AlgoLab ETH 2013

name	problem	solution	P
Dominoes	Given a list of tiles of different heights, determine how many tiles will fall after toppling the left-most tile.	-	15
Aliens	Given a set of intervals, count those that have an element that is not contained in any other interval.	-	20
Shelves	Given two different types of shelves with length m and n, $m \leq n$ and an empty space with length l, determine the optimal number of shelves x and y such that $x * m + y * n = l - \epsilon$, $epsilon > 0$, epsilon is minimized. Second objective is minimizing y.	clever loop with branching that reduces runtime to $O(\sqrt{l})$	16
False Coin	Given a series weighing outcomes of several coins, determine which coin has a different weight than the other coins.	-	24
Light Pattern	Given a sequence of n bulbs that are in the states "on" or "off", determine the minimal number of "changes" that are necessary such that each interval $i * k$ to $(i + 1) * k$ of the bulbs follows a given pattern. A "change" is either changing the state of a single bulb or change the state of every bulb from 0 up to $t * k - 1$.	-	35
Divisor Distance	Given a number n. Let G_n be a graph with 1 to n as vertices. Further, in G_n there is an (undirected) edge between vertices i and j with $i > j$ iff j is the largest proper divisor of i. Determine the minimal distance of two given vertices in this graph.	-	101
Formula One	Given a sequence of integers, determine the minimum number of "swaps" of numbers at position i and $i + 1$ are necessary to sort the list.	merge sort and count swaps	26
Boats	Given a set of boat lengths l_i and positions of rings p_i, determine the maximum number of boats that can be tied.	Greedy (Boat with earliest end then longest)	22
Checking Change	Given a number of different coin-values c_i, output the minimum number of coins that are necessary to represent m_i.	DP	12
Even Pairs	Given a list $x_1, \ldots, x_n$, count the number of pairs $1 \leq i \leq j \leq n$ for which the sum is even.	DP	18
Burning Coins	Given a sequence of coins $c_1, \ldots, c_n$ with values $v_1, \ldots, v_n$. Each player $\in \{A, B\}$ is alternately allowed to remove the leftmost or rightmost coin of the sequence until the sequence is empty. Output the maximum value of coins that A can remove.	DP	33
Jump	Given n cells, a number representing how far the agent can jump and the cost a_i to land on a cell. Determine the minimum cost to get from cell 1 to cell n.	DP	34

Poker Chips	Given some stacks S_i with chips c_{ij} each having a color. An agent A is allowed to remove the topmost chip from each stack provided they have the same color. If A removed $k > 1$ chips, he is awarded with 2^{k-2} points. Determine the maximum number of points for the given S_i.	DP	89
Connecting capitals	Given a tree. Determine the largest set of vertex disjoint edges.	DP	lec. 12
Race Tracks	Given a rectangular grid with obstacles, a start S and goal position. An agent A starts at S, has a x-y-velocity (a, b) $-3 \le a, b \le 3$. After each move A can adjust $a' = a + n_a, b' = b + n_b, n_a, n_b \in \{-1, 0, 1\}$. Determine the minimum number of moves necessary for A to reach the goal.	represent $(x, y, (a, b))$ as a node in a graph, BFS	29
Next Path	Given a directed Graph with vertices s and t, determine the length of the second shortest path.	BFS	115
Longest Path	Determine the length of the longest path in a tree.	DFS while recording tree height	37
Odd Route	Given a directed weighted graph G and two vertices s, t of G. Find the shortest path from s to t such that both its number of edges and total weight are odd.	Construct G', then Dijkstra	117
Build The Graph		MST and Dijkstra	42
Ants	Given an undirected graph G, and nodes S_i of subgraphs i. Build the subgraphs i by starting at S_i and do a uniform cost search until all nodes are explored. Find the cheapest path by combining all S_i (not G) from node a to b.	MST, then Dijkstra	38
Deleted Entries	Given a undirected graph G. Find k-coloring such that the graph stays connected after deleting all edges inside color classes.	MST and BFS to k-color	43
Monkey Island	Let G be a undirected graph (V, E) with costs on vertices. Let S be a subset of vertices, such that $\forall v \in V, \exists s \in S$ and there is a path from s to v. Find S that minimises $\sum_{s \in S} cost(s)$	Strongly connected components	81
Bridges	Given a connected undirected graph G, find the edges whose lacking would result in two connected components	Biconnected components	40
Deleted Entries Strike Back	Given a connected undirected graph G, find 3-coloring such that the graph stays biconnected after deleting all edges inside color classes.	Biconnected components, DFS-coloring using lowpoint map	103
Shy Programmers	Given a undirected graph G, decide if it is outerplanar, i.e., you can draw it in a plane so that all vertices lie on a circle and all edges are straight and don't intersect.	construct G' then planarity test	46
Buddy Selection	Given n students with c hobbies each. Determine if they can be matched in pairs such that each pair shares at least f hobbies.	Maximum matching size	50

Tiles	Given a two-dimensional array of where each enty is either marked as "obstacle" or "no obstacle". Determine the maximum number of "tiles" consisting of two neighboring cells fit in the array. No cell of a tile can be placed on an "obstacle" cell.	Maximum matching	102
Consecutive Constructions	Given a DAG. Determine the maximum number of edges that can be packed in vertex-disjoint paths.	Maximum matching	lec. 12
Satellites	Minimum vertex cover of unweighted bipartite undirect graph.	Compute maximum matching and use Konigs Theorem	51
Placing Knights	Determine the maximum number of chess-knights on a field with given size and obstacles.	Maximum independent set (n-maximum matching)	82
Algocoon	Given a directed graph G find the global minimum (directed) cut.	Global minimum cut: find source/sink that minimize flow. Start BFS from source to get the cut.	47
Coin Tossing	Given a sequence of m games, some with known result, some without and $s_1, \ldots, s_n$ the number of won games for each player. Determine if it is possible to assign outcomes to unknown games and get standings.	Maximum flow	55
Shopping Trip	Given a Graph and a set of nodes S and a node s. Determine if its possible to find $|S|$ edge disjoint paths from s to $S_i \forall S_i \in S$.	Maximum flow	85
Tetris	Given an interval of width w and a list B of intervals with coordinates $[a_i, b_i], 0 \le a_i, b_i \le w$ each. A "full line" L is a sequence of intervals of B such that $[0, w]$ is contained in the concatenation of l. Determine the maximum number of "full lines" F that can be constructed using the intervals of B while each interval can only be used once. Moreover each interval border in F is only allowed to occur once in F.	Maximum flow	95
Kingdom Defense	Given directed graph with maximum and minimum edge capacities and starting and minimum ending vertex budget. Determine if its possible to send have a flow in the network such that min and max edge capacities and the minimum ending vertex budget holds.	Model minimal edge capacities, then maximum flow problem	53
Sweepers	Given an undirected graph G, a list of start vertices S and a list of end vertices $E, |S| = |E|$. Determine if it is possible to find $|S|$ paths each starting at a vertex of S and end at a vertex of E, (where every start and end vertex can only be used in one path) such that all edges of the graph occur in exactly one path.	Maximum flow and eulerian path check	109
Knights	Given an undirected graph G, a list of start vertices S and a list of end vertices E. Determine the maximum number of edge and vertex disjoint paths from S to E.	Maximum flow	113

The Bracelet	Given a list B of tuples $t_i = (c_{i,1}, c_{i,2})$ and a set of colors $c_{i,j} \in C \subset \mathbb{R}$. Determine if it is possible to construct a strongly connected directed graph such that all tuples in B occur once as vertices, each tuple has exactly one ingoing and exacly one outgoing edge and the following edges are allowed: There can be an edge from t_i to t_k or from t_k to t_i if $c_{i,l_1} = c_{k,l_2}, l_1, l_2 \in \{1, 2\}$.	Eulerian path construction	111
Hit	Given a ray and some line segments. Determine if ray intersects any segment.	loop & test intersection	60
First Hit	Given a ray and some line segments, where does the ray first intersect a segment?	randomize segment order, loop while minimizing intersection	61
Antenna	Given a set of points, find the minimum enclosing circle.	Minimum enclosing circle	57
Almost Antenna	Given a set of points, find the minimum circle that encloses all but one point.	Minimum enclosing circle	58
Theev	Given a set of points P and the center of a circle C_1 in the plane. Determine the radius of two circles with centers C_1 and C_2 such that all points in P are covered.	Sorting and Min circle	87
Search Snippets	Given a set of numbers and their 1D-positions (many for a single number) in a sequence, find the minimum length of a sequence such that all numbers are occuring.	Sweepline	63
Search Snippets	Given a set of numbers and their 1D-positions (many for a single number) in a sequence, find the minimum length of a sequence such that all numbers are occuring.	Sweepline	63
Beach Bar	Given a set of points P in the line. Determine the integral position of a point x such that $|P_x|$ is maximum, where P_x are the points in P that are closer than 100 units away from x. Further, if multiple x are possible minimize the distance d_x to the farthest point in P_x. Output all optimal positions, $|P_x|$ and d_x	Sweepline	97
Hiking Maps	Given a polygonal path $p_0, \ldots, p_{m-1}$ and triangles $t_0, \ldots, t_{n-1}$, what is the minimum length of an interval $[b, e)$ contained in $[0, n)$ such that each leg $p_i p_{i+1}$ of the path is contained in at least one of $t_b, \ldots, t_{e-1}$	Fast triangle intersection test and Sweepline	72
Bistro	Given a set of points in the plane S and a set of points T. Determine for each point in T the distance to the closest point in S.	Delaunay triangulation on S	65
Graypes	Given a set of points G find the shortest distance between any pair of the points.	Delaunay triangulation	68
Collisions	Given a set P of points in the plane and a number d, how many points from P have at least one other point from P in distance $< d$?	Delaunay triangulation	77
Germs	Given a rectangle R and a set of points G inside the rectangle. For every point determine the minimal distance between either any other point or the rectangle boundary.	Delaunay triangulation, then find shortest incident triangulation edge	66

4

H1N1	How to move a disk without colliding with a given point set?	Precompute for each Delaunay-face the escape radius using DFS.	69
Cover	Given a rectangle and the center of n disks. Determine the minimal radius s.t. the rectangle is fully covered.	Delaunay triangulation (Voronoi)	99
Light the Stage	Given a set D of m disks (players) in the plane, a number h and a sequence $I_1, \ldots, I_n$ of n disks of radius h (lamps), denote by $A_i, i \in \{0, \ldots, i\}$ the set of all disks $d \in D$ such that d does not overlap with I_j, for any $j \in \{1, \ldots, i\}$. Determine the largest k such that A_k is not empty. Output A_k	Delaunay and binary search	105
Radiation 2	Given n healthy points and m cancer points in the plane, find a disk containing no healthy point but a maximum number of cancer points.	Delaunay and Sweepline along Voronoi Edge	??
Diet	Given the price of a food product and some nutritional constraints, calculate the cheapest diet.	Linear Programming	78
Inball	Given a cave $C = \{x \in \mathbb{R}^d | a_i^T x \le b_i, i = 1, \ldots, n\}$. Find the ball with the largest radius that is contained in C.	Linear Programming	80
Radiation	Given a set of points H and a set of points T in the plane. Determine the smallest number d, such that a polynomial of degree d can completely distinguish all points in H from all points T.	Linear programming	107
Missing Roads	Given a weighted undirected graph and a number k, determine the matching M consisting of k edges where the sum of the weights in M is minimized.	Linear programming.	120
Software update	Given a list of centers of disks in the plane. Determine the maximal sum of the radii such that no two disks overlap	Linear programming.	lec.13
Maximize It	Solve two quadratic programs.	Quadratic Programming	75
Portfolios	Given price, expected return and the covariance matrix of some assets and the minimum expected portfolio return, maximum portfolio cost and maximum portfolio variance of some investor. Determine if a portfolio for the investor exists.	Quadratic Programming	79
Stamp Exhibition	Given a set of stamps S, a set of lamps L and a set of segments W in the plane. Each $s_i \in S$ has a maximal allowed light intensity M_i. Determine if its possible to set the power p_i of the i-th lamp such that each stamp is illuminated with intensity > 1 but smaller M_i.	Quadratic Programming	93
Portfolios revisited	Given price, expected return and the covariance matrix of some assets and the maximum portfolio cost and maximum portfolio variance of some investor. Determine the maximum possible expected return.	Quadratic Programming and Binary search	91

Misc

Methods

- -
- Sorting
- Binary search
- Greedy
- Dynamic programming (Trigger: current computation does not depend on previous computation)
- BFS/DFS
- Shortest path/Dijkstra
- Minimum spanning tree
- (strongly) Connected components
- Biconnected components
- Planarity test
- Maximum matching (Biconnected: Max Vertex Cover/Maximum Independent Set)
- (global) Minimum cut
- Network flows
- Eulerian path
- Minimum enclosing circle
- Sweepline
- Delaunay triangulation
- Linear Programming / Quadratic Programming
- Backtracking

Potential problems: too slow

- `std::ios_base::sync_with_stdio(false);`
- check if there is possibility for a cache over testcases
- check if every non-primitive variable is called by reference
- find upper/lower bound
- try binary search or combination of binary search and linear search
- If CGAL: try randomization of input
- If QP/LP:solve LP with **linear** program solver
- If QP/LP:use nonnegative program where possible

Potential problems: error

- Assertion Error: off-by-one/memory
- CGAL: Think of infinite vertex
- Input size requirements

Useful Code Snippets

```cpp
#include <iostream>
#include <vector>
#include <sstream>
#include <string.h>
#include <stdio.h>

using namespace std;

/*
 * debug macro
 */
#define DEBUG
#ifdef DEBUG
#define D(x) x
#else
#define D(x)
#endif

/*
 * fast I/O
 */
ios_base::sync_with_stdio(false);

/*
 * output floating point numbers in fixed point notation
 * setprecision: number of digits after the decimal point
 */
std::cout<<std::setiosflags(std::ios::fixed) << std::setprecision(0)<< ceil_to_double(sqrt(CGAL::to_double(maxDist)))
         <<std::endl;

/*
 * number to string
 */
template <typename T>
    string NumberToString ( T Number )
    {
     ostringstream ss;
     ss << Number;
     return ss.str();
    }

/*
 * highest and lowest of orderable types
 */
#include <limits>
int max = numeric_limits<int>::max();
int min = numeric_limits<int>::min();

/*
 * enums
 */
enum enumeration_name {
  value1,
  value2,
  value3,
};

/*
 * power
 */
#include <math.h>
pow(7,3) //7^3

/*
 * sqrt
 */
#include <math.h>
double result = sqrt(1024.0);

#include <cstdlib>
double result = std::abs(-1024.0);

/*
 * sum vector contents
```

```
 75    */
 76   #include<numeric>
 77   std::accumulate(newTopChipPosition.begin(),
 78                                    newTopChipPosition.end(),0);
 79
 80   /*
 81    * sort array
 82    */
 83   #include<algorithm>
 84   std::sort(parasols.begin(), parasols.end());
 85
 86   /*
 87    * find in vector
 88    */
 89   std::find(aVector.begin(), aVector.end(), item)!=aVector.end()
 90
 91   /*
 92    * get index of max/min element in vector
 93    */
 94   #include <algorithm>
 95   std::vector<int> v(10,0);
 96   int a[] = {1,2,3,2};
 97   int maxIndex = std::max_element(v.begin(), v.end());
 98   int maxIndex2 = std::max_element(a, a+4);
 99
100   /*
101    * set intersection
102    */
103   std::vector<string> inters;
104   std::set_intersection(chars[i].begin(), chars[i].end(),
105       chars[j].begin(), chars[j].end(),
106       std::back_inserter(inters));
107
108   /*
109    * queue reverse order
110    */
111   priority_queue<int, vector<int>, greater<int>> Q;
112
113   /*
114    * comparator
115    */
116   struct faceComparator {
117       bool operator() (const Face_handle &x, const Face_handle &y) {
118           return (x->info() < y->info());
119       }
120   };
121   std::set<Face_handle, faceComparator> faces;
122
123   /*
124    * non-static comparator
125    */
126   struct CityComparator{
127       P mainCity;
128       bool operator()(P x, P y) {
129           K::FT d1 = S(mainCity, x).squared_length();
130           K::FT d2 = S(mainCity, y).squared_length();
131           return(d1 <d2);
132       }
133
134   };
135
136   /*
137    * Binary Search
138    */
139   int midpoint(int lowerBound, int upperBound) {
140       return lowerBound + (upperBound-lowerBound)/2;
141   }
142
143   int binary_search(int A[], int key, int imin, int imax)
144   {
145     // continue searching while [imin,imax] is not empty
146     while (imax >= imin)
147       {
148         // calculate the midpoint for roughly equal partition
149         int imid = midpoint(imin, imax);
150         if(A[imid] == key)
```

```
151        // key found at index imid
152        return imid;
153      // determine which subarray to search
154      else if (A[imid] < key)
155        // change min index to search upper subarray
156        imin = imid + 1;
157      else
158        // change max index to search lower subarray
159        imax = imid - 1;
160    }
161  // key was not found
162  return KEY_NOT_FOUND;
163 }



/*
167 /*
168  * BOOST
169  */
170 /*
171  * print graph
172  */
173 void printGraph(Graph g, WeightMap weight) {
174     graph_traits<Graph>::edge_iterator eiter, eiter_end;
175     for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
176         std::cout << source(*eiter, g) << " <--> " << target(*eiter, g)
177           << " with weight of " << weight[*eiter]
178           << std::endl;
179     }
180 }

182 /*
183  * shortest path BFS
184  */
185 int shortestPath(const Graph &g, int x, int y) {
186         //BFS
187         int n=num_vertices(g);
188         vector<bool> visited(n, false);
189         vector<int> distance(n, -1);
190         std::queue<int> bfs_queue;
191
192         vector<int> bfs_num(n);
193         bfs_queue.push(x);
194         visited[x] = true;
195         distance[x] = 0;
196         while (!bfs_queue.empty()) {
197         int v = bfs_queue.front();
198         bfs_queue.pop();
199         //cerr << "visit   : " << v << "\n";
200         GraphTraits::out_edge_iterator e, e_end;
201         for (tie(e, e_end) = out_edges(v, g); e != e_end; ++e) {
202             Vertex t = target(*e, g);
203             if (!visited[t]) {
204             visited[t] = true;
205             distance[t] = distance[v] + 1;
206             //cerr << "enqueue " << t << "\n";
207             if(t==y) return distance[t];
208             bfs_queue.push(t);
209             }
210         }
211     }
212     return 0;
213 }

215 /*
216  * non-recursive DFS
217  */
218 int main ()
219 {
220     int n;
221     vector<vector<int> > edges;
222     read_graph(n, edges);
223     /* 8< */
224     vector<bool> visited(n, false);
225     vector<int> dfs_stack;
226     vector<int> dfs_neighbor_pos;
```

```
227    int next_num = 0;
228    vector<int> dfs_num(n);
229    for (int u = 0; u < n; u++) {
230    if (visited[u])
231        continue;
232    dfs_stack.push_back(u);
233    dfs_neighbor_pos.push_back(0);
234    visited[u] = true;
235    dfs_num[u] = next_num++;
236    cerr << "visit␣" << dfs_num[u] << ":␣" << u << "\n";
237    while (!dfs_stack.empty()) {
238        int v = dfs_stack.back();
239        int i = dfs_neighbor_pos.back();
240        dfs_stack.pop_back();
241        dfs_neighbor_pos.pop_back();
242        vector<int>& neighbors = edges[v];
243        for (; i < (int)neighbors.size(); i++) {
244        int w = neighbors[i];
245        if (!visited[w]) {
246            /* defer looking at v */
247            dfs_stack.push_back(v);
248            dfs_neighbor_pos.push_back(i+1);
249            /* look at w next */
250            visited[w] = true;
251            dfs_num[w] = next_num++;
252            dfs_stack.push_back(w);
253            dfs_neighbor_pos.push_back(0);
254            cerr << "visit␣" << dfs_num[w] << ":␣" << w << "\n";
255            break;
256        }
257        /* if we fall off this loop (as opposed to break), we
258         * are done looking at v */
259        }
260    }
261    }
262    /* >8 */
263 }
264
265 /*
266  * Boost named parameters and visitor
267  */
268
269 vector<int> discoverTimeToVertex;
270
271 // Custom visitor used to record DFS order.
272 struct CustomVisitor : public default_dfs_visitor {
273     void discover_vertex(int u, const Graph& G) {
274         discoverTimeToVertex.push_back(u);
275     }
276 };
277
278 vector<Vertex> low(numVertices);
279 vector<Vertex> vertexToDiscoverTime(numVertices);
280 vector<Vertex> predecessor(numVertices);
281 discoverTimeToVertex.clear();
282 size_t num_comps
283     = biconnected_components(g,component,
284                              lowpoint_map(&low[0])
285                              .discover_time_map(&vertexToDiscoverTime[0])
286                              .predecessor_map(&predecessor[0])
287                              .visitor(CustomVisitor()));
288
289 /*
290  * Boost: defining a custom property
291  */
292 namespace boost {
293     enum edge_info_t { edge_info = 219 }; // A unique ID.
294     BOOST_INSTALL_PROPERTY(edge, info);
295 }
296
297 struct EdgeInfo {
298 ...
299 }
300
301 typedef adjacency_list<vecS, vecS, directedS,
302     no_property, // vertex property, none this time
```

```
303      property<edge_info_t, EdgeInfo> //edge property, could also use a predefined property like edge_name_t and define ↙
              ↳ it with integers
304      > Graph;
305  typedef property_map<Graph, edge_info_t>::type InfoMap;
306  ...
307  InfoMap info_map = get(edge_info, G);
308  info_map[e] = ...
309
310  /*
311   * CGAL: add property to a point
312   */
313  template <typename I>
314  struct MyP : public K::Point_2 {
315      MyP(const K::FT& x, const K::FT& y, const I& i_) : K::Point_2(x,y), i(i_) {}
316      I i;
317  };
318  typedef MyP<std::pair<std::size_t,K::FT> > Participant;
319  Participant(x,y,make_pair(i,h));
320
321  /*
322   * CGAL: Voronoi dual
323   */
324  // process all Voronoi vertices
325  for (Face_iterator f = t.finite_faces_begin(); f != t.finite_faces_end(); ++f) {
326      K::Point_2 p = t.dual(f);
327      ...
328  }
329  // process all Voronoi edges
330  for(Edge_iterator e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
331      CGAL::Object o = t.dual(e);
332      // o can be a segment, a ray or a line ...
333      ...
334  }
335
336  /*
337   * CGAL: Enhancing faces/vertices
338   */
339  //Can use maps for {Face,Vertices,Edges(with some additional work)}-handles
340  //or:
341  #include <CGAL/Triangulation_face_base_with_info_2.h>
342  enum Color { Black = 0, White = 1, Red = 2};
343  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
344  typedef CGAL::Triangulation_vertex_base_2<K> Vb;
345  //New face class, vertex class stays the same. With info parameter. Here each face gets a color
346  typedef CGAL::Triangulation_face_base_with_info_2<Color,K> Fb;
347  //change the underlying triangulation structure
348  typedef CGAL::Triangulation_data_structure_2<Vb,Fb> Tds;
349  typedef CGAL::Delaunay_triangulation_2<K,Tds> Triangulation;
350
351  Triangulation t;
352
353  // color all infinite faces black
354  Triangulation::Face_circulator f = t.incident_faces(t.infinite_vertex());
355  do {
356      f->info() = Black;
357  } while (++f != t.incident_faces(t.infinite_vertex()));
358
359
360  /*
361   * QP Debugging
362   */
363  cout<<qp.get_n()<<std::endl;
364  cout<<qp.get_m()<<std::endl;
365  CGAL::print_quadratic_program(std::cerr, qp, "qp");
366
367  CGAL::Quadratic_program_options options;
368  options.set_pricing_strategy(CGAL::QP_BLAND);
369  Solution s = CGAL::solve_linear_program(qp, ET(), options);
```

Checking Change

```cpp
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <stdio.h>
#include <string.h>

using namespace std;

/*
 * c[] coin denominations
 * ci |c[]|
 * n change
 */
int *change(int c[], int ci, int n) {
    int *C = new int[n+1];
    C[0] = 0;
    for(int p=1; p <= n; p++) {
        int min = numeric_limits<int>::max() - 1;
        for(int i=0; i < ci; i++) {
            if(c[i] <= p) {
                if (1 + C[p-c[i]] < min) {
                    min = 1 + C[p-c[i]];
                }
            }
        }
        C[p] = min;
    }
    return C;
}

int changeRec(int c[], int ci, int n) {
    if(n==0) {
        return 0;
    } else {
        int min = numeric_limits<int>::max() - 1;
        for(int i=ci-1; i >= 0; i--) {
            if(c[i] <= n) {
                int tryChange = changeRec(c, ci, n-c[i]);
                if (1 + tryChange < min) {
                    min = 1 + tryChange;
                }
            }
        }
        return min;
    }
}


//C[n] initialised with -1
int changeFast(vector<int> &C, int c[], int ci, int n) {
    if(n==0) {
        C[n] = 0;
    } else if(C[n] == -1) {
        int min = numeric_limits<int>::max() - 1;
        for(int i=ci-1; i >= 0; i--) {
            if(c[i] <= n) {
                int tryChange = changeFast(C, c, ci, n-c[i]);
                if (1 + tryChange < min) {
                    min = 1 + tryChange;
                }
            }
        }
        C[n] = min;
    }
    return C[n];
}

void reverse(int *word, int len)
{
    int temp;
    for (int i=0;i<len/2;i++)
    {
```

```
74              temp=word[i];
75              word[i]=word[len-i-1];
76              word[len-i-1]=temp;
77          }
78  }
79
80  int *changeFaster(int c[],int n, const int A, int **C) {
81      reverse(c, n);
82      for(int j=0; j<A; j++) {
83          C[n-1][j] =j;
84      }
85      for(int i=n-2; i>=0; i--) {
86          for(int j=0; j<A; j++) {
87              if(c[i] > j || C[i+1][j] < 1 + C[i][j-c[i]]) {
88                  C[i][j] = C[i+1][j];
89              } else {
90                  C[i][j] = 1 + C[i][j-c[i]];
91              }
92          }
93      }
94  }
95
96
97  int count( int S[], int m, int n )
98  {
99      // table[i] will be storing the number of solutions for
100     // value i. We need n+1 rows as the table is consturcted
101     // in bottom up manner using the base case (n = 0)
102     int table[n+1];
103
104     // Initialize all table values as 0
105     memset(table, 0, sizeof(table));
106
107     // Base case (If given value is 0)
108     table[0] = 0;
109
110     // Pick all coins one by one and update the table[] values
111     // after the index greater than or equal to the value of the
112     // picked coin
113     for(int i=0; i<m; i++)
114         for(int j=S[i]; j<=n; j++)
115             table[j] += table[j-S[i]];
116
117     return table[n];
118 }
119
120 // fill vector up such that all indices n - former C.size()  are -1
121 void fillVector(vector<int> &C, int n) {
122     for(int i=C.size(); i < n+1; i++) {
123         C.push_back(-1);
124     }
125 }
126
127 void printVector(vector<int> &C) {
128     for(int i=0; i < C.size(); i++) {
129         cout << C[i] << "␣";
130     }
131     cout << endl;
132 }
133
134 int main() {
135     cin.sync_with_stdio(false);
136     cout.sync_with_stdio(false);
137     int testCases;
138     cin >> testCases;
139
140     for(int i=0; i < testCases; i++) {
141         int ci, mi;
142         cin >> ci >> mi;
143         int c[ci];
144         for(int j=0; j < ci; j++) {
145             int coinValue;
146             cin >> coinValue;
147             c[j] = coinValue;
148         }
149         vector<int> C;
```

```cpp
        for(int k=0; k < mi; k++) {
            int n;
            cin >> n;
            fillVector(C, n);

            changeFast(C, c, ci, n);
            int numCoins = C[n];
            if(numCoins == numeric_limits<int>::max() - 1) {
                cout << "not possible" << endl;
            } else {
                cout << numCoins << endl;
            }
            // delete C;
        }
    }
}
```

