name	problem	solution	author	P
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	Ben	??
Dominoes	Given a list of tiles of different heights, determine how many tiles will fall after toppling the left-most tile.	-	Ben	??
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 $O(\sqrt{l})$	Ben	??
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	Jonas	??
Aliens	Given a set of intervals, count those that have an element that is not contained in any other interval.	-	Jonas	??
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)	Jonas	??
False Coin	Given a series weighing outcomes of several coins, determine which coin has a different weight than the other coins.	-	Jonas	??
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	Jonas	??
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) $-3 \leq a, b \leq 3$. 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	Jonas	??
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	Jonas	??
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	Jonas	??
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 $t * k - 1$.	-	Jonas	??

Longest Path	Determine the length of the longest path in a tree.	DFS while recording tree height	Jonas	??
Ants	Given a 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	Jonas	??
Bridges	Given a connected graph G, find the edges whose lacking would result in two connected components	Biconnected components	Jonas	??
Build The Graph		MST and Dijkstra	Jonas	??
Deleted Entries	Given a graph G. Determine whether it is possible to divide the graph into k groups such that every node can reach every node in every other group after deleting the in-group edges.	MST and BFS to k-color	Jonas	??
Shy Programmers	Given a 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	Jonas	??
Algocoon	Given a directed graph G find the global minimum (directed) cut.	Find source/sink that minimize flow. Start BFS from source to get the cut.	Jonas	??
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	Jonas	??
Satellites	Minimum vertex cover of unweighted bipartite graph.	Compute maximum matching and use Konigs Theorem	Jonas	??
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	Jonas	??
Kingdom Defense	Given 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	Jonas	??
Hit	Given a ray and some line segments. Determine if ray intersects any segment.	loop & test intersection	Jonas	??
First Hit	Given a ray and some line segments, where does the ray first intersect a segment?	randomize segment order, loop while minimizing intersection	Jonas	??
Antenna	Given a set of points, find the minimum enclosing circle.	Minimum enclosing circle	Jonas	??

Almost Antenna	Given a set of points, find the minimum circle that encloses all but one point.	Minimum enclosing circle	Jonas	??
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	Jonas	??
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	Jonas	??
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	Jonas	??
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	??	
Graypes	Given a set of points G find the shortest distance between any pair of the points.	Delaunay triangulation	??	
H1N1	How to move a disk without colliding with a given point set?	Precompute for each Delaunay-face the escape radius using DFS.	Jonas	??
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	Jonas	??
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	Jonas	??
Diet	Given the price of a food product and some nutritional constraints, calculate the cheapest diet.	Linear Programming	Jonas	??
Inball	Given a cave $C = \{x \in \mathbb{R}^d \mid a_i^T x \leq b_i, i = 1, \ldots, n\}$. Find the ball with the largest radius that is contained in C.	Linear Programming	Jonas	??
Maximize It	Solve two quadratic programs.	Quadratic Programming	Jonas	??
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	Jonas	??
Monkey Island	Let G be a 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	Jonas	??

Placing Knights	Determine the maximum number of chess-knights on a field with given size and obstacles.	Maximum independent set (n-maximum matching)	Jonas	**??**
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	Jonas	**??**
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	Jonas	**??**
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.	Dynamic Programming	Jonas	**??**
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	Jonas	**??**

Checking Change

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <string>
#include <sstream>
using namespace std;

vector<string> answers;

int main(int argc, char const *argv[])
{

    int currencies;
    cin >> currencies;

    for (int currency = 0; currency < currencies; currency++)
    {

        int coins_count;
        int testcases;

        cin >> coins_count >> testcases;

        vector<int> coins;
        for (int coins_it = 0; coins_it < coins_count; coins_it++)
        {
            int coin;
            cin >> coin;
            coins.push_back(coin);
        }

        vector<int> tests;
        for (int testcase = 0; testcase < testcases; testcase++)
        {
            int test;
            cin >> test;
            tests.push_back(test);
        }

        // find maximum of tests
        vector<int>::iterator max_test_it = max_element(tests.begin(), tests.end());
        int max_test = *max_test_it;
        int N = max_test + 1;

        vector<int>::iterator max_coin_it = max_element(coins.begin(), coins.end());
        int max_coin = *max_coin_it;

        vector<int>::iterator min_coin_it = min_element(coins.begin(), coins.end());
        int min_coin = *min_coin_it;

        // instantiate array with size max(tests)
        int arraysize = 2;
        vector<int> counts(arraysize);

        // fill indices we already know -> coins, set to zero where index smaller than index of smallest coin.
        for (int i = 0; i < min_coin; i++)
        {
            if (min_coin >= arraysize)
            {
                arraysize += min_coin + 10;
                counts.resize(arraysize);
                //cout << "vector size now " << arraysize;
            }
            counts[i] = 0;
        }

        for (vector<int>::iterator coins_it = coins.begin(); coins_it != coins.end(); coins_it++)
        {
            if (*coins_it <= max_coin)
            {
                if (*coins_it >= arraysize)
                {
                    arraysize += *coins_it + 1;
```

```
                    counts.resize(arraysize);
                    //cout << "vector size now " << arraysize;
                }
                counts[*coins_it] = 1;
            }
        }

        // iterate over counts, combine all minimums.
        for (int n = min_coin + 1; n < N; n++)
        {
            if (arraysize <= n)
            {
                arraysize += 1;
                counts.resize(arraysize);
                //cout << "vector size now " << arraysize;
            }

            signed int min = -1;
            for(int backward = n-1; backward >= min_coin; backward--) {

                if (counts[n] == 1)
                {
                    min = 1;
                } else {
                    if(counts[backward] != 0 && counts[n-backward] != 0) {
                        int new_min = counts[backward] + counts[n-backward];
                        //cout << n << ": counts[backward]: " << counts[backward] << " counts[n-backward]: " << ↙
                            ↳ counts[n-backward] << "new_min: "<< new_min << "\n";
                        if (min > new_min || min == -1)
                        {
                            min = new_min;
                        }
                    }
                }

