HackPack

Adam Doussan
Conner Tharp
Ross Centanni

Contents

```java
// Adam Doussan AD844156 04/13/2017

public class CombPerm
{
    // ans needs to be intialized before calling, take.length needs to equal
    // words.length
    public static <T> void genCombs(T[] words, ArrayList<ArrayList<T>> ans,
                                    boolean [] take, int depth)
    {
        if(depth == words.length)
        {
            ArrayList<T> temp = new ArrayList<>();

            for(int i = 0; i < take.length; i++)
                if(take[i])
                    temp.add(words[i]);

            ans.add(temp);
            return;
        }

        take[depth] = true;
        genCombs(words,ans,take,depth+1);

        take[depth] = false;
        genCombs(words,ans,take,depth+1);
    }

    // ans & now needs to be intialized before calling, take.length needs to equal
    // words.length
    public static <T> void genPerms(T[] words, ArrayList<ArrayList<T>> ans,
                                    boolean [] take, int depth, ArrayList<T> now)
    {
        if(depth == words.length)
        {
            ans.add(new ArrayList<>(now));
            return;
        }

        for(int i = 0; i < words.length; i++)
        {
            if(!take[i])
            {
                take[i] = true;
                now.add(words[i]);
                genPerms(words, ans, take, depth+1, now);
                now.remove(now.size()-1);
                take[i] = false;
            }
        }
    }
}

// Adam Doussan AD844156 04/11/2017

public class Graphs
{
    public static void dfs(ArrayList[] graph, boolean [] visited, int v)
    {
        visited[v] = true;
        for(Integer next : ((ArrayList<Integer>[])graph)[v])
            if(!visited[next])
                dfs(graph, visited, next);
    }

    public static int[] bfs(ArrayList[] graph, int v)
    {
        int n = graph.length;
        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;
```

```
    }

    public int compareTo(Edge o)
    {
        return this.w - o.w;
    }
}

// Adam Doussan AD844156 04/13/2017

public class BellmanFord
{
    final public static int oo = (int) 1e9;

    // returns null if there is a negative cycle, edges are directional
    public static int[] run(Edge[] eList, int n, int s)
    {
        int [] dist = new int [n];
        Arrays.fill(dist, oo);
        dist[s] = 0;

        for(int i = 0; i < n - 1; i++)
            for(Edge e : eList)
                if(dist[e.v1] + e.w < dist[e.v2])
                    dist[e.v2] = dist[e.v1] + e.w;

        for(Edge e : eList)
            if(dist[e.v1] + e.w < dist[e.v2])
                return null;

        return dist;
    }
}

class Edge
{
    public int v1, v2, w;

    public Edge(int v1, int v2, int w)
    {
        this.v1 = v1; this.v2 = v2; this.w = w;
    }
}

// Adam Doussan AD844156 04/13/2017

public class Floyd
{
    public static int [][] path;
    final public static int oo = (int)1e9;

    // gets answer without destroying passed in array
    public static int [][] run(int [][] adj)
    {
        int n = adj.length;
        int [][] m = copy(adj);
        path = new int[adj.length][adj.length];

        for(int i = 0; i < adj.length; i++)
            for(int j = 0; j < adj.length; j++)
                if(adj[i][j] != oo)
                    path[i][j] = j;
                else
                    path[i][j] = oo;

        for(int k = 0; k < n; k++)
            for(int i = 0; i < n; i++)
                for(int j = 0; j < n; j++)
                    if(m[i][k] != oo && m[k][j] != oo)
```

```
                        if(m[i][k] + m[k][j] < m[i][j])
                        {
                            m[i][j] = m[i][k] + m[k][j];
                            path[i][j] = path[i][k];
                        }

            return m;
        }

        public static int [][] copy(int [][] a)
        {