Dominoes

```cpp
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <algorithm>

using namespace std;

int main() {
    std::ios_base::sync_with_stdio(false);
    int testCases;
    cin >> testCases;

    for(int k=0; k < testCases; k++) {
        int n;
        cin >> n;
        int numFall = 0;

        int h0; //height of the first dominoe which is surely falling
        cin >> h0;
        int accH = h0; //accH is the currently highest falling dominoe

        for(numFall=1; numFall < n; numFall++) {
            int h;
            cin >> h;
            accH= accH-1;
            if(accH > 0) {
                accH = max(accH, h);
            } else {
                break; //this dominoe does not fall
            }
        }
        //read rest of the file
        for(int r=numFall+1; r<n;r++) {
            int h;
            cin >> h;
        }

        cout << numFall << endl;
    }
}
```

Shelves

```cpp
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <math.h>

using namespace std;

struct triple {
    int first;
    int second;
    int third;
};

void computeSlow(triple &result, int lTotal, int lM, int lN) {
        for(int uncovered = 0; uncovered < lTotal; uncovered++) {
            int rest = lTotal - uncovered;
            // cout << uncovered << " rest:" << rest << " ";
            for(int numN = rest/lN; numN >= 0; numN--) {
                rest = lTotal - uncovered - numN*lN;
                // cout << " numN: " << numN << " rest:"<< rest << " ";
                for(int numM = rest/lM; numM >= 0; numM--) {
                    rest = lTotal - uncovered -numN*lN - numM*lM;
                    // cout << " numM: " << numM << " rest:"<< rest << " "<< endl;
                    if(rest==0) {
                        result.first=numM;
                        result.second=numN;
                        result.third=uncovered;

                        return;
                    }
                }
            }
        }
        result.first = 0;
        result.second = 0;
        result.third = lTotal;
}

void computeFast(triple &result, int lTotal, int lM, int lN) {
    int rest = lTotal;
    int a, b = 0;
    for(int bp=0; bp<=lTotal/lN; bp++) {
        int ap = (lTotal-bp*lN)/lM;
        int restp = lTotal - ap*lM - bp*lN;
        if(restp <= rest) {
            rest = restp;
            b = bp;
            a = ap;
        }
    }
    result.first = a;
    result.second = b;
    result.third = rest;
    return;
}

// runtime sqrt(lTotal)
void computeVeryFast(triple &result, int lTotal, int lM, int lN) {
    int rest = lTotal;
    int a, b = 0;
    if(lN < sqrt(lTotal)) { // lTotal is big
        for(int ap=lN-1; ap >=0; ap--) {
            int bp = (lTotal-ap*lM)/lN;
            int restp = lTotal - ap*lM - bp*lN;
            if(restp <= rest) {
                rest = restp;
                b = bp;
                a = ap;
            }
        }

    } else { //lN > sqrt(lTotal) => loop iterates < lTotal/sqrt(lTotal) < sqrt(lTotal)
```

```cpp
        for(int bp=0; bp<=lTotal/lN; bp++) {
            int ap = (lTotal-bp*lN)/lM;
            int restp = lTotal - ap*lM - bp*lN;
            if(restp <= rest) {
                rest = restp;
                b = bp;
                a = ap;
            }
        }
    }
    result.first = a;
    result.second = b;
    result.third = rest;
    return;
}


int main() {
    int testCases;

    cin >> testCases;

    for(int i=0; i < testCases; i++) {
        int lTotal, lM, lN, nM, nN, uncovered;
        cin >> lTotal >> lM >> lN;
        triple result;
        computeVeryFast(result, lTotal, lM, lN);
        cout << result.first << "␣" << result.second << "␣" << result.third << endl;
    }

    return 0;
}
```

Even Pairs

```cpp
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <math.h>

using namespace std;

struct triple {
    int first;
    int second;
    int third;
};

void computeSlow(triple &result, int lTotal, int lM, int lN) {
        for(int uncovered = 0; uncovered < lTotal; uncovered++) {
            int rest = lTotal - uncovered;
            // cout << uncovered << " rest:" << rest << " ";
            for(int numN = rest/lN; numN >= 0; numN--) {
                rest = lTotal - uncovered - numN*lN;
                // cout << " numN: " << numN << " rest:"<< rest << " ";
                for(int numM = rest/lM; numM >= 0; numM--) {
                    rest = lTotal - uncovered -numN*lN - numM*lM;
                    // cout << " numM: " << numM << " rest:"<< rest << " "<< endl;
                    if(rest==0) {
                        result.first=numM;
                        result.second=numN;
                        result.third=uncovered;

                        return;
                    }
                }
            }
        }
        result.first = 0;
        result.second = 0;
        result.third = lTotal;
}

void computeFast(triple &result, int lTotal, int lM, int lN) {
    int rest = lTotal;
    int a, b = 0;
    for(int bp=0; bp<=lTotal/lN; bp++) {
        int ap = (lTotal-bp*lN)/lM;
        int restp = lTotal - ap*lM - bp*lN;
        if(restp <= rest) {
            rest = restp;
            b = bp;
            a = ap;
        }
    }
    result.first = a;
    result.second = b;
    result.third = rest;
    return;
}

// runtime sqrt(lTotal)
void computeVeryFast(triple &result, int lTotal, int lM, int lN) {
    int rest = lTotal;
    int a, b = 0;
    if(lN < sqrt(lTotal)) { // lTotal is big
        for(int ap=lN-1; ap >=0; ap--) {
            int bp = (lTotal-ap*lM)/lN;
            int restp = lTotal - ap*lM - bp*lN;
            if(restp <= rest) {
                rest = restp;
                b = bp;
                a = ap;
            }
        }

    } else { //lN > sqrt(lTotal) => loop iterates < lTotal/sqrt(lTotal) < sqrt(lTotal)
```

```
74        for(int bp=0; bp<=lTotal/lN; bp++) {
75            int ap = (lTotal-bp*lN)/lM;
76            int restp = lTotal - ap*lM - bp*lN;
77            if(restp <= rest) {
78                rest = restp;
79                b = bp;
80                a = ap;
81            }
82        }
83    }
84    result.first = a;
85    result.second = b;
86    result.third = rest;
87    return;
88 }


91 int main() {
92     int testCases;
93
94     cin >> testCases;
95
96     for(int i=0; i < testCases; i++) {
97         int lTotal, lM, lN, nM, nN, uncovered;
98         cin >> lTotal >> lM >> lN;
99         triple result;
100         computeVeryFast(result, lTotal, lM, lN);
101         cout << result.first << "␣" << result.second << "␣" << result.third << endl;
102     }
103
104     return 0;
105 }
```

Aliens

```cpp
#include <iostream>
#include <limits>
#include <set>
#include <vector>
#include <algorithm>

using namespace std;

int numSuperior(vector<pair<int, int> > &interval, int numHumans) {
    sort(interval.begin(), interval.end());
    int rightmost = 0;
    for(int i=0; i<interval.size(); i++) {
        if(interval[i].first > rightmost + 1)
            return 0;
        else
            rightmost = max(rightmost, interval[i].second);
    }
    if(rightmost < numHumans)
        return 0;

    rightmost = 0;
    int rightMostI = -1;
    int beforeLeft = 0;
    int numSuperiorAliens;
    int beforeI;
    vector<bool> isSuperior(interval.size(), true);

    for(int i=0; i<interval.size(); i++) {
        if(interval[i].first != beforeLeft) {
            if(interval[i].second <= rightmost) {
                isSuperior[i] = false;
                // cout << "right intervall smaller equal rightmost" << endl;
            }
        } else {
            if(interval[i].second < rightmost) {
                isSuperior[i] = false;
                // cout << "right intervall smaller equal rightmost" << endl;
            } else if(interval[i].second == rightmost) {
                isSuperior[i] = false;
                isSuperior[beforeI] = false;
                //cout << "right intervall equal rightmost";
            } else {
                isSuperior[beforeI] = false;
            }
        }
        if(rightmost <= interval[i].second) {
            rightmost = interval[i].second;
        }
        beforeI = i;
        beforeLeft = interval[i].first;
    }
    numSuperiorAliens=0;
    for(int i=0; i<isSuperior.size(); i++) {
        // cout << "alien: " << interval[i].first << " " << interval[i].second << " ";
        if(isSuperior[i]) {
            numSuperiorAliens++;
            // cout << "is superior!!";
        }
        // cout << endl;
    }

    return numSuperiorAliens;
}

int main() {
    ios_base::sync_with_stdio(false);
    int testCases;
    cin >> testCases;

    for(int t=0; t < testCases; t++) {
        int numHumans, numAliens;
        cin >> numAliens >> numHumans;
```

```
74          vector<pair<int, int> > interval;
75          for(int i=0; i < numAliens; i++) {
76              int intervalLeft;
77              int intervalRight;
78              cin >> intervalLeft >> intervalRight;
79              if(intervalLeft != 0 && intervalRight!=0) {
80                  pair<int, int> anInterval = pair<int, int>(intervalLeft, intervalRight);
81                  interval.push_back(anInterval);
82              }
83          }
84          cout << numSuperior(interval, numHumans) << endl;
85      }
86  }
```

Boats

```
1   #include <iostream>
2   #include <limits>
3   #include <set>
4   #include <vector>
5   #include <algorithm>
6   using namespace std;
7
8   class boat {
9       public:
10          int length;
11          int position;
12          boat(int len, int pos) {
13              length = len;
14              position = pos;
15          }
16          bool operator< (const boat& other) const {
17              return position < other.position;
18          }
19          bool operator== (const boat& other) const {
20              return length == other.length
21                  && position == other.position;
22          }
23  };
24
25  int numBoats(int numWizards, set<boat> &boats) {
26      int curPos = numeric_limits<int>::min();
27      int numBoats = 0;
28      int i = 0;
29      while(boats.size() > 0 ) {
30          set<boat>::iterator it;
31          for(it=boats.begin(); (*it).position < curPos; it++) {
32              boats.erase(it);
33          }
34          // if(it==boats.end()) {
35          //     return numBoats;
36          // }
37          int bestEnd = numeric_limits<int>::max();
38          // boat bestBoat(bestEnd, bestEnd);
39          const boat *bestBoat = NULL;
40          set<boat>::iterator bestBoatIterator;
41          for(it=boats.begin();it!=boats.end(); it++) {
42              int begin = ((*it).position - (*it).length > curPos) ? ((*it).position-(*it).length) : curPos;
43              int end = begin + (*it).length;
44              if(end < bestEnd) {
45                  bestEnd = end;
46                  bestBoat = &(*it);
47                  bestBoatIterator = it;
48              }
49              if((*it).position > bestEnd) {
50                  break;
51              }
52          }
53          curPos=bestEnd;
54          bestBoat = bestBoat;
55          numBoats+=1;
56          i++;
57          // cout << "boat " << i << " with p:" << bestBoat->position << " l:" << bestBoat->length << " , end is at " ↙
                    ↘ << curPos << endl;
58          boats.erase(bestBoatIterator);
59      }
60      return numBoats;
61
62  }
63
64  int numBoatsFast(int numWizards, vector<boat> &boats) {
65      int i = 0;
66      sort(boats.begin(), boats.end());
67      // int curPos = (*boats.begin()).position;
68      int curPos = numeric_limits<int>::min();
69      // i++;
70      // int numBoats = 1;
71      int numBoats = 0;
72      while(i < numWizards) {
```

```cpp
        for(;i < numWizards && boats[i].position < curPos; i++);
        if(i==numWizards) break;
        int newMaxPos = numeric_limits<int>::max();
        int j = i;
        while(j < numWizards && boats[j].position < newMaxPos){
            int proposed = max(boats[j].position, curPos + boats[j].length);
            if(proposed < newMaxPos) {
                newMaxPos = proposed;
            }
            j++;
        }
        curPos = newMaxPos;
        // cout << "boat " << j-1 << " with p:" << boats[j-1].position << " l:" << boats[j-1].length << " , end is at ↙
            ↘ " << curPos << endl;
        numBoats++;
        i+=(j-i);
        // i++;
    }
    return numBoats;
}

int main() {
    ios_base::sync_with_stdio(false);
    int testCases;
    cin >> testCases;

    for(int i=0; i < testCases; i++) {
        int numWizards;
        cin >> numWizards;
        vector<boat> boats (numWizards, boat(0,0));
        for(int j=0; j < numWizards; j++) {
            int length;
            int position;
            cin >> length >> position;
            boats[j] = boat(length, position);
        }
        cout << numBoatsFast(numWizards, boats) << endl;
    }
}
```

False Coin

```cpp
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <stdio.h>
#include <string.h>
#include <sstream>

using namespace std;

bool *initializeArray(bool init, int n) {
    bool *array = new bool[n];
    for(int i=0; i < n; i++) {
        array[i] = init;
    }
    return array;
}

string falseCoin(bool suspicious[], int numCoins) {
    int fCoin = -1;
    for(int i=0; i<numCoins; i++) {
        if(suspicious[i]) {
            if(fCoin!=-1) {
                return "0";
            } else {
                fCoin = i+1;
            }
        }
    }
    return static_cast<ostringstream*>( &(ostringstream() << fCoin) )->str();
}

int main() {
    cin.sync_with_stdio(false);
    int testCases;
    cin >> testCases;

    for(int i=0; i < testCases; i++) {
        // char newline;
        // cin >> newline;

        int numCoins, numWeighings;
        cin >> numCoins >> numWeighings;
        bool *suspicious = initializeArray(true, numCoins);

        for(int j=0; j < numWeighings; j++) {
            int amountInPan;
            cin >> amountInPan;
            bool *weighingCoins = initializeArray(false, numCoins);
            // read currently weighted coins
            for(int k=0; k < 2*amountInPan; k++) {
                int currentCoin;
                cin >> currentCoin;
                weighingCoins[currentCoin-1] = true;
            }

            char outcome;
            cin >> outcome;
            if(outcome == '=') {
                //every weighing coin true, is false in coins
                for(int k=0; k<numCoins; k++) {
                    if(weighingCoins[k]) {
                        suspicious[k] = false;
                    }
                }
            } else {
                //set all others to false
                for(int k=0; k<numCoins; k++) {
                    if(!weighingCoins[k]) {
                        suspicious[k] = false;
                    }
                }
            }
```

```
74            delete weighingCoins;
75        }
76
77        cout << falseCoin(suspicious, numCoins) << endl;
78        delete suspicious;
79    }
80 }
```

Formula One

```
1    #include <iostream>
2    #include <iterator>
3    #include <vector>
4
5    using namespace std;
6
7    void swap(int* array, int i, int j) {
8        int tmp;
9        tmp = array[j];
10       array[j] = array[i];
11       array[i] = tmp;
12   }
13
14   int overpasses;
15
16   //! \brief Merges two sorted vectors into one sorted vector
17   //! \param left A sorted vector of integers
18   //! \param right A sorted vector of integers
19   //! \return A sorted vector that is the result of merging two sorted
20   //! vectors.
21   vector<int> merge(const vector<int>& left, const vector<int>& right)
22   {
23       // Fill the resultant vector with sorted results from both vectors
24       vector<int> result;
25       unsigned left_it = 0, right_it = 0;
26
27       while(left_it < left.size() && right_it < right.size())
28       {
29           // If the left value is smaller than the right it goes next
30           // into the resultant vector
31           if(left[left_it] < right[right_it])
32           {
33               result.push_back(left[left_it]);
34               left_it++;
35               overpasses= (overpasses + right_it) % 10000;
36           }
37           else
38           {
39               result.push_back(right[right_it]);
40               right_it++;
41           }
42       }
43
44       // Push the remaining data from both vectors onto the resultant
45       while(left_it < left.size())
46       {
47           result.push_back(left[left_it]);
48           overpasses= (overpasses + right_it) % 10000;
49           left_it++;
50       }
51
52       while(right_it < right.size())
53       {
54           result.push_back(right[right_it]);
55           right_it++;
56       }
57
58       return result;
59   }
60   //! \brief Performs a recursive merge sort on the given vector
61   //! \param vec The vector to be sorted using the merge sort
62   //! \return The sorted resultant vector after merge sort is
63   //! complete.
64   vector<int> merge_sort(vector<int>& vec)
65   {
66       // Termination condition: List is completely sorted if it
67       // only contains a single element.
68       if(vec.size() == 1)
69       {
70           return vec;
71       }
72
73       // Determine the location of the middle element in the vector
```

```cpp
        std::vector<int>::iterator middle = vec.begin() + (vec.size() / 2);

        vector<int> left(vec.begin(), middle);
        vector<int> right(middle, vec.end());

        // Perform a merge sort on the two smaller vectors
        left = merge_sort(left);
        right = merge_sort(right);

        return merge(left, right);
}

/*
 * return number of bubble sort steps necessary
 */
int bubbleSort(int* order, int numRacers) {
        bool swapped = false;
        int numSwaps = 0;
        int n = numRacers;

        do {
                int newN = 0;
                swapped = false;
                for(int i=1; i<n; i++) {
                        if(order[i-1] > order[i]) {
                                swap(order, i-1, i);
                                swapped = true;
                                newN = i;
                                numSwaps++;
                        }
                }
                n = newN;
        } while(!n==0);

        return numSwaps;
}

// uses bubbleSort to control for correctness of the merge sort overpasses counting
void controlIfCorrect(vector<int> order) {
                int orderArray[order.size()];
                copy(order.begin(), order.end(), orderArray);

                int overpBubble = bubbleSort(orderArray, order.size());

                if(overpBubble == overpasses) {
                        // cout << "ok: " << overpasses << endl;
                } else {
                        cout << "different␣answer:␣bubble:␣" << overpBubble << "merge:␣" << overpasses << endl;
                        cout << "testset:␣" << endl;
                        copy(order.begin(), order.end(), ostream_iterator<int>(cout, "␣"));
                }
}

int main() {
        cin.sync_with_stdio(false);
        int testCases;
        cin >> testCases;

        for(int i=0; i < testCases; i++) {
                int numRacers;
                cin >> numRacers;

                // int order[numRacers];
                vector<int> order;
                for(int j=0; j < numRacers; j++) {
                        int racer;
                        cin >> racer;
                        order.push_back(racer);
                }

                overpasses = 0;
                merge_sort(order);

                // controlIfCorrect(order);

                cout << overpasses % 10000 << endl;
```

```
150
151        }
152    }
```

Race Tracks

```cpp
#include <iostream>
#include <limits>
#include <vector>
#include <queue>
#include <set>
#include <sstream>
#include <cstdlib>

using namespace std;

template <typename T>
    string NumberToString ( T Number )
    {
     ostringstream ss;
     ss << Number;
     return ss.str();
    }

class point {
    private:
        int x;
        int y;

    public:
        point(int X, int Y) {
            x = X;
            y = Y;
        }
        point(const point &p) {
            x = p.x;
            y = p.y;
        }
        bool operator== (const point &other) const {
            return (x == other.getX() &&
                    y == other.getY());
        }
        bool operator!= (const point &other) const {
            return !(*this == other);
        }
        bool operator< (const point &other) const {
            if(x < other.x) {
                return true;
            } else if((x == other.x) && (y < other.y)) {
                return true;
            }
            return false;
        }
        int getX() const { return x; }
        int getY() const { return y; }
        string str() const { return "(" + NumberToString(x) + "␣" + NumberToString(y) + ")"; }
};

class vertex {
    private:
        point position;
        point velocity;
        int distance;
    public:
        vertex(point &i_position, point &i_velocity) :
                position(0,0), velocity(0,0) {
            position = i_position;
            velocity = i_velocity;
            distance = numeric_limits<int>::max();
        }
        bool operator== (const vertex &other) const {
            return (position == other.position
                    && velocity == other.velocity
                );
        }
        bool operator!= (const vertex &other) const {
            return !(*this == other);
        }
        bool operator < (const vertex& other) const {
```

```
74            if(position < other.getPosition()) {
75                    return true;
76                } else if(position == other.position) {
77                    return (velocity < other.velocity);
78                }
79            return false;
80        }
81        point getPosition() const { return position; }
82        point getVelocity() const { return velocity; }
83        int getDistance() const { return distance; }
84        void setDistance(int distance_arg) { distance = distance_arg; return;}
85        string str() const { return "(" + NumberToString(distance) + "," + position.str() + "," + velocity.str() + ↙
              ↳ ")"; }
86 };
87
88 struct OrderByDistance {
89     bool operator() (const vertex& v1, const vertex& v2) const {
90         return (v1.getDistance() > v2.getDistance());
91     }
92 };
93
94 struct Obstacle {
95     int x1;
96     int x2;
97     int y1;
98     int y2;
99 };
100
101 bool validPosition(point p, int width, int height, Obstacle obstacles[], int numObstacles) {
102     if(!((0<=p.getX()) && (p.getX()<width) && (0<=p.getY()) && (p.getY()<height))) {
103         return false;
104     }
105     for(int i=0; i < numObstacles; i++) {
106         Obstacle o = obstacles[i];
107         if((o.x1 <= p.getX()) && (p.getX() <= o.x2) &&
108                 (o.y1 <= p.getY()) && (p.getY() <= o.y2)) {
109             // cout << o.x1 << " " << p.getX() << " " << o.x2 << endl;
110             // cout << o.y1 << " " << p.getY() << " " << o.y2 << endl;
111             return false;
112
113         }
114     }
115     return true;
116 }
117
118 void printQueue(priority_queue<vertex,vector<vertex>,OrderByDistance> queue) {
119     cout << "Queue:␣";
120     while(queue.size() > 0) {
121         cout << queue.top().str() << ","; queue.pop();
122     }
123     cout << endl;
124 }
125
126 void printSet(set<vertex> visit) {
127     cout << "Visited:␣";
128     while(visit.size() > 0) {
129         set<vertex>::iterator firstEle = visit.begin();
130         cout << (*firstEle).str() << ","; visit.erase(firstEle);
131     }
132     cout << endl;
133 }
134
135 int vertexToHash(vertex v, int width, int height) {
136     // x y xa ya
137     return (v.getPosition().getX() * (height * 7 * 7)
138             + v.getPosition().getY() * 7 * 7
139             + (v.getVelocity().getX()+3) * 7
140             + (v.getVelocity().getY()+3));
141 }
142
143 vertex hashToVertex(int hash, int width, int height) {
144     int ya = (hash%7)-3;
145     hash = hash/7;
146     int xa = (hash%7)-3;
147     hash = hash/7;
148     int y = hash%height;
```

```
149        hash = hash/height;
150        int x = hash;
151        point pos =point(x,y);
152        point vel = point(xa,ya);
153        return vertex(pos,vel);
154    }
155
156    void printVisited(bool *visit, int width, int height) {
157        int n = width*height*7*7;
158        cout << "Visited:␣";
159        for(int i=0; i < n; i++) {
160            if(visit[i]) {
161                cout << hashToVertex(i, width, height).str() << ",";
162            }
163        }
164    }
165
166
167    int shortestPath(point start, point end, int width, int height, Obstacle obstacles[], int numObstacles) {
168        // set<vertex> visit;
169        int numVisit = width*height*7*7;
170        bool visit[numVisit];
171        for(int v=0; v < numVisit; v++) {
172            visit[v] = false;
173        }
174        // flags to set nodes state to      visited
175        priority_queue<vertex,vector<vertex>,OrderByDistance> queue ;
176
177        point zero(0,0);
178        vertex startVertex(start, zero);
179        startVertex.setDistance(-1);
180        queue.push(startVertex);
181
182        while (!queue.empty()) {
183            vertex cur = queue.top(); queue.pop();
184            if(visit[vertexToHash(cur,width,height)])
185                continue;
186
187            // printQueue(queue);
188            // cout<<"current vertex: " << cur.str() << endl;
189            // printVisited(visit, width, height);
190            visit[vertexToHash(cur,width,height)] = true;
191            if(cur.getPosition() == end) {
192                return cur.getDistance();
193            }
194
195            point newPosition(cur.getPosition().getX() + cur.getVelocity().getX(),
196                              cur.getPosition().getY() + cur.getVelocity().getY());
197
198            if(validPosition(newPosition, width, height, obstacles, numObstacles)) {
199                for(int xa=-1; xa <=1; xa++) {
200                    for(int ya=-1; ya <= 1; ya++) {
201                        // if(xa == 0 && ya == 0 && ) {
202                        //     continue;
203                        // }
204                        if(abs(cur.getVelocity().getX() + xa) > 3
205                                || abs(cur.getVelocity().getY() + ya) > 3) {
206                            continue;
207                        }
208
209                        point newVelocity(cur.getVelocity().getX()+xa,
210                                                cur.getVelocity().getY()+ya);
211                        vertex to = vertex(newPosition, newVelocity);
212                        int altDistance = cur.getDistance() + 1;
213                        if((!visit[vertexToHash(to,width,height)]) //if not in visited set
214                            && altDistance < to.getDistance() ) {
215                                to.setDistance(altDistance);
216                                queue.push(to); //push again to the queue? not bad I think
217                        }
218
219                    }
220
221                }
222
223            }
224        }
```

```
225        return -2;
226    }
227
228
229    int main() {
230        cin.sync_with_stdio(false);
231        int testCases;
232        cin >> testCases;
233
234        for(int i=0; i < testCases; i++) {
235
236            int width, height;
237            cin >> width >> height;
238            int sx, sy, ex, ey;
239            cin >> sx >> sy >> ex >> ey;
240            point start(sx,sy), end(ex,ey);
241
242            int numObstacles;
243            cin >> numObstacles;
244            Obstacle obstacles[numObstacles];
245            for(int j=0; j<numObstacles; j++) {
246                cin >> obstacles[j].x1 >> obstacles[j].y1 >> obstacles[j].x2 >> obstacles[j].y2;
247            }
248            int sp = shortestPath(start, end, width, height, obstacles, numObstacles);
249            if(sp<-1) {
250                cout << "No solution." << endl;
251            } else if(sp==-1) {
252                cout << "Optimal solution takes 0 hops." << endl;
253            } else {
254                cout << "Optimal solution takes " << sp << " hops." << endl;
255            }
256        }
257    }
```

