

```

1 // Adam Doussan AD844156 04/13/2017
2
3 public class CombPerm
4 {
5     // ans needs to be intialized before calling, take.length needs to equal
6     // words.length
7     public static <T> void genCombs(T[] words, ArrayList<ArrayList<T>> ans,
8                                     boolean [] take, int depth)
9     {
10         if(depth == words.length)
11         {
12             ArrayList<T> temp = new ArrayList<>();
13
14             for(int i = 0; i < take.length; i++)
15                 if(take[i])
16                     temp.add(words[i]);
17
18             ans.add(temp);
19             return;
20         }
21
22         take[depth] = true;
23         genCombs(words, ans, take, depth+1);
24
25         take[depth] = false;
26         genCombs(words, ans, take, depth+1);
27     }
28
29     // ans & now needs to be intialized before calling, take.length needs to equal
30     // words.length
31     public static <T> void genPerms(T[] words, ArrayList<ArrayList<T>> ans,
32                                     boolean [] take, int depth, ArrayList<T> now)
33     {
34         if(depth == words.length)
35         {
36             ans.add(new ArrayList<>(now));
37             return;
38         }
39
40         for(int i = 0; i < words.length; i++)
41         {
42             if(!take[i])
43             {
44                 take[i] = true;
45                 now.add(words[i]);
46                 genPerms(words, ans, take, depth+1, now);
47                 now.remove(now.size()-1);
48                 take[i] = false;
49             }
50         }
51     }
52 }
53
54 // Adam Doussan AD844156 04/11/2017
55
56 public class Graphs
57 {
58     public static void dfs(ArrayList[] graph, boolean [] visited, int v)
59     {
60         visited[v] = true;
61         for(Integer next : ((ArrayList<Integer>[])graph)[v])
62             if(!visited[next])
63                 dfs(graph, visited, next);
64     }
65
66     public static int[] bfs(ArrayList[] graph, int v)
67     {
68         int n = graph.length;
69         int [] distance = new int [n];

```

```

70
71     Arrays.fill(distance, -1);
72     distance[n] = 0;
73
74     Queue<Integer> q = new ArrayDeque<Integer>();
75     q.add(v);
76
77     while(!q.isEmpty())
78     {
79         int cur = q.remove();
80         for(Integer next : ((ArrayList<Integer>[])graph)[cur])
81         {
82             if(distance[next] == -1)
83             {
84                 distance[next] = distance[cur] + 1;
85                 q.add(next);
86             }
87         }
88     }
89
90     return distance;
91 }
92
93
94 // Adam Doussan AD844156 04/13/2017
95
96 public class Dijkstras
97 {
98     final public static int oo = (int) 1e9;
99     private static int n;
100
101     // pass in a value of true to stop if you want early termination for
102     // destination d, false if you want the entire dist array to be filled in
103     public static int[] run(ArrayList<Edge>[] g, int s, int d, boolean stop)
104     {
105         n = g.length;
106
107         boolean [] visited = new boolean[n];
108         int [] dist = new int [n];
109         Arrays.fill(dist, oo);
110
111         PriorityQueue<Edge> pq = new PriorityQueue<Edge>();
112         pq.add(new Edge(s, 0));
113         dist[s] = 0;
114
115         while(!pq.isEmpty())
116         {
117             Edge at = pq.remove();
118             if(visited[at.e]) continue;
119             visited[at.e] = true;
120
121             if(stop && at.e == d) return dist;
122
123             for(Edge adj : g[at.e])
124                 if(!visited[adj.e] && adj.w + at.w < dist[adj.e])
125                     pq.add(new Edge(adj.e, dist[adj.e] = adj.w + at.w));
126         }
127
128         return dist;
129     }
130 }
131
132 class Edge implements Comparable<Edge>
133 {
134     int e, w;
135
136     public Edge(int e, int w)
137     {
138         this.e = e; this.w = w;