            int [][] res = new int [a.length][a[0].length];
                for(int i = 0; i < a.length; i++)
                    for(int j = 0; j < a[0].length; j++)
                        res[i][j] = a[i][j];
            return res;
        }

        public static ArrayList<Integer> getPath(int i, int j)
        {
            if(path[i][j] == oo)
                return null;

            ArrayList<Integer> myPath = new ArrayList<>();

            myPath.add(i);

            while(i != j)
            {
                i = path[i][j];
                myPath.add(i);
            }

            return myPath;
        }
    }

    // Adam Doussan AD844156 05/01/2017

    public class myFordFulkerson
    {
        public int n, s, t;
        public int [][] g;

        public myFordFulkerson(int n)
        {
            this.n = n+2; this.s = n; this.t = n+1;
            g = new int [n+2][n+2];
        }

        public void addEdge(int v1, int v2, int cap)
        {
            g[v1][v2] = cap;
        }

        public int flow()
        {
            int ans = 0;
            int [] prev = new int [n];

            while(bfs(prev))
            {
                int myNext = prev[t];
                int myFlow = g[myNext][t];

                while(myNext != s)
                {
                    myFlow = Math.min(myFlow, g[prev[myNext]][myNext]);
                    myNext = prev[myNext];
```

```
            }

            ans += myFlow;
            myNext = t;

            while(myNext != s)
            {
                int temp = prev[myNext];
                g[myNext][temp] += myFlow;
                g[temp][myNext] -= myFlow;
                myNext = temp;
            }
        }

        return ans;
    }

    public boolean bfs(int[] prev)
    {
        Queue<Integer> q = new ArrayDeque<>();
        boolean [] visited = new boolean [n];

        q.add(s);
        visited[s] = true;

        while(!q.isEmpty())
        {
            int myNext = q.remove();

            for(int i = 0; i < n; i++)
            {
                if(!visited[i] && g[myNext][i] > 0)
                {
                    visited[i] = true;
                    q.add(i);
                    prev[i] = myNext;
                }
            }
        }

        return visited[t];
    }
}

// Credit Arup Guha

class FordFulkerson
{
    int[][] cap;  boolean[] vis;  int n, s, t, oo = (int)(1E9);

    public FordFulkerson(int size) { n = size + 2;  s = n - 2;  t = n - 1;  cap = new
    int[n][n]; }
    void add(int v1, int v2, int c) {  cap[v1][v2] = c;  }

    int ff()
    {
        vis = new boolean[n];  int f = 0;
        while (true)
        {
            Arrays.fill(vis, false);
            int res = dfs(s, oo);
            if (res == 0) break;
            f += res;
        }
        return f;
    }

    int dfs(int pos, int min)
    {
```

```
        if (pos == t)  return min;
        if (vis[pos])  return 0;
        vis[pos] = true;  int f = 0;

        for (int i = 0; i < n; i++)
        {
            if (cap[pos][i] > 0)  f = dfs(i, Math.min(cap[pos][i], min));
            if (f > 0) { cap[pos][i] -= f;  cap[i][pos] += f;  return f; }
        }
        return 0;
    }
}

// Credit Arup Guha

public class Dinic
{
    ArrayDeque<Integer> q;
    ArrayList<Edge>[] adj;
    int n, s, t, oo = (int)1E9;
    boolean[] blocked;
    int[] dist;

    public Dinic (int N)
    {
        n = N; s = n++; t = n++;
        blocked = new boolean[n];
        dist = new int[n];
        q = new ArrayDeque<Integer>();
        adj = new ArrayList[n];
        for(int i = 0; i < n; ++i)
            adj[i] = new ArrayList<Edge>();
    }

    void add(int v1, int v2, int cap, int flow)
    {
        Edge e = new Edge(v1, v2, cap, flow);
        Edge rev = new Edge(v2, v1, 0, 0);
        adj[v1].add(rev.rev = e);
        adj[v2].add(e.rev = rev);
    }

    boolean bfs()
    {