Burning Coins

```cpp
#include <iostream>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    int testCases;
    cin >> testCases;

    for(int i=0; i < testCases; i++) {
        int numCoins;
        cin >> numCoins;
        int minOpt[numCoins][numCoins];
        for(int j=0; j < numCoins; j++) {
            cin >> minOpt[j][j];
        }
        for(int j=0; j < numCoins-1; j++) {
            minOpt[j][j+1] = max(minOpt[j][j], minOpt[j+1][j+1]);
            // cout << j << " " << j+1 << " " << minOpt[j][j+1] << endl;
        }
        for(int m=2; m < numCoins; m++) {
            for(int j=0; j < numCoins-m; j++) {
                int k= j+m;

                minOpt[j][k]
                    = max(minOpt[j][j] + ((minOpt[j+2][k] > minOpt[j+1][k-1]) ? minOpt[j+1][k-1] : minOpt[j+2][k]),
                            minOpt[k][k] + ((minOpt[j+1][k-1] > minOpt[j][k-2]) ? minOpt[j][k-2] : minOpt[j+1][k-1]));
                // cout << j << " " << k << " " << minOpt[j][k] << endl;


            }
        }
        cout << minOpt[0][numCoins-1] << endl;
    }
}
```

Jump

```
1   #include <iostream>
2   #include <limits>
3
4   using namespace std;
5
6   int main() {
7       ios_base::sync_with_stdio(false);
8       int testCases = 0;
9       cin >> testCases;
10      for(int testCase=0; testCase < testCases; testCase++) {
11          int numCells, maxJump;
12          cin >> numCells >> maxJump;
13
14          int vCell[numCells];
15          long long minCost[numCells];
16
17          minCost[0] = 0;
18          cin >> vCell[0];
19          int globalBestPos = 0;
20
21          for(int curCell=1; curCell < numCells; curCell++) {
22              if(globalBestPos < curCell-maxJump) {
23                  // find minCost
24                  long long min = numeric_limits<long long>::max();
25                  long long pos = curCell-maxJump;
26                  for(int k=curCell-maxJump; k < curCell; k++) {
27                          long long curMin = minCost[k];
28                          if(curMin < min) {
29                              min = curMin;
30                              pos = k;
31                          }
32                  }
33                  globalBestPos = pos;
34
35              }
36              cin >> vCell[curCell];
37              long long curMinPos = minCost[globalBestPos] + vCell[curCell];
38              minCost[curCell] = curMinPos;
39              if(minCost[globalBestPos] >= curMinPos) {
40                  globalBestPos = curCell;
41              }
42
43              // cout << "Min on pos " << i << " is " << minCost[curCell] << endl;
44          }
45
46          cout << minCost[numCells-1] << endl;
47      }
48      return 0;
49  }
```

Light Pattern

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   void decToBin(int dec, vector<int> &bin, int binLen) {
7       int i=binLen-1;
8       while(dec > 1) {
9           if(dec%2==0) {
10              bin[i]=0;
11          } else {
12              bin[i]=1;
13          }
14          dec=dec/2;
15          i--;
16      }
17      bin[i] = (dec==0) ? 0 : 1;
18      i--;
19      while(i >=0) {
20          bin[i] = 0;
21          i--;
22      }
23  }
24
25  void printVector(vector<int> &v) {
26      for(vector<int>::iterator it = v.begin(); it != v.end(); it++) {
27          cout << *it << "␣";
28      }
29      cout << endl;
30  }
31
32  int computeSteps(vector<int> &bulb, vector<int> &resultPattern, int numBulbs, int patternLength) {
33      int numPatterns = numBulbs/patternLength;
34      int curReplacesToPattern = 0;
35      int curReplacesToOpposite = 0;
36      //go over every pattern
37      for(int i=0; i<numPatterns; i++) {
38          int replacesToPattern=0;
39          int replacesToOpposite=0;
40          for(int j=0; j<patternLength; j++) {
41              int iBulb = i*patternLength+j;
42              if(bulb[iBulb] != resultPattern[j]) {
43                  replacesToPattern++;
44                  // cout << "bulb " << iBulb << " is like opposite" << endl;
45              } else {
46                  replacesToOpposite++;
47                  // cout << "bulb " << iBulb << " is like pattern" << endl;
48              }
49          }
50          // cout << "pattern " << i << " needs " << replacesToPattern << " and " << replacesToOpposite << endl;
51          int bothReplacesToPattern = curReplacesToPattern+replacesToPattern;
52          int bothReplacesToOpposite = curReplacesToOpposite+replacesToOpposite;
53          curReplacesToPattern = (bothReplacesToPattern < bothReplacesToOpposite + 1)
54                              ? bothReplacesToPattern : bothReplacesToOpposite+1;
55          curReplacesToOpposite = (bothReplacesToPattern+1 < bothReplacesToOpposite)
56                              ? bothReplacesToPattern+1 : bothReplacesToOpposite;
57      }
58
59      return curReplacesToPattern;
60  }
61
62  int main() {
63      ios_base::sync_with_stdio(false);
64      int testCases;
65      cin >> testCases;
66
67      for(int i=0; i < testCases; i++) {
68          int numBulbs, patternLength, x;
69          cin >> numBulbs >> patternLength >> x;
70
71          vector<int> bulb(numBulbs);
72          for(int j=0; j<numBulbs; j++) {
73              int state;
```

```cpp
74              cin >> state;
75              bulb[j] = state;
76          }
77          // cout << "bulbs: " ;
78          // printVector(bulb);
79
80          vector<int> binX(patternLength);
81          decToBin(x, binX, patternLength);
82          // cout << "result pattern: " ;
83          // printVector(binX);
84          cout << computeSteps(bulb, binX, numBulbs, patternLength) <<endl;
85
86      }
87  }
```

Longest Path

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  pair<int, int> longestPath(int node, int ancestor, const vector<vector<int> > &adjList) {
7      const vector<int> *neighbors = &adjList[node];
8      if(neighbors->size() == 1 && ancestor != -1) {
9          return pair<int, int>(1,1);
10     } else if (neighbors->size() == 2 && ancestor != -1) {
11         for(vector<int>::const_iterator it = neighbors->begin(); it != neighbors->end(); it++) {
12             if(*it != ancestor) {
13                 pair<int, int> child = longestPath(*it, node, adjList);
14                 if(child.first + 1 > child.second) {
15                     return pair<int, int>(child.first+1, child.first+1);
16                 } else {
17                     return pair<int, int>(child.first+1, child.second);
18                 }
19             }
20         }
21     } else {
22         int maxHeight = 0;
23         int secondMaxHeight = 0;
24         int maxPath = 0;
25         for(vector<int>::const_iterator it = neighbors->begin(); it != neighbors->end(); it++) {
26             if(*it != ancestor) {
27                 pair<int, int> child = longestPath(*it, node, adjList);
28                 if(child.first > maxHeight) {
29                     secondMaxHeight = maxHeight;
30                     maxHeight = child.first;
31                 } else if(child.first > secondMaxHeight) {
32                     secondMaxHeight = child.first;
33                 }
34                 if(child.second > maxPath) {
35                     maxPath = child.second;
36                 }
37             }
38         }
39         int altMaxPath = maxHeight+secondMaxHeight+1;
40         if(altMaxPath > maxPath) {
41             return pair<int, int>(maxHeight+1, altMaxPath);
42         } else {
43             return pair<int, int>(maxHeight+1, maxPath);
44         }
45     }
46  }
47
48  int main() {
49      ios_base::sync_with_stdio(false);
50      int testCases = 0;
51      cin >> testCases;
52      for(int testCase=0; testCase < testCases; testCase++) {
53          int numVertices;
54          cin >> numVertices;
55          vector<vector<int> > adjList(numVertices);
56          int accArray[numVertices];
57
58          for(int i=0; i<numVertices-1; i++) {
59              int v1, v2;
60              cin >> v1 >> v2;
61              adjList[v1].push_back(v2);
62              adjList[v2].push_back(v1);
63              accArray[i] = -1;
64          }
65          accArray[numVertices-1] = -1;
66          cout << longestPath(0, -1, adjList).second << endl;
67      }
68      return 0;
69  }
```

Ants

```
1   // would be faster, when not every species has its own graph but only the property maps change.
2   #include <iostream>
3   #include <vector>
4   #include <boost/config.hpp>
5   #include <boost/graph/adjacency_list.hpp>
6   #include <boost/tuple/tuple.hpp>
7   #include <boost/graph/graphviz.hpp>
8   #include <boost/graph/kruskal_min_spanning_tree.hpp>
9   #include <boost/graph/dijkstra_shortest_paths.hpp>
10
11  using namespace std;
12  using namespace boost;
13
14  typedef adjacency_list<vecS, vecS, undirectedS, no_property,
15  property<edge_weight_t, int> > Graph;
16  typedef graph_traits<Graph> Traits;
17  typedef Traits::vertex_descriptor Vertex;
18  typedef Traits::edge_descriptor Edge;
19  typedef property_map<Graph, edge_weight_t>::type WeightMap;
20
21  void printGraph(Graph g, WeightMap weight) {
22      graph_traits<Graph>::edge_iterator eiter, eiter_end;
23      for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
24          std::cout << source(*eiter, g) << " <--> " << target(*eiter, g)
25            << " with weight of " << weight[*eiter]
26            << std::endl;
27      }
28  }
29
30
31  int main() {
32      ios_base::sync_with_stdio(false);
33      int testCases = 0;
34      cin >> testCases;
35      for(int testCase=0; testCase < testCases; testCase++) {
36          int numTreeNodes, numTreeEdges;
37          int numSpecies, startTree, finishTree;
38          cin >> numTreeNodes >> numTreeEdges >> numSpecies
39                  >> startTree >> finishTree;
40
41          //construct graph for each species
42          Graph speciesGraph[numSpecies];
43          WeightMap weightMap[numSpecies];
44          for(int i=0; i < numSpecies; i++) {
45              speciesGraph[i] = Graph(numTreeNodes);
46              weightMap[i] = get(edge_weight, speciesGraph[i]);
47          }
48          for(int e=0; e < numTreeEdges; e++) {
49              int v1, v2;
50              cin >> v1 >> v2;
51              for(int i=0; i < numSpecies; i++) {
52                  Edge e;
53                  int w;
54                  cin >> w;
55                  tie(e,tuples::ignore) = add_edge(v1,v2,speciesGraph[i]);
56                  weightMap[i][e] = w;
57              }
58
59          }
60          int hive[numSpecies];
61          for(int i=0; i < numSpecies; i++) {
62              int aHive;
63              cin >> aHive;
64              hive[i] = aHive;
65              // cout << "species " << i << endl;
66              // printGraph(speciesGraph[i], weightMap[i]);
67          }
68
69          // compute minimum spanning tree for each species
70          vector<Edge> spanning_tree[numSpecies];
71          Graph speciesMST[numSpecies];
72          WeightMap weightMapMST[numSpecies];
73          for(int i=0; i < numSpecies; i++) {
```

```
74              kruskal_minimum_spanning_tree(speciesGraph[i], back_inserter(spanning_tree[i]));
75              //construct new tree
76              for(vector<Edge>::iterator it=spanning_tree[i].begin(); it!=spanning_tree[i].end(); it++) {
77                  Edge e;
78                  tie(e,tuples::ignore) = add_edge(source(*it,speciesGraph[i]),
79                                                    target(*it,speciesGraph[i]),
80                                                    speciesMST[i]);
81                  weightMapMST[i][e] = weightMap[i][*it];
82              }
83              // cout << "species MST" << i << endl;
84              // printGraph(speciesMST[i], weightMapMST[i]);
85          }
86
87          // build combined graph
88          Graph combinedGraph(numTreeNodes);
89          WeightMap combinedWeightMap;
90          combinedWeightMap = get(edge_weight, combinedGraph);
91
92          graph_traits<Graph>::edge_iterator eiter, eiter_end;
93          //iterate over all possible edges
94          for (tie(eiter, eiter_end) = edges(speciesGraph[0]); eiter != eiter_end; ++eiter) {
95              Vertex u = get(vertex_index, speciesGraph[0], source(*eiter, speciesGraph[0]));
96              Vertex v = get(vertex_index, speciesGraph[0], target(*eiter, speciesGraph[0]));
97
98              int minWeight = numeric_limits<int>::max();
99              for(int s=0; s < numSpecies;s++) {
100                 Edge speciesEdge;
101                 bool hasEdge = false;
102                 tie(speciesEdge, hasEdge) = edge(u,v,speciesMST[s]);
103                 if(hasEdge) {
104                     int weight = get(weightMapMST[s],speciesEdge);
105                     if(weight < minWeight) {
106                         minWeight = weight;
107                     }
108                 }
109             }
110             if(minWeight < numeric_limits<int>::max()) {
111                 Edge e;
112                 tie(e,tuples::ignore) = add_edge(u,v,combinedGraph);
113                 combinedWeightMap[e] = minWeight;
114             }
115
116         }
117
118
119         // cout << "combined graph" << endl;
120         // printGraph(combinedGraph, combinedWeightMap);
121
122         // dijkstra
123         vector<int> distances(num_vertices(combinedGraph));
124         dijkstra_shortest_paths(combinedGraph, startTree,
125           distance_map(&distances[0]));
126         // for(vector<int>::iterator it=distances.begin(); it!=distances.end(); it++) {
127         //     cout << "in dlist: " << *it << endl;
128         // }
129         cout << distances[finishTree] << endl;
130     }
131 }
```

Bridges

```
1   #include <iostream>
2   #include <vector>
3   #include <map>
4   #include <set>
5   #include <boost/config.hpp>
6   #include <boost/graph/adjacency_list.hpp>
7   #include <boost/tuple/tuple.hpp>
8   #include <boost/graph/biconnected_components.hpp>
9   #include <boost/graph/connected_components.hpp>
10
11  using namespace std;
12  using namespace boost;
13
14  namespace boost
15  {
16    struct edge_component_t
17    {
18      enum
19      { num = 555 };
20      typedef edge_property_tag kind;
21    }
22    edge_component;
23  }
24
25
26  typedef adjacency_list<setS, vecS, undirectedS, no_property,
27          property<edge_component_t, std::size_t> > Graph;
28  typedef graph_traits<Graph> Traits;
29  typedef Traits::vertex_descriptor Vertex;
30  typedef Traits::edge_descriptor Edge;
31  typedef property_map<Graph, vertex_index_t>::type IndexMap;
32  typedef property_map<Graph, edge_component_t>::type ComponentMap;
33
34  void bridgesWithBiconnected(Graph &g, std::set<pair<int, int> > &criticalBridges) {
35      ComponentMap componentMap = get(edge_component, g);
36      size_t num_comps = biconnected_components(g, componentMap);
37      vector<vector<Edge> > numEleOfComp(num_comps, vector<Edge>());
38
39      IndexMap index = get(vertex_index, g);
40      Traits::edge_iterator ei, ei_end;
41      // cout << "new:"<<endl;
42      for (tie(ei, ei_end) = edges(g); ei != ei_end; ++ei) {
43          // std::cout << "(" << index[source(*ei, g)]
44          //        << "," << index[target(*ei, g)] << ") ";
45          // cout << " comp: " << componentMap[*ei] << endl;
46          numEleOfComp[componentMap[*ei]].push_back(*ei);
47      }
48
49
50      //every edge that is alone in a biconnected component is an important bridge
51      for(int i=0; i<num_comps; i++) {
52          if(numEleOfComp[i].size() ==1) {
53              Edge e = numEleOfComp[i][0];
54              int u = index[source(e, g)];
55              int v = index[target(e, g)];
56              pair<int,int> uv = (u<v)?pair<int,int>(u, v):pair<int,int>(v, u);
57              criticalBridges.insert(uv);
58          }
59      }
60  }
61  void bridgesWithBruteForce(Graph &g, std::set<pair<int, int> > &criticalBridges) {
62      vector<Edge> edgeVector;
63      std::vector<int> componentMap(num_vertices(g));
64      int components = connected_components(g, &componentMap[0]);
65
66      Traits::edge_iterator ei, ei_end;
67      for (tie(ei, ei_end) = edges(g); ei != ei_end; ++ei) {
68          edgeVector.push_back(*ei);
69      }
70      for(int i=0; i<edgeVector.size(); i++) {
71          Edge anEdge = edgeVector[i];
72          int sourceNode = source(anEdge, g);
73          int targetNode = target(anEdge,g);
```

```
74          remove_edge(sourceNode, targetNode, g);
75          if(connected_components(g, &componentMap[0]) > components)
76              criticalBridges.insert(pair<int,int>(source(anEdge, g), target(anEdge,g)));
77          Edge e;
78          tie(e, tuples::ignore) = add_edge(sourceNode,targetNode,g);
79      }
80
81  }
82
83  int main() {
84      ios_base::sync_with_stdio(false);
85      int testCases = 0;
86      cin >> testCases;
87      for(int testCase=0; testCase < testCases; testCase++) {
88          int numCities, numBridges;
89          cin >> numCities >> numBridges;
90
91          Graph g(numCities);
92          for(int i=0; i < numBridges; i++) {
93              Edge e;
94              int u,v;
95              cin >> u >> v;
96              tie(e, tuples::ignore) = add_edge(u,v,g);
97          }
98
99          std::set<pair<int, int> > criticalBridges;
100
101         // if(numCities > 50) {
102             bridgesWithBiconnected(g,criticalBridges);
103         // } else {
104             // bridgesWithBruteForce(g,criticalBridges);
105         // }
106
107         int nCriticalBridges = criticalBridges.size();
108         cout << nCriticalBridges << endl;
109         for(set<pair<int,int> >::iterator eit=criticalBridges.begin();
110                 eit!=criticalBridges.end(); eit++){
111             cout << (*eit).first
112              << "␣" << (*eit).second << endl;
113         }
114     }
115 }
```

Build The Graph

```cpp
1   #include <iostream>
2   #include <vector>
3   #include <boost/config.hpp>
4   #include <boost/graph/adjacency_list.hpp>
5   #include <boost/tuple/tuple.hpp>
6   #include <boost/graph/graphviz.hpp>
7   #include <boost/graph/kruskal_min_spanning_tree.hpp>
8   #include <boost/graph/dijkstra_shortest_paths.hpp>
9
10  using namespace std;
11  using namespace boost;
12
13  typedef adjacency_list<vecS, vecS, undirectedS, no_property,
14  property<edge_weight_t, int> > Graph;
15  typedef graph_traits<Graph> Traits;
16  typedef Traits::vertex_descriptor Vertex;
17  typedef Traits::edge_descriptor Edge;
18  typedef property_map<Graph, edge_weight_t>::type WeightMap;
19
20  pair<int, int> computeTask(Graph &g, WeightMap &weightMap) {
21      vector<Edge> spanning_tree;
22      kruskal_minimum_spanning_tree(g, back_inserter(spanning_tree));
23
24      int sumMSTWeights = 0;
25      // cout << "Print the edges in the MST:" << endl;
26      for (vector < Edge >::iterator ei = spanning_tree.begin();
27          ei != spanning_tree.end(); ++ei) {
28      // cout << source(*ei, g) << " <--> " << target(*ei, g) << " with weight of " << weightMap[*ei] << endl;
29      sumMSTWeights += weightMap[*ei];
30      }
31
32      //get vertex 0
33      graph_traits<Graph>::vertex_iterator vert_0;
34      tie(vert_0, tuples::ignore) = vertices(g);
35      vector<int> distances(num_vertices(g));
36      vector<Vertex> predecessor(num_vertices(g));
37
38      dijkstra_shortest_paths(g, 0,
39          predecessor_map(&predecessor[0]).distance_map(&distances[0]));
40      return pair<int, int>(sumMSTWeights, *max_element(distances.begin(), distances.end()));
41
42  }
43  int main() {
44      ios_base::sync_with_stdio(false);
45      int testCases = 0;
46      cin >> testCases;
47      for(int testCase=0; testCase < testCases; testCase++) {
48          int numVertices, numEdges;
49          cin >> numVertices >> numEdges;
50          Graph g(numVertices);
51          WeightMap weightMap = get(edge_weight, g);
52          for(int edgeIndex=0; edgeIndex < numEdges; edgeIndex++) {
53              int vo,vi,w;
54              cin >> vo >> vi >> w;
55              Edge e;
56              tie(e,tuples::ignore) = add_edge(vi,vo,g);
57              weightMap[e] = w;
58          }
59          // cout << computeTask(g) << endl;
60          pair<int, int> result = computeTask(g, weightMap);
61          cout << result.first << "␣" << result.second << endl;
62
63
64      }
65
66      return 0;
67  }
```

Deleted Entries

```
1   #include <iostream>
2   #include <vector>
3   #include <boost/config.hpp>
4   #include <boost/graph/adjacency_list.hpp>
5   #include <boost/tuple/tuple.hpp>
6   #include <boost/graph/graphviz.hpp>
7   #include <boost/graph/kruskal_min_spanning_tree.hpp>
8   #include <boost/graph/dijkstra_shortest_paths.hpp>
9   #include <boost/graph/sequential_vertex_coloring.hpp>
10
11  using namespace std;
12  using namespace boost;
13
14  typedef adjacency_list<vecS, vecS, undirectedS > Graph;
15  typedef graph_traits<Graph> Traits;
16  typedef Traits::vertex_descriptor Vertex;
17  typedef Traits::edge_descriptor Edge;
18  typedef property_map<Graph, vertex_index_t>::type IndexMap;
19
20  void printGraph(Graph g) {
21      graph_traits<Graph>::edge_iterator eiter, eiter_end;
22      for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
23          std::cout << source(*eiter, g) << "␣<-->␣" << target(*eiter, g)
24            << std::endl;
25      }
26  }
27
28  // typedef vector<Vertex> PredecessorMap;
29  // typedef vector<int> ColorMap;
30
31  // class colorize : public default_bfs_visitor {
32  //     private:
33  //         PredecessorMap m_predecessor;
34  //         ColorMap m_colorMap;
35  //         ColorMap m_prevColorMap;
36  //         int maxColors;
37  //     public:
38  //         //give numColors as parameter
39  //         colorize(PredecessorMap p, ColorMap c, ColorMap prevC, int numColors)
40  //             : m_predecessor(p), m_colorMap(c),
41  //               m_prevColorMap(prevC) {
42  //                 maxColors = numColors;
43  //
44  //             }
45  //         int newColor(int c) {
46  //             return (c+1)%maxColors;
47  //         }
48  //         void tree_edge(Edge e, Graph g) {
49  //             Vertex s = source(e, g);
50  //             Vertex t = target(e, g);
51  //             put(m_predecessor, t, s);
52  //             int color = newColor(m_prevColorMap[s]);
53  //             if(color == m_colorMap[s]) {
54  //                 color = newColor(color);
55  //             }
56  //             put(m_colorMap, t, color);
57  //             put(m_prevColorMap, s, color);
58  //         }
59  //         //void on_start_vertex default color
60  // };
61
62  void colorize(Graph &g, int numColors, vector<int> &color) {
63      IndexMap index = get(vertex_index, g);
64      vector<bool> visited(num_vertices(g), false);
65      std::queue<Vertex> bfs_queue;
66
67      //get start vertex
68      Traits::vertex_iterator vi;
69      tie(vi, tuples::ignore) = vertices(g);
70      Vertex startV = *vi;
71
72      color[index[startV]] = 0;
73      visited[index[startV]] = true;
```

```
74          bfs_queue.push(startV);
75
76          int curColor = 0;
77          while (!bfs_queue.empty()) {
78              Vertex v = bfs_queue.front();
79              bfs_queue.pop();
80              int parentColor = color[index[v]];
81              // cout << "current " << index[v] << " col:" << color[index[v]] << endl;
82
83              Traits::adjacency_iterator adjV, adEnd;
84              tie(adjV, adEnd) = adjacent_vertices(v, g);
85              for (; adjV != adEnd; adjV++) {
86                  int vIndex = index[*adjV];
87                  if(!visited[vIndex]) {
88                      visited[vIndex] = true;
89                      curColor=(curColor+1)%numColors;
90                      if(curColor==parentColor)
91                          curColor=(curColor+1)%numColors;
92                      color[vIndex] = curColor;
93                      bfs_queue.push(*adjV);
94                  }
95
96              }
97          }
98      }
99
100     int main() {
101         ios_base::sync_with_stdio(false);
102         int testCases = 0;
103         cin >> testCases;
104         for(int testCase=0; testCase < testCases; testCase++) {
105             int numStudents, numEdges, numGroups;
106             cin >> numStudents >> numEdges >> numGroups;
107             Graph g(numStudents);
108             for(int i = 0; i < numEdges; i++) {
109                 Edge e;
110                 int u,v;
111                 cin >> u >> v;
112                 tie(e, tuples::ignore) =add_edge(u,v,g);
113             }
114             // cout << "whole graph: " << endl;
115             // printGraph(g);
116
117             // vector<Edge> spanning_tree;
118             // //connected components
119             // kruskal_minimum_spanning_tree(g, back_inserter(spanning_tree));
120
121             // vector<vector<int> > groupList(numGroups);
122             Graph MST = g;
123             // Graph MST(spanning_tree.begin(), spanning_tree.end(), numStudents);
124             // for(vector<Edge>::iterator it=spanning_tree.begin(); it !=spanning_tree.end(); it++) {
125             //     Edge e;
126             //     tie(e, tuples::ignore) =add_edge(source(*it,g),target(*it,g),MST);
127             // }
128
129             // cout << "MS tree: " << endl;
130             // printGraph(MST);
131
132             //not enough vertices? -> no
133             if(num_vertices(MST) < numGroups) {
134                 cout << "no" << endl;
135                 continue;
136             }
137             vector<int> color(num_vertices(g), -1);
138             colorize(MST, numGroups, color);
139
140             // not connected? -> no
141             if(find(color.begin(), color.end(), -1) != color.end()) {
142                 cout << "no" << endl;
143                 continue;
144             }
145
146
147             vector< vector<int> > groups(numGroups, vector<int>());
148             for(int i=0; i<numStudents; i++) {
149                 int studGroup = color[i];
```