```

```

139     }
140
141     public int compareTo(Edge o)
142     {
143         return this.w - o.w;
144     }
145 }
146
147 // Adam Doussan AD844156 04/13/2017
148
149 public class BellmanFord
150 {
151     final public static int oo = (int) 1e9;
152
153     // returns null if there is a negative cycle, edges are directional
154     public static int[] run(Edge[] eList, int n, int s)
155     {
156         int [] dist = new int [n];
157         Arrays.fill(dist, oo);
158         dist[s] = 0;
159
160         for(int i = 0; i < n - 1; i++)
161             for(Edge e : eList)
162                 if(dist[e.v1] + e.w < dist[e.v2])
163                     dist[e.v2] = dist[e.v1] + e.w;
164
165         for(Edge e : eList)
166             if(dist[e.v1] + e.w < dist[e.v2])
167                 return null;
168
169         return dist;
170     }
171 }
172
173 class Edge
174 {
175     public int v1, v2, w;
176
177     public Edge(int v1, int v2, int w)
178     {
179         this.v1 = v1; this.v2 = v2; this.w = w;
180     }
181 }
182
183 // Adam Doussan AD844156 04/13/2017
184
185 public class Floyd
186 {
187     public static int [][] path;
188     final public static int oo = (int)1e9;
189
190     // gets answer without destroying passed in array
191     public static int [][] run(int [][] adj)
192     {
193         int n = adj.length;
194         int [][] m = copy(adj);
195         path = new int[adj.length][adj.length];
196
197         for(int i = 0; i < adj.length; i++)
198             for(int j = 0; j < adj.length; j++)
199                 if(adj[i][j] != oo)
200                     path[i][j] = j;
201                 else
202                     path[i][j] = oo;
203
204         for(int k = 0; k < n; k++)
205             for(int i = 0; i < n; i++)
206                 for(int j = 0; j < n; j++)
207                     if(m[i][k] != oo && m[k][j] != oo)

```

```

208         if(m[i][k] + m[k][j] < m[i][j])
209         {
210             m[i][j] = m[i][k] + m[k][j];
211             path[i][j] = path[i][k];
212         }
213
214     return m;
215 }
216
217 public static int [][] copy(int [][] a)
218 {
219     int [][] res = new int [a.length][a[0].length];
220     for(int i = 0; i < a.length; i++)
221         for(int j = 0; j < a[0].length; j++)
222             res[i][j] = a[i][j];
223     return res;
224 }
225
226 public static ArrayList<Integer> getPath(int i, int j)
227 {
228     if(path[i][j] == oo)
229         return null;
230
231     ArrayList<Integer> myPath = new ArrayList<>();
232
233     myPath.add(i);
234
235     while(i != j)
236     {
237         i = path[i][j];
238         myPath.add(i);
239     }
240
241     return myPath;
242 }
243 }
244
245 // Adam Doussan AD844156 05/01/2017
246
247 public class myFordFulkerson
248 {
249     public int n, s, t;
250     public int [][] g;
251
252     public myFordFulkerson(int n)
253     {
254         this.n = n+2; this.s = n; this.t = n+1;
255         g = new int [n+2][n+2];
256     }
257
258     public void addEdge(int v1, int v2, int cap)
259     {
260         g[v1][v2] = cap;
261     }
262
263     public int flow()
264     {
265         int ans = 0;
266         int [] prev = new int [n];
267
268         while(bfs(prev))
269         {
270             int myNext = prev[t];
271             int myFlow = g[myNext][t];
272
273             while(myNext != s)
274             {
275                 myFlow = Math.min(myFlow, g[prev[myNext]][myNext]);
276                 myNext = prev[myNext];

```

```

277     }
278
279     ans += myFlow;
280     myNext = t;
281
282     while(myNext != s)
283     {
284         int temp = prev[myNext];
285         g[myNext][temp] += myFlow;
286         g[temp][myNext] -= myFlow;
287         myNext = temp;
288     }
289 }
290
291 return ans;
292 }
293
294 public boolean bfs(int[] prev)
295 {
296     Queue<Integer> q = new ArrayDeque<>();
297     boolean [] visited = new boolean [n];
298
299     q.add(s);
300     visited[s] = true;
301
302     while(!q.isEmpty())
303     {
304         int myNext = q.remove();
305
306         for(int i = 0; i < n; i++)
307         {
308             if(!visited[i] && g[myNext][i] > 0)
309             {
310                 visited[i] = true;
311                 q.add(i);
312                 prev[i] = myNext;
313             }
314         }
315     }
316
317     return visited[t];
318 }
319 }
320
321 // Credit Arup Guha
322
323 class FordFulkerson
324 {
325     int[][] cap; boolean[] vis; int n, s, t, oo = (int)(1E9);
326
327     public FordFulkerson(int size) { n = size + 2; s = n - 2; t = n - 1; cap = new
int[n][n]; }
328     void add(int v1, int v2, int c) { cap[v1][v2] = c; }
329
330     int ff()
331     {
332         vis = new boolean[n]; int f = 0;
333         while (true)
334         {
335             Arrays.fill(vis, false);
336             int res = dfs(s, oo);
337             if (res == 0) break;
338             f += res;
339         }
340         return f;
341     }
342
343     int dfs(int pos, int min)
344     {