        q.clear();
        Arrays.fill(dist, -1);
        dist[t] = 0;
        q.add(t);

        while(!q.isEmpty()) {
            int node = q.poll();
            if(node == s)
                return true;
            for(Edge e : adj[node])
            {
                if(e.rev.cap > e.rev.flow && dist[e.v2] == -1)
                {
                    dist[e.v2] = dist[node] + 1;
                    q.add(e.v2);
                }
            }
        }
        return dist[s] != -1;
    }

    int dfs(int pos, int min)
    {
        if(pos == t)
            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 
506     class Graph
507     {
508         int numVerts;
509         ArrayList<Integer> matrix[];
510 
511         public Graph(int numVerts)
512         {
513             this.numVerts = numVerts;
514             matrix = new ArrayList[numVerts];
515 
516             for(int i = 0; i < matrix.length; i++)
517                 matrix[i] = new ArrayList<Integer>();
518         }
519 
520         public void newEdge(int v1, int v2)
521         {
522             matrix[v1].add(v2);
523         }
524     }
525 
526     // Conner  04/29/2017
527 
528     public class Kruskals
529     {
530         private static int count = 0;
531         private static int sort = 0;
532 
533         public static Edge[] run(Graph g)
534         {
535             Arrays.sort(g.edgeArray);
536             Edge mst[] = new Edge[g.numVerts];
537             for(int i = 0; i < g.numVerts; i++)
538             {
539                 mst[i] = new Edge(0,0,0);
540             }
541 
542             SubTree stArray[] = new SubTree[g.numVerts];
543             for(int i = 0; i < g.numVerts; i++)
544             {
545                 stArray[i] = new SubTree();
546                 stArray[i].parent = i;
547                 stArray[i].rank = 0;
548             }
549 
550             while(sort < g.numVerts - 1)
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)
```

```
        {
            return this.w - e.w;
        }
}

class SubTree
{
    int parent;
    int rank;
}

// Adam Doussan AD844156 04/13/2017

public class MyMath
{
    public static long lcm(long a, long b)
    {
        return a / gcd(a,b) * b;
    }

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

    public static int MCSS(int [] days)
    {
        int maxSum = days[0];
        int runningSum = days[0];

        for(int i = 0; i < days.length; i++)
        {
            if(runningSum < 0)
                runningSum = 0;

            runningSum += days[i];

            if(runningSum > maxSum)
                maxSum = runningSum;
        }

        return maxSum;
    }

    public static boolean[] primeSieve(int n)
    {
        boolean[] isPrime = new boolean[n+1];
        Arrays.fill(isPrime, true);
        isPrime[0] = false;
        isPrime[1] = false;

        for(int i = 2; i*i <= n; i++)
            if(isPrime[i])
                for(int j = i + i; j <= n; j += i)
                    isPrime[j] = false;

        return isPrime;
    }

    public static long[] extendedEuclid(long a, long b)
    {
        if(b == 0)
            return new long []{1,0,a};
        else
        {
            long q = a / b;
            long r = a % b;
            long [] rec = extendedEuclid(b,r);
            return new long [] {rec[1], rec[0] - q*rec[1], rec[2]};
```

```
        }
    }

    public static BigInteger sumOfDivisors(ArrayList<pair> primeFac)
    {
        BigInteger sum = new BigInteger("1");

        for(pair a : primeFac)
        {
            BigInteger prime = new BigInteger("" + a.prime);

            BigInteger temp = prime.pow((int)a.exp + 1).subtract(BigInteger.ONE);
            temp = temp.divide(prime.subtract(BigInteger.ONE));

            sum = sum.multiply(temp);
        }

        return sum;
    }

    public static ArrayList<pair> primeFactorize(int n)
    {