44

```
150              groups[studGroup].push_back(i);
151          }
152
153          cout << "yes" << endl;
154          for(vector<vector<int> >::iterator it=groups.begin(); it != groups.end(); it++) {
155              cout << (*it).size();
156              for(int i=0; i < (*it).size(); i++) {
157                  cout << " " <<(*it)[i];
158              }
159              cout << endl;
160          }
161
162      }
163      return 0;
164  }
```

Shy Programmers

```cpp
#include <iostream>
#include <vector>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/boyer_myrvold_planar_test.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, undirectedS> Graph;
typedef graph_traits<Graph> Traits;
typedef Traits::vertex_descriptor Vertex;
typedef Traits::edge_descriptor Edge;

int main() {
    ios_base::sync_with_stdio(false);
    int testCases = 0;
    cin >> testCases;
    for(int testCase=0; testCase < testCases; testCase++) {
        int numCoders, numFriends;
        cin >> numCoders >> numFriends;
        Graph g(numCoders+1);

        for(int i=0; i<numFriends; i++) {
            Edge e;
            int u,v;
            cin >> u >> v;
            tie(e, tuples::ignore) =add_edge(u,v,g);
        }


        //add vertex, connected to all vertices
        for(int i=0; i<numCoders; i++) {
            Edge e;
            tie(e, tuples::ignore) = add_edge(numCoders,i,g);
        }
        if(boyer_myrvold_planarity_test(g)){
            cout << "yes" << endl;
        } else {
            cout << "no" << endl;
        }
    }
    return 0;
}
```

Algocoon Group

```
1   #include <boost/config.hpp>
2   #include <boost/tuple/tuple.hpp>
3   #include <boost/graph/adjacency_list.hpp>
4   #include <boost/graph/push_relabel_max_flow.hpp>
5   #include <iostream>
6   #include <limits>
7   #include <queue>
8
9   using namespace std;
10  using namespace boost;
11
12  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13  typedef adjacency_list<vecS, vecS, directedS, no_property,
14    property<edge_capacity_t, long,
15    property<edge_residual_capacity_t, long,
16    property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
17  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
18  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
19  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
20  typedef property_map<Graph, vertex_index_t>::type IndexMap;
21
22  typedef graph_traits<Graph>::vertex_descriptor Vertex;
23  typedef graph_traits<Graph>::edge_descriptor Edge;
24  typedef graph_traits<Graph> GraphTraits;
25
26  void printGraph(Graph g) {
27      graph_traits<Graph>::edge_iterator eiter, eiter_end;
28      for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
29          std::cout << source(*eiter, g) << " <--> " << target(*eiter, g)
30            << std::endl;
31      }
32  }
33
34  void findCut(Graph &g, EdgeCapacityMap &capacity,
35                ResidualCapacityMap &res_capacity,
36                std::set<Vertex> &ret,
37                Vertex sourceV, Vertex sinkV) {
38      IndexMap index = get(vertex_index, g);
39      vector<bool> visited(num_vertices(g), false);
40      std::queue<Vertex> bfs_queue;
41
42      //get start vertex
43      Vertex startV = sourceV;
44
45      visited[index[startV]] = true;
46      bfs_queue.push(startV);
47      ret.insert(startV);
48
49      while (!bfs_queue.empty()) {
50          Vertex v = bfs_queue.front();
51          bfs_queue.pop();
52
53          graph_traits<Graph>::adjacency_iterator adjV, adEnd;
54          GraphTraits::out_edge_iterator out_i, out_end;
55          for (tie(out_i, out_end) = out_edges(v, g);
56            out_i != out_end; ++out_i) {
57            Edge e = *out_i;
58            Vertex src = source(e, g), targ = target(e, g);
59            int flow = capacity[e] - res_capacity[e];
60            // cout << "edge " << src << " " << targ << " c:" << capacity[e] << " f:" << flow << endl;
61            int vIndex = index[targ];
62            if(flow < capacity[e] && !visited[vIndex]) {
63                bfs_queue.push(targ);
64                visited[vIndex] = true;
65                ret.insert(targ);
66            }
67          }
68      }
69  }
70
71  int cutCapacity(Graph &g, set<Vertex> &sSet, EdgeCapacityMap &capacity) {
72      int cutCapacity=0;
73      for(set<Vertex>::iterator it=sSet.begin();
```

```
74                  it != sSet.end(); it++) {
75          Vertex v = *it;
76          GraphTraits::out_edge_iterator out_i, out_end;
77          for (tie(out_i, out_end) = out_edges(v, g);
78              out_i != out_end; ++out_i) {
79              Edge e = *out_i;
80              Vertex targ = target(e, g);
81              if(find(sSet.begin(), sSet.end(), targ) == sSet.end()) {
82                  //in T set
83                  cutCapacity+=capacity[e];
84              }
85          }
86      }
87      return cutCapacity;
88  }
89
90  pair<int, int> findBestSourceSinkSlow(Graph &g, int numFigures) {
91      pair<int, int> bestSourceSink;
92      int minMaxFlow = numeric_limits<int>::max();
93      for(int i=0; i<numFigures; i++) {
94          for(int j=i+1; j<numFigures; j++) {
95              long flow = push_relabel_max_flow(g, i, j);
96              if(flow < minMaxFlow) {
97                  bestSourceSink = pair<int, int>(i, j);
98                  minMaxFlow = flow;
99              }
100             flow = push_relabel_max_flow(g, j, i);
101             if(flow < minMaxFlow) {
102                 bestSourceSink = pair<int, int>(j, i);
103                 minMaxFlow = flow;
104             }
105         }
106     }
107     return bestSourceSink;
108 }
109
110
111 pair<int, int> findBestSourceSinkFast(Graph &g, int numFigures) {
112     pair<int, int> bestSourceSink;
113     int minMaxFlow = numeric_limits<int>::max();
114     int u = 0;
115     for(int i=1; i<numFigures; i++) {
116         long flow = push_relabel_max_flow(g, u, i);
117         if(flow < minMaxFlow) {
118             bestSourceSink = pair<int, int>(u,i);
119             minMaxFlow = flow;
120         }
121     }
122     for(int i=1; i<numFigures; i++) {
123         long flow = push_relabel_max_flow(g, i, u);
124         if(flow < minMaxFlow) {
125             bestSourceSink = pair<int, int>(i,u);
126             minMaxFlow = flow;
127         }
128     }
129     return bestSourceSink;
130 }
131 int main() {
132     ios_base::sync_with_stdio(false);
133     int testCases = 0;
134     cin >> testCases;
135     for(int testCase=0; testCase < testCases; testCase++) {
136         int numFigures, numLimbs;
137         cin >> numFigures >> numLimbs;
138         Graph g(numFigures);
139         EdgeCapacityMap capacity = get(edge_capacity, g);
140         ReverseEdgeMap rev_edge = get(edge_reverse, g);
141         ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
142
143         for(int i=0; i<numLimbs;i++) {
144             int u, v, c;
145             cin >> u >> v >> c;
146             Edge e, reverseE;
147             tie(e, tuples::ignore) = add_edge(u,v,g);
148             tie(reverseE, tuples::ignore) = add_edge(v, u, g);
149             capacity[e] = c;
```

```
150            capacity[reverseE] = 0;
151            rev_edge[e] = reverseE;
152            rev_edge[reverseE] = e;
153        }
154
155        pair<int, int> bestSourceSink = findBestSourceSinkFast(g, numFigures);
156        int sourceV = bestSourceSink.first;
157        int sinkV = bestSourceSink.second;
158        long flow = push_relabel_max_flow(g, sourceV, sinkV);
159        cout << flow << endl;
160        std::set<Vertex> oneSet;
161        findCut(g, capacity, res_capacity, oneSet, sourceV, sinkV);
162        IndexMap index = get(vertex_index, g);
163        cout << oneSet.size() << "␣";
164        for(set<Vertex>::iterator it=oneSet.begin();
165                it != oneSet.end(); it++) {
166            cout << index[*it] << "␣";
167        }
168        cout << endl;
169        // cout << "cut capacity " << cutCapacity(g, oneSet, capacity) << endl;
170
171    }
172 }
```

Buddy Selection

```cpp
#include <boost/config.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/max_cardinality_matching.hpp>
#include <iostream>
#include <set>

using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, undirectedS > Graph;
typedef graph_traits<Graph> Traits;
typedef Traits::vertex_descriptor Vertex;
typedef Traits::edge_descriptor Edge;
typedef property_map<Graph, vertex_index_t>::type IndexMap;

int main() {
    ios_base::sync_with_stdio(false);
    int testCases = 0;
    cin >> testCases;
    for(int testCase=0; testCase < testCases; testCase++) {
        int numStudents, numChars, numCommChar;
        cin >> numStudents >> numChars >> numCommChar;

        Graph g(numStudents);
        vector<set<string> > chars(numStudents, set<string>());

        // cout << "testset:" << testCase << endl;
        for(int i=0; i < numStudents; i++) {
            for(int j=0; j < numChars; j++) {
                string characteristic;
                cin >> characteristic;
                chars[i].insert(characteristic);
            }
            for(int j=0; j < i; j++) {
                std::vector<string> inters;
                std::set_intersection(chars[i].begin(), chars[i].end(),
                                      chars[j].begin(), chars[j].end(),
                                      std::back_inserter(inters));
                if(inters.size() > numCommChar) {
                    // cout << "v:" << i << " u:" << j << endl;
                    //add edge
                    Edge e;
                    tie(e, tuples::ignore)=add_edge(i,j,g);
                }
            }
        }
        vector<Vertex> mate(numStudents);
        bool foundMatching = checked_edmonds_maximum_cardinality_matching(g, &mate[0]);
        if(matching_size(g, &mate[0])*2 >= numStudents) {
            cout << "not␣optimal" << endl;
        } else {
            cout << "optimal" << endl;
        }
    }
}
```

Satellites

```
1   /* minimum vertex cover */
2   #include <iostream>
3   #include <vector>
4   #include <boost/config.hpp>
5   #include <boost/graph/adjacency_list.hpp>
6   #include <boost/tuple/tuple.hpp>
7   #include <boost/graph/max_cardinality_matching.hpp>
8
9   using namespace std;
10  using namespace boost;
11
12  typedef adjacency_list<vecS, vecS, undirectedS > Graph;
13  typedef graph_traits<Graph> Traits;
14  typedef Traits::vertex_descriptor Vertex;
15  typedef Traits::edge_descriptor Edge;
16  typedef property_map<Graph, vertex_index_t>::type IndexMap;
17
18  void minVCoverBipDfs(Graph &g, vector<Vertex> &mate, vector<bool> &visited,
19          int node, vector<bool> &isLeft) {
20      visited[node] = true;
21      Traits::out_edge_iterator out_i, out_end;
22      for (tie(out_i, out_end) = out_edges(node, g);
23              out_i != out_end; ++out_i) {
24          Vertex targ = target(*out_i, g);
25          int targetNode = get(vertex_index, g)[targ];
26          if(visited[targetNode])
27              continue;
28          if(isLeft[node]) {
29              if(mate[node] != targ) {
30                  minVCoverBipDfs(g, mate, visited, targetNode, isLeft);
31              }
32          } else {
33              if(mate[node] == targ) {
34                  minVCoverBipDfs(g, mate, visited, targetNode, isLeft);
35              }
36          }
37      }
38  }
39
40
41  /* takes bipartite graph and a vector that specifies whether
42   * a node is on the left side.
43   * Returns a vector containing the min vertex cover
44   */
45  vector<int> findMinVertexCoverBipartite(Graph &g, vector<bool> &isLeft) {
46      int numNodes = isLeft.size();
47      vector<Vertex> mate(numNodes);
48      edmonds_maximum_cardinality_matching(g, &mate[0]);
49
50      vector<bool> visited(numNodes, false);
51      const Vertex NULL_VERTEX = graph_traits<Graph>::null_vertex();
52      for(int i=0;i<numNodes;i++) {
53          if(isLeft[i] && mate[i] == NULL_VERTEX) {
54              visited[i]=true;
55          }
56      }
57
58      for(int i = 0; i < numNodes; i++) {
59          if(isLeft[i] && visited[i]) {
60              vector<bool> dfsVisited(numNodes, false);
61              minVCoverBipDfs(g, mate, visited, i, isLeft);
62          }
63      }
64
65      vector<int> minVertexCover;
66      for(int i = 0; i < numNodes; i++) {
67          if(isLeft[i] && !visited[i]) {
68              minVertexCover.push_back(i);
69          } else if(!isLeft[i] && visited[i]){
70              minVertexCover.push_back(i);
71          }
72      }
73      return minVertexCover;
```

```
 74    }
 75
 76    int main() {
 77        ios_base::sync_with_stdio(false);
 78        int testCases = 0;
 79        cin >> testCases;
 80        for(int testCase=0; testCase < testCases; testCase++) {
 81            int numStations, numSatellites, numLinks;
 82            cin >> numStations >> numSatellites >> numLinks;
 83            Graph g(numStations + numSatellites);
 84            vector<bool> isLeft(numStations+numSatellites,false);
 85            for(int i=0; i<numLinks; i++) {
 86                int station, satellite;
 87                cin >> station >> satellite;
 88                Edge e;
 89                tie(e, tuples::ignore) =add_edge(station,numStations+satellite,g);
 90                isLeft[station] = true;
 91            }
 92
 93            vector<int> minVertexCover = findMinVertexCoverBipartite(g, isLeft);
 94
 95            vector<int> minStations;
 96            vector<int> minSatellites;
 97
 98            for(int i=0; i<minVertexCover.size(); i++) {
 99                int node = minVertexCover[i];
100                if(node<numStations)
101                    minStations.push_back(node);
102                else
103                    minSatellites.push_back(node-numStations);
104            }
105
106            cout << minStations.size() << " " << minSatellites.size() << endl;
107            for(vector<int>::iterator it=minStations.begin(); it!=minStations.end(); it++) {
108                cout << *it << " ";
109            }
110            for(vector<int>::iterator it=minSatellites.begin(); it!=minSatellites.end(); it++) {
111                cout << *it << " ";
112            }
113
114            cout << endl;
115        }
116    }
```

Kingdom Defense

```cpp
1   #include <boost/config.hpp>
2   #include <boost/tuple/tuple.hpp>
3   #include <boost/graph/adjacency_list.hpp>
4   #include <boost/graph/push_relabel_max_flow.hpp>
5   #include <iostream>
6   #include <limits>
7   #include <queue>
8
9   using namespace std;
10  using namespace boost;
11
12  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13  typedef adjacency_list<vecS, vecS, directedS, no_property,
14    property<edge_capacity_t, long,
15    property<edge_residual_capacity_t, long,
16    property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
17  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
18  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
19  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
20  typedef property_map<Graph, vertex_index_t>::type IndexMap;
21
22  typedef graph_traits<Graph>::vertex_descriptor Vertex;
23  typedef graph_traits<Graph>::edge_descriptor Edge;
24  typedef graph_traits<Graph> GraphTraits;
25
26  void printGraph(Graph g, EdgeCapacityMap &capacity, vector<pair<int, int> > &locProps) {
27      graph_traits<Graph>::edge_iterator eiter, eiter_end;
28      for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
29          if(capacity[*eiter] > 0) {
30              int aSource = source(*eiter, g);
31              int aTarget = target(*eiter, g);
32              std::cout << aSource <<":␣("
33                  << locProps[aSource].first<< "|" << locProps[aSource].second << ")" <<
34                  "␣-" << capacity[*eiter] << "->␣" << aTarget
35                  << ":␣(" << locProps[aTarget].first<< "|" << locProps[aTarget].second << ")"
36                  << std::endl;
37          }
38      }
39  }
40
41  void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
42      Edge e, reverseE;
43      tie(e, tuples::ignore) = add_edge(u,v,g);
44      tie(reverseE, tuples::ignore) = add_edge(v, u, g);
45      capacity[e] = c;
46      capacity[reverseE] = 0;
47      rev_edge[e] = reverseE;
48      rev_edge[reverseE] = e;
49  }
50
51  int main() {
52      ios_base::sync_with_stdio(false);
53      int testCases = 0;
54      cin >> testCases;
55      for(int testCase=0; testCase < testCases; testCase++) {
56          int numVertices, numEdges;
57          cin >> numVertices >> numEdges;
58          vector<pair<int, int> > locProp(numVertices, pair<int, int>(0,0));
59
60          for(int i=0; i<numVertices; i++) {
61              int numStationed,numNeeded;
62              cin >> numStationed >> numNeeded;
63              locProp[i] = pair<int, int>(numStationed, numNeeded);
64          }
65
66          Graph g(numVertices);
67          EdgeCapacityMap capacity = get(edge_capacity, g);
68          ReverseEdgeMap rev_edge = get(edge_reverse, g);
69          ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
70          for(int i=0; i<numEdges; i++) {
71              int from,to, minPassing, maxPassing;
72              cin >> from >> to >> minPassing >> maxPassing;
73
```

```
74                //force soldiers to walk along
75                locProp[from].first = locProp[from].first - minPassing;
76                locProp[to].first = locProp[to].first + minPassing;
77                maxPassing-=minPassing;
78
79                //build actual graph
80                addFlowEdge(g, capacity, rev_edge, from, to, maxPassing);
81            }
82
83        // printGraph(g, capacity, locProp);
84
85        const int SOURCE = numVertices;
86        const int SINK = numVertices+1;
87        int numNonForcedSoldiers=0;
88        //add source and sink
89        for(int i=0; i<numVertices; i++) {
90            int soldiersStationed = locProp[i].first;
91            int soldiersNeeded = locProp[i].second;
92            if(soldiersStationed > 0) {
93                addFlowEdge(g, capacity, rev_edge, SOURCE, i, soldiersStationed);
94                addFlowEdge(g, capacity, rev_edge, i, SINK, soldiersNeeded);
95            } else {
96                soldiersNeeded-=soldiersStationed;
97                addFlowEdge(g, capacity, rev_edge, i, SINK, soldiersNeeded);
98            }
99            numNonForcedSoldiers+=soldiersNeeded;
100        }
101        // cout << "after adding source and sink:" << endl;
102        // printGraph(g, capacity, locProp);
103
104        //compute maxFlow
105        long flow = push_relabel_max_flow(g, SOURCE, SINK);
106        // long flow = 0;
107
108        // cout << numNonForcedSoldiers << endl;
109        if(flow==numNonForcedSoldiers) {
110            cout << "yes";
111        } else {
112            cout << "no";
113        }
114        cout << endl;
115    }
116 }
```

Coin Tossing

```
1   #include <iostream>
2   #include <vector>
3   #include <boost/config.hpp>
4   #include <boost/graph/adjacency_list.hpp>
5   #include <boost/tuple/tuple.hpp>
6   #include <boost/graph/push_relabel_max_flow.hpp>
7
8   using namespace std;
9   using namespace boost;
10
11  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
12  typedef adjacency_list<vecS, vecS, directedS, no_property,
13    property<edge_capacity_t, long,
14    property<edge_residual_capacity_t, long,
15    property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
16
17  typedef graph_traits<Graph> GraphTraits;
18  typedef GraphTraits::vertex_descriptor Vertex;
19  typedef GraphTraits::edge_descriptor Edge;
20  typedef property_map<Graph, vertex_index_t>::type IndexMap;
21  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
22  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
23  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
24
25
26
27  void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
28      Edge e, reverseE;
29      tie(e, tuples::ignore) = add_edge(u,v,g);
30      tie(reverseE, tuples::ignore) = add_edge(v, u, g);
31      capacity[e] = c;
32      capacity[reverseE] = 0;
33      rev_edge[e] = reverseE;
34      rev_edge[reverseE] = e;
35  }
36
37  int sum(vector<int> &aVector) {
38      int sum_of_elems = 0;
39      for(std::vector<int>::iterator j=aVector.begin();j!=aVector.end();++j)
40          sum_of_elems += *j;
41      return sum_of_elems;
42  }
43
44
45  int main() {
46      ios_base::sync_with_stdio(false);
47      int testCases = 0;
48      cin >> testCases;
49      for(int testCase=0; testCase < testCases; testCase++) {
50          int numPlayers, numRounds;
51          int outcome = 0; //-1 false, 0 don't know, 1 true
52          cin >> numPlayers >> numRounds;
53
54          vector<int> scoreBoard(numPlayers, 0);
55          const int SOURCE = numPlayers;
56          const int SINK = numPlayers+1;
57          vector<vector<int> > helperNode(numPlayers, vector<int>(numPlayers, -1));
58          Graph g;
59          EdgeCapacityMap capacity = get(edge_capacity, g);
60          ReverseEdgeMap rev_edge = get(edge_reverse, g);
61          ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
62          int outOfSink = 0;
63
64          for(int i=0; i<numRounds;i++) {
65              int playerA, playerB, outcome;
66              cin >> playerA >> playerB >> outcome;
67              if(outcome == 1) {
68                  scoreBoard[playerA] -=1;
69              } else if(outcome == 2) {
70                  scoreBoard[playerB] -=1;
71              } else {
72                  //construct graph
73                  if(helperNode[playerA][playerB] == -1) {
```

```
74                      //no helper node before
75                      int aHelperNode = SINK+1+i;
76                      helperNode[playerA][playerB] = aHelperNode;
77                      helperNode[playerB][playerA] = aHelperNode;
78
79                      addFlowEdge(g, capacity, rev_edge, SOURCE, aHelperNode, 1);
80                      addFlowEdge(g, capacity, rev_edge, aHelperNode, playerA, 1);
81                      addFlowEdge(g, capacity, rev_edge, aHelperNode, playerB, 1);
82                  } else {
83                      int aHelperNode = helperNode[playerA][playerB];
84                      Edge sourceToHelper, helperToA, helperToB;
85                      sourceToHelper = edge(SOURCE, aHelperNode, g).first;
86                      helperToA = edge(aHelperNode, playerA, g).first;
87                      helperToB = edge(aHelperNode, playerB, g).first;
88                      capacity[sourceToHelper] += 1;
89                      capacity[helperToA] += 1;
90                      capacity[helperToB] += 1;
91                  }
92                  outOfSink++;
93              }
94          }
95          for(int i=0; i<numPlayers; i++) {
96              int desiredScore;
97              cin >> desiredScore;
98              scoreBoard[i] += desiredScore;
99              if(scoreBoard[i] < 0) {
100                 //a player won more often than in the desired scoreboard
101                 outcome = -1;
102             }
103         }
104         if(outcome >= 0 && outOfSink==sum(scoreBoard)) {
105
106             // if(scoreBoard.sum() < outCapacitySource) {
107             //     outcome = -1;
108             // }
109
110             // add edges to sink
111             for(int i=0; i<numPlayers; i++) {
112                 if(scoreBoard[i] > 0) {
113                     addFlowEdge(g, capacity, rev_edge, i, SINK, scoreBoard[i]);
114                 }
115             }
116
117
118             long flow = push_relabel_max_flow(g, SOURCE, SINK);
119             if(flow != sum(scoreBoard)) {
120                 // cout << "flow != sum(scoreBoard)" << endl;
121                 cout << "no" << endl;
122             } else {
123                 cout << "yes" << endl;
124                 // cout << flow << " " << sum(scoreBoard) << endl;
125             }
126         } else {
127             // cout << "a player won more often than in the desired scoreboard" << endl;
128             cout << "no" << endl;
129         }
130     }
131 }
```

Antenna

```cpp
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
#include <iostream>

using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
typedef CGAL::Min_circle_2_traits_2<K> Traits;
typedef CGAL::Min_circle_2<Traits> Min_circle;

double floor_to_double(const K::FT& x) {
    double a = std::floor(CGAL::to_double(x));
    while(a>x) a-=1;
    while(a+1<=x) a+=1;
    return a;
}

double ceil_to_double(const K::FT& x) {
    double a = std::ceil(CGAL::to_double(x));
    while(a<x) a+=1;
    while(a-1>=x) a-=1;
    return a;
}


int main() {
    ios_base::sync_with_stdio(false);

    while(true) {
        int numCitizens;
        cin >> numCitizens;
        if(numCitizens == 0)
            break;

        K::Point_2 P[numCitizens];
        for(int i=0; i<numCitizens; i++) {
            double x,y;
            cin >> x >> y;
            P[i] = K::Point_2(x, y);
        }
        Min_circle radio(P, P+numCitizens, true);
        Traits::Circle c = radio.circle();
        K::FT radius = sqrt(c.squared_radius());
        cout << std::setiosflags(std::ios::fixed) << setprecision(0) << ceil_to_double(radius) << endl;
        // cout << std::round(CGAL::to_double(radius)) << endl;
    }
}
```

Almost Antenna

```cpp
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
#include <iostream>

using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
typedef CGAL::Min_circle_2_traits_2<K> Traits;
typedef CGAL::Min_circle_2<Traits> Min_circle;

double floor_to_double(const K::FT& x) {
    double a = std::floor(CGAL::to_double(x));
    while(a>x) a-=1;
    while(a+1<=x) a+=1;
    return a;
}

double ceil_to_double(const K::FT& x) {
    double a = std::ceil(CGAL::to_double(x));
    while(a<x) a+=1;
    while(a-1>=x) a-=1;
    return a;
}

double computeSlow(vector<K::Point_2> P) {
    K::FT radius;
    for(vector<K::Point_2>::iterator it = P.begin();
            it != P.end(); it++) {
        vector<K::Point_2> tmpP(P.begin(), it);
        tmpP.insert(tmpP.end(), it+1, P.end());

        Min_circle radio(tmpP.begin(), tmpP.end(), true);
        Traits::Circle c = radio.circle();
        K::FT aRadius = c.squared_radius();
        if(it!=P.begin()) {
            if(aRadius < radius)
                radius = aRadius;
        } else {
            radius = aRadius;
        }
    }
    return ceil_to_double(sqrt(radius));
}

//remove only the defining points of the whole minCircle
double computeFast(vector<K::Point_2> P) {
    Min_circle radio(P.begin(), P.end(), true);

    K::FT radius;
    for(int i=0; i<radio.number_of_support_points(); i++) {
        K::Point_2 sp = radio.support_point(i);
        vector<K::Point_2> tmpP(P.begin(), P.end());
        vector<K::Point_2>::iterator toSp = find(tmpP.begin(),tmpP.end(),sp);
        tmpP.erase(toSp);

        Min_circle partRadio(tmpP.begin(), tmpP.end(), true);
        Traits::Circle c = partRadio.circle();
        K::FT aRadius = c.squared_radius();
        if(i!=0) {
            if(aRadius < radius)
                radius = aRadius;
        } else {
            radius = aRadius;
        }

    }

    return ceil_to_double(sqrt(radius));
}

int main() {
    ios_base::sync_with_stdio(false);
```

```cpp
    while(true) {
        int numCitizens;
        cin >> numCitizens;
        if(numCitizens == 0)
            break;

        vector<K::Point_2> P(numCitizens);
        for(int i=0; i<numCitizens; i++) {
            double x,y;
            cin >> x >> y;
            P[i] = K::Point_2(x, y);
        }
            cout << std::setiosflags(std::ios::fixed) << setprecision(0)
                << computeFast(P) << endl;
    }
}
```

