throws NumberFormatException,
    IOException {

        if(args.length == 0) {
            System.out.println("args fails.");
            return;
        }

        FastReader in = new FastReader(args);

        int test = in.nextInt();
        for(int t=0; t<test; t++) {

            int n = in.nextInt();
            int q = in.nextInt();

            int[] height = new int[n];
            for(int i=0; i<n; i++)
                height[i] = in.nextInt();

            int[][] index = new int[q][2];
            for(int i=0; i<q; i++)
                for(int j=0; j<2; j++)
                    index[i][j] = in.nextInt();

            Mordor mordor = new Mordor(n, q, height, index);
```

```
72                  mordor.solve();
73              }
74          }
75
76      }
77
```

{

Exception,

```java
package algorithms.apss.tree;

import java.util.Iterator;
import java.util.StringTokenizer;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem Nerd
 * @reference apss - 702p
 *
 * @description
 *
 */
public class Nerd {

    private BST<Integer, Integer> bst;  // x, y
    private int n;
    private int[][] data;

    public Nerd(int n, int[][] data) {

        bst = new BST<Integer, Integer>();
        this.n = n;
        this.data = data;
    }

    public int solve() {

        int r = 0;
        for(int i=0; i<n; i
            r += register(d
        return r;
    }

    private int register(int x, int y) {

        if(isDominated(x, y))
            return bst.size();

        removeDominated(x, y);
        bst.put(x, y);

        return bst.size();
    }

    private boolean isDominated(int x, int y) {

        if(bst.size() == 0)
            return false;

        Integer cx = bst.ceiling(x);
        if(cx == null)
            return false;

        return bst.get(cx) >= y;
    }

    private void removeDominated(int x, int y) {

        if(bst.size() == 0)
            return;
        if(bst.floor(x) == null)
            return;

        Iterator<Integer> it = bst.keys(0, x).iterator();
}
```

```
73              while(it.hasNext()) {
74                  int cx = it.next();
75                  if(bst.get(cx) <= y)
76                      bst.delete(cx);
77              }
78          }
79
80          public static void main(String[] args) throws NumberFormatException,
            IOException {
81
82              if(args.length == 0) {
83                  System.out.println("args fails.");
84                  return;
85              }
86
87              FastReader in = new FastReader(args);
88
89              int test = in.nextInt();
90              for(int t=0; t<test; t++) {
91
92                  int n = in.nextInt();
93                  int[][] data = new int[n][2];
94                  for(int i=0; i<n; i++)
95                      for(int j=0; j<2; j++)
96                          data[i][j] = in.nextInt();
97
98                  Nerd nerd = new Nerd(n, data);
99                  System.out.println(nerd.solve());
100             }
101         }
102
103 }
104
```

```java
package algorithms.apss.tree;

public class RMQ {

    private int[] array;

    private int size;
    private int[] minOfRange;

    public RMQ(int[] array, int cmp) {

        this.array = new int[array.length];
        for(int i=0; i<array.length; i++)
            this.array[i] = array[i]*cmp;

        size = 1;
        while((size *= 2) < 2*array.length);
        minOfRange = new int[size];

        System.out.println(size);

        initialize(0, array.length-1, 1);
    }

    // index is for range.
    private int initialize(int left, int right, int index) {

        if(left == right)
            return minOfRange[index] = array[left];

        int mid = (left+right)/2;
        int leftMin = initialize(left, mid, 2*index);
        int rightMin = initialize(mid+1, right, 2*index+1);

        return minOfRange[i                    rightMin);
    }

    // both inclusive. find min.
    public int query(int left, int right) {

        return query(left, right, 1, 0, array.length-1);
    }

    private int query(int left, int right, int index, int indexLeft, int
    indexRight) {

        if(right<indexLeft || left>indexRight)
            return Integer.MAX_VALUE;
        if(left<=indexLeft && right>=indexRight)
            return minOfRange[index];

        int mid = (indexLeft+indexRight)/2;

        return min(query(left, right, 2*index, indexLeft, mid),
                   query(left, right, 2*index+1, mid+1, indexRight));
    }

    private int min(int a, int b) {
        return a<b? a:b;
    }

    public static void main(String[] args) {

        int[] array = new int[6];
        for(int i=0; i<6; i++)
            array[i] = i;

        RMQ rmq = new RMQ(array, -1);
    }

}
```

```java
package algorithms.apss.tree;

import java.util.PriorityQueue;
import java.util.StringTokenizer;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem RunningMedian
 * @reference apss - 731p
 *
 * @description
 *
 */
public class RunningMedian {

    private int N;
    private int a, b;

    private int sequence;
    private static final long INITIAL = 1983;
    private static final long MODULUS = 20090711;

    public RunningMedian(int N, int a, int b) {

        this.N = N;
        this.a = a;
        this.b = b;

        this.sequence = (int
    }

    public int solve() {

        PriorityQueue<Integer> maxPQ = new PriorityQueue<Integer>();
        PriorityQueue<Integer> minPQ = new PriorityQueue<Integer>();