```

```

345         if (pos == t) return min;
346         if (vis[pos]) return 0;
347         vis[pos] = true; int f = 0;
348
349         for (int i = 0; i < n; i++)
350         {
351             if (cap[pos][i] > 0) f = dfs(i, Math.min(cap[pos][i], min));
352             if (f > 0) { cap[pos][i] -= f; cap[i][pos] += f; return f; }
353         }
354         return 0;
355     }
356 }
357
358 // Credit Arup Guha
359
360 public class Dinic
361 {
362     ArrayDeque<Integer> q;
363     ArrayList<Edge>[] adj;
364     int n, s, t, oo = (int)1E9;
365     boolean[] blocked;
366     int[] dist;
367
368     public Dinic (int N)
369     {
370         n = N; s = n++; t = n++;
371         blocked = new boolean[n];
372         dist = new int[n];
373         q = new ArrayDeque<Integer>();
374         adj = new ArrayList[n];
375         for(int i = 0; i < n; ++i)
376             adj[i] = new ArrayList<Edge>();
377     }
378
379     void add(int v1, int v2, int cap, int flow)
380     {
381         Edge e = new Edge(v1, v2, cap, flow);
382         Edge rev = new Edge(v2, v1, 0, 0);
383         adj[v1].add(rev.rev = e);
384         adj[v2].add(e.rev = rev);
385     }
386
387     boolean bfs()
388     {
389         q.clear();
390         Arrays.fill(dist, -1);
391         dist[t] = 0;
392         q.add(t);
393
394         while(!q.isEmpty()) {
395             int node = q.poll();
396             if(node == s)
397                 return true;
398             for(Edge e : adj[node])
399             {
400                 if(e.rev.cap > e.rev.flow && dist[e.v2] == -1)
401                 {
402                     dist[e.v2] = dist[node] + 1;
403                     q.add(e.v2);
404                 }
405             }
406         }
407         return dist[s] != -1;
408     }
409
410     int dfs(int pos, int min)
411     {
412         if(pos == t)
413             return min;

```

```

414     int flow = 0;
415     for(Edge e : adj[pos])
416     {
417         int cur = 0;
418         if(!blocked[e.v2] && dist[e.v2] == dist[pos]-1 && e.cap - e.flow > 0)
419         {
420             cur = dfs(e.v2, Math.min(min-flow, e.cap - e.flow));
421             e.flow += cur;
422             e.rev.flow = -e.flow;
423             flow += cur;
424         }
425         if(flow == min)
426             return flow;
427     }
428     blocked[pos] = flow != min;
429     return flow;
430 }
431
432 int flow()
433 {
434     clear();
435     int ret = 0;
436     while(bfs())
437     {
438         Arrays.fill(blocked, false);
439         ret += dfs(s, oo);
440     }
441     return ret;
442 }
443
444 void clear()
445 {
446     for(ArrayList<Edge> edges : adj)
447         for(Edge e : edges)
448             e.flow = 0;
449 }
450 }
451
452 class Edge
453 {
454     int v1, v2, cap, flow;
455     Edge rev;
456     Edge(int V1, int V2, int Cap, int Flow)
457     {
458         v1 = V1; v2 = V2; cap = Cap; flow = Flow;
459     }
460 }
461
462 // Conner 04/29/2017
463
464 public class TopologicalSort
465 {
466     public static ArrayList<Integer> run(Graph g)
467     {
468         int degrees[] = new int[g.numVerts];
469         for(int i = 0; i < degrees.length; i++)
470         {
471             ArrayList<Integer> tempArr = g.matrix[i];
472
473             for(int k = 0; k < tempArr.size(); k++)
474                 degrees[tempArr.get(k)]++;
475         }
476
477         Queue<Integer> q = new ArrayDeque<Integer>();
478
479         for(int i = 0; i < g.numVerts; i++)
480             if(degrees[i] == 0)
481                 q.add(i);
482