Hit

```
1   #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2   #include <iostream>
3
4   using namespace std;
5
6   typedef CGAL::Exact_predicates_exact_constructions_kernel K;
7
8
9   bool hitsObstacle(int numObstacles) {
10
11      bool ret = false;
12      double x, y, a, b;
13      cin >> x >> y >> a >> b;
14      K::Ray_2 phil(K::Point_2(x,y), K::Point_2(a,b));
15      // cout << "Phil " << phil << endl;
16
17      for(int i=0; i<numObstacles; i++) {
18          double r,s,t,u;
19          cin >> r >> s >> t >> u;
20          K::Segment_2 obstacle(K::Point_2(r,s), K::Point_2(t,u));
21          if(!ret && CGAL::do_intersect(phil, obstacle)) {
22              ret=true;
23          }
24      }
25
26      return ret;
27  }
28
29  int main() {
30      ios_base::sync_with_stdio(false);
31
32      while(true) {
33          int numObstacles;
34          cin >> numObstacles;
35          // cout << "numObstacles " << numObstacles << endl;
36          if(numObstacles == 0) {
37              return 0;
38          }
39          if(hitsObstacle(numObstacles)) {
40              cout << "yes" << endl;
41          } else {
42              cout << "no" << endl;
43          }
44      }
45  }
```

First Hit

```
1   #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2   #include <iostream>
3
4   using namespace std;
5
6   typedef CGAL::Exact_predicates_exact_constructions_kernel K;
7   typedef K::Point_2 P;
8   typedef K::Segment_2 S;
9
10  double floor_to_double(const K::FT& x) {
11      double a = std::floor(CGAL::to_double(x));
12      while(a>x) a-=1;
13      while(a+1<=x) a+=1;
14      return a;
15  }
16
17  bool hitsObstacle(int numObstacles, K::Point_2 &isPos) {
18
19      bool isHit = false;
20      double x, y, a, b;
21      cin >> x >> y >> a >> b;
22      P philStart(x,y);
23      K::Ray_2 phil(philStart, K::Point_2(a,b));
24      P minIntersectionPoint;
25      // cerr << "Phil " << phil << endl;
26
27      for(int i=0; i<numObstacles; i++) {
28          // cerr << "Loop " << i << endl;
29          double r,s,t,u;
30          cin >> r >> s >> t >> u;
31          K::Segment_2 obstacle(K::Point_2(r,s), K::Point_2(t,u));
32          if(CGAL::do_intersect(phil, obstacle)) {
33              // cerr << "Intersection " << i << endl;
34              P intersectionPoint;
35              CGAL::Object o = CGAL::intersection(phil, obstacle);
36              if(const P* op = CGAL::object_cast<P>(&o)) {
37                  intersectionPoint = *op;
38              } else if(const S* os = CGAL::object_cast<S>(&o)) {
39                  //take the segment end that is nearer
40                  intersectionPoint = CGAL::squared_distance(philStart, os->source())
41                                      < CGAL::squared_distance(philStart, os->target())
42                                      ? os->source() : os->target();
43              }
44              else {
45                  throw std::runtime_error("strange segment intersection");
46              }
47
48              // cerr << "Know Intersection " << i << endl;
49              if(isHit) {
50                  if(CGAL::squared_distance(philStart, intersectionPoint)
51                      < CGAL::squared_distance(philStart, minIntersectionPoint)) {
52                      // cerr << "new intersection" << i << endl;
53                      minIntersectionPoint = intersectionPoint;
54                  }
55              } else {
56                  // cerr << "min intersection NULL " << i << endl;
57                  minIntersectionPoint = intersectionPoint;
58              }
59
60              // cerr << "end of loop" << i << endl;
61              isHit=true;
62          }
63      }
64
65      // cerr << "return" << endl;
66      if(isHit) {
67          isPos = minIntersectionPoint;
68      }
69      // cerr << "return" << endl;
70      return isHit;
71  }
72
73  int main() {
```

```cpp
74        ios_base::sync_with_stdio(false);

76        while(true) {
77            // cerr << "beginLoop" << endl;
78            int numObstacles;
79            cin >> numObstacles;
80            // cerr << "numObstacles " << numObstacles << endl;
81            if(numObstacles == 0) {
82                return 0;
83            }
84            K::Point_2 isPos;
85            if(hitsObstacle(numObstacles, isPos)) {
86                // cerr << "afterreturn" << endl;
87                cout << std::setiosflags(std::ios::fixed) << setprecision(0)
88                    << floor_to_double(isPos.x())
89                    << "␣"
90                    << floor_to_double(isPos.y()) << endl;
91                // cerr << "afterout" << endl;
92            } else {
93                cout << "no" << endl;
94            }
95            // cerr << "endloop" << endl;
96        }
97    }
```

```cpp
        K::Point_2 isPos;
```

Search Snippets

```
1    #include <iostream>
2    #include <queue>
3    #include <vector>
4    #include <limits>
5
6    using namespace std;
7
8    const int MAX = numeric_limits<int>::max();
9
10   pair<int,int> minOfVector(vector<int> &v) {
11           int min = MAX;
12           int minPos;
13           for(vector<int>::iterator it=v.begin(); it!= v.end(); it++) {
14                   if(min > *it) {
15                           min = *it;
16                           minPos = it-v.begin();
17                   }
18           }
19           return pair<int,int>(min,minPos);
20   }
21
22   int computeInterval(priority_queue<pair<int,int> > &Q, int numWords) {
23           vector<int> currentPosition(numWords);
24           vector<bool> hasObserved(numWords, false);
25           int numNotObserved = numWords;
26
27           int minInterval = MAX;
28           int interval1 = 0;
29           int interval2 = 0;
30           while(Q.size() > 0){
31                   pair<int, int> occurence = Q.top();
32                   Q.pop();
33                   int word = occurence.second;
34                   int pos = -occurence.first;
35   //              cout << "current wordpos of " << word << " is " << pos << endl;
36
37                   currentPosition[word] = pos;
38                   if(hasObserved[word] == false) {
39                           hasObserved[word] = true;
40                           numNotObserved--;
41                   }
42                   if(numNotObserved > 0)
43                           continue;
44
45                   pair<int,int> aMinOfVector = minOfVector(currentPosition);
46                   int curInterval = pos - aMinOfVector.first +1;
47
48                   if(curInterval<minInterval) {
49                           minInterval=curInterval;
50                           interval1 =aMinOfVector.second;
51                           interval2 =pos;
52   //                      cout << minInterval << " between ["<<  interval1<<  ","<<  interval2<<  "]" << endl;
53                   }
54           }
55
56           return minInterval;
57   }
58
59   int main() {
60       ios_base::sync_with_stdio(false);
61       int testCases;
62       cin >> testCases;
63       for(int i=0; i<testCases; i++) {
64           int numWords;
65           cin >> numWords;
66           int numAppears[numWords];
67           for(int j=0;j<numWords; j++) {
68                   int num;
69                   cin >> num;
70                   numAppears[j] = num;
71           }
72
73           priority_queue<pair<int,int> > Q;
```

```
for(int j=0;j<numWords; j++) {
        for(int k=0; k<numAppears[j]; k++) {
                pair<int, int> occurence;
                int position;
                cin>>position;
                Q.push(pair<int,int>(-position, j));
        }
}
cout << computeInterval(Q, numWords) << endl;

        }
}
```

Bistro

```
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <iostream>
4   #include <cmath>
5
6   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7   typedef CGAL::Delaunay_triangulation_2<K>  Triangulation;
8   typedef Triangulation::Edge_iterator  Edge_iterator;
9   typedef CGAL::Segment_2<K> Segment;
10  int main() {
11      std::ios_base::sync_with_stdio(false);
12
13      int numBistros;
14      while(std::cin >> numBistros) {
15          if(numBistros==0) break;
16
17          // std::cout << numBistros << std::endl;
18          std::vector<K::Point_2> bistro(numBistros);
19          Triangulation t;
20          for(int i=0; i<numBistros;i++) {
21              double x, y;
22              std::cin >> x >> y;
23              bistro[i] = K::Point_2(x,y);
24              // std::cout << bistro[i] << std::endl;
25          }
26          t.insert(bistro.begin(), bistro.end());
27
28          int numNewBistros;
29          std::cin >> numNewBistros;
30          // std::cout << numNewBistros << std::endl;
31          for(int i=0; i<numNewBistros;i++) {
32              double x, y;
33              std::cin >> x >> y;
34              K::Point_2 newBistro(x,y);
35              Triangulation::Vertex_handle nearestBistro = t.nearest_vertex(newBistro);
36              Segment shortestPath(nearestBistro->point(), newBistro);
37              std::cout << std::setiosflags(std::ios::fixed) << std::setprecision(0)
38                  << shortestPath.squared_length() << std::endl;
39          }
40
41
42      }
43  }
```

Germs

```
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <iostream>
4   #include <cmath>
5   #include <queue>
6
7   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8   typedef CGAL::Delaunay_triangulation_2<K>  Triangulation;
9   typedef Triangulation::Edge_iterator  Edge_iterator;
10  typedef CGAL::Segment_2<K> Segment;
11  typedef K::Point_2 Point;
12
13  double ceil_to_double(const K::FT& x) {
14      double a = std::ceil(CGAL::to_double(x));
15      while(a<x) a+=1;
16      while(a-1>=x) a-=1;
17      return a;
18  }
19
20  int main() {
21      std::ios_base::sync_with_stdio(false);
22
23      int numBacteria;
24      while(std::cin >> numBacteria) {
25          if(numBacteria==0) break;
26          int l,b,r,t;
27          std::cin >> l >> b >> r >> t;
28          std::vector<Segment> dish(4);
29          dish[0] = Segment(Point(l,b),Point(r,b));
30          dish[1] = Segment(Point(r,b),Point(r,t));
31          dish[2] = Segment(Point(r,t),Point(l,t));
32          dish[3] = Segment(Point(l,t),Point(l,b));
33
34
35          std::vector<K::Point_2> bacteria(numBacteria);
36          for(int i=0;i<numBacteria;i++) {
37              int x,y;
38              std::cin >> x >> y;
39              bacteria[i] = K::Point_2(x,y);
40          }
41
42          std::priority_queue<K::FT> q;
43          if(numBacteria > 1) {
44              Triangulation tria;
45              tria.insert(bacteria.begin(), bacteria.end());
46
47              Triangulation::Vertex_iterator v = tria.finite_vertices_begin();
48              for(Triangulation::Vertex_iterator v = tria.finite_vertices_begin();
49                  v != tria.finite_vertices_end(); ++v) {
50                  bool firstEdge = true;
51                  K::FT minDist;
52
53
54                  Triangulation::Edge_circulator c = tria.incident_edges(v);
55                  do {
56                      if(!tria.is_infinite(c)) {
57                          Triangulation::Vertex_handle v1 = c->first->vertex((c->second + 1) % 3);
58                          Triangulation::Vertex_handle v2 = c->first->vertex((c->second + 2) % 3);
59
60                          K::FT candidateDist = Segment(v1->point(),v2->point()).squared_length();
61                          if(firstEdge == true || minDist > candidateDist) {
62                              minDist = candidateDist;
63                              // std::cout << "canididate minimal distance for "<< v->point() << " between " << ⤸
                                    ↳ v1->point() << " and " << v2->point() << ": " << minDist << std::endl;
64                              firstEdge = false;
65                          }
66                      }
67                  } while(++c != tria.incident_edges(v));
68                  //other bacterium extends as well
69                  minDist = sqrt(minDist)/2;
70                  // std::cout << "tentative minimal distance for "<< v->point() <<": " << minDist << std::endl;
71
72                  //compare with dish
```

```cpp
73              for(std::vector<Segment>::iterator s = dish.begin(); s!=dish.end(); s++) {
74                  K::FT candidateDist = sqrt(squared_distance(v->point(),*s));
75                  if(firstEdge == true || minDist > candidateDist) {
76                      minDist = candidateDist;
77                      firstEdge = false;
78                  }
79
80              }
81              //compute time
82              K::FT time = sqrt(minDist-0.5);
83
84              q.push(-time);
85              // std::cout << "minimal distance for "<< v->point() <<": " << minDist << std::endl;
86
87          }
88
89      } else {
90          bool firstEdge = true;
91          K::FT minDist;
92
93          //compare with dish
94          for(std::vector<Segment>::iterator s = dish.begin(); s!=dish.end(); s++) {
95              K::FT candidateDist = squared_distance(bacteria[0],*s);
96              if(firstEdge == true || minDist > candidateDist) {
97                  minDist = candidateDist;
98                  bool firstEdge = false;
99              }
100
101          }
102          // std::cout << "minimal distance: " << minDist << std::endl;
103          //compute time
104          K::FT time = sqrt(sqrt(minDist)-0.5);
105
106          q.push(-time);
107
108      }
109
110      std::cout << ceil_to_double(-q.top()) << "␣";
111      int mid = numBacteria/2;
112      for(int i=0; i < mid; i++) {
113          q.pop();
114          // std::cout << "1popped" << std::endl;
115      }
116      std::cout << ceil_to_double(-q.top()) << "␣";
117      // std::cout << "mid: " << mid << "numBacteria: " << numBacteria  << std::endl;
118      for(int i=mid; i < numBacteria-1; i++) {
119          q.pop();
120          // std::cout << "2popped" << std::endl;
121      }
122      std::cout << ceil_to_double(-q.top());
123      std::cout << std::endl;
124  }
125 }
```

Graypes

```
1   // #include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
2   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
3   #include <CGAL/Delaunay_triangulation_2.h>
4   #include <iostream>
5   #include <cmath>
6
7   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8   typedef CGAL::Delaunay_triangulation_2<K>  Triangulation;
9   typedef Triangulation::Edge_iterator  Edge_iterator;
10  typedef CGAL::Segment_2<K> Segment;
11
12  double ceil_to_double(const K::FT& x) {
13      double a = std::ceil(CGAL::to_double(x));
14      while(a<x) a+=1;
15      while(a-1>=x) a-=1;
16      return a;
17  }
18
19  int main() {
20      std::ios_base::sync_with_stdio(false);
21
22      int numGraypes;
23      while(std::cin >> numGraypes) {
24          if(numGraypes==0) break;
25          std::vector<K::Point_2> graype(numGraypes);
26          Triangulation t;
27          for(int i=0; i<numGraypes;i++) {
28              double x, y;
29              std::cin >> x >> y;
30              graype[i] = K::Point_2(x,y);
31          }
32          t.insert(graype.begin(), graype.end());
33
34          Segment minEdge;
35          bool firstEdge = true;
36          // output all edges
37          for(Edge_iterator e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
38              if(firstEdge == true || minEdge.squared_length() > t.segment(e).squared_length()) {
39                  firstEdge = false;
40                  minEdge = t.segment(e);
41              }
42          }
43          std::cout << ceil_to_double(50*sqrt(minEdge.squared_length())) << std::endl;
44      }
45  }
```

H1N1

```
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <CGAL/Triangulation_face_base_with_info_2.h>
4   #include <iostream>
5   #include <vector>
6   #include <algorithm>
7
8   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
9   typedef CGAL::Triangulation_vertex_base_2<K> Vb;
10  typedef CGAL::Triangulation_face_base_with_info_2<K::FT,K> Fb;
11  typedef CGAL::Triangulation_data_structure_2<Vb,Fb> Tds;
12  typedef CGAL::Delaunay_triangulation_2<K,Tds>  Triangulation;
13  typedef Triangulation::Face_iterator Face_iterator;
14  typedef Triangulation::All_vertices_iterator Vertex_iterator;
15  typedef Triangulation::Face_handle Face_handle;
16  typedef Triangulation::Edge Edge;
17  typedef CGAL::Point_2<K> Point;
18
19
20  struct faceComparator {
21      bool operator() (const Face_handle &x, const Face_handle &y) const {
22          return (x->info() > y->info());
23      }
24  };
25
26  bool isInfinite(Triangulation &t, Face_handle &f) {
27      return t.is_infinite(Edge(f,0))
28              || t.is_infinite(Edge(f,1))
29              || t.is_infinite(Edge(f,2));
30
31  }
32
33  bool has_infinite_vertex(Triangulation &t, Face_handle &f) {
34      return t.is_infinite(f->vertex(0))
35          || t.is_infinite(f->vertex(1))
36          || t.is_infinite(f->vertex(2));
37  }
38
39  void initFaces(Triangulation &t, int startValue) {
40      //initialize escape radius
41      for (Triangulation::All_faces_iterator i = t.all_faces_begin();
42              i != t.all_faces_end(); i++) {
43          i->info() = K::FT(startValue);
44      }
45  }
46
47
48  void maxEscapeRadiusPerFace(Triangulation &t) {
49      //initialize escape radius
50      for (Face_iterator i = t.finite_faces_begin();
51              i != t.finite_faces_end(); i++) {
52          i->info() = K::FT(0);
53      }
54
55
56      //initialize boundary
57      Triangulation::Face_circulator fib = t.incident_faces(t.infinite_vertex());
58      Triangulation::Face_circulator fie = fib;
59
60      do {
61          for(int i=0; i<3; i++) {
62              if(!t.is_infinite(fib->neighbor(i))) {
63                  fib->neighbor(i)->info()
64                      = t.segment(Triangulation::Edge(fib,i)).squared_length()/4;
65                  Face_handle mf = fib->neighbor(i);
66                  break;
67              }
68          }
69      } while(++fib != fie);
70
71      //build search data structure -> Dijkstra
72      std::set<Face_handle, faceComparator> faces;
73      for(Face_iterator i=t.finite_faces_begin();
```

```
74              i!=t.finite_faces_end(); i++) {
75          faces.insert(Face_handle(i));
76      }
77
78      while(!faces.empty()) {
79          //remove face with the best escape radius
80          Face_handle mf = *(faces.begin());
81          faces.erase(faces.begin());
82          // std::cout << mf->info() << ": " << mf->vertex(1)->point() << "," << mf->vertex(2)->point() << "," << ↙
                  ↘ mf->vertex(3)->point() << std::endl;
83          //and relax all edges
84          for(int i=0; i<3; i++) {
85              if(!t.is_infinite(mf->neighbor(i))) {
86                  K::FT tentativeRadius = t.segment(Edge(mf,i)).squared_length()/4;
87                  K::FT w;
88                  if(mf->info() < tentativeRadius) {
89                      w = mf->info();
90                  } else {
91                      w = tentativeRadius;
92                  }
93
94                  if(w > mf->neighbor(i)->info()) {
95                      faces.erase(mf->neighbor(i));
96                      mf->neighbor(i)->info() = w;
97                      faces.insert(mf->neighbor(i));
98                  }
99              }
100         }
101     }
102 }
103
104
105 bool canEscape(Triangulation &t, Point &p, K::FT &squaredRadius) {
106     if(squaredRadius <= 0) {
107         std::cout << "one" << std::endl;
108         return true;
109     }
110
111     Face_handle f = t.locate(p);
112
113     // std::cout << CGAL::squared_distance(p, t.nearest_vertex(p,f)->point()) << " " << squaredRadius << std::endl;
114     if(CGAL::squared_distance(p, t.nearest_vertex(p,f)->point()) < squaredRadius) {
115         // std::cout << "three" << std::endl;
116         return false;
117     }
118
119     if(t.is_infinite(f)) {
120         // std::cout << "two" << std::endl;
121         return true;
122     }
123
124     std::cout << f->info() << ": " << f->vertex(0)->point() << "," << f->vertex(1)->point() << "," << ↙
                  ↘ f->vertex(2)->point() << (has_infinite_vertex(t,f) ? " has infinite vertex" : "") << std::endl;
125     return(f->info() >= squaredRadius);
126 }
127
128 bool canEscapeSlow(Triangulation &t, Point &s, K::FT &r, int queryNumber) {
129     int i = queryNumber;
130     if(r <= 0)
131         return true;
132     Face_handle f = t.locate(s);
133     if(CGAL::squared_distance(s, t.nearest_vertex(s, f)->point()) < r)
134         return false;
135
136     // DFS
137     std::vector<Face_handle> stack;
138     stack.push_back(f);
139     f->info() = i;
140     while(!stack.empty()) {
141         f = stack.back();
142         stack.pop_back();
143         // std::cout << f->info() << ": " << f->vertex(0)->point() << "," << f->vertex(1)->point() << "," << ↙
                  ↘ f->vertex(2)->point() << (has_infinite_vertex(t,f) ? " has infinite vertex" : "") << std::endl;
144         if(t.is_infinite(f))
145             return true;
146         for(int j = 0; j < 3; ++j) {
```

```cpp
147              if(f->neighbor(j)->info() < i
148                      && t.segment(Triangulation::Edge(f,j)).squared_length() >= 4* r) {
149                  stack.push_back(f->neighbor(j));
150                  // std::cout << "push back" << std::endl;
151                  f->neighbor(j)->info() = i;
152              }
153          }
154      }
155      // std::cout << "four" << std::endl;
156      return false;
157  }
158
159  int main() {
160      std::ios_base::sync_with_stdio(false);
161
162      int numVertices;
163      while(std::cin >> numVertices) {
164          if(numVertices <= 0) break;
165
166          std::vector<Point> verticesList(numVertices);
167          for(int i=0; i<numVertices; i++) {
168              int x,y;
169              std::cin >> x >> y;
170              Point p(x,y);
171              verticesList[i] = p;
172          }
173
174          Triangulation t;
175          t.insert(verticesList.begin(), verticesList.end());
176          // maxEscapeRadiusPerFace(t);
177
178          // std::cout << "faces: " << std::endl;
179          // for(Face_iterator i = t.faces_begin(); i !=t.faces_end(); i++) {
180          //     std::cout << i->info() << ": " << i->vertex(1)->point() << "," << i->vertex(2)->point() << "," << ↙
181                  ↳ i->vertex(3)->point() << std::endl;
181          // }
182          // std::cout << "faces end " << std::endl;
183          initFaces(t,-1);
184
185          int numCircles;
186          std::cin >> numCircles;
187          for(int i=0; i<numCircles; i++) {
188              int x,y;
189              K::FT squaredRadius;
190              std::cin >> x >> y >> squaredRadius;
191              Point p(x,y);
192              std::cout << (canEscapeSlow(t, p, squaredRadius,i) ? "y" : "n");
193          }
194
195          std::cout << std::endl;
196      }
197  }
```

Hiking Maps

```
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <iostream>
3   #include <set>
4   #include <vector>
5   #include <queue>
6   #include <limits>
7
8   using namespace std;
9
10  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
11  typedef K::Point_2 Point;
12  typedef pair<Point, Point> Segment;
13  typedef vector<Point> Triangle;
14
15  pair<int,int> minOfVector(vector<int> &v) {
16          int min = numeric_limits<int>::max();
17          int minPos;
18          for(vector<int>::iterator it=v.begin(); it!= v.end(); it++) {
19                  if(min > *it) {
20                          min = *it;
21                          minPos = it-v.begin();
22                  }
23          }
24          return pair<int,int>(min,minPos);
25  }
26
27  bool contained(const Segment &s, const Triangle &t) {
28      //test whether endpoints of the segments have the same orientation to
29      //the points on the side of the triangle as those points have to each
30      //other (same orientation or collinear)
31
32      bool right1 = CGAL::right_turn(t[0], t[1], t[2]);
33      bool right2 = CGAL::right_turn(t[2], t[3], t[4]);
34      bool right3 = CGAL::right_turn(t[4], t[5], t[0]);
35
36      CGAL::Orientation orient1 = !right1 ? CGAL::RIGHT_TURN : CGAL::LEFT_TURN;
37      CGAL::Orientation orient2 = !right2 ? CGAL::RIGHT_TURN : CGAL::LEFT_TURN;
38      CGAL::Orientation orient3 = !right3 ? CGAL::RIGHT_TURN : CGAL::LEFT_TURN;
39
40
41      bool ret = CGAL::orientation(t[0], t[1], s.first) != orient1
42                  && CGAL::orientation(t[2], t[3], s.first) != orient2
43                  && CGAL::orientation(t[4], t[5], s.first) != orient3
44                  && CGAL::orientation(t[0], t[1], s.second) != orient1
45                  && CGAL::orientation(t[2], t[3], s.second) != orient2
46                  && CGAL::orientation(t[4], t[5], s.second) != orient3;
47      return ret;
48  }
49
50  void myApproach(const vector<Segment>& legs, int numLegs, int numMaps) {
51      vector<Triangle > maps(numMaps, Triangle(6));
52      int minInterval = numeric_limits<int>::max();
53      vector<int> newestCover(numLegs); //map from leg to triangle
54      vector<bool> isCovered(numLegs, false); //for the beginning to test whether all are overed
55      int numNotCovered = numLegs;
56
57      int curInterval = 0;
58      bool mustRecompute = true;
59      int curMinIndex = -1;
60
61      for(int i=0;i<numMaps;i++) {
62          for(int j=0; j<6; j++) {
63              int x,y;
64              cin >> x >> y;
65              maps[i][j] = Point(x,y);
66          }
67
68          //test intersection
69          for(int j=0; j<legs.size(); j++) {
70              if(contained(legs[j], maps[i])) {
71                  newestCover[j] = i;
72                  if(j==curMinIndex) mustRecompute = true;
73                  if(isCovered[j] == false) {
```

```
74                    isCovered[j] = true;
75                    numNotCovered--;
76                }
77            }
78
79        }
80        if(numNotCovered > 0)
81            continue;
82
83        if(mustRecompute) {
84            pair<int,int> resultOfMin = minOfVector(newestCover);
85            curInterval = i-resultOfMin.first+1;
86            curMinIndex = resultOfMin.second;
87            mustRecompute = false;
88            if(curInterval < minInterval)
89                minInterval=curInterval;
90
91        }
92    }
93
94    cout << minInterval << endl;
95
96 }
97
98 void benApproach(const vector<Segment> &legs, int numLegs, int numMaps) {
99     vector<Triangle > maps(numMaps, Triangle(6));
100    int minInterval = numeric_limits<int>::max();
101    vector<queue<int> > coveredBy(numLegs); //map from leg to triangle
102
103    for(int i=0;i<numMaps;i++) {
104        for(int j=0; j<6; j++) {
105            int x,y;
106            cin >> x >> y;
107            maps[i][j] = Point(x,y);
108        }
109
110        //test intersection
111        for(int j=0; j<legs.size(); j++) {
112            if(contained(legs[j], maps[i])) {
113                // cout<<"leg "<<j<<" covered by map "<< i<<std::endl;
114                coveredBy[j].push(i);
115            }
116        }
117    }
118
119    //find the shortest interval s.t. all legs are covered
120    priority_queue<int> maxMapSTAllCovered;
121    priority_queue<pair<int,int>,vector<pair<int,int> >, greater<pair<int,int> > >
122        minMapSTAllCovered;
123
124    for(int i=0; i<numLegs; i++) {
125        int val = coveredBy[i].front(); coveredBy[i].pop();
126        maxMapSTAllCovered.push(val);
127        minMapSTAllCovered.push(make_pair(val,i));
128    }
129
130    while(true) {
131        int minMap, minMapLeg;
132        boost::tie(minMap,minMapLeg)
133            = minMapSTAllCovered.top(); minMapSTAllCovered.pop();
134        int maxMap = maxMapSTAllCovered.top(); //maxMapSTAllCovered.pop();
135
136        int curInterval = maxMap - minMap + 1;
137        minInterval = min(curInterval, minInterval);
138
139        if(coveredBy[minMapLeg].empty()) break;
140
141        int newMapInInterval = coveredBy[minMapLeg].front();
142        coveredBy[minMapLeg].pop();
143        maxMapSTAllCovered.push(newMapInInterval);
144        minMapSTAllCovered.push(make_pair(newMapInInterval, minMapLeg));
145    }
146
147    cout << minInterval << endl;
148
149 }
```

```
151  int main() {
152      ios_base::sync_with_stdio(false);
153
154      int numTestCases;
155      cin >> numTestCases;
156
157      for(int t=0; t<numTestCases; t++) {
158          int numLegs, numMaps;
159          cin >> numLegs >> numMaps;
160
161          numLegs-=1;
162          vector<Segment> legs(numLegs);
163          int x,y;
164          cin >> x>>y;
165          Point p1(x,y);
166          for(int i=0;i<numLegs;i++) {
167              cin >> x >> y;
168              Point p2(x,y);
169              legs[i]=Segment(p1,p2);
170              p1 = p2;
171          }
172
173          //100points
174          // myApproach(legs, numLegs, numMaps);
175          //gives only 80 points for me, but interesting sweepline algorithm
176          benApproach(legs, numLegs, numMaps);
177      }
178  }
```

Maximize It!