        ArrayList<pair> res = new ArrayList<>();
        int div = 2;

        while(div*div <= n)
        {
            int exp = 0;
            while(n % div == 0)
            {
                n /= div;
                exp++;
            }

            if(exp > 0) res.add(new pair(div, exp));
            div++;
        }

        if(n > 1) res.add(new pair(n,1));
        return res;
    }
}

class pair
{
    public int prime, exp;

    public pair(int p, int e)
    {
        prime = p; exp = e;
    }
}

// Credit Arup Guha
// Cleaned up by Adam Doussan AD844156 04/15/2017

public class ConvexHull
{
    public static void main(String [] args)
    {
        Scanner in = new Scanner(System.in);
        int cases = in.nextInt();

        for(int i = 0; i < cases; i++)
        {
            int numCase = in.nextInt();
            int numPoints = in.nextInt();
            pt[] pts = new pt[numPoints];

```

```
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 <
                ans.x))
775                 ans = pts[i];
776 
777         return ans;
778     }
779 
780     public static ArrayList<pt> grahamScan(pt[] pts, int n)
781     {
782         Arrays.sort(pts);
783 
784         Stack<pt> st = new Stack<>();
785         st.push(pts[0]);
786         st.push(pts[1]);
787 
788         for(int i = 2; i < n; i++)
789         {
790             pt next = pts[i];
791             pt mid = st.pop();
792             pt prev = st.pop();
793 
794             while(!prev.isRightTurn(mid, next))
795             {
796                 mid = prev;
797                 prev = st.pop();
798             }
799 
800             st.push(prev);
801             if(!prev.isStraight(mid, next)) // deleting collinear points
802                 st.push(mid);
803             st.push(next);
804         }
805 
806         ArrayList<pt> ans = new ArrayList<>();
807 
808         while(!st.isEmpty())
809             ans.add(st.pop());
810 
811         return ans;
812     }
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                 }
```

```
            catch(Exception e)
            {
            }
        }

        return (2*Math.PI) - totalAngle < 1e-9;
    }

    // assumes polygon is 2d, z is unused and set to 0, and points are ordered
    // such that pt[x] -> pt[(x+1) % pt.length] is connected with a line. Also
    // assumes simple polygon
    public static double area(pt [] poly)
    {
        double ans = 0;
        pt ori = new pt(0,0,0);

        for(int i = 0; i < poly.length; i++)
        {
            pt v1 = ori.vect(poly[i]);
            pt v2 = ori.vect(poly[(i+1)%poly.length]);
            ans += v1.cross(v2).z;
        }

        return 0.5 * Math.abs(ans);
    }
}

class pt
{
    public double x, y, z;

    pt(double x, double y, double z)
    {
        this.x = x; this.y = y; this.z = z;
    }

    public pt vect(pt pt2)
    {
        return new pt(pt2.x-x, pt2.y-y, pt2.z-z);
    }

    public pt cross(pt pt2)
    {
        double x = this.y * pt2.z - pt2.y * this.z;
        double y = this.z * pt2.x - pt2.z * this.x;
        double z = this.x * pt2.y - pt2.x * this.y;

        return new pt(x,y,z);
    }

    public double mag()
    {
        return Math.sqrt(Math.pow(this.x,2) + Math.pow(this.y,2) + Math.pow(this.z,2));
    }

    public double dot(pt vect)
    {
        return (this.x * vect.x) + (this.y * vect.y) + (this.z * vect.z);
    }
}

// Ross 4/24/2017

public class LineCircleIntersect
{
    // returns true if the segment intersects the circle, false otherwise
    public static boolean intersect(pt start, pt end, circ circle)
    {
        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.");
```

```
        }

        else if(left == 0)
        {
            System.out.println("There is no intersection.");
        }

        else
        {
            double lam = ((double)right) / left;
            System.out.format("The intersection is the point (%.1f, %.1f, %.1f).\n",