```

```

483         int visited = 0;
484         ArrayList<Integer> result = new ArrayList<Integer>();
485
486         while(!q.isEmpty())
487         {
488             int currentEdge = q.poll();
489             result.add(currentEdge);
490
491             for(int i = 0; i < g.matrix[currentEdge].size(); i++)
492                 if(--degrees[g.matrix[currentEdge].get(i)] == 0)
493                     q.add(g.matrix[currentEdge].get(i));
494
495             visited++;
496         }
497
498         if(visited != g.numVerts) // cycle
499             return null;
500
501         return result;
502     }
503 }
504
505 class Graph
506 {
507     int numVerts;
508     ArrayList<Integer> matrix[];
509
510     public Graph(int numVerts)
511     {
512         this.numVerts = numVerts;
513         matrix = new ArrayList[numVerts];
514
515         for(int i = 0; i < matrix.length; i++)
516             matrix[i] = new ArrayList<Integer>();
517     }
518
519     public void newEdge(int v1, int v2)
520     {
521         matrix[v1].add(v2);
522     }
523 }
524
525 // Conner 04/29/2017
526
527 public class Kruskals
528 {
529     private static int count = 0;
530     private static int sort = 0;
531
532     public static Edge[] run(Graph g)
533     {
534         Arrays.sort(g.edgeArray);
535         Edge mst[] = new Edge[g.numVerts];
536         for(int i = 0; i < g.numVerts; i++)
537         {
538             mst[i] = new Edge(0,0,0);
539         }
540
541         SubTree stArray[] = new SubTree[g.numVerts];
542         for(int i = 0; i < g.numVerts; i++)
543         {
544             stArray[i] = new SubTree();
545             stArray[i].parent = i;
546             stArray[i].rank = 0;
547         }
548
549         while(sort < g.numVerts - 1)
550         {
551

```



```

552         Edge next = g.edgeArray[count++];
553
554         int v1Base = g.findSet(stArray, next.v1);
555         int v2Base = g.findSet(stArray, next.v2);
556
557         if(v1Base != v2Base)
558         {
559             mst[sort++] = next;
560             g.union(stArray, v1Base, v2Base);
561         }
562     }
563     return mst;
564 }
565 }
566
567 class Graph
568 {
569     int numVerts;
570     int numEdges;
571     Edge edgeArray[];
572
573     public Graph(int numVerts, int numEdges)
574     {
575         this.numVerts = numVerts;
576         this.numEdges = numEdges;
577         edgeArray = new Edge[numEdges];
578         for(int i = 0; i < numEdges; i++)
579         {
580             edgeArray[i] = new Edge(0,0,0);
581         }
582     }
583     public int findSet(SubTree stArray[], int key)
584     {
585         if(stArray[key].parent != key)
586         {
587             stArray[key].parent = findSet(stArray, stArray[key].parent);
588         }
589
590         return stArray[key].parent;
591     }
592     public void union(SubTree[] stArray, int v1, int v2)
593     {
594         int v1Base = findSet(stArray, v1);
595         int v2Base = findSet(stArray, v2);
596
597         if(stArray[v1Base].rank < stArray[v2Base].rank)
598             stArray[v1].parent = v2Base;
599
600         else if(stArray[v2Base].rank < stArray[v1Base].rank)
601             stArray[v2Base].parent = v1Base;
602         else
603         {
604             stArray[v2Base].parent = v1Base;
605             stArray[v1Base].rank++;
606         }
607     }
608 }
609
610
611 class Edge implements Comparable<Edge>
612 {
613     int v1, v2, w;
614
615     public Edge(int v1, int v2, int w)
616     {
617         this.v1 = v1; this.v2 = v2; this.w = w;
618     }
619
620     public int compareTo(Edge e)

```

```

621         {
622             return this.w - e.w;
623         }
624     }
625
626     class SubTree
627     {
628         int parent;
629         int rank;
630     }
631
632     // Adam Doussan AD844156 04/13/2017
633
634     public class MyMath
635     {
636         public static long lcm(long a, long b)
637         {
638             return a / gcd(a,b) * b;
639         }
640
641         public static long gcd(long a, long b)
642         {
643             return (b == 0) ? a : gcd(b, a%b);
644         }
645
646         public static int MCSS(int [] days)
647         {
648             int maxSum = days[0];
649             int runningSum = days[0];
650
651             for(int i = 0; i < days.length; i++)
652             {
653                 if(runningSum < 0)
654                     runningSum = 0;
655
656                 runningSum += days[i];
657
658                 if(runningSum > maxSum)
659                     maxSum = runningSum;
660             }
661
662             return maxSum;
663         }
664
665         public static boolean[] primeSieve(int n)
666         {
667             boolean[] isPrime = new boolean[n+1];
668             Arrays.fill(isPrime, true);
669             isPrime[0] = false;
670             isPrime[1] = false;
671
672             for(int i = 2; i*i <= n; i++)
673                 if(isPrime[i])
674                     for(int j = i + i; j <= n; j += i)
675                         isPrime[j] = false;
676
677             return isPrime;
678         }
679
680         public static long[] extendedEuclid(long a, long b)
681         {
682             if(b == 0)
683                 return new long [] {1,0,a};
684             else
685             {
686                 long q = a / b;
687                 long r = a % b;
688                 long [] rec = extendedEuclid(b,r);
689                 return new long [] {rec[1], rec[0] - q*rec[1], rec[2]};