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>

// choose exact integral type
#ifdef CGAL_USE_GMP
#include <CGAL/Gmpz.h>
typedef CGAL::Gmpz ET;
#else
#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;
#endif

// program and solution types
typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

double ceil_to_double(const CGAL::Quotient<ET>& x) {
    double a = std::ceil(CGAL::to_double(x));
    while(a<x) a+=1;
    while(a-1>=x) a-=1;
    return a;
}

double floor_to_double(const CGAL::Quotient<ET>& x) {
    double a = std::floor(CGAL::to_double(x));
    while(a>x) a-=1;
    while(a+1<=x) a+=1;
    return a;
}

int main() {

  int problemType;
  while(std::cin >> problemType) {
        if(problemType==0) break;
        int a,b;
        std::cin >> a >> b;

        const int X = 0;
        const int Y = 1;
        Program lp (CGAL::SMALLER, false, 0, false, 0);

        if(problemType==1) {
            //minimize the negative problem
            lp.set_c(Y, -b);
            lp.set_d(X,X,2*a);

            lp.set_l(X,true,0);
            lp.set_l(Y,true,0);

            lp.set_a(X,0,1); lp.set_a(Y,0,1); lp.set_b(0, 4);
            lp.set_a(X,1,4); lp.set_a(Y,1,2); lp.set_b(1, a*b);
            lp.set_a(X,2,-1); lp.set_a(Y,2,1); lp.set_b(2, 1);
        } else {
            const int Z = 2;
            lp.set_d(X,X,2*a);
            lp.set_d(Z,Z,2);
            lp.set_c(Y, b);

            lp.set_u(X,true,0);
            lp.set_l(X,false);
            lp.set_u(Y,true,0);
            lp.set_l(Y,false);
            lp.set_u(Z,false);
            lp.set_l(Z,false);

            lp.set_a(X,0,1); lp.set_a(Y,0,1); lp.set_b(0, -4);
            lp.set_r(0, CGAL::LARGER);
            lp.set_a(X,1,4); lp.set_a(Y,1,2); lp.set_a(Z,1,1);
            lp.set_b(1, -a*b);
```

```
74              lp.set_r(1, CGAL::LARGER);
75              lp.set_a(X,2,-1); lp.set_a(Y,2,1); lp.set_b(2, -1);
76              lp.set_r(2, CGAL::LARGER);
77
78          }
79          Solution s= CGAL::solve_quadratic_program(lp, ET());
80          assert(s.solves_quadratic_program(lp));
81
82          // std::cout << s;
83
84          if(s.is_infeasible())
85              std::cout << "no" << std::endl;
86          else if(s.is_unbounded())
87              std::cout << "unbounded" << std::endl;
88          else {
89              CGAL::Quotient<ET> exactValue = s.objective_value();
90              int value = problemType==1 ?
91                              floor_to_double(-exactValue) :
92                              ceil_to_double(exactValue);
93
94              std::cout << value << std::endl;
95          }
96      }
97  }
```

Collisions

```
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <iostream>
4   #include <vector>
5
6   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7   typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
8   typedef Triangulation::Vertex_iterator Vertex_iterator;
9   typedef K::Point_2 Point;
10  typedef CGAL::Segment_2<K> Segment;
11
12  int main() {
13      std::ios_base::sync_with_stdio(false);
14      int testCases;
15      std::cin >> testCases;
16      for(int i=0; i<testCases; i++) {
17          int numPlanes;
18          K::FT minDistance;
19          std::cin >> numPlanes>> minDistance;
20
21      std::vector<Point> planes(numPlanes);
22
23      Triangulation t;
24
25          for(int j=0; j<numPlanes;j++) {
26              int x, y;
27              std::cin >> x >> y;
28              //t.insert(Triangulation::Point(x,y));
29              planes[j] = Point(x,y);
30          }
31
32      t.insert(planes.begin(), planes.end());
33
34      int numEndangered = 0;
35      K::FT minSquaredDistance = minDistance * minDistance;
36
37      for(Vertex_iterator v=t.finite_vertices_begin();
38          v!=t.finite_vertices_end(); v++) {
39          // std::cout<<v->point() << std::endl;
40          Triangulation::Edge_circulator c = t.incident_edges(v);
41          K::FT minDist;
42          bool firstEdge = true;
43
44          do {
45            if(!t.is_infinite(c)) {
46              Triangulation::Vertex_handle v1 = c->first->vertex((c->second+1)%3);
47              Triangulation::Vertex_handle v2 = c->first->vertex((c->second+2)%3);
48
49              K::FT candidateDist = Segment(v1->point(), v2->point()).squared_length();
50              // std::cout << "candidate min dist for " << v->point() << " between " << v1->point() << " " << ↲
                           ↳ v2->point() << " is " << candidateDist << std::endl;
51              if(firstEdge == true || minDist > candidateDist) {
52                minDist = candidateDist;
53                firstEdge = false;
54              }
55            }
56          } while(++c != t.incident_edges(v));
57
58          if(!firstEdge && minDist < minSquaredDistance)
59          numEndangered++;
60      }
61
62      std::cout << numEndangered << std::endl;
63
64      }
65  }
```

Diet

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>

// choose exact integral type
#ifdef CGAL_USE_GMP
#include <CGAL/Gmpz.h>
typedef CGAL::Gmpz ET;
#else
#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;
#endif

// program and solution types
typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

double ceil_to_double(const CGAL::Quotient<ET>& x) {
    double a = std::ceil(CGAL::to_double(x));
    while(a<x) a+=1;
    while(a-1>=x) a-=1;
    return a;
}

double floor_to_double(const CGAL::Quotient<ET>& x) {
    double a = std::floor(CGAL::to_double(x));
    while(a>x) a-=1;
    while(a+1<=x) a+=1;
    return a;
}

int main() {

  int numNutrients, numFoods;
  while(std::cin >> numNutrients >> numFoods) {
      if(numNutrients == 0 && numFoods == 0)
          break;
      Program lp (CGAL::LARGER, true, 0, false, 0);
      for(int nutrient=0;nutrient<numNutrients;nutrient++) {
          int l,u;
          std::cin >> l >> u;
          lp.set_b(2*nutrient, l);
          lp.set_b(2*nutrient+1, u);
          lp.set_r(2*nutrient+1, CGAL::SMALLER);
      }

      for(int food=0;food<numFoods;food++) {
          int price;
          std::cin >> price;
          for(int nutrient=0; nutrient<numNutrients; nutrient++) {
              int price,C;
              std::cin >> C;
              lp.set_a(food, 2*nutrient, C);
              lp.set_a(food, 2*nutrient+1, C);
          }
          lp.set_c(food, price);
      }

      Solution s= CGAL::solve_quadratic_program(lp, ET());
      assert(s.solves_quadratic_program(lp));

      if(!s.is_optimal())
          std::cout << "No such diet." << std::endl;
      else
          std::cout << floor_to_double(s.objective_value()) << std::endl;
  }
}
```

Porfolios

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6
7  // choose exact integral type
8  #ifdef CGAL_USE_GMP
9  #include <CGAL/Gmpz.h>
10 typedef CGAL::Gmpz ET;
11 #else
12 #include <CGAL/MP_Float.h>
13 typedef CGAL::MP_Float ET;
14 #endif
15
16 // program and solution types
17 typedef CGAL::Quadratic_program<int> Program;
18 typedef CGAL::Quadratic_program_solution<ET> Solution;
19
20 int main() {
21
22     int numAssets, numPeople;
23     while(std::cin >> numAssets >> numPeople) {
24         if(numAssets == 0 && numPeople == 0)
25             break;
26
27         Program qp (CGAL::SMALLER, true, 0, false, 0);
28         for(int i=0; i<numAssets; i++) {
29             int cost, expectedReturn;
30             std::cin >> cost >> expectedReturn;
31             qp.set_a(i, 0, expectedReturn);
32             qp.set_r(0, CGAL::LARGER);
33             qp.set_a(i, 1, cost);
34         }
35         for(int i=0; i<numAssets; i++) {
36             for(int j=0; j<numAssets; j++) {
37                 int covariance;
38                 std::cin >> covariance;
39                 if(j<=i)
40                     qp.set_d(i,j,2*covariance);
41             }
42
43         }
44         for(int i=0; i<numPeople; i++) {
45             int maxCost, minReturn, maxVariance;
46             std::cin >> maxCost >> minReturn >> maxVariance;
47             qp.set_b(0, minReturn);
48             qp.set_b(1, maxCost);
49             Solution s= CGAL::solve_quadratic_program(qp, ET());
50             assert(s.solves_quadratic_program(qp));
51
52             if(!s.is_optimal() || s.objective_value() > maxVariance)
53                 std::cout << "No." << std::endl;
54             else
55                 std::cout << "Yes." << std::endl;
56         }
57     }
58 }
```

Inball

```
1   #include <iostream>
2   #include <cassert>
3   #include <CGAL/basic.h>
4   #include <CGAL/QP_models.h>
5   #include <CGAL/QP_functions.h>
6
7   // choose exact integral type
8   #ifdef CGAL_USE_GMP
9   #include <CGAL/Gmpz.h>
10  typedef CGAL::Gmpz ET;
11  #else
12  #include <CGAL/MP_Float.h>
13  typedef CGAL::MP_Float ET;
14  #endif
15
16  // program and solution types
17  typedef CGAL::Quadratic_program<int> Program;
18  typedef CGAL::Quadratic_program_solution<ET> Solution;
19
20  using namespace std;
21
22  double floor_to_double(const CGAL::Quotient<ET>& x) {
23      double a = std::floor(CGAL::to_double(x));
24      while(a>x) a-=1;
25      while(a+1<=x) a+=1;
26      return a;
27  }
28
29  int main() {
30      ios_base::sync_with_stdio(false);
31
32      int numInequalities, numDimensions;
33      while(true) {
34          cin >> numInequalities;
35          if(numInequalities==0) break;
36          cin >> numDimensions;
37
38          Program lp (CGAL::SMALLER, false, 0, false, 0);
39          for(int i=0; i<numDimensions; i++) {
40              lp.set_c(i,0);
41          }
42          lp.set_c(numDimensions,-1);
43
44          for(int i=0; i<numInequalities; i++) {
45              long normA=0;
46              for(int j=0; j<numDimensions; j++) {
47                  int a;
48                  cin >> a;
49                  normA = normA + a*a;
50                  lp.set_a(j,i,a);
51              }
52              lp.set_a(numDimensions, i, sqrt(normA));
53              int b;
54              cin >> b;
55              lp.set_b(i,b);
56          }
57          lp.set_l(numDimensions, true, 0);
58          Solution s= CGAL::solve_quadratic_program(lp, ET());
59          assert(s.solves_quadratic_program(lp));
60
61          if(s.is_infeasible())
62              std::cout << "none" << std::endl;
63          else if(s.is_unbounded())
64              std::cout << "inf" << std::endl;
65          else {
66              CGAL::Quotient<ET> exactValue = s.objective_value();
67              cout << floor_to_double(-exactValue) << endl;
68          }
69      }
70  }
```

Monkey Island

```cpp
#include <iostream>
#include <vector>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/strong_components.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, directedS> Graph;
typedef graph_traits<Graph> Traits;
typedef Traits::edge_descriptor Edge;
typedef Traits::vertex_descriptor Vertex;
typedef Traits::edge_iterator Eit;

int main() {
    ios_base::sync_with_stdio(false);
    int testCases = 0;
    cin >> testCases;
    for(int testCase=0; testCase < testCases; testCase++) {
        int numLocations, numRoads;
        cin >> numLocations >> numRoads;

        Graph g(numLocations);
        vector<int> cost(numLocations);
        for(int i=0;i<numRoads;i++) {
            Edge e;
            int u,v;
            cin >> u >> v;
            tie(e, tuples::ignore) = add_edge(u-1,v-1,g);
        }
        for(int i=0;i<numLocations;i++) {
            int c;
            cin >> c;
            cost[i] = c;
        }

        vector<int> componentMap(numLocations);
        int numComponents = strong_components(g, &componentMap[0]);
        vector<int> componentCosts(numComponents, 100);

        // cout << "numcomp " << numComponents << endl;

        for(int i=0; i<numLocations; i++) {
            if(cost[i] < componentCosts[componentMap[i]]) {
                componentCosts[componentMap[i]] = cost[i];
            }
        }

        Eit ei, ei_end;
        for(tie(ei, ei_end) = edges(g); ei != ei_end; ++ei) {
            Vertex sV = source(*ei,g);
            Vertex tV = target(*ei,g);

            if(componentMap[sV] != componentMap[tV]) {
                componentCosts[componentMap[tV]] = 0;
            }
        }

        int minPStation = 0;
        for(int i=0;i<numComponents;i++) {
            minPStation += componentCosts[i];
        }
        cout << minPStation << endl;
    }
}
```

Placing Knights

```
1   #include <iostream>
2   #include <vector>
3   #include <set>
4   #include <cstdlib>
5   #include <boost/graph/max_cardinality_matching.hpp>
6   #include <boost/tuple/tuple.hpp>
7   #include <boost/graph/adjacency_list.hpp>
8   #include <boost/config.hpp>
9
10  using namespace std;
11  using namespace boost;
12
13  typedef pair<int,int> Pos;
14  typedef adjacency_list<setS, vecS, undirectedS > Graph;
15  typedef graph_traits<Graph> Traits;
16  typedef Traits::vertex_descriptor Vertex;
17  typedef Traits::edge_descriptor Edge;
18  typedef property_map<Graph, vertex_index_t>::type IndexMap;
19
20
21  //returns the knights that the given knight threatens
22  vector<Pos> threatens(vector<vector<int> > &board, Pos &knight) {
23      vector<Pos> ret;
24      for(int i=-2;i<=2;i++) {
25          for(int j=-2;j<=2;j++) {
26              if(!((abs(i)== 1 && abs(j) ==2)
27                      || (abs(i)== 2 && abs(j) ==1))) continue; //move not allowed
28              unsigned int newX = knight.first+i;
29              unsigned int newY = knight.second+j;
30              if(0 <= newX && newX < board.size()
31                      && 0 <= newY && newY < board.size()
32                      && board[newX][newY] != 0) {
33                  ret.push_back(Pos(newX, newY));
34              }
35          }
36      }
37      return ret;
38  }
39
40  void setToZero(vector<vector<int> > &board, vector<Pos> &threatened) {
41      for(unsigned int i=0; i<threatened.size(); i++) {
42          Pos pos = threatened[i];
43          board[pos.first][pos.second] = 0;
44      }
45  }
46
47  void printBoard(vector<vector<int> > &board) {
48      for(unsigned int i=0; i<board.size(); i++) {
49          for(unsigned int j=0; j<board.size(); j++) {
50              cout << board[i][j];
51          }
52          cout << endl;
53      }
54  }
55
56
57  void reduceNumThreatened(set<pair<int, pair<int, int> > > &Q, vector<vector<int> > &board, vector<vector<int> > > ↗
        ↘ &boardThreatening, vector<Pos> &threatened) {
58      for(unsigned int i=0; i<threatened.size(); i++) {
59          Pos pos = threatened[i];
60          int numThreatens = boardThreatening[pos.first][pos.second];
61          Q.erase(pair<int, Pos>(numThreatens, pos));
62
63          vector<Pos> toReduce = threatens(board, pos);
64          for(unsigned int j=0; j<toReduce.size(); j++) {
65              Pos toReducePos = toReduce[j];
66              numThreatens = boardThreatening[toReducePos.first][toReducePos.second];
67              Q.erase(pair<int, Pos>(numThreatens, toReducePos));
68              numThreatens--;
69              Q.insert(pair<int, Pos>(numThreatens, toReducePos));
70              boardThreatening[toReducePos.first][toReducePos.second] = numThreatens;
71          }
72      }
```

```cpp
73    }
74
75    int numKnights(vector<vector<int> > board) {
76        int numKnights = 0;
77        int numPosLoc = 0;
78        set<pair<int, pair<int, int> > > Q;
79        int n = board.size();
80        vector<vector<int> > boardThreatening(n,vector<int>(n));
81
82        for(int i=0;i<n;i++) {
83            for(int j=0;j<n;j++) {
84                if(board[i][j] != 0) {
85                    Pos pos =Pos(i,j);
86                    int numThreatening = threatens(board, pos).size();
87                    Q.insert(pair<int, pair<int, int> >(numThreatening,pos));
88                    boardThreatening[pos.first][pos.second] = numThreatening;
89                    numPosLoc++;
90                }
91            }
92        }
93        while(Q.size() > 0 && numPosLoc > 0) {
94            // cout<< "board" << std::endl;
95            // printBoard(board);
96            // cout<<"num threatening" <<std::endl;
97            // printBoard(boardThreatening);
98            pair<int, pair<int, int> > p = *(Q.begin());
99            Q.erase(Q.begin());
100           pair<int, int> pos = p.second;
101           if(board[pos.first][pos.second] != 0) {
102               vector<Pos> threatened = threatens(board, pos);
103               setToZero(board, threatened);
104               board[pos.first][pos.second] = 0;
105               numPosLoc -= (threatened.size() + 1);
106               numKnights++;
107               threatened.push_back(pos);
108               reduceNumThreatened(Q, board, boardThreatening, threatened);
109           }
110       }
111       return numKnights;
112   }
113
114   int posToInt(pair<int,int> pos, int n) {
115       return pos.first*n+pos.second;
116   }
117
118   int numKnightsCorrect(vector<vector<int> > board) {
119       Graph g;
120       int n = board.size();
121       int numVertices = 0;
122
123       //build bipartite graph
124       for(int i=0;i<n;i++) {
125           for(int j=0;j<n;j++) {
126               if(board[i][j] == 0) continue;
127               Pos curPos(i,j);
128               vector<Pos> threatened = threatens(board, curPos);
129               numVertices++;
130               for(int k=0; k<threatened.size(); k++) {
131                   Edge e;
132                   tie(e, tuples::ignore) = add_edge(
133                           posToInt(curPos,n),
134                           posToInt(threatened[k],n),g);
135               }
136           }
137       }
138       vector<Vertex> mate(num_vertices(g));
139       edmonds_maximum_cardinality_matching(g, &mate[0]);
140       const Vertex NULL_VERTEX = graph_traits<Graph>::null_vertex();
141       int numMatched = 0;
142       for(int i=0; i<mate.size(); i++) {
143           if(mate[i] != NULL_VERTEX) numMatched++;
144       }
145       return numVertices - (numMatched/2);
146   }
147
148   int main() {
```

```cpp
        ios_base::sync_with_stdio(false);
        int testCases;
        cin >> testCases;
        for(int t=0; t < testCases; t++) {
            int n;
            cin >> n;
            vector<vector<int> > board(n,vector<int>(n));

            for(int i=0;i<n;i++) {
                for(int j=0;j<n;j++) {
                    int status;
                    cin >> status;
                    board[i][j] = status;
                }
            }

            cout << numKnightsCorrect(board) << std::endl;

        }
        return 0;
    }
```

Shopping Trip

```cpp
1   #include <boost/config.hpp>
2   #include <boost/tuple/tuple.hpp>
3   #include <boost/graph/adjacency_list.hpp>
4   #include <boost/graph/push_relabel_max_flow.hpp>
5   #include <iostream>
6   #include <limits>
7
8   using namespace std;
9   using namespace boost;
10
11  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
12  typedef adjacency_list<vecS, vecS, directedS, no_property,
13    property<edge_capacity_t, long,
14    property<edge_residual_capacity_t, long,
15    property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
16  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
17  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
18  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
19  typedef property_map<Graph, vertex_index_t>::type IndexMap;
20
21  typedef graph_traits<Graph>::vertex_descriptor Vertex;
22  typedef graph_traits<Graph>::edge_descriptor Edge;
23  typedef graph_traits<Graph> GraphTraits;
24
25  void printGraph(Graph g, EdgeCapacityMap &capacity) {
26      graph_traits<Graph>::edge_iterator eiter, eiter_end;
27      for (tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
28          if(capacity[*eiter] > 0) {
29              int aSource = source(*eiter, g);
30              int aTarget = target(*eiter, g);
31              std::cout << aSource <<" "
32                  " -" << capacity[*eiter] << "-> " << aTarget
33                  << std::endl;
34          }
35      }
36  }
37
38  void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
39      Edge e, reverseE;
40      tie(e, tuples::ignore) = add_edge(u,v,g);
41      tie(reverseE, tuples::ignore) = add_edge(v, u, g);
42      capacity[e] = c;
43      capacity[reverseE] = 0;
44      rev_edge[e] = reverseE;
45      rev_edge[reverseE] = e;
46  }
47
48
49  int main() {
50      ios_base::sync_with_stdio(false);
51
52      int testCases = 0;
53      cin >> testCases;
54      for(int testCase=0; testCase < testCases; testCase++) {
55          int numVertices, numEdges, numStores;
56          cin >> numVertices >> numEdges >> numStores;
57
58          Graph g(numVertices);
59          EdgeCapacityMap capacity = get(edge_capacity, g);
60          ReverseEdgeMap rev_edge = get(edge_reverse, g);
61          ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
62
63          //add sink
64          const int SOURCE = 0;
65          //addFlowEdge(g, capacity, rev_edge, isStore, SINK, 1);
66          const int SINK = numVertices;
67
68          for(int i=0; i<numStores;i++) {
69              int isStore;
70              cin >> isStore;
71              addFlowEdge(g, capacity, rev_edge, isStore, SINK, 1);
72          }
73
```

```cpp
        for(int i=0; i<numEdges;i++) {
            int u,v;
            cin >> u >> v;
            addFlowEdge(g, capacity, rev_edge, u, v,1);
            addFlowEdge(g, capacity, rev_edge, v, u,1);
        }

        //printGraph(g, capacity) ;

        long flow = push_relabel_max_flow(g, SOURCE, SINK);

        cout << ((flow >= numStores) ? "yes" : "no") << endl;
    }
}
```

TheeV

```
1   #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2   #include <CGAL/Min_circle_2.h>
3   #include <CGAL/Min_circle_2_traits_2.h>
4   #include <iostream>
5
6   using namespace std;
7
8   typedef CGAL::Exact_predicates_exact_constructions_kernel K;
9   typedef CGAL::Min_circle_2_traits_2<K> Traits;
10  typedef CGAL::Min_circle_2<Traits> Min_circle;
11  typedef K::Point_2 P;
12  typedef K::Segment_2 S;
13
14  double ceil_to_double(const K::FT& x) {
15      double a = std::ceil(CGAL::to_double(x));
16      while(a<x) a+=1;
17      while(a-1>=x) a-=1;
18      return a;
19  }
20
21  struct CityComparator{
22      P mainCity;
23      bool operator()(P x, P y) {
24          K::FT d1 = S(mainCity, x).squared_length();
25          K::FT d2 = S(mainCity, y).squared_length();
26          return(d1 > d2);
27      }
28
29  };
30
31  int main() {
32      ios_base::sync_with_stdio(false);
33      int testCases;
34      cin>>testCases;
35
36      for(int t=0; t<testCases;t++) {
37          int numCities;
38          cin>>numCities;
39          vector<P> cities;
40
41
42          for(int i=0; i<numCities;i++) {
43              int x,y;
44              cin>>x>>y;
45              P p(x,y);
46              cities.push_back(p);
47          }
48
49          P mainCity = cities[0];
50
51          CityComparator comp;
52          comp.mainCity = mainCity;
53
54          sort(cities.begin(), cities.end(), comp);
55          //sort from highest distance to lowest
56
57          K::FT otherRadius;
58          K::FT mainCityRadius;
59          K::FT beforeMainCityRadius;
60          Min_circle otherRadio;
61          for(vector<P>::iterator i = cities.begin();
62                  i!=cities.end(); i++) {
63
64              // cout << "begin " << *(cities.begin()) << endl;
65              // cout << "i " << *i << endl;
66              // cout << "end " << *(cities.end()-1) << endl;
67              mainCityRadius = S(*(i+1), mainCity).squared_length();
68
69              otherRadio.insert(*i);
70              Traits::Circle otherCircle = otherRadio.circle();
71              otherRadius = otherCircle.squared_radius();
72
73              // cout << fixed<<setprecision(2)<<"radii " << mainCityRadius << " " << otherRadius << ":" << ⤸
```