        // initialize
        maxPQ.add(-sequence);
        minPQ.add(next());
        int sum = -2*maxPQ.peek();

        for(int i=2; i<N; i++) {

            next();

            if(i%2 == 0) {
                if(sequence > minPQ.peek()) {
                    maxPQ.add(-minPQ.remove());
                    minPQ.add(sequence);
                }
                else {
                    maxPQ.add(sequence);
                }
            }
            else {
                if(sequence < -maxPQ.peek()) {
                    minPQ.add(-maxPQ.remove());
                    minPQ.add(-sequence);
                }
                else {
                    minPQ.add(sequence);
                }
            }

            sum += maxPQ.peek();
        }
```

```
73
74          return sum;
75      }
76
77      private int next() {
78
79          return sequence = (int) ((sequence*(long)a+(long)b)%MODULUS);
80      }
81
82      private class Node {
83
84          private int key;
85          private int value;
86
87          public Node(int key, int value) {
88
89              this.key = key;
90              this.value = value;
91          }
92      }
93
94      public static void main(String[] args) throws NumberFormatException,
        IOException {
95
96          if(args.length == 0) {
97              System.out.println("args fails.");
98              return;
99          }
100
101         FastReader in = new FastReader(args);
102
103         int test = in.nextInt();
104         for(int t=0; t<test; t++) {
105
106             int N = in.nex
107             int a = in.nex
108             int b = in.nextInt();
109
110             RunningMedian rm = new RunningMedian(N, a, b);
111             System.out.println(rm.solve());
112         }
113     }
114
115 }
116
```

```java
package algorithms.apss.tree;

import java.util.StringTokenizer;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 *
 * @author taehyeok.jang
 *
 * @problem Traversal
 * @reference apss - 686p
 *
 * @description
 * transformation from pre, in order traversal into post order traversal.
 */
public class Traversal {

    private int[] preOrder, inOrder;

    private Node root;

    public Traversal(int n, int[] preOrder, int[] inOrder) {

        this.preOrder = preOrder;
        this.inOrder = inOrder;

        this.root = parse(0, 0, n);
    }

    public void postOrder()

        postOrder(root);
    }

    private void postOrder(Node node) {

        if(node == null)
            return ;

        postOrder(node.left);
        postOrder(node.right);

        System.out.println(node.data);
    }

    private Node parse(int s1, int s2, int length) {

        if(length == 0)
            return null;

        Node node = new Node(preOrder[s1]);

        int L;
        for(L=0; inOrder[s2+L] != preOrder[s1]; L++);
        int R = length-1-L;

        node.left = parse(s1+1, s2, L);
        node.right = parse(s1+1+L, s2+1+L, R);

        return node;
    }

    private class Node {

        private int data;
        private Node left, right;

        public Node(int data) {
```

```
73
74              this.data = data;
75              left = null;
76              right = null;
77          }
78      }
79
80      public static void main(String[] args) throws NumberFormatException,
        IOException {
81
82          if(args.length == 0) {
83              System.out.println("args fails.");
84              return;
85          }
86
87          FastReader in = new FastReader(args);
88
89          int test = in.nextInt();
90          for(int t=0; t<test; t++) {
91
92              int n = in.nextInt();
93              int[] preOrder = new int[n];
94              int[] inOrder = new int[n];
95              for(int i=0; i<n; i++)
96                  preOrder[i] = in.nextInt();
97              for(int i=0; i<n; i++)
98                  inOrder[i] = in.nextInt();
99
100             Traversal traversal = new Traversal(n, preOrder, inOrder);
101             traversal.postOrder();
102         }
103     }
104
105 }
106
```

versal.

```java
package algorithms.apss.tree;

public class UnionFind {

    protected int[] id;
    protected int[] size;

    public UnionFind(int n) {

        id = new int[n];
        size = new int[n];

        for(int i=0; i<n; i++)  id[i] = i;
        for(int i=0; i<n; i++)  size[i] = 1;
    }

    protected int root(int i) {

        while(i!=id[i])
            i = id[i];
        return i;
    }

    public int union(int p, int q) {

        if(p == -1) return q;
        if(q == -1) return p;

        int i = root(p);
        int j = root(q);

        if(i == j)
            return i;

        if(size[i]<size[j])
            id[i] = j;
            size[j] += size[i];
            return j;
        }
        else {
            id[j] = i;
            size[i] += size[j];
            return i;
        }
    }

    public boolean connected(int p, int q) {

        return root(p) == root(q);
    }
}

```