```

```

690     }
691 }
692
693 public static BigInteger sumOfDivisors (ArrayList<pair> primeFac)
694 {
695     BigInteger sum = new BigInteger("1");
696
697     for(pair a : primeFac)
698     {
699         BigInteger prime = new BigInteger("" + a.prime);
700
701         BigInteger temp = prime.pow((int)a.exp + 1).subtract(BigInteger.ONE);
702         temp = temp.divide(prime.subtract(BigInteger.ONE));
703
704         sum = sum.multiply(temp);
705     }
706
707     return sum;
708 }
709
710 public static ArrayList<pair> primeFactorize(int n)
711 {
712     ArrayList<pair> res = new ArrayList<>();
713     int div = 2;
714
715     while(div*div <= n)
716     {
717         int exp = 0;
718         while(n % div == 0)
719         {
720             n /= div;
721             exp++;
722         }
723
724         if(exp > 0) res.add(new pair(div, exp));
725         div++;
726     }
727
728     if(n > 1) res.add(new pair(n,1));
729     return res;
730 }
731 }
732
733 class pair
734 {
735     public int prime, exp;
736
737     public pair(int p, int e)
738     {
739         prime = p; exp = e;
740     }
741 }
742
743 // Credit Arup Guha
744 // Cleaned up by Adam Doussan AD844156 04/15/2017
745
746 public class ConvexHull
747 {
748     public static void main(String [] args)
749     {
750         Scanner in = new Scanner(System.in);
751         int cases = in.nextInt();
752
753         for(int i = 0; i < cases; i++)
754         {
755             int numCase = in.nextInt();
756             int numPoints = in.nextInt();
757             pt[] pts = new pt[numPoints];
758

```

```

759         for(int k = 0; k < numPoints; k++)
760             pts[k] = new pt(in.nextInt(), in.nextInt());
761
762         pt.ref = getRef(pts, numPoints);
763
764         ArrayList<pt> ans = grahamScan(pts, numPoints);
765     }
766 }
767
768 // returns the point in pts with min y breaking tie by min x
769 public static pt getRef(pt[] pts, int n)
770 {
771     pt ans = pts[0];
772
773     for(int i = 1; i < n; i++)
774         if(pts[i].y < ans.y || (pts[i].y == ans.y && pts[i].x <
775             ans.x))
776             ans = pts[i];
777
778     return ans;
779 }
780
781 public static ArrayList<pt> grahamScan(pt[] pts, int n)
782 {
783     Arrays.sort(pts);
784
785     Stack<pt> st = new Stack<>();
786     st.push(pts[0]);
787     st.push(pts[1]);
788
789     for(int i = 2; i < n; i++)
790     {
791         pt next = pts[i];
792         pt mid = st.pop();
793         pt prev = st.pop();
794
795         while(!prev.isRightTurn(mid, next))
796         {
797             mid = prev;
798             prev = st.pop();
799         }
800
801         st.push(prev);
802         if(!prev.isStraight(mid, next)) // deleting collinear points
803             st.push(mid);
804         st.push(next);
805     }
806
807     ArrayList<pt> ans = new ArrayList<>();
808
809     while(!st.isEmpty())
810         ans.add(st.pop());
811
812     return ans;
813 }
814
815
816 class pt implements Comparable<pt>
817 {
818     public static pt ref;
819     public int x, y;
820
821     public pt(int x, int y)
822     {
823         this.x = x; this.y = y;
824     }
825
826     public pt vect(pt pt2)