```cpp
                        ↳ otherRadio.number_of_points()<< endl;

            if(otherRadius >= mainCityRadius)
                break;
            beforeMainCityRadius=mainCityRadius;
        }
        cout<<fixed<<setprecision(0)<<ceil_to_double(min(beforeMainCityRadius,otherRadius))<<endl;

    }

}
```

Poker Chips

```
1   #include <iostream>
2   #include <numeric>
3   #include <vector>
4   #include <map>
5   #include <bitset>
6
7   using namespace std;
8
9   int optimalPoints(vector<vector<int> > &chips,
10          map<vector<int>,int> &table, vector<int> topChipPosition){
11      int numStacks = chips.size();
12
13      map<vector<int>,int>::iterator found = table.find(topChipPosition);
14      if(found != table.end()) {
15          return found->second;
16      }
17
18      //return min over all subsets of possible taking
19      int maxPoints = 0;
20      int maxSubset = 0;
21      //iterate over all subsets of taking coins from the top
22      for(int s=1; s < (1<<numStacks); s++) {
23          int numChipsTaken = 0;
24          //s represents a subset of {0,...,n-1}
25          // cout<<"take subset " << std::bitset<5>(s).to_string() << std::endl;
26          vector<int> newTopChipPosition(numStacks);
27          int color=-1;
28          for(int k=0; k<numStacks; k++) {
29              //iterate over all elements of s
30              newTopChipPosition[k] = topChipPosition[k];
31              if((s & (1<<k)) != 0){
32                  // cout<<"stack " << k << " is in subset" << std::endl;
33                  if(topChipPosition[k]<0) {
34                      // cout<<"stack " << k << " is empty" << std::endl;
35                      continue;
36                  }
37
38                  //k is in S
39                  //ensure that colors are the same
40                  int stackColor = chips[k][topChipPosition[k]];
41                  if(color == -1) {
42                      color = stackColor;
43                      // cout<<"take color " << color << std::endl;
44                  } else if (color!=stackColor) {
45                      // cout<<"stack " << k << "has different color"<< std::endl;
46                      continue;
47                  }
48                  // cout<<"remove chip from stack " << k << std::endl;
49                  ++numChipsTaken;
50                  newTopChipPosition[k] = newTopChipPosition[k]-1;
51              }
52          }
53          int sumChips=0;
54          for(int i=0; i< numStacks; i++) {
55              sumChips +=newTopChipPosition[i]+1;
56          }
57
58          int newPoints = 0;
59          if(numChipsTaken > 1)
60              newPoints = (1<<(numChipsTaken-2));
61
62          if(sumChips > 0 && numChipsTaken>0)
63              newPoints = newPoints + optimalPoints(chips, table,
64                          newTopChipPosition);
65
66          // cout<<"points: "<<newPoints<<endl;
67          if(newPoints > maxPoints) {
68              maxPoints = newPoints;
69              maxSubset = s;
70          }
71      }
72      // cout<<"max points for this stack: "<< maxPoints<< " using "
73          // << std::bitset<5>(maxSubset).to_string()<< endl;
```

```
74          table[topChipPosition] = maxPoints;
75          return maxPoints;
76  }

77
78  int main() {
79          std::ios_base::sync_with_stdio(false);
80          int testCases;
81          cin >> testCases;

82
83          for(int k=0; k < testCases; k++) {
84                  int numStacks;
85                  cin >> numStacks;

86
87                  vector< vector<int> > chips(numStacks);
88                  vector<int> topChipPosition(numStacks);
89                  for(int i=0; i<numStacks; i++) {
90                          int stackHeight;
91                          cin >> stackHeight;
92                          chips[i] = vector<int>(stackHeight);
93                          topChipPosition[i] = stackHeight-1;
94                  }

95
96                  for(int i=0; i<numStacks; i++) {
97                          for(int j=0; j<chips[i].size(); j++) {
98                                  //the chip at the top of the stack is added last
99                                  int c;
100                                 cin >> c;
101                                 chips[i][j] = c;
102                         }
103                 }
104                 map<vector<int>, int> table;
105                 cout << optimalPoints(chips,table,topChipPosition) << endl;
106         }
107 }
```

Portfolios Revisited

```cpp
1   #include <iostream>
2   #include <cassert>
3   #include <CGAL/basic.h>
4   #include <CGAL/QP_models.h>
5   #include <CGAL/QP_functions.h>
6
7   // choose exact integral type
8   #ifdef CGAL_USE_GMP
9   #include <CGAL/Gmpz.h>
10  typedef CGAL::Gmpz ET;
11  #else
12  #include <CGAL/MP_Float.h>
13  typedef CGAL::MP_Float ET;
14  #endif
15
16  // program and solution types
17  typedef CGAL::Quadratic_program<int> Program;
18  typedef CGAL::Quadratic_program_solution<ET> Solution;
19
20  int midPoint(int lowerBound, int upperBound) {
21      return lowerBound + (upperBound-lowerBound)/2;
22  }
23
24  int main() {
25      int numAssets, numPeople;
26      while(std::cin >> numAssets >> numPeople) {
27          if(numAssets == 0 && numPeople == 0)
28              break;
29
30          Program qp (CGAL::SMALLER, true, 0, false, 0);
31          std::vector<int> costs(numAssets);
32          std::vector<int> returns(numAssets);
33          std::vector<double> returnPerCost(numAssets);
34          for(int i=0; i<numAssets; i++) {
35              int cost, expectedReturn;
36              std::cin >> cost >> expectedReturn;
37              qp.set_a(i, 0, expectedReturn);
38              qp.set_r(0, CGAL::LARGER);
39              qp.set_a(i, 1, cost);
40              costs[i] = cost;
41              returns[i] = expectedReturn;
42              returnPerCost[i] = (double)expectedReturn/(double)cost;
43          }
44          for(int i=0; i<numAssets; i++) {
45              for(int j=0; j<numAssets; j++) {
46                  int covariance;
47                  std::cin >> covariance;
48                  if(j<=i)
49                      qp.set_d(i,j,2*covariance);
50              }
51
52          }
53          for(int i=0; i<numPeople; i++) {
54              int maxCost, minReturn, maxVariance;
55              std::cin >> maxCost >> maxVariance;
56              qp.set_b(1, maxCost);
57
58              int indexMinCost = std::max_element(returnPerCost.begin(), returnPerCost.end()) - returnPerCost.begin();
59              long returnUpperBound = ceil(returns[indexMinCost]*maxCost/costs[indexMinCost])+1;
60
61              // the following lower bound is not correct, since it might not be possible to buy all high cost assets ↙
                        ↘ because this might exceed the risk
62              // long returnLowerBound = returns[indexMaxCost]*(double)maxCost/costs[indexMaxCost];
63
64              long returnLowerBound = 0;
65
66              // binary search
67              while(returnLowerBound <= returnUpperBound) {
68                  long maxReturn = midPoint(returnLowerBound,returnUpperBound);
69
70                  qp.set_b(0, maxReturn);
71                  Solution s= CGAL::solve_quadratic_program(qp, ET());
72                  assert(s.solves_quadratic_program(qp));
```

```
73              if(s.is_optimal() && s.objective_value() <= maxVariance) {
74                  returnLowerBound = maxReturn+1;
75              } else {
76                  returnUpperBound = maxReturn-1;
77              }
78          }
79          std::cout << returnUpperBound << std::endl;
80      }
81   }
82 }
```

Stamp Exhibition

```cpp
1   #include <iostream>
2   #include <utility>
3   #include <cmath>
4   #include <cassert>
5   #include <CGAL/basic.h>
6   #include <CGAL/QP_models.h>
7   #include <CGAL/QP_functions.h>
8   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
9
10  // choose exact integral type
11  #ifdef CGAL_USE_GMP
12  #include <CGAL/Gmpzf.h>
13  typedef CGAL::Gmpzf ET;
14  #else
15  #include <CGAL/MP_Float.h>
16  typedef CGAL::MP_Float ET;
17  #endif
18
19  using namespace std;
20
21  // program and solution types
22  typedef pair<int,int> Pos;
23  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
24  typedef CGAL::Quadratic_program<ET> Program;
25  typedef CGAL::Quadratic_program_solution<ET> Solution;
26  typedef K::Point_2 P;
27  typedef K::Segment_2 S;
28
29
30  int main() {
31      ios_base::sync_with_stdio(false);
32
33      int testCases;
34      cin>>testCases;
35      for(int tcase=0; tcase<testCases; tcase++)  {
36          Program lp (CGAL::SMALLER, true, 1, true, 4096);
37
38      int numLights, numStamps, numWalls;
39      cin >> numLights >> numStamps >> numWalls;
40      vector<P> lights(numLights);
41      vector<P> stamps(numStamps);
42      vector<S> walls(numWalls);
43      vector<long> stampIntensity(numStamps);
44
45      for(int l=0; l<numLights; l++) {
46          int x,y;
47          cin>>x>>y;
48          lights[l]=P(x,y);
49          lp.set_c(l,1);
50      }
51
52      for(int s=0; s<numStamps; s++) {
53          int x,y,intensity;
54          cin>>x>>y>>intensity;
55          stamps[s] = P(x,y);
56          stampIntensity[s] = intensity;
57
58      }
59
60      for(int w=0; w<numWalls; w++) {
61          int x1,y1,x2,y2;
62          cin>>x1>>y1>>x2>>y2;
63          walls[w] = S(P(x1,y1),P(x2,y2));
64      }
65
66
67      for(int s=0; s<numStamps; s++) {
68          for(int l=0; l<numLights; l++) {
69              S segment =S(stamps[s],lights[l]);
70              K::FT r2 = segment.squared_length();
71              double quotient2 = 1.0/r2;
72              lp.set_a(l,s,quotient2);
73              lp.set_a(l,numStamps+s,quotient2);
```

```
74              for(int w=0; w<numWalls; w++) {
75                  if(CGAL::do_intersect(walls[w], segment)) {
76                  // cout<<"stamp"<<s<<" lamp"<<l<<" " <<quotient2;
77                      lp.set_a(l,s,0);
78                      lp.set_a(l,numStamps+s,0);
79                      break;
80                  }
81
82              }
83          }
84          lp.set_b(s, stampIntensity[s]);
85          lp.set_r(numStamps+s, CGAL::LARGER);
86          lp.set_b(numStamps+s, 1);
87      }
88
89      Solution s=CGAL::solve_linear_program(lp,ET());
90      assert(s.solves_quadratic_program(lp));
91      // CGAL::print_linear_program(std::cerr, lp, "lp");
92
93      if(s.is_optimal())
94          cout<< "yes" << endl;
95      else
96          cout << "no" << endl;
97      }
98  }
```

Tetris

```cpp
#include <iostream>
#include <vector>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
typedef adjacency_list<vecS, vecS, directedS, no_property,
  property<edge_capacity_t, long,
  property<edge_residual_capacity_t, long,
  property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;

typedef graph_traits<Graph> GraphTraits;
typedef GraphTraits::vertex_descriptor Vertex;
typedef GraphTraits::edge_descriptor Edge;
typedef property_map<Graph, vertex_index_t>::type IndexMap;
typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;

void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
    Edge e, reverseE;
    tie(e, tuples::ignore) = add_edge(u,v,g);
    tie(reverseE, tuples::ignore) = add_edge(v, u, g);
    capacity[e] = c;
    capacity[reverseE] = 0;
    rev_edge[e] = reverseE;
    rev_edge[reverseE] = e;
}

void testCase() {
    int width, numBricks;
    cin >> width >> numBricks;

    Graph g;
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap rev_edge = get(edge_reverse, g);
    ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);


    // does not work for bordercases
    // for(int i=0; i<numBricks; i++) {
    //     int u,v;
    //     cin>>u>>v;
    //     int from = min(u,v);
    //     int to = max(u,v);
    //     addFlowEdge(g, capacity, rev_edge, 2*from+1, 2*to, 1);
    // }
    // for(int i=1; i<width; i++) {
    //     addFlowEdge(g, capacity, rev_edge, 2*i, 2*i+1, 1);
    // }

    for(int i=0; i<numBricks; i++) {
        int u,v;
        cin>>u>>v;
        int from = min(u,v);
        int to = max(u,v);
        if(from == 0) {
            addFlowEdge(g, capacity, rev_edge, from, to, 1);
        } else {
            addFlowEdge(g, capacity, rev_edge, from+width, to, 1);
        }
    }
    for(int i=1; i<width; i++) {
        addFlowEdge(g, capacity, rev_edge, i, i+width, 1);
    }

    long flow = push_relabel_max_flow(g, 0, width);
```

```
74      cout<<flow<<std::endl;
75  }
76
77  int main() {
78      ios_base::sync_with_stdio(false);
79      int testCases = 0;
80      cin >> testCases;
81      while(testCases--) testCase();
82  }
```

Beach Bar

```cpp
#include <iostream>
#include <queue>
#include <set>
#include <algorithm>
#include <limits>

using namespace std;

const int MAX = numeric_limits<int>::max();

pair<int,int> optimalPosition(vector<int> &parasols, set<int> &bestPos) {
    int numParasols = parasols.size();
    sort(parasols.begin(), parasols.end());

    int bestNumParasols = 0;
    int bestMaxDistToWalk = MAX;
    priority_queue<int, vector<int>, greater<int> > Q;
    for(int i=0; i<numParasols; i++) {
        int rightMostParasol = parasols[i];
        Q.push(rightMostParasol);
        while(Q.top() < rightMostParasol-200) {
            Q.pop();
        }
        int leftMostParasol = Q.top();
        int numCovered = Q.size();
        int span = rightMostParasol-leftMostParasol;
        int barPos = (span/2)+ leftMostParasol;
        int barPos2 = barPos;
        if(span%2!=0) barPos2 = barPos+1;
        int maxDistToWalk = max(rightMostParasol-barPos,
                                -(leftMostParasol-barPos));

        if(numCovered > bestNumParasols) {
            bestNumParasols = numCovered;
            bestMaxDistToWalk = maxDistToWalk;
            bestPos.clear();
            bestPos.insert(barPos);
            bestPos.insert(barPos2);
        } else if(numCovered == bestNumParasols) {
            if(maxDistToWalk < bestMaxDistToWalk) {
                bestPos.clear();
                bestPos.insert(barPos);
                bestPos.insert(barPos2);
                bestMaxDistToWalk = maxDistToWalk;
            } else if(maxDistToWalk == bestMaxDistToWalk) {
                bestPos.insert(barPos);
                bestPos.insert(barPos2);
            }
        }
    }

    return pair<int,int>(bestNumParasols, bestMaxDistToWalk);
}

int main() {
    ios_base::sync_with_stdio(false);
    int testCases;
    cin >> testCases;
    for(int t=0;t<testCases;t++) {
        int numParasol;
        cin >> numParasol;
        vector<int> parasols(numParasol);
        for(int i=0;i<numParasol;i++) {
            int pos;
            cin >> pos;
            parasols[i] = pos;
        }
        set<int> optimalPos;
        pair<int, int> optimalValues = optimalPosition(parasols, optimalPos);
        cout<<optimalValues.first<<" "<<optimalValues.second<<endl;
        for(set<int>::iterator it = optimalPos.begin();
                it!=optimalPos.end(); it++) {
            if(it!=optimalPos.begin()) {
```

```
74              cout<<" ";
75          }
76          cout<<*it;
77      }
78      cout<<endl;
79  }
80 }
```

Cover

```
1   #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <iostream>
4   #include <cmath>
5   #include <vector>
6   #include <limits>
7
8   typedef CGAL::Exact_predicates_exact_constructions_kernel K;
9   typedef CGAL::Delaunay_triangulation_2<K>  Triangulation;
10  typedef Triangulation::Edge_iterator  Edge_iterator;
11  typedef CGAL::Segment_2<K> Segment;
12  typedef CGAL::Point_2<K> Point;
13  typedef CGAL::Ray_2<K> Ray;
14  typedef Triangulation::Finite_faces_iterator FFiter;
15  typedef CGAL::Iso_rectangle_2<K> Rect;
16  typedef CGAL::Line_2<K> Line;
17  // typedef CGAL::Triangulation_data_structure_2 Tds;
18  typedef Triangulation::Face_handle Face_handle;
19  typedef Triangulation::Vertex_handle Vertex_handle;
20
21  const int MAX = std::numeric_limits<int>::max();
22
23
24  double ceil_to_double(double x) {
25      double a = std::ceil(x);
26      while(a<x) a+=1;
27      while(a-1>=x) a-=1;
28      return a;
29  }
30
31  K::FT sdistance(Point x, Point y) {
32      return Segment(x,y).squared_length();
33  }
34
35  std::vector<Segment> rectToSegment(Rect r) {
36      std::vector<Segment> v(4);
37      v[0] = Segment(r.vertex(0), r.vertex(1));
38      v[1] = Segment(r.vertex(1), r.vertex(2));
39      v[2] = Segment(r.vertex(2), r.vertex(3));
40      v[3] = Segment(r.vertex(3), r.vertex(4));
41      return v;
42  }
43
44  template<typename T>
45  K::FT check_intersection_segments(T &voronoiEdge, Rect &area, Triangulation &t) {
46          std::vector<Segment> areaSegments = rectToSegment(area);
47          Segment intersectedAreaEdge;
48          bool isIntersected =false;
49          for(int i=0;i<4;i++) {
50              if(CGAL::do_intersect(voronoiEdge,areaSegments[i])) {
51                  intersectedAreaEdge = areaSegments[i];
52                  isIntersected =true;
53              }
54          }
55          if(!isIntersected) return -1;
56
57          CGAL::Object o2 = CGAL::intersection(voronoiEdge, intersectedAreaEdge);
58          if(const Point* op = CGAL::object_cast<Point>(&o2)) {
59              Vertex_handle vh = t.nearest_vertex(*op);
60              return sdistance(*op, vh->point());
61
62          } else {
63              std::cerr<<"runtime error";
64              return -1;
65              throw std::runtime_error("strange segment intersection");
66          }
67  }
68
69
70  void testcase(int numAntenna) {
71      double x1,x2,y1,y2;
72      std::cin >> x1 >> y1 >> x2 >> y2;
73      Rect area = Rect(Point(x1,y1),Point(x2,y2));
```

```
74        std::vector<Point > antennas(numAntenna);
75        Triangulation t;
76        for(int i=0; i<numAntenna; i++) {
77            double x,y;
78            std::cin>>x>>y;
79            antennas[i] = Point(x,y);
80        }
81
82        t.insert(antennas.begin(), antennas.end());
83        K::FT maxDist = -1;
84
85        //case 1: go over finite faces get circumcenter
86        for(FFiter i=t.finite_faces_begin();
87                i!=t.finite_faces_end();
88                i++) {
89            // Point c = t.circumcenter(i);
90            Point c = t.dual(i);
91            if(c.x() >= x1 && c.x() <= x2 && c.y() >= y1 && c.y() <= y2) {
92                K::FT tentativeMax = sdistance(c, i->vertex(1)->point());
93                if(tentativeMax > maxDist)
94                    maxDist = tentativeMax;
95            }
96        }
97        // std::cout<<"maxDist " <<ceil_to_double_sqrt(maxDist) << std::endl;
98
99        //case 2: go over corners of area, get nearest vertex
100       //for every corner
101       for(int i=0; i<4; i++) {
102           Point corner = area.vertex(i);
103           //for every nearestVertex
104           Vertex_handle vh = t.nearest_vertex(corner);
105           K::FT tentMax = sdistance(vh->point(), corner);
106
107           if(tentMax > maxDist) {
108               maxDist = tentMax;
109           }
110       }
111       // std::cout<<"maxDist " <<ceil_to_double_sqrt(maxDist) << std::endl;
112
113       //case 3: go over infinite faces
114       // process all Voronoi edges
115       for(Edge_iterator e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
116           CGAL::Object o = t.dual(e);
117           // o can be a segment, a ray or a line ...
118           Point* areaIntersection;
119           if(const Ray* oray = CGAL::object_cast<Ray>(&o)) {
120               maxDist = max(maxDist,check_intersection_segments(*oray, area, t));
121           } else if(const Line* oray = CGAL::object_cast<Line>(&o)) {
122               maxDist = max(maxDist,check_intersection_segments(*oray, area, t));
123           } else if(const Segment* oray = CGAL::object_cast<Segment>(&o)) {
124               maxDist = max(maxDist,check_intersection_segments(*oray, area, t));
125           }
126       }
127       std::cout<<std::setiosflags(std::ios::fixed) << std::setprecision(0)<< ↙
                ↳ ceil_to_double(sqrt(CGAL::to_double(maxDist)))
128               <<std::endl;
129       // std::cout<< numAntenna<<std::endl;
130   }
131   int main() {
132       std::ios_base::sync_with_stdio(false);
133       while(true) {
134           int numAntenna;
135           std::cin>>numAntenna;
136           if(numAntenna == 0) return 0;
137           testcase(numAntenna);
138       }
139   }
```

Divisor Distance

```cpp
#include <vector>
#include <iostream>
#include <cmath>

using namespace std;

int greatestDivisor(int number) {
    int i;
    for (i = 2; i <=sqrt(number); i++) {
        if (number % i == 0) {
            return number/i;
        }
    }
    return 1;
}

int cacheDivisor(vector<int> &graph, int i) {
    if(graph[i] == -1) {
        int gcd = greatestDivisor(i);
        graph[i] = gcd;
        return gcd;
    } else {
        return graph[i];
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    int testCases = 0;
    cin >> testCases;

    const int maxN = 10000000;
    vector<int> g(maxN,-1);
    for(int testCase=0; testCase < testCases; testCase++) {
        int n, numPairs;

        cin >> n;

        cin >> numPairs;
        for(int i=0; i<numPairs; i++) {
            int x,y;
            cin >> x >> y;
            int pathLen=0;
            while(true) {
                if(x==y) break;
                if(x>y) {
                    x=cacheDivisor(g, x);
                } else {
                    y=cacheDivisor(g, y);
                }
                ++pathLen;
            }
            cout<<pathLen<<endl;
        }
    }
}
```

Tiles

```cpp
#include<iostream>
#include <vector>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/max_cardinality_matching.hpp>


using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, undirectedS > Graph;
typedef graph_traits<Graph> Traits;
typedef Traits::vertex_descriptor Vertex;
typedef Traits::edge_descriptor Edge;
typedef property_map<Graph, vertex_index_t>::type IndexMap;


void testCase() {
    int width,height;
    cin >> width >> height;

    vector<vector<int> > field =
        vector<vector<int> >(width, vector<int>(height, -1));
    int curNumTilable = 0;
    Graph g;

    for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
            char place;
            cin>>place;
            if(place == '.') {
                field[j][i] = curNumTilable;
                if(i>0 && field[j][i-1] != -1) {
                    Edge e;
                    tie(e, tuples::ignore)=add_edge(field[j][i-1],curNumTilable,g);
                }
                if(j>0 && field[j-1][i] != -1) {
                    Edge e;
                    tie(e, tuples::ignore)=add_edge(field[j-1][i],curNumTilable,g);
                }
                curNumTilable++;
            }
        }
    }

    vector<Vertex> mate(curNumTilable);
    edmonds_maximum_cardinality_matching(g, &mate[0]);
    // for(i      get(mate,v)
    // graph_traits::null_vertex();
    if(matching_size(g,&mate[0])*2 == curNumTilable) {
        cout<<"yes"<<std::endl;
    } else {
        cout<<"no"<<std::endl;
    }

}

int main() {
    int testCases;
    cin>>testCases;
    while(testCases--) testCase();
    return 0;
}
```

Deleted Entries Stike Back

not correct

```
1   #include <iostream>
2   #include <vector>
3   #include <queue>
4   #include <boost/config.hpp>
5   #include <boost/graph/adjacency_list.hpp>
6   #include <boost/tuple/tuple.hpp>
7   #include <boost/graph/biconnected_components.hpp>
8   #include <boost/graph/connected_components.hpp>
9
10  using namespace std;
11  using namespace boost;
12
13  // #define DEBUG
14
15  #ifdef DEBUG
16  #define D(x) x
17  #else
18  #define D(x)
19  #endif
20
21  namespace boost {
22      struct edge_component_t {
23          enum {
24              num=555
25          };
26          typedef edge_property_tag kind;
27      }
28      edge_component;
29  }
30
31  typedef adjacency_list<vecS,vecS,undirectedS,no_property,
32              property<edge_component_t,std::size_t> > Graph;
33  typedef graph_traits<Graph> Traits;
34  typedef Traits::vertex_descriptor Vertex;
35  typedef Traits::edge_descriptor Edge;
36  typedef property_map<Graph, vertex_index_t>::type IndexMap;
37  typedef property_map<Graph, edge_component_t>::type ComponentMap;
38
39  vector<int> discoverTimeToVertex;
40
41  // Custom visitor used to record DFS order.
42  struct CustomVisitor : public default_dfs_visitor {
43      void discover_vertex(int u, const Graph& G) {
44          discoverTimeToVertex.push_back(u);
45      }
46  };
47
48  int newColor(int col1, int col2) {
49      int newCol = (col1+1)%3;
50      if(newCol == col2) {
51          newCol = (newCol+1)%3;
52      }
53      return newCol;
54  }
55
56  void testCase() {
57      int numVertices, numEdges;
58      cin>>numVertices>>numEdges;
59
60      Graph g(numVertices);
61      ComponentMap component = get(edge_component,g);
62
63      for(int i=0; i<numEdges; i++) {
64          Edge e;
65          int u,v;
66          cin>>u>>v;
67          tie(e,tuples::ignore) = add_edge(u,v,g);
68      }
69
70      //is connected?
71
72      //is biconnected?
```

```cpp
        vector<Vertex> low(numVertices);
        vector<Vertex> vertexToDiscoverTime(numVertices);
        vector<Vertex> predecessor(numVertices);
        discoverTimeToVertex.clear();
        size_t num_comps
            = biconnected_components(g,component,
                                     lowpoint_map(&low[0])
                                     .discover_time_map(&vertexToDiscoverTime[0])
                                     .predecessor_map(&predecessor[0])
                                     .visitor(CustomVisitor()));
        if(num_comps!=1) {
            cout<<"no"<<endl;
            return;
        }

        cout<<"yes"<<endl;
        vector<int> color(numVertices, 0);
        vector<bool> visited(numVertices, false);