            }
            if (min == -1)
            {
                min = 0;
            }
            counts[n] = min;
        }

        /*int i = 0;
        for (vector<int>::iterator elements = counts.begin(); elements != counts.end(); elements++)
        {
            cout << i++ << ": " << *elements << " \n";
        }*/

        for (vector<int>::iterator test = tests.begin(); test != tests.end(); test++)
        {
            int answer = counts[*test];

            stringstream ss;
            if (answer == 0)
            {
                ss << "not␣possible";
            } else {
                ss << answer;
            }

            answers.push_back(ss.str());
        }

    }

    for (vector<string>::iterator answer = answers.begin(); answer != answers.end(); answer++)
        cout << *answer << "\n";

    return 0;
}
```

Dominoes

```cpp
/*
* Benjamin Grhbiel
* Domino
*/

#include <iostream>
#include <vector>
#include <map>
using namespace std;

int main (int argc, const char *argv[])
{

  ios_base::sync_with_stdio(false);

  int testcases;
  cin >> testcases;

  map< int, vector<int> > index;

  for (int testcase = 0; testcase < testcases; testcase++) {

    long int dominoes;
    cin >> dominoes;

    for (int dominoPos = 1; dominoPos <= dominoes; dominoPos++) {
      int height;
      cin >> height;
      index[testcase].push_back(height);
    }

  }

  for (map<int, vector<int> >::iterator it = index.begin(); it != index.end(); it++) {
      //cout << "Testcase: " << it->first << " Tiles: " << it->second.size() << "\n";

      vector<int> tiles = it->second;

      if (tiles.size() == 0) {
        cout << 0;
      }
      else
      {
        int intervalRight = 0;
        int iteration = 0;
        int counter = 0;

        for (vector<int>::iterator tile_it = tiles.begin(); tile_it != tiles.end(); tile_it++) {

          if (iteration > intervalRight) {
              //cout << "Break; iteration > intervalRight \n";
              break;
          }

          int h = *tile_it;
          int newIntervalRight = h + iteration - 1;

          if(newIntervalRight > intervalRight) {
            intervalRight = newIntervalRight;
          }

          iteration++;
          //cout << "intervalRight: " << intervalRight << " iteration: " << iteration << "\n";
          counter++;
        }

        cout << counter << "\n";

      }
  }

  return 0;

```

```
74 }
```

Shelves

```cpp
#include <iostream>

using namespace std;

int main(void) {
    // speeds up read and write
    ios_base::sync_with_stdio(false);

    // number of testcases we need to run
    int nrCases;
    cin >> nrCases;

    for(int i = 0; i < nrCases; i++) {
        // read the input for the test case
        int l, m, n;
        cin >> l >> m >> n;

        // number of the two shelves and remaining length
        int cm = 0;
        int cn = 0;
        int r = l;

        for(int tmpCn = l/n; tmpCn >= 0 && r != 0; tmpCn--) {
            // calculate the number of the small shelves
            int tmpCm  = (l - tmpCn * n) / m;
            if(tmpCm >= n) {
                break;
            }

            // calculate the new remaining space and use it when smaller
            int tmpR = l - tmpCn * n - tmpCm * m;
            if(tmpR < r) {
                cn = tmpCn;
                cm = tmpCm;
                r = tmpR;
            }
        }

        // output the result
        cout << cm << "␣" << cn << "␣" << r << '\n';
    }

    return 0;
}
```

Even Pairs

Even Pairs missing

Aliens

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

typedef vector<pair<int, int> > vii;      // sorted by left, right.
bool sortDescAsc(const pair<int, int>& lhs, const pair<int, int>& rhs) {
    if(lhs.first == rhs.first)
        return (lhs.second > rhs.second);
    else
        return lhs.first < rhs.first;
}

void testcase() {
    int n, m;
    cin >> n >> m;
    vii intervals;
    int superior = n;
    for(int i = 0; i < n; ++i) {
        int pi, qi;
        cin >> pi >> qi;
        if(pi == 0 && qi == 0) {
            --superior;
            continue;
        }
        pair<int, int> entry = make_pair(pi, qi);
        intervals.push_back(entry);
    }

    sort(intervals.begin(), intervals.end(), sortDescAsc);

    int left = 0;
    int right = 0;
    for(int i = 0; i < intervals.size(); ++i) {
        if(i+1 < intervals.size() && intervals[i+1].first == intervals[i].first && intervals[i+1].second == ↙
            ↳ intervals[i].second)
            --superior;
        else if(left == intervals[i].first && right == intervals[i].second)
            --superior;
        else if(right >= intervals[i].second)
            --superior;

        if(right < intervals[i].second) {
            left = intervals[i].first;
            if(right != 0 && left-right > 1) {
                cout << "0\n";
                return;
            }
            right = intervals[i].second;
        }
   }

    cout << superior << "\n";
}

int main() {
    int TC;
    cin >> TC;
    while(TC--) testcase();
}
```

Boats

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

struct Boat {
    int ring;
    int length;
    bool taken;

};
inline bool operator<( const Boat& lhs, const Boat& rhs ) {
    return lhs.ring < rhs.ring;
}
inline bool operator<( int lhs, const Boat& rhs ) {
    return lhs <= rhs.ring;
}
inline bool operator<(const Boat& lhs, const int &val) {
    return (lhs.ring < val);
}

void testcase() {
    int boats; cin >> boats;
    vector<Boat> boat_list;

    for (int i = 0; i < boats; ++i)
    {
        int length, ring; cin >> length >> ring;
        Boat boat;
        boat.length = length;
        boat.ring = ring;
        boat.taken = false;
        boat_list.push_back(boat);
    }

    std::sort(boat_list.begin(), boat_list.end());

    int counter = 1;
    int rightmost = boat_list[0].ring;
    boat_list[0].taken = true;