```

```

827     {
828         return new pt(pt2.x-x, pt2.y-y);
829     }
830
831     public double dist(pt pt2)
832     {
833         return Math.sqrt(Math.pow(pt2.x-x, 2) + Math.pow(pt2.y-y, 2));
834     }
835
836     public int cross(pt pt2)
837     {
838         return x*pt2.y - y*pt2.x;
839     }
840
841     public boolean isRightTurn(pt mid, pt next)
842     {
843         pt v1 = vect(mid);
844         pt v2 = mid.vect(next);
845         return v1.cross(v2) >= 0;
846     }
847
848     // used to get rid of collinear points that are already accounted for by
849     // points on either end
850     public boolean isStraight(pt mid, pt next)
851     {
852         pt v1 = vect(mid);
853         pt v2 = mid.vect(next);
854         return v1.cross(v2) == 0;
855     }
856
857     public boolean isZero()
858     {
859         return x == 0 && y == 0;
860     }
861
862     public int compareTo(pt other)
863     {
864         pt v1 = ref.vect(this);
865         pt v2 = ref.vect(other);
866
867         if (v1.isZero()) return -1;
868         if (v2.isZero()) return 1;
869
870         if (v1.cross(v2) != 0)
871             return -v1.cross(v2);
872
873         if (ref.dist(v1) < ref.dist(v2))
874             return -1;
875         return 1;
876     }
877 }
878
879 // Adam Doussan AD844156 04/28/2017
880
881 public class Polygon
882 {
883     public static boolean ptInPoly(pt intersect, pt [] poly)
884     {
885         double totalAngle = 0;
886
887         for(int i = 0; i < poly.length; i++)
888         {
889             pt v1 = intersect.vect(poly[i]);
890             pt v2 = intersect.vect(poly[(i+1)%poly.length]);
891
892             try
893             {
894                 totalAngle += Math.acos((v1.dot(v2)) / (v1.mag() * v2.mag()));
895             }

```

```

896         catch(Exception e)
897         {
898         }
899     }
900
901     return (2*Math.PI) - totalAngle < 1e-9;
902 }
903
904 // assumes polygon is 2d, z is unused and set to 0, and points are ordered
905 // such that pt[x] -> pt[(x+1) % pt.length] is connected with a line. Also
906 // assumes simple polygon
907 public static double area(pt [] poly)
908 {
909     double ans = 0;
910     pt ori = new pt(0,0,0);
911
912     for(int i = 0; i < poly.length; i++)
913     {
914         pt v1 = ori.vect(poly[i]);
915         pt v2 = ori.vect(poly[(i+1)%poly.length]);
916         ans += v1.cross(v2).z;
917     }
918
919     return 0.5 * Math.abs(ans);
920 }
921 }
922
923 class pt
924 {
925     public double x, y, z;
926
927     pt(double x, double y, double z)
928     {
929         this.x = x; this.y = y; this.z = z;
930     }
931
932     public pt vect(pt pt2)
933     {
934         return new pt(pt2.x-x, pt2.y-y, pt2.z-z);
935     }
936
937     public pt cross(pt pt2)
938     {
939         double x = this.y * pt2.z - pt2.y * this.z;
940         double y = this.z * pt2.x - pt2.z * this.x;
941         double z = this.x * pt2.y - pt2.x * this.y;
942
943         return new pt(x,y,z);
944     }
945
946     public double mag()
947     {
948         return Math.sqrt(Math.pow(this.x,2) + Math.pow(this.y,2) + Math.pow(this.z,2));
949     }
950
951     public double dot(pt vect)
952     {
953         return (this.x * vect.x) + (this.y * vect.y) + (this.z * vect.z);
954     }
955 }
956
957 // Ross 4/24/2017
958
959 public class LineCircleIntersect
960 {
961     // returns true if the segment intersects the circle, false otherwise
962     public static boolean intersect(pt start, pt end, circ circle)
963     {
964         double dx = end.x - start.x;