        D(for(int i=0; i<numVertices; i++) {
            cout<<i << "␣" << vertexToDiscoverTime[i]<< "␣" << predecessor[i] << "␣" << low[i] << endl;
        })

        for(int i=1; i< numVertices; i++) {
            int v = discoverTimeToVertex[i];
            color[v] = newColor(color[predecessor[v]], color[low[v]-1]);
        }

        vector<vector<int> > colors(3);
        for(int i=0; i<numVertices; i++) {
            colors[color[i]].push_back(i);
        }

        for(int i=0; i<3; i++) {
            cout<<colors[i].size();
            for(int j=0; j<colors[i].size();j++) {
                cout<< "␣";
                cout<<colors[i][j];
            }
            cout<<endl;
        }

}

int main() {
    ios_base::sync_with_stdio(false);
    int testCases;
    cin>>testCases;
    while(testCases--) testCase();
}
```

Light The Stage

```cpp
1   #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2   #include <CGAL/Delaunay_triangulation_2.h>
3   #include <iostream>
4   #include<vector>
5
6   typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7   typedef CGAL::Delaunay_triangulation_2<K>  Triangulation;
8   typedef Triangulation::Face_handle Face;
9   typedef Triangulation::Vertex_handle Vertex;
10  typedef K::Point_2 P;
11  typedef K::Segment_2 S;
12  typedef K::Circle_2 C;
13
14  using namespace std;
15
16  int midpoint(int lowerBound, int upperBound) {
17      return lowerBound + (upperBound-lowerBound)/2;
18  }
19
20  void winnersGivenNumLamps(const vector<P> &lamps, const int numLamps, const int height,
21                            const vector<int> &peopleR, const vector<P> &people,
22                            vector<int> &winners) {
23      // construct triangulation
24      Triangulation t;
25      t.insert(lamps.begin(), lamps.begin()+numLamps+1);
26
27      for(int i=0; i<people.size(); i++) {
28          Vertex nearestLamp = t.nearest_vertex(people[i]);
29
30          K::FT maxDist = height;
31          maxDist+=peopleR[i];
32          maxDist = maxDist * maxDist;
33          K::FT dist = CGAL::squared_distance(nearestLamp->point(),people[i]);
34
35          if(dist >= maxDist) winners.push_back(i);
36      }
37
38  }
39
40  void testcase() {
41      int numPeople, numLamps;
42      cin >> numPeople >> numLamps;
43
44      vector<P> people(numPeople);
45      vector<int> peopleR(numPeople);
46      for(int i=0; i<numPeople; i++) {
47          int x,y,r;
48          cin>>x>>y>>r;
49          people[i] = P(x,y);
50          peopleR[i] = r;
51      }
52
53      int height;
54      cin >> height;
55      vector<P> lamps(numLamps);
56      for(int i=0; i<numLamps; i++) {
57          int x,y;
58          cin>>x>>y;
59          lamps[i] = P(x,y);
60      }
61
62      vector<int> winners;
63
64      // try all lamps
65      winnersGivenNumLamps(lamps, numLamps, height,
66                           peopleR, people, winners);
67      if(winners.size() != 0) {
68          for(int i=0; i< winners.size();i++) {
69              cout<<winners[i]<<"␣";
70          }
71          cout<<endl;
72          return;
73      }
```

```
74
75
76        //try binary search
77        int maxLampsMin = 0;
78        int maxLampsMax = lamps.size()-1;
79
80        while (maxLampsMax >= maxLampsMin) {
81            winners.clear();
82
83            int maxLamps = midpoint(maxLampsMin, maxLampsMax);
84
85            winnersGivenNumLamps(lamps, maxLamps, height,
86                                 peopleR, people, winners);
87
88            if(winners.size() == 0) {
89                maxLampsMax = maxLamps-1;
90            } else {
91                maxLampsMin = maxLamps+1;
92            }
93        }
94
95        winners.clear();
96        winnersGivenNumLamps(lamps, maxLampsMax, height,
97                             peopleR, people, winners);
98        for(int i=0; i< winners.size();i++) {
99            cout<<winners[i]<<"␣";
100        }
101        cout<<endl;
102
103    }
104
105    int main() {
106        ios_base::sync_with_stdio(false);
107        int testcases;
108        cin >> testcases;
109        while(testcases--) testcase();
110
111    }
```

Radiation

```
1   #include <iostream>
2   #include <cassert>
3   #include <CGAL/basic.h>
4   #include <CGAL/QP_models.h>
5   #include <CGAL/QP_functions.h>
6
7   // choose exact integral type
8   #ifdef CGAL_USE_GMP
9   #include <CGAL/Gmpz.h>
10  typedef CGAL::Gmpz ET;
11  #else
12  #include <CGAL/MP_Float.h>
13  typedef CGAL::MP_Float ET;
14  #endif
15
16  // program and solution types
17  typedef CGAL::Quadratic_program<ET> Program;
18  typedef CGAL::Quadratic_program_solution<ET> Solution;
19
20  using namespace std;
21
22
23  vector<vector<double> > powArray(2048,vector<double>(31,-1));
24
25  struct Point3 {
26      int x,y,z;
27  };
28
29  Point3 P(int x, int y, int z) {
30      Point3 p;
31      p.x = x;
32      p.y = y;
33      p.z = z;
34      return p;
35  }
36
37  double getPowerArray(int x, int y) {
38      double tmp;
39      if(powArray[x+1024][y] == -1) {
40          tmp = pow(x,y);
41          powArray[x+1024][y] = tmp;
42          return tmp;
43      } else {
44          return powArray[x+1024][y];
45      }
46  }
47
48  int midpoint(int lowerBound, int upperBound) {
49      return lowerBound + (upperBound-lowerBound)/2;
50  }
51
52  bool testDegree(vector<Point3> &cells, int numHealthy, int degree) {
53      Program lp (CGAL::SMALLER, false, 0, false, 0);
54      for(int c=0; c<cells.size(); c++) {
55          int row = c;
56          int rowIndex = 0;
57          for(int i=0; i <= degree; i++) {
58              for(int j=0; j <= degree-i; j++) {
59                  for(int k=0; k <= degree-i-j; k++) {
60                      lp.set_a(rowIndex, row,
61                              getPowerArray(cells[c].x, i)
62                              * getPowerArray(cells[c].y, j)
63                              * getPowerArray(cells[c].z, k)
64                              );
65                      if(row<numHealthy) {
66                          //healthy cell
67                          lp.set_b(row,-1);
68                          lp.set_r(row, CGAL::SMALLER);
69                      } else {
70                          //tumor cell
71                          lp.set_b(row,1);
72                          lp.set_r(row, CGAL::LARGER);
73                      }
```

```
 74                        ++rowIndex;
 75                    }
 76                }
 77            }
 78        }
 79        CGAL::Quadratic_program_options options;
 80        options.set_pricing_strategy(CGAL::QP_BLAND);
 81        Solution s = CGAL::solve_linear_program(lp, ET(), options);
 82        assert(s.solves_linear_program(lp));
 83
 84        return !s.is_infeasible();
 85    }
 86
 87    int findBestDegree(vector<Point3> &cells, int numHealthy) {
 88        const int maxLinear = -1;
 89        for(int degree = 0; degree <= maxLinear; degree++) {
 90            if(testDegree(cells, numHealthy, degree)) {
 91                return degree;
 92            }
 93        }
 94
 95        int degreeMin = maxLinear+1;
 96        int degreeMax = 30;
 97        while(degreeMax >= degreeMin) {
 98            int degree = midpoint(degreeMin, degreeMax);
 99
100            if(testDegree(cells, numHealthy, degree)) {
101                degreeMax = degree-1;
102            } else {
103                degreeMin = degree+1;
104            }
105        }
106
107        return degreeMin;
108    }
109
110    void testCase() {
111        int numHealthy, numTumor;
112        cin>>numHealthy>>numTumor;
113
114        vector<Point3> cells(numHealthy+numTumor);
115
116        for(int i=0; i<cells.size(); i++) {
117            int x,y,z;
118            cin>>x>>y>>z;
119            cells[i] = P(x,y,z);
120        }
121
122        int degree = findBestDegree(cells, numHealthy);
123        if(degree <= 30) {
124            cout<< degree << std::endl;
125        } else {
126            cout << "Impossible!" << std::endl;
127        }
128    }
129
130    int main() {
131        ios_base::sync_with_stdio(false);
132        int testCases;
133        cin >> testCases;
134        while(testCases--) {
135            testCase();
136        }
137    }
```

Sweepers

```
1   #include <iostream>
2   #include <vector>
3   #include <queue>
4   #include <boost/config.hpp>
5   #include <boost/graph/adjacency_list.hpp>
6   #include <boost/tuple/tuple.hpp>
7   #include <boost/graph/push_relabel_max_flow.hpp>
8
9   using namespace std;
10  using namespace boost;
11
12  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13  typedef adjacency_list<vecS, vecS, directedS, no_property,
14    property<edge_capacity_t, long,
15    property<edge_residual_capacity_t, long,
16    property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
17
18  typedef graph_traits<Graph> GraphTraits;
19  typedef GraphTraits::vertex_descriptor Vertex;
20  typedef GraphTraits::edge_descriptor Edge;
21  typedef GraphTraits::out_edge_iterator edge_iterator;
22  typedef property_map<Graph, vertex_index_t>::type IndexMap;
23  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
24  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
25  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
26
27  void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
28      Edge e, reverseE;
29      tie(e, tuples::ignore) = add_edge(u,v,g);
30      tie(reverseE, tuples::ignore) = add_edge(v, u, g);
31      capacity[e] = c;
32      capacity[reverseE] = 0;
33      rev_edge[e] = reverseE;
34      rev_edge[reverseE] = e;
35  }
36
37  void testCase() {
38      int numVertices, numEdges, numSweepers;
39      cin>>numVertices>>numEdges>>numSweepers;
40
41      Graph g(numVertices+2);
42      EdgeCapacityMap capacity = get(edge_capacity, g);
43      ReverseEdgeMap rev_edge = get(edge_reverse, g);
44      ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
45
46      const int SOURCE = numVertices;
47      const int SINK = numVertices+1;
48      vector<int> startLocations, exitLocations;
49
50      //read start and exit locations
51      for(int i=0; i<numSweepers; i++) {
52          int startLocation;
53          cin>>startLocation;
54          startLocations.push_back(startLocation);
55      }
56      for(int i=0; i<numSweepers; i++) {
57          int exitLocation;
58          cin>>exitLocation;
59          exitLocations.push_back(exitLocation);
60      }
61
62
63      // the following is a incorrect interpretation of the task:
64      // correct: if there are sweepers they should clean every corridor just once
65      // there seems to be no requirement that corridors have to be cleaned
66      // if(numEdges > 0 && numSweepers == 0) {
67      //     cout<< "no"<< std::endl;
68      //     return;
69      // }
70
71      //build flow graph
72      for(int i=0; i<numEdges; i++) {
73          int u,v;
```

```
 74          cin>>u>>v;
 75          addFlowEdge(g, capacity, rev_edge, u, v, 1);
 76          addFlowEdge(g, capacity, rev_edge, v, u, 1);
 77      }
 78
 79      // check if all vertices of non-zero degree are reachable from some source
 80      vector<bool> visited(numVertices,false);
 81      std::queue<Vertex> Q;
 82      for(int i=0; i<startLocations.size(); i++) {
 83          int v = startLocations[i];
 84          visited[v] = true;
 85          Q.push(v);
 86          while(Q.size() > 0) {
 87              v = Q.front(); Q.pop();
 88              edge_iterator out_i, out_end;
 89              for(tie(out_i, out_end) = out_edges(v,g);
 90                      out_i!=out_end; ++out_i) {
 91                  Edge e = *out_i;
 92                  int targ = target(e,g);
 93                  if(targ<visited.size() && !visited[targ]) {
 94                      Q.push(targ);
 95                      visited[targ] = true;
 96                  }
 97              }
 98          }
 99      }
100
101      for(int i=0; i<visited.size(); i++) {
102          if(!visited[i] && out_degree(i,g)>0) {
103              cout<< "no"<< std::endl;
104              return;
105          }
106      }
107
108      //check eulerian tour
109      for(int i=0; i<numVertices; i++) {
110          int v = vertex(i, g);
111          int numStartLocation = count(startLocations.begin(), startLocations.end(), v);
112          int numExitLocation = count(exitLocations.begin(), exitLocations.end(), v);
113
114          if((out_degree(v,g)/2+numStartLocation+numExitLocation)%2 == 1) {
115              cout<<"no";
116              cout<<std::endl;
117              return;
118          }
119      }
120
121      // add source and sink
122      for(int i=0; i<startLocations.size(); i++) {
123          addFlowEdge(g, capacity, rev_edge, SOURCE, startLocations[i], 1);
124      }
125      for(int i=0; i<exitLocations.size(); i++) {
126          addFlowEdge(g, capacity, rev_edge, exitLocations[i], SINK, 1);
127      }
128
129      //compute flow
130      long flow = push_relabel_max_flow(g, SOURCE, SINK);
131
132      if(flow == numSweepers)
133          cout<<"yes"<<std::endl;
134      else
135          cout<<"no"<<std::endl;
136
137      return;
138
139  }
140
141  int main() {
142      ios_base::sync_with_stdio(false);
143      int testCases = 0;
144      cin >> testCases;
145      while(testCases--) testCase();
146  }
```

The Bracelet

```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <map>
#include <set>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/connected_components.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, undirectedS,
                        no_property, property<edge_weight_t, int> > Graph;
typedef graph_traits<Graph> Traits;
typedef Traits::vertex_descriptor Vertex;
typedef Traits::edge_descriptor Edge;
typedef Traits::vertex_iterator vertex_iter;
typedef Traits::out_edge_iterator edge_iterator;
typedef property_map<Graph, edge_weight_t>::type WeightMap;

void eulerTourDfs(Graph &g, WeightMap &marked, Vertex v, vector<int> &tour) {

    edge_iterator out_i, out_end;
    for (tie(out_i, out_end) = out_edges(v, g);
            out_i!=out_end; ++out_i) {
        if(marked[*out_i]==0) {
            marked[*out_i]=1;
            eulerTourDfs(g,marked,target(*out_i, g),tour);
        }
    }
    tour.push_back(v);
}
void eulerTour(Graph &g, WeightMap &marked, Vertex v, vector<int> &tour) {
    stack<int> S;
    S.push(v);
    while(!S.empty()) {
        Vertex v = S.top();
        edge_iterator out_i, out_end;
        bool hasUnmarkedEdge=false;
        for (tie(out_i, out_end) = out_edges(v, g);
                out_i!=out_end; ++out_i) {
            if(marked[*out_i]==0) {
                hasUnmarkedEdge=true;
                marked[*out_i]=1;
                S.push(target(*out_i,g));
                break;
            }
        }

        if(!hasUnmarkedEdge) {
            S.pop();
            tour.push_back(v);
        }
    }
}


void testCase(int num) {
    cout<<"Case #"<<num<<std::endl;
    int numBeads;
    cin>>numBeads;

    Graph g;
    WeightMap marked = get(edge_weight, g);
    vector<int> colorMap(51,-1);
    int colorIndex = 0;

    for(int i=0;i<numBeads;i++) {
        int color1, color2;
        cin>>color1>>color2;

```

```
74          // map colors to gapless indexes starting with 0
75          if(colorMap[color1] == -1) {
76              colorMap[color1] = colorIndex;
77              ++colorIndex;
78          }
79          if(colorMap[color2] == -1) {
80              colorMap[color2] = colorIndex;
81              ++colorIndex;
82          }
83
84          Edge e;
85          tie(e, tuples::ignore) =add_edge(colorMap[color1],colorMap[color2],g);
86          marked[e]=0;
87      }
88
89      vector<int> componentMap(num_vertices(g));
90      int components = connected_components(g, &componentMap[0]);
91      if(components > 1) {
92          cout<<"some␣beads␣may␣be␣lost"<<std::endl;
93          // cout<<"reason 1: "<<components<<std::endl;
94          return;
95      }
96
97      std::pair<vertex_iter,vertex_iter> vi;
98      for(vi=vertices(g); vi.first != vi.second; ++vi.first) {
99          int degree = out_degree(*(vi.first),g);
100         if(degree%2==1) {
101             cout<<"some␣beads␣may␣be␣lost"<<std::endl;
102             // cout<<"reason 2"<<std::endl;
103             return;
104         }
105     }
106
107     //print eulerian tour
108     vector<int> tour;
109     // eulerTourDfs(g,marked,0,tour);
110     eulerTour(g,marked,0,tour);
111
112     // reverse colorMap
113     vector<int> inverseColorMap(51,-1);
114     for(int i=0; i<colorMap.size(); i++) {
115         if(colorMap[i] != -1) {
116             inverseColorMap[colorMap[i]] = i;
117         }
118     }
119
120     for(int i=0; i<tour.size()-1; i++) {
121         cout<<inverseColorMap[tour[i]]<<"␣"<<inverseColorMap[tour[i+1]]<<std::endl;
122     }
123 }
124
125 int main() {
126     ios_base::sync_with_stdio(false);
127     int testCases;
128     cin >> testCases;
129     int i = 0;
130     while(testCases--) {
131         testCase(++i);
132         if(testCases>0) cout<<std::endl;
133     }
134 }
```

Knights

```cpp
#include <iostream>
#include <vector>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>

using namespace std;
using namespace boost;

// #define DEBUG

#ifdef DEBUG
#define D(x) x
#else
#define D(x)
#endif

typedef adjacency_list_traits<vecS,vecS,directedS> Traits;
typedef adjacency_list<vecS, vecS, directedS, no_property,
        property<edge_capacity_t, long,
        property<edge_residual_capacity_t, long,
        property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;

typedef graph_traits<Graph> GraphTraits;
typedef GraphTraits::vertex_descriptor Vertex;
typedef GraphTraits::edge_descriptor Edge;
typedef property_map<Graph,vertex_index_t>::type IndexMap;
typedef property_map<Graph,edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph,edge_residual_capacity_t>::type ResidualCapacityMap;
typedef property_map<Graph,edge_reverse_t>::type ReverseEdgeMap;

void addFlowEdge(Graph &g, EdgeCapacityMap &capacity, ReverseEdgeMap &rev_edge, int u, int v, int c) {
    Edge e, reverseE;
    tie(e,tuples::ignore) = add_edge(u,v,g);
    tie(reverseE,tuples::ignore) = add_edge(v,u,g);
    capacity[e] = c;
    capacity[reverseE] = 0;
    rev_edge[e] = reverseE;
    rev_edge[reverseE] = e;
}

void testCase() {
    // graph is given as a grid
    int gridX, gridY, numSources;
    cin>>gridX>>gridY>>numSources;

    Graph g;
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap rev_edge = get(edge_reverse, g);
    ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);

    //need two "layers" one for in-vertices, the other for out-vertices because
    //the "vertex"-capacity is 1
    //the in-vertex is on the first layer, meaning its index <= numVerticesOnOneLayer
    int numVerticesOnLayer = gridX*gridY;

    const int SOURCE = numVerticesOnLayer*2;
    const int SINK = numVerticesOnLayer*2+1;

    for(int i=0; i<gridX; i++) {
        for(int j=0; j<gridY; j++) {
            int curVertex = numVerticesOnLayer + i*gridY+j;
            // upper row
            if(j>0) {
                addFlowEdge(g, capacity, rev_edge, curVertex, i*gridY+j-1,1);
            } else {
                addFlowEdge(g, capacity, rev_edge, curVertex, SINK,1);
            }
            // right column
            if(i<gridX-1) {
                addFlowEdge(g, capacity, rev_edge, curVertex, (i+1)*gridY+j,1);
            } else {
```

```
74              addFlowEdge(g, capacity, rev_edge, curVertex, SINK,1);
75          }
76          // lower row
77          if(j<gridY-1) {
78              addFlowEdge(g, capacity, rev_edge, curVertex, i*gridY+j+1,1);
79          } else {
80              addFlowEdge(g, capacity, rev_edge, curVertex, SINK,1);
81          }
82          // left column
83          if(i>0) {
84              addFlowEdge(g, capacity, rev_edge, curVertex, (i-1)*gridY+j,1);
85          } else {
86              addFlowEdge(g, capacity, rev_edge, curVertex, SINK,1);
87          }
88
89          //connect out-vertex to its in-counterpart
90          addFlowEdge(g, capacity, rev_edge, curVertex-numVerticesOnLayer, curVertex,1);
91      }
92  }
93
94  for(int i=0; i<numSources; i++) {
95      int x,y;
96      cin>>x>>y;
97      addFlowEdge(g, capacity, rev_edge, SOURCE, x*gridY+y,1);
98  }
99
100 D(cout<<"num vertices "<<num_vertices(g)<<endl;)
101
102 long flow = push_relabel_max_flow(g, SOURCE, SINK);
103
104 cout<<flow<<endl;
105
106 }
107
108 int main() {
109     int numCases;
110     cin >> numCases;
111     while(numCases--) testCase();
112     return 0;
113 }
```

Next Path

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

// #define DEBUG
#ifdef DEBUG
#define D(x) x
#else
#define D(x)
#endif

typedef vector<vector<int> > AdjacencyList;

const int MAX = numeric_limits<int>::max();

int shortestPath(const AdjacencyList &adjacencyList, int start, int end, vector<int> &predecessorMap) {
    queue<int> Q;
    vector<bool> visited(adjacencyList.size(),false);
    vector<int> distance(adjacencyList.size());
    Q.push(start);
    visited[start] = true;
    distance[start] = 0;
    if(start == end) {
        return 0;
    }

    while(!Q.empty()) {
        int v = Q.front();
        Q.pop();
        for(int i =0; i< adjacencyList[v].size(); i++) {
            int childV = adjacencyList[v][i];
            if(!visited[childV]) {
                visited[childV] = true;
                predecessorMap[childV] = v;
                distance[childV] = distance[v] + 1;
                if(childV == end) {
                    return distance[childV];
                } else {
                    Q.push(childV);
                }
            }
        }
    }
    return -1;
}

void testCase() {
    int numEdges, numVertices;
    cin>>numVertices>>numEdges;
    int start, end;
    cin>>start>>end;
    start-=1; end-=1;

    AdjacencyList adjacencyList(numVertices, vector<int>(0));
    for(int i=0; i<numEdges; i++) {
        int u,v;
        cin>>u>>v;
        adjacencyList[u-1].push_back(v-1);
    }

    vector<int> predecessorMap(numVertices, -1);
    int secondShortestPathLength = MAX;
    int shortestPathLength = shortestPath(adjacencyList, start, end, predecessorMap);
    D(cout<<"shortest: "<<shortestPathLength<<endl;)

    if(shortestPathLength == -1) {
        cout<<"no"<<endl;
        D(cout<<"reason1"<<endl;)
        return;
    }
```

```
74
75      vector<int> path;
76      int curVertex = end;
77      D(cout<<"backwards␣path:␣";)
78      while(curVertex != -1) {
79          D(cout<<curVertex <<"␣";)
80          path.push_back(curVertex);
81          curVertex = predecessorMap[curVertex];
82      }
83      D(cout<<endl;)
84
85      for(int i = 0; i<path.size(); i++) {
86          int v = path[i];
87          int pathLenSoFar = path.size() - i - 1;
88          for(int j =0; j< adjacencyList[v].size(); j++) {
89              int childV = adjacencyList[v][j];
90              if(i==0 || childV != path[i-1]) {
91                  int shortestPathFromChild = shortestPath(adjacencyList, childV , end, predecessorMap);
92                  D(cout<<"vertex␣"<<v<< "␣path␣len␣so␣far␣on␣shortest:␣" << pathLenSoFar <<";␣shortest␣path␣from␣" << ↙
                      ↳ childV << "␣is␣" << shortestPathFromChild << endl;)
93                  if(shortestPathFromChild >= 0) {
94                      secondShortestPathLength
95                          = min(secondShortestPathLength, pathLenSoFar + 1 + shortestPathFromChild);
96                  }
97              }
98          }
99      }
100
101     if(secondShortestPathLength==MAX) {
102         // secondShortestPathLength = shortestPathLength;
103         cout<<"no"<<endl;
104         D(cout<<"reason2"<<endl;)
105         return;
106     }
107
108     cout<<secondShortestPathLength<<endl;
109
110 }
111
112 int main() {
113     int testCases;
114     cin>>testCases;
115     while(testCases--) testCase();
116     return 0;
117 }
```

Odd Route

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
#include <boost/config.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, directedS, no_property,
property<edge_weight_t, int> > Graph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;

void testCase() {
    int numVertices, numEdges;
    cin>>numVertices>>numEdges;
    Graph g(numVertices*4);

    int source,target;
    cin>>source>>target;

    target = numVertices*3 + target;

    property_map<Graph, edge_weight_t>::type weightMap = get(edge_weight, g);
    for(int i = 0; i < numEdges; ++i)
    {
        int u,v,w;
        cin>>u>>v>>w;
        bool success;

        Edge e;
        if(w%2==1) {
            tie(e, success) = add_edge(u, v+3*numVertices, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+numVertices, v+2*numVertices, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+2*numVertices, v+numVertices, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+3*numVertices, v, g);
            weightMap[e] = w;

        } else {
            tie(e, success) = add_edge(u, v+2*numVertices, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+numVertices, v+3*numVertices, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+2*numVertices, v, g);
            weightMap[e] = w;
            tie(e, success) = add_edge(u+3*numVertices, v+numVertices, g);
            weightMap[e] = w;


        }
    }

    std::vector<Vertex> predecessors(num_vertices(g));
    std::vector<int> distancesFromSource(num_vertices(g));

    dijkstra_shortest_paths(g, source,
        predecessor_map(&predecessors[0]).distance_map(&distancesFromSource[0]));

    if(distancesFromSource[target] < numeric_limits<int>::max()) {
        cout<<distancesFromSource[target]<<endl;
    } else {
        cout<<"no"<<endl;
    }

}


```

```cpp
int main()
{
    int testCases;
    cin>>testCases;

    while(testCases--) testCase();
}
```

Radiation 2

Missing Roads

Standard maximum matching in Linear Programming formulation:

For each edge between u and v introduce variable x_{uv}. If the variable is set to 1 the the edge is part of the matching, and if it set to 0, the edge is not in the matching. $\max \sum x_{uv}$ Subject to the constraint that no for every vertex u $x_{uv_1} + x_{uv_2} + \cdots \leq 1$