    // Problem 1: rightmost < boat_list.back().ring ... meaning, we stopped too early, neglecting the last boat.
    // Problem 2: Endless loop in the scenario of just one boat... as righmost = boat_list.back().ring.
    while((rightmost <= boat_list.back().ring) && (boat_list.size() != 1)) {

        vector<Boat>::iterator up = lower_bound(boat_list.begin(), boat_list.end(), rightmost);
        int index = (up - boat_list.begin());
        int next = index;
        //cerr << "next: " << next << "\n";

        // check if already taken, if yes, move pointer to the right.
        if(boat_list[next].taken == true) next++;

        int local_rightmost;
        int min_rightmost = -1;
        int boat_index;
        do {
            int ring = boat_list[next].ring;
            int left = ring - rightmost;
            int right = boat_list[next].length - left;

            if(right < 0) local_rightmost = ring;
            else local_rightmost = ring + right;

            //cerr << "local_rightmost: " << local_rightmost << " min_rightmost: " << min_rightmost << "\n";
            if((local_rightmost < min_rightmost) || (min_rightmost == -1)) {
                min_rightmost = local_rightmost;
                boat_index = next;
                //cerr << "local minimum set: " << local_rightmost << " boat_index: " << boat_index << "\n";
            }
            next++;
        }
        // Problem 4: while condition was wrong - running through example revealed mistake.
```

```
74          while( (boat_list[next].ring < min_rightmost) && (next < boat_list.size()) );
75
76          boat_list[boat_index].taken = true;
77          rightmost = min_rightmost;
78          counter++;
79
80          // Problem 2: break out as soon as the last boat has been assigned.
81          // Needed because rightmost <= boat_list.back().ring. boat_index not available in while header.
82          if(boat_index == (boat_list.size() - 1)) break;
83      }
84
85      cout << counter << "\n";
86  }
87
88  int main() {
89      int TC; cin >> TC;
90      while(TC--) testcase();
91      return 0;
92  }
```

False Coin

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

int solve(int numberOfCoins, vector< pair<char, vector<int> > > equations);

vector<int> answers;

int main(int argc, char const *argv[])
{
    int datasets;
    cin >> datasets;

    for (int dataset = 0; dataset < datasets; dataset++) {
        //cout << "data set: " << dataset << "\n";

        int numberOfCoins, numberOfWeighings;
        cin >> numberOfCoins >> numberOfWeighings;

        vector< pair<char, vector<int> > > equations;
        equations.clear();
        for (int i = 0; i < numberOfWeighings; i++) {
            //cout << "reading weighing: " << i << " \n";
            int coinsInPan;
            cin >> coinsInPan;

            vector<int> coins;
            coins.clear();
            for (int j = 0; j < (coinsInPan*2); j++)
            {
                int coin;
                cin >> coin;
                coins.push_back(coin);
                //cout << "reading coin: " << j << "\n";
            }

            char operatorSymbol;
            cin >> operatorSymbol;

            equations.push_back(make_pair(operatorSymbol, coins));
        }


        int result = solve(numberOfCoins, equations);
        if (result != 0)
        {
            answers.push_back(result);
        } else {
            answers.push_back(result);
        }

    }

    for (vector<int>::iterator answer = answers.begin(); answer != answers.end(); answer++) {
        cout << *answer << "\n";
    }

    return 0;
}

int solve(int numberOfCoins, vector< pair<char, vector<int> > > equations) {

    vector<int> falseCoins;

    for(int n = 1; n <= numberOfCoins; n++) {
        int coin_id = n;

        bool holding = true;
        //cout << "coin_id: " << coin_id << "\n";

        vector<int> lightWeightedCoins (numberOfCoins+1, 1);
        lightWeightedCoins.at(coin_id) = 0;
        vector<int> heavyWeightedCoins (numberOfCoins+1, 0);
```

```
74          heavyWeightedCoins.at(coin_id) = 1;
75
76          //cout << "initialized weighted vectors \n";
77
78          for (vector< pair<char, vector<int> > >::iterator eq_it = equations.begin(); eq_it != equations.end(); eq_it++)
79          {
80              //cout << "evaluationg equation... coin_id: " << coin_id << "\n";
81              vector<int> coins = eq_it->second;
82              int pan = coins.size() / 2;
83
84              vector<int> leftSum (2, 0);
85              vector<int> rightSum (2, 0);
86
87              int i = 1;
88              for (vector<int>::iterator coin_it = coins.begin(); coin_it != coins.end(); coin_it++)
89              {
90                  //cout << "iterating over coin: " << *coin_it << " adding: " << lightWeightedCoins[*coin_it] << "\n";
91
92                  if (i <= pan) {
93                      leftSum[0] = leftSum[0] + lightWeightedCoins[*coin_it];
94                      leftSum[1] = leftSum[1] + heavyWeightedCoins[*coin_it];
95                  } else {
96                      rightSum[0] = rightSum[0] + lightWeightedCoins[*coin_it];
97                      rightSum[1] = rightSum[1] + heavyWeightedCoins[*coin_it];
98                  }
99
100                 i++;
101             }
102             //cout << "coin_id: " << coin_id << " leftSum light: " << leftSum[0] << " rightSum light: " << ↙
                    ↪ rightSum[0] << "\n";
103             //cout << "coin_id: " << coin_id << " leftSum heavy: " << leftSum[1] << " rightSum heavy: " << ↙
                    ↪ rightSum[1] << "\n";
104
105             char symbol = eq_it->first;
106             if (symbol == '<')
107             {
108                 bool verdict_light = leftSum[0] < rightSum[0];  // assuming false coin is lighter than others
109                 bool verdict_heavy = leftSum[1] < rightSum[1];  // assuming false coin is heavier than others
110
111                 if (verdict_light || verdict_heavy)
112                 {
113                     // possible
114                 } else {
115                     holding = false;
116                     break;
117                 }
118             }
119             if (symbol == '>')
120             {
121                 bool verdict_light = leftSum[0] > rightSum[0];  // assuming false coin is lighter than others
122                 bool verdict_heavy = leftSum[1] > rightSum[1];  // assuming false coin is heavier than others
123
124                 if (verdict_light || verdict_heavy)
125                 {
126                     // possible
127                 } else {
128                     holding = false;
129                     break;
130                 }
131             }
132             if (symbol == '=')
133             {
134                 bool verdict_light = leftSum[0] == rightSum[0]; // assuming false coin is lighter than others
135                 bool verdict_heavy = leftSum[1] == rightSum[1]; // assuming false coin is heavier than others
136
137                 if (verdict_light || verdict_heavy)
138                 {
139                     // possible
140                     //cout << "checking equation: " << leftSum[0] << "=" << rightSum[0] << " OR " << leftSum[1] << ↙
                            ↪ "=" << rightSum[1];
141                 } else {
142                     //cout << "does not hold...";
143                     holding = false;
144                     break;
145                 }
146             }
```

```
147
148             }
149
150             if (holding == true)
151             {
152                 falseCoins.push_back(coin_id);
153             }
154
155         }
156
157         if(falseCoins.size() == 1) {
158             return falseCoins[0];
159         } else {
160             return 0;
161         }
162
163     }
```