```

```

965         double dy = end.y - start.y;
966
967         double a = ((dx*dx)+(dy*dy));
968         double b = 2 * (dx * (start.x - circle.x) + dy * (start.y - circle.y));
969         double c = ((start.x - circle.x) * (start.x - circle.x))
970                 + ((start.y - circle.y) * (start.y - circle.y))
971                 - (circle.radius * circle.radius);
972
973         double det = (b*b) - (4 * a * c);
974
975         // Answer will be non-real; discard.
976         if (det < 0)
977             return false;
978
979         double soln1 = ((-1 * b) + Math.sqrt(det))/(2*a);
980         double soln2 = ((-1 * b) - Math.sqrt(det))/(2*a);
981
982         if (((soln1 < 0) || (soln1 > 1)) && ((soln2 < 0) || (soln2 > 1)))
983             return false;
984         return true;
985     }
986 }
987
988 class pt
989 {
990     double x,y;
991 }
992
993 class circ
994 {
995     double x,y;
996     double radius;
997 }
998
999 // Adam Doussan 4/24/2017
1000
1001 public class LineLineIntersection
1002 {
1003     // java.awt.geom.Line2D.linesIntersect()
1004
1005     public static boolean intersect(line a, line b)
1006     {
1007         return a.intersect(b);
1008     }
1009 }
1010
1011 class vect
1012 {
1013     public double x,y;
1014
1015     public vect(pt start, pt end)
1016     {
1017         x = end.x - start.x;
1018         y = end.y - start.y;
1019     }
1020 }
1021
1022 class pt
1023 {
1024     public double x, y;
1025
1026     public pt(double x, double y)
1027     {
1028         this.x = x; this.y = y;
1029     }
1030 }
1031
1032 class line
1033 {

```

```

1034     public pt p;
1035     public vect dir;
1036
1037     public line(pt start, pt end)
1038     {
1039         p = start;
1040         dir = new vect(start, end);
1041     }
1042
1043     public boolean intersect(line other)
1044     {
1045         double den = det(dir.x, -other.dir.x, dir.y, -other.dir.y);
1046         double numLam = det(other.p.x-p.x, -other.dir.x, other.p.y-p.y, -other.dir.y);
1047
1048         // paralell
1049         if(den < 1e-9)
1050             return false;
1051
1052         else if(numLam/den > 1)
1053             return false;
1054
1055         else
1056             return true;
1057     }
1058
1059     public static double det(double a,double b ,double c ,double d)
1060     {
1061         return a*d-b*c;
1062     }
1063 }
1064
1065 // Adam Doussan AD844156 04/14/2017
1066
1067 public class LinePlaneIntersect
1068 {
1069     // example where input is given as points
1070     public static void main(String [] args)
1071     {
1072         Scanner in = new Scanner(System.in);
1073
1074         int run = in.nextInt();
1075
1076         for(int r = 1; r <= run; r++)
1077         {
1078             pt p1 = new pt(in.nextInt(),in.nextInt(),in.nextInt());
1079             pt p2 = new pt(in.nextInt(),in.nextInt(),in.nextInt());
1080
1081             line l = new line(p1.vect(p2), p1);
1082
1083             pt p3 = new pt(in.nextInt(),in.nextInt(),in.nextInt());
1084             pt p4 = new pt(in.nextInt(),in.nextInt(),in.nextInt());
1085             pt p5 = new pt(in.nextInt(),in.nextInt(),in.nextInt());
1086
1087             plane p = new plane(p3.vect(p4), p3.vect(p5), p3);
1088
1089             System.out.format("Data Set #%d:\n", r);
1090             solve(l,p);
1091             System.out.println();
1092         }
1093     }
1094
1095     public static void solve(line l, plane p)
1096     {
1097         int right = p.d - (l.p.x*p.orth.x + l.p.y*p.orth.y + l.p.z*p.orth.z);
1098         int left = (l.vect.x*p.orth.x + l.vect.y*p.orth.y + l.vect.z*p.orth.z);
1099
1100         if(right == 0)
1101         {
1102             System.out.println("The line lies on the plane.");