            l.p.x + l.vect.x*lam, l.p.y + l.vect.y*lam, l.p.z + l.vect.z*lam);
        }
    }
}

class pt
{
    int x, y, z;

    pt(int x, int y, int z)
    {
        this.x = x; this.y = y; this.z = z;
    }

    public pt vect(pt pt2)
    {
        return new pt(pt2.x-x, pt2.y-y, pt2.z-z);
    }

    public pt cross(pt pt2)
    {
        int x = this.y * pt2.z - pt2.y * this.z;
        int y = this.z * pt2.x - pt2.z * this.x;
        int z = this.x * pt2.y - pt2.x * this.y;

        return new pt(x,y,z);
    }

    public int solve(pt pt2)
    {
        return this.x*pt2.x + this.y*pt2.y + this.z*pt2.z;
    }
}

class line
{
    pt vect, p;

    public line(pt vect, pt p)
    {
        this.vect = vect; this.p = p;
    }
}

class plane
{
    pt orth;
    int d;

    public plane(pt vect1, pt vect2, pt ori)
    {
        orth = vect1.cross(vect2);
        d = orth.solve(ori);
    }
}

// Adam Doussan AD844156 04/14/2017

```

```java
public class KnapSack
{
    // unlim = true means can take unlimited times, otherwise each thing can be
    // taken only once. Add multiple times if multiple limited number of same
    // object. kp[money] contains the answer for money.
    public static long[] solve(int [] cost, int [] value, int money, boolean unlim)
    {
        long [] kp = new long [money+1];
        int n = cost.length;

        if(unlim)
        {
            for(int j = 0; j < n; j++)
            {
                for(int k = cost[j]; k < money+1; k++)
                {
                    if(kp[k-cost[j]] + value[j] > kp[k])
                        kp[k] = kp[k - cost[j]] + value[j];
                }
            }
        }

        else
        {
            for(int j = 0; j < n; j++)
            {
                for(int k = money+1; k >= cost[j] ; k--)
                {
                    if(kp[k-cost[j]] + value[j] > kp[k])
                        kp[k] = kp[k - cost[j]] + value[j];
                }
            }
        }

        return kp;
    }
}

// Ross 04/30/2017

public class MatrixChain
{
    public static void main(String [] args)
    {
        //Matrix A is 2x4, Matrix B is 4x2, Matrix C is 2x3 and so on.
        int matr[] = new int[] {2, 4, 2, 3, 1, 4};
        int size = matr.length;

        int res = MCM(matr, size);
        System.out.println(res);
    }

    static int MCM(int[] matr, int size)
    {
        int a;
        int memo[][] = new int[size][size];

        for(int i=0; i<size; i++)
        {
            memo[i][i] = 0;
        }

        for(int sub=2; sub <= (size-1); sub++)
        {
            for(int j=1; j <= ((size-1) - sub + 1); j++)
            {
                memo[j][sub+j-1] = Integer.MAX_VALUE;
                for(int k=j; k <= (sub+j-2); k++)
                {
```

```
                    a = memo[j][k] + memo[k+1][j+sub-1] + (matr[j-1] * matr[k] *
                    matr[sub+j-1]);
                    if (a < memo[j][j+sub-1])
                    {
                        memo[j][j+sub-1] = a;
                    }
                }
            }
        }

        return memo[1][size-1];
    }
}

// Ross 04/30/2017

public class LCS
{
    static int LCS(String str1, String str2, int len1, int len2)
    {
        int[][] memo = new int[len1+1][len2+1];

        for(int i=1; i<=len1; i++)
        {
            for(int j=1; j<=len2; j++)
            {
                if((i == 0) || (j == 0))
                {
                    memo[i][j] = 0;
                }
                else if(str1.charAt(i-1) == str2.charAt(j-1))
                {
                    memo[i][j] = 1 + memo[i-1][j-1];
                }
                else
                {
                    int sub = Math.max(memo[i][j-1], memo[i-1][j]);
                    memo[i][j] = sub;
                }
            }
        }
        return memo[len1][len2];
    }
}
```