Formulas

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

void init_mergesort(vector<int> &racers, vector<int> &aux, int left, int right);
void sort(vector<int> &racers, vector<int> &aux, int left, int right);
void merge(vector<int> &racers, vector<int> &aux, int left, int pivot, int right);

vector<unsigned long> answers;

unsigned long overpasses;

int main(int argc, char const *argv[])
{

    int testcases;
    cin >> testcases;

    for (int testcase = 0; testcase < testcases; testcase++)
    {
        int size;
        cin >> size;

        vector<int> racers;
        vector<int> aux;

        for (int racer = 0; racer < size; racer++)
        {
            int pos;
            cin >> pos;
            racers.push_back(pos);
        }

        aux = racers;

        overpasses = 0;
        init_mergesort(racers, aux, 0, size-1);
        answers.push_back(overpasses % 10000);
    }

    for(vector<unsigned long>::iterator iter = answers.begin(); iter != answers.end(); iter++) {
        cout << *iter << "\n";
    }

    return 0;
}

void init_mergesort(vector<int> &racers, vector<int> &aux, int left, int right) {
    int pivot = (left + right) / 2;

    sort(racers, aux, left, pivot);
    sort(racers, aux, pivot + 1, right);
    merge(racers, aux, left, pivot, right);
}

void sort(vector<int> &racers, vector<int> &aux, int left, int right) {
    if (left < right)
    {
        int pivot = (left + right) / 2;
        sort(racers, aux, left, pivot);
        sort(racers, aux, pivot+1, right);
        merge(racers, aux, left, pivot, right);
    }
}

void merge(vector<int> &racers, vector<int> &aux, int left, int pivot, int right) {

    unsigned long local_overpasses = 0;
    int a = left;
    int i = left;
    int j = pivot + 1;

    // TODO: if left - right smaller than threshold, then use insertion sort!
```

```
74      while( (i <= pivot) && (j <= right) )
75      {
76          if (racers[i] == racers[j]) {
77              aux[a++] = racers[i++];
78          }
79          if (racers[i] < racers[j]) {
80              aux[a++] = racers[i++];
81          }
82          if (racers[i] > racers[j]) {
83              aux[a++] = racers[j++];
84              local_overpasses += (pivot + 1 - left) - (i - left);
85          }
86
87      }
88
89      if (i <= pivot) for (int k = i; k <= pivot; k++) { aux[a++] = racers[k]; i++; }
90      if (j <= right) for (int k = j; k <= right; k++) { aux[a++] = racers[k]; j++; }
91
92      //TODO: make it faster!
93      for (int k = left; k <= right; k++) {
94          racers[k] = aux[k];
95      }
96
97      overpasses += local_overpasses;
98  }
```

Race Tracks

```cpp
#include <vector>
#include <set>
#include <queue>
#include <sstream>
#include <string>
#include <iostream>
using namespace std;

vector<string> answers;

int main(int argc, char const *argv[])
{

    int testsets;
    cin >> testsets;

    for (int testset = 0; testset < testsets; testset++) {

        int m, n;
        cin >> m >> n;

        int s1, s2;
        cin >> s1 >> s2;

        int f1, f2;
        cin >> f1 >> f2;

        int numberObstacles;
        cin >> numberObstacles;

        vector< vector<bool> > obstacles (m, vector<bool>(n));
        for (int o = 0; o < numberObstacles; o++)
        {
            int x1, y1, x2, y2;
            cin >> x1 >> y1 >> x2 >> y2;

            for (int x = x1; x <= x2; x++)
            {
                for (int y = y1; y <= y2; y++) {
                    obstacles[x][y] = true;
                }
            }
        }

        if (obstacles[f1][f2] == true)
        {
            answers.push_back("No solution.");
            continue;
        }

        // visited states
        vector< vector< set<pair<int,int> > > > visited(m, vector<set<pair<int,int> > >(n) );

        // fifo queue for BFS
        queue<pair< pair< pair<int, int>, int>, pair<int, int> > > fifo;

        // adding starting point to fifo queue
        pair<pair< pair<int, int>,int>, pair<int, int> > start_point = make_pair( make_pair( make_pair(s1,s2) , 0), ↙
            ↳ make_pair(0,0));
        fifo.push(start_point);
        visited[s1][s2].insert(make_pair(0,0));

        bool success = false;

        while (!fifo.empty()) {
            pair<
                pair<
                    pair<int, int>, int>,
                    pair<int,int>
                > current_element = fifo.front();

            // remove current element
            fifo.pop();
```

```
73
74              // add to visited
75              int current_x = current_element.first.first.first;
76              int current_y = current_element.first.first.second;
77              int current_hops = current_element.first.second;
78              int current_xv = current_element.second.first;
79              int current_yv = current_element.second.second;
80
81              if ( (current_x == f1) && (current_y == f2) )
82              {
83                  stringstream ss;
84                  ss << "Optimal␣solution␣takes␣" << current_hops << "␣hops.";
85                  answers.push_back(ss.str());
86                  success = true;
87                  break;
88              }
89
90              // get children, add to queue
91              for (int xv = -1; xv <= 1; xv++)
92              {
93                  for (int yv = -1; yv <= 1; yv++) {
94
95                      // updated velocity
96                      int new_vx = current_xv + xv;
97                      int new_vy = current_yv + yv;
98
99                      // potential x and y coordinates
100                     int new_x = current_x + new_vx;
101                     int new_y = current_y + new_vy;
102
103                     // check for velocity range (-3,3), grid range (m,n) and obstacles
104                     if ((new_vx <= 3) && (new_vy <= 3)
105                         && (new_vx >= -3) && (new_vy >= -3)
106                         && (new_x < m) && (new_y < n)
107                         && (new_y >= 0) && (new_x >= 0)
108                         && obstacles[new_x][new_y] != true)
109                     {
110                         pair<int, int> child_velocity = make_pair(new_vx, new_vy);
111                         if (visited[new_x][new_y].find(child_velocity) == visited[new_x][new_y].end())
112                         {
113
114                             if ( (new_x == f1) && (new_y == f2) )
115                             {
116                                 stringstream ss;
117                                 ss << "Optimal␣solution␣takes␣" << current_hops + 1 << "␣hops.";
118                                 answers.push_back(ss.str());
119                                 success = true;
120                                 goto loopend;
121                             }
122
123                             pair< pair<int, int>, int> child_position = make_pair(make_pair(new_x, new_y), ↙
                                    ↳ current_hops + 1);
124                             pair< pair< pair<int, int>, int>, pair<int,int> > fifo_element = ↙
                                    ↳ make_pair(child_position, child_velocity);
125                             fifo.push(fifo_element);
126
127                             // add to visited nodes
128                             visited[new_x][new_y].insert(child_velocity);
129                         }
130                     }
131                 }
132             }
133
134             if(success == true) {
135                 loopend:
136                     break;
137             }
138
139         }
140
141         if (success == false) answers.push_back("No␣solution.");
142     }
143
144
145     for (vector<string>::iterator iter = answers.begin(); iter != answers.end(); iter++)
146     {
```

```
147             cout << *iter << "\n";
148         }
149 
150         return 0;
151     }
```

Burning Coins

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

#define UNDEFINED -1
typedef vector<int> vi;
typedef vector<vi> vii;

int subsequence(int left, int right, vi& coins, vii& dp_table) {
    if(dp_table[left][right] != UNDEFINED) return dp_table[left][right];

    if(left > right) left = right;
    if(left == right) return dp_table[left][right] = coins[left];
    if(right - left == 1) return dp_table[left][right] = max(coins[left], coins[right]);

    int min_left = min(subsequence(left+2, right, coins, dp_table), subsequence(left+1, right-1, coins, dp_table));
    int min_right = min(subsequence(left, right-2, coins, dp_table), subsequence(left+1, right-1, coins, dp_table));
    return dp_table[left][right] = max(coins[left]+min_left, coins[right]+min_right);
}

void testcase() {
    int n; cin >> n;
    vi coins(n);
    for(int i = 0; i < n; ++i) {
        int input; cin >> input;
        coins[i] = input;
    }

    vii dp_table(n, vi(n, UNDEFINED));
    subsequence(0, n-1, coins, dp_table);
    cout << dp_table[0][n-1] << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Jump

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

typedef vector<unsigned long int> vi;

void testcase() {
    int n, k; cin >> n >> k;
    int input; cin >> input; // ignore first input.

    priority_queue<pair<long unsigned int, int>, vector<pair<long unsigned int, int> >, greater<pair<long unsigned int, int> > > min_heap;

    vi dp_table;
    dp_table.push_back(0);

    for(int i = 1; i < n; ++i) {
        while((!min_heap.empty()) && (min_heap.top().second < max(0, i - k))) min_heap.pop();
        min_heap.push(make_pair(dp_table[i-1], i-1));

        int input; cin >> input;
        long unsigned int new_min = input + min_heap.top().first;
        dp_table.push_back(new_min);
    }
    cout << dp_table[n-1] << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Light Pattern

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

#define SWAP 1
#define NO_SWAP 0
typedef vector<int> vi;
typedef vector<vi> vii;

void testcase() {
    int n, k, x; cin >> n >> k >> x;

    vi pattern;
    for(int i = k-1; i >= 0; i--) if(x - pow(2.0, i) >= 0) { x -= pow(2.0, i); pattern.push_back(1); } else { ↙
        ↳ pattern.push_back(0); }

    vii changes(n/k, vi(2));
    for(int i = 0, p = 0, b = 0; i < n; ++i, ++p) {
        int input; cin >> input;
        (pattern[p] == input) ? changes[b][SWAP] += 1 : changes[b][NO_SWAP] += 1;
        if(p == k-1) { p = -1; ++b; }
    }

    vii dp_table(n/k, vi(2));
    dp_table[0][SWAP] = changes[0][SWAP] + 1;
    dp_table[0][NO_SWAP] = changes[0][NO_SWAP];
    for(int b = 1; b < (n/k); ++b) {
        dp_table[b][SWAP] = min(dp_table[b-1][SWAP] + changes[b][SWAP], dp_table[b-1][NO_SWAP] + 2 + changes[b][SWAP]);
        dp_table[b][NO_SWAP] = min(dp_table[b-1][NO_SWAP] + changes[b][NO_SWAP], dp_table[b-1][SWAP] + ↙
            ↳ changes[b][NO_SWAP]);
    }

    cout << min(dp_table[(n/k)-1][SWAP], dp_table[(n/k)-1][NO_SWAP]) << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Longest Path

```cpp
#include <vector>
#include <queue>
#include <iostream>
#include <algorithm>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> AdjacencyList;

void drill(int target, int comingFrom, AdjacencyList& adj, vi& max, vector<priority_queue<int> >& incomingPaths, vi& longest, bool start) {
    if(adj[target].size() == 1 && !start) {
        max[target] = 0;
        incomingPaths[comingFrom].push(1);
        return;
    }

    for(unsigned int outgoing = 0; outgoing < adj[target].size(); ++outgoing) {
        if(adj[target][outgoing] != comingFrom)
            drill(adj[target][outgoing], target, adj, max, incomingPaths, longest, false);
    }

    int first = incomingPaths[target].top(); incomingPaths[target].pop();
    int second = 0;
    if(!incomingPaths[target].empty()) {
        second = incomingPaths[target].top(); incomingPaths[target].pop();
    }

    max[target] = first;
    longest[target] = first + second;
    incomingPaths[comingFrom].push(first+1);
}

void testcase() {
    int vertices; cin >> vertices;

    if(vertices == 1) { int v1, v2; cin >> v1 >> v2; cerr << 1 << "\n"; return; }

    AdjacencyList adj(vertices);
    vi max(vertices, 0);
    vi longest(vertices, 0);
    vector<priority_queue<int> > incomingPaths(vertices);

    for(int input = 0; input < vertices-1; ++input) {
        int v1, v2; cin >> v1 >> v2;
        adj[v1].push_back(v2);
        adj[v2].push_back(v1);
    }

    drill(0, 0, adj, max, incomingPaths, longest, true);
    cout << *max_element(longest.begin(), longest.end())+1 << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

```cpp
#include <vector>
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vii;
int N;

pair<int, int> DFS(int start, vii& adj, vi& dist, vi& visited) {
    queue<int> fifo;
    fifo.push(start);
```

```
    visited[start] = 1;

    while(!fifo.empty()) {
        int parent_id = fifo.front(); fifo.pop();
        for(int child = 0; child < adj[parent_id].size(); ++child) {
            int child_id = adj[parent_id][child];
            if(visited[child_id] == 0) {
                fifo.push(child_id);
                visited[child_id] = 1;
                dist[child_id] = dist[parent_id] + 1;
            }
        }
    }
    vi::iterator it = max_element(dist.begin(), dist.end());
    pair<int, int> val;
    val.first = it - dist.begin();
    val.second = *it;
    return val;
}

void testcase() {
    cin >> N;    // N vertices, by definition N-1 edges.
    vii adj(N);
    vi dist(N, 0);
    vi visited(N, 0);

    for(int n = 0; n < N-1; ++n) {
        int v1, v2; cin >> v1 >> v2;
        adj[v1].push_back(v2);
        adj[v2].push_back(v1);
    }
    if(N == 1) { cout << 0 << "\n"; return; }

    pair<int, int> pass1 = DFS(0, adj, dist, visited);
    dist.assign(N, 0); visited.assign(N, 0);
    pair<int, int> pass2 = DFS(pass1.first, adj, dist, visited);
    cout << pass2.second+1 << "\n";
}

int main() {
    cin.sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Ants

```cpp
#include <vector>
#include <iostream>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/graph_traits.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/kruskal_min_spanning_tree.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
using namespace std;
using namespace boost;

typedef property<edge_weight_t, int, property<edge_index_t, int> > EdgeProperties;
typedef property<vertex_index_t, int> VertexProperties;
typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, EdgeProperties> Graph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef property_map<Graph, edge_weight_t>::type WeightMap;
typedef property_map<Graph, edge_index_t>::type EIndexMap;
typedef property_map<Graph, vertex_index_t>::type VIndexMap;
typedef graph_traits<Graph>::edge_iterator EdgeIterator;
typedef vector<int> vi;
typedef vector<vi> vii;
typedef vector<Edge> ve;

void testcase() {
    int N, M, S, a, b; cin >> N >> M >> S >> a >> b;

    Graph g;
    WeightMap weightMap = get(edge_weight, g);
    EIndexMap eIndexMap = get(edge_index, g);
    vii weights(M);

    for(int e = 0; e < M; ++e) {
        int t1, t2; cin >> t1 >> t2;
        for(int s = 0; s < S; ++s) {
            int s_weight; cin >> s_weight;
            weights[e].push_back(s_weight);
        }

        Edge edge; bool success;
        tie(edge, success) = add_edge(t1, t2, g);
        eIndexMap[edge] = e;
    }

    Graph final;
    WeightMap weightMapFinal = get(edge_weight, final);

    for(int s = 0; s < S; ++s) {
        int hive; cin >> hive;

        EdgeIterator eit, eend;
        for(tie(eit, eend) = edges(g); eit != eend; ++eit) weightMap[*eit] = weights[eIndexMap[*eit]][s];

        ve mst(num_vertices(g)-1);
        kruskal_minimum_spanning_tree(g, mst.begin());
        for(ve::iterator edge = mst.begin(); edge != mst.end(); ++edge) {
            Edge newEdge; bool success;
            tie(newEdge, success) = add_edge(source(*edge, g), target(*edge, g), final);
            weightMapFinal[newEdge] = weightMap[*edge];
        }
    }

    vi d(num_vertices(final));
    dijkstra_shortest_paths(final, vertex(a, final), distance_map(&d[0]));
    cout << d[b] << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Bridges

```cpp
#include <vector>
#include <iostream>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/biconnected_components.hpp>
using namespace std;
using namespace boost;

typedef property<vertex_index_t, int> VertexProperties;
typedef adjacency_list< vecS, vecS, undirectedS, VertexProperties, no_property> Graph;
typedef property_map<Graph, vertex_index_t>::type VIndexMap;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::adjacency_iterator AIter;
typedef vector<int> vi;
typedef vector<Vertex> vv;
typedef pair<int, int> pi;

void testcase() {
    int N, M; cin >> N >> M;

    if(N == 0 || M == 0) { cout << "0\n"; return; }

    Graph g(N);
    VIndexMap index = get(vertex_index, g);

    for(int m = 0; m < M; ++m) {
        int v1, v2; cin >> v1 >> v2;
        add_edge(v1, v2, g);
    }

    vv art_points;
    vi discover_time(num_vertices(g));
    vi low_point(num_vertices(g));
    vector<pi> bridges;
    articulation_points(g,
                        back_inserter(art_points),
                        discover_time_map(&discover_time[0]).lowpoint_map(&low_point[0]));

    // workaround for "root not chosen as articulation point if only one child".
    if(out_degree(vertex(1, g), g) == 1) {
        Vertex root = vertex(1, g);
        art_points.insert(art_points.begin(), root);
    }
    for(int v = 0; v < art_points.size(); ++v) {
        Vertex art_point = art_points[v];
        AIter neighbour, neighbour_end;
        for(tie(neighbour, neighbour_end) = adjacent_vertices(art_point, g); neighbour != neighbour_end; ++neighbour) {
            if(low_point[*neighbour] > discover_time[art_point]) {
                //cout << "bridge found between: " << index[art_point] << "-" << index[*neighbour] << "\n";
                bridges.push_back(make_pair(min(index[art_point], index[*neighbour]), max(index[art_point], ↙
                    ↳ index[*neighbour])));
            }
        }
    }

    sort(bridges.begin(), bridges.end());
    cout << bridges.size() << "\n";
    for(int b = 0; b < bridges.size(); ++b) {
        cout << bridges[b].first << "␣" << bridges[b].second << "\n";
    }
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

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

```cpp
using namespace std;

#define UNVISITED 0
#define VISITED 1
#define EXPLORED 2

typedef vector<int> vi;
typedef vector<vi> vii;
typedef pair<int, int> pi;

vi visited;
vi dfs_num;
vi dfs_low;

void dfs(int vertex, int parent, vii& adj, int counter) {
    for(signed int child = 0; child < adj[vertex].size(); ++child) {
        int child_vertex = adj[vertex][child];
        if(child_vertex != parent) {
            if(visited[child_vertex] == EXPLORED) {
                dfs_low[vertex] = min(dfs_num[child_vertex], dfs_low[vertex]);
            }

            if(visited[child_vertex] == UNVISITED) {
                visited[child_vertex] = EXPLORED;
                dfs_num[child_vertex] = ++counter;
                dfs_low[child_vertex] = dfs_num[child_vertex];
                dfs(child_vertex, vertex, adj, counter);
            }
        }
    }

    dfs_low[parent] = min(dfs_low[parent], dfs_low[vertex]);
    visited[vertex] = VISITED;
}

void testcase() {
    int N, M; cin >> N >> M;
    visited.clear(); dfs_low.clear(); dfs_num.clear();
    vii adj(N); visited.assign(N, UNVISITED); dfs_num.assign(N, 0); dfs_low.assign(N, 0);

    if(N == 0 || N == 0) { cout << "0\n"; return; }

    for(int m = 0; m < M; ++m) {
        int v1, v2; cin >> v1 >> v2;
        adj[(v1-1)].push_back(v2-1);
        adj[(v2-1)].push_back(v1-1);
    }

    dfs_num[0] = 0; dfs_low[0] = 0; visited[0] = EXPLORED;
    dfs(0, 0, adj, 0);

    vector<pi> bridges;
    set<int> art_points;
    for(int u = 0; u < N; ++u) {
        for(int v = 0; v < adj[u].size(); ++v) {
            if(dfs_low[adj[u][v]] > dfs_num[u]) {
                bridges.push_back(make_pair(min(u, adj[u][v]), max(u, adj[u][v])));
            }
            if(dfs_low[adj[u][v]] >= dfs_num[u]) {
                // if it is not root, or it is root but has more than 1 child:
                art_points.insert(u);
            }
        }
    }
    sort(bridges.begin(), bridges.end());
    cout << bridges.size() << "\n";
    for(signed int b = 0; b < bridges.size(); ++b) {
        cout << bridges[b].first+1 << "␣" << bridges[b].second+1 << "\n";
    }
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Build The Graph

```cpp
#include <iostream>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/kruskal_min_spanning_tree.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
using namespace std;
using namespace boost;

// create internal properties
typedef property<vertex_index_t, int> IndexProperty;
typedef property<edge_weight_t, int> WeightProperty;

// adjacency list with properties
typedef adjacency_list<vecS, vecS, undirectedS, no_property, WeightProperty, IndexProperty> Graph;

// Vertex and edge type
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::edge_iterator EdgeIterator;

// Property maps for accessing the properties
typedef property_map<Graph, edge_weight_t>::type WeightMap;
typedef property_map<Graph, vertex_index_t>::type IndexMap;

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

    for(int i = 0; i < t; i++) {
        int m, n; cin >> n >> m;

        Graph G(n);
        WeightMap weightMap = get(edge_weight, G);

        for(int j = 0; j < m; j++) {
            int v1, v2, w;
            cin >> v1 >> v2 >> w;
            Edge e;
            tie(e, tuples::ignore) = add_edge(v1, v2, G);
            weightMap[e] = w;
        }

        vector<Edge> spanningTree;
        kruskal_minimum_spanning_tree(G, back_inserter(spanningTree));
        int sumOfWeights = 0;

        Graph mstGraph(n);
        WeightMap mstWeightMap = get(edge_weight, mstGraph);
        for (vector<Edge>::iterator ei = spanningTree.begin(); ei != spanningTree.end(); ++ei) {
            sumOfWeights += weightMap[*ei];
        }

        vector<int> distances(n);
        vector<Vertex> p_map(num_vertices(G));

        Vertex startVertex = vertex(0, G);
        dijkstra_shortest_paths(G, startVertex, predecessor_map(&p_map[0]).distance_map(&distances[0]));

        int longestDistance = 0;
        for(int k = 0; k < n; k++) {
            int distance = distances[k];

            if(distance > longestDistance) {
                longestDistance = distance;
            }
        }

        cout << sumOfWeights << "␣" << longestDistance << endl;

        /* Playing around with backtracking shortest path.
        IndexMap index;
        int target = 3;
        while(target != p_map[index[vertex(target, G)]]) {
```

```
74              cout << target << "-" << p_map[index[vertex(target, G)]] << "\n";
75              target = p_map[index[vertex(target, G)]];
76          }
77          */
78      }
79  }
```

Deleted Entries

```cpp
#include <vector>
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vii;

int k;

void testcase() {
    int n, m, k;
    cin >> n >> m >> k;

    vii adj(n);
    vii groups(k);
    vi col(n, -1);

    for(int e = 0; e < m; ++e) {
        int v1, v2; cin >> v1 >> v2;
        adj[v1].push_back(v2);
        adj[v2].push_back(v1);
    }

    queue<int> q; // lifo
    int c = 0;
    q.push(0);
    col[0] = c;
    groups[c].push_back(0);

    while(!q.empty()) {
        const int v = q.front(); q.pop();
        for(int child = 0; child < adj[v].size(); ++child) {
            const int u = adj[v][child];
            if(col[u] != -1) continue;

            c = (c == k-1) ? 0 : ++c;
            if(col[v] == c) { c = (c == k-1) ? 0 : ++c; }
            col[u] = c;
            groups[c].push_back(u);
            q.push(u);
        }
    }

    if(n >= k && find(col.begin(), col.end(), -1) == col.end()) {
        cout << "yes\n";
        for(int g = 0; g < k; ++g) {
            cout << groups[g].size();
            for(int i = 0; i < groups[g].size(); ++i) {
                cout << "␣" << groups[g][i];
            }
            cout << "\n";
        }
    } else {
        cout << "no\n";
    }
    col.clear();
    adj.clear();
    groups.clear();
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

Shy Programmers

```cpp
#include <iostream>
#include <vector>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/boyer_myrvold_planar_test.hpp>
using namespace std;
using namespace boost;

typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;

void testcase() {
    int N, M; cin >> N >> M;
    Graph g(N+1);
    vector<int> processed(N, 0);
    for(int m = 0; m < M; ++m) {
        int a, b; cin >> a >> b;
        add_edge(a, b, g);
        if(!processed[a]) { add_edge(a, N+1, g); processed[a] = 1; }
        if(!processed[b]) { add_edge(b, N+1, g); processed[b] = 1; }
    }

    if(boyer_myrvold_planarity_test(g))
        cout << "yes\n";
    else
        cout << "no\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Algocoon Group

Missing.

Buddies

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <utility>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/max_cardinality_matching.hpp>
using namespace std;
using namespace boost;

typedef vector<int> vi;
typedef pair<int, int> ii;

typedef property<vertex_index_t, int> VertexProperties;
typedef property<edge_weight_t, int> EdgeProperties;
typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, EdgeProperties> Graph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef property_map<Graph, vertex_index_t>::type VIndexMap;
typedef graph_traits<Graph>::edge_iterator EdgeIterator;

void testcase() {
    int n, c, f; cin >> n >> c >> f;
    map<string, vi> char_map;

    for(int student = 0; student < n; ++student) {
        for(int characteric = 0; characteric < c; ++characteric) {
            string input; cin >> input;
            if(char_map.count(input) == 0) {
                vi students; students.push_back(student);
                char_map.insert(make_pair(input, students));
            }
            else { char_map[input].push_back(student); }
        }
    }

    map<ii, int> edges;
    for(map<string, vi>::iterator iter = char_map.begin(); iter != char_map.end(); ++iter) {
        pair<string, vi> value_pair = *iter;
        vi& values = value_pair.second;
        for(int s1 = 0; s1 < values.size()-1; ++s1) {
            for(int s2 = s1+1; s2 < values.size(); ++s2) {
                ii edge = make_pair(values[s1], values[s2]);
                if(edges.count(edge) == 0) { edges.insert(make_pair(edge, 1)); }
                else { edges[edge]++; }
            }
        }
    }

    Graph g(n);
    for(map<ii, int>::iterator iter = edges.begin(); iter != edges.end(); ++iter) {
        pair<ii, int> edge_pair = *iter;