```



```

1103     }
1104
1105     else if(left == 0)
1106     {
1107         System.out.println("There is no intersection.");
1108     }
1109
1110     else
1111     {
1112         double lam = ((double)right) / left;
1113         System.out.format("The intersection is the point (%.1f, %.1f, %.1f).\n",
1114             l.p.x + l.vect.x*lam, l.p.y + l.vect.y*lam, l.p.z + l.vect.z*lam);
1115     }
1116 }
1117
1118
1119 class pt
1120 {
1121     int x, y, z;
1122
1123     pt(int x, int y, int z)
1124     {
1125         this.x = x; this.y = y; this.z = z;
1126     }
1127
1128     public pt vect(pt pt2)
1129     {
1130         return new pt(pt2.x-x, pt2.y-y, pt2.z-z);
1131     }
1132
1133     public pt cross(pt pt2)
1134     {
1135         int x = this.y * pt2.z - pt2.y * this.z;
1136         int y = this.z * pt2.x - pt2.z * this.x;
1137         int z = this.x * pt2.y - pt2.x * this.y;
1138
1139         return new pt(x,y,z);
1140     }
1141
1142     public int solve(pt pt2)
1143     {
1144         return this.x*pt2.x + this.y*pt2.y + this.z*pt2.z;
1145     }
1146 }
1147
1148 class line
1149 {
1150     pt vect, p;
1151
1152     public line(pt vect, pt p)
1153     {
1154         this.vect = vect; this.p = p;
1155     }
1156 }
1157
1158 class plane
1159 {
1160     pt orth;
1161     int d;
1162
1163     public plane(pt vect1, pt vect2, pt ori)
1164     {
1165         orth = vect1.cross(vect2);
1166         d = orth.solve(ori);
1167     }
1168 }
1169
1170 // Adam Doussan AD844156 04/14/2017
1171

```

```

1172 public class KnapSack
1173 {
1174     // unlim = true means can take unlimited times, otherwise each thing can be
1175     // taken only once. Add multiple times if multiple limited number of same
1176     // object. kp[money] contains the answer for money.
1177     public static long[] solve(int [] cost, int [] value, int money, boolean unlim)
1178     {
1179         long [] kp = new long [money+1];
1180         int n = cost.length;
1181
1182         if(unlim)
1183         {
1184             for(int j = 0; j < n; j++)
1185             {
1186                 for(int k = cost[j]; k < money+1; k++)
1187                 {
1188                     if(kp[k-cost[j]] + value[j] > kp[k])
1189                         kp[k] = kp[k - cost[j]] + value[j];
1190                 }
1191             }
1192         }
1193         else
1194         {
1195             for(int j = 0; j < n; j++)
1196             {
1197                 for(int k = money+1; k >= cost[j] ; k--)
1198                 {
1199                     if(kp[k-cost[j]] + value[j] > kp[k])
1200                         kp[k] = kp[k - cost[j]] + value[j];
1201                 }
1202             }
1203         }
1204     }
1205     return kp;
1206 }
1207
1208 // Ross 04/30/2017
1209
1210 public class MatrixChain
1211 {
1212     public static void main(String [] args)
1213     {
1214         //Matrix A is 2x4, Matrix B is 4x2, Matrix C is 2x3 and so on.
1215         int matr[] = new int[] {2, 4, 2, 3, 1, 4};
1216         int size = matr.length;
1217
1218         int res = MCM(matr, size);
1219         System.out.println(res);
1220     }
1221
1222     static int MCM(int[] matr, int size)
1223     {
1224         int a;
1225         int memo[][] = new int[size][size];
1226
1227         for(int i=0; i<size; i++)
1228         {
1229             memo[i][i] = 0;
1230         }
1231
1232         for(int sub=2; sub <= (size-1); sub++)
1233         {
1234             for(int j=1; j <= ((size-1) - sub + 1); j++)
1235             {
1236                 memo[j][sub+j-1] = Integer.MAX_VALUE;
1237                 for(int k=j; k <= (sub+j-2); k++)
1238                 {
1239
1240

```

```

1241         a = memo[j][k] + memo[k+1][j+sub-1] + (matr[j-1] * matr[k] *
1242         matr[sub+j-1]);
1243         if (a < memo[j][j+sub-1])
1244         {
1245             memo[j][j+sub-1] = a;
1246         }
1247     }
1248 }
1249
1250     return memo[1][size-1];
1251 }
1252 }
1253
1254 // Ross 04/30/2017
1255
1256 public class LCS
1257 {
1258     static int LCS(String str1, String str2, int len1, int len2)
1259     {
1260         int[][] memo = new int[len1+1][len2+1];
1261
1262         for(int i=1; i<=len1; i++)
1263         {
1264             for(int j=1; j<=len2; j++)
1265             {
1266                 if((i == 0) || (j == 0))
1267                 {
1268                     memo[i][j] = 0;
1269                 }
1270                 else if(str1.charAt(i-1) == str2.charAt(j-1))
1271                 {
1272                     memo[i][j] = 1 + memo[i-1][j-1];
1273                 }
1274                 else
1275                 {
1276                     int sub = Math.max(memo[i][j-1], memo[i-1][j]);
1277                     memo[i][j] = sub;
1278                 }
1279             }
1280         }
1281         return memo[len1][len2];
1282     }
1283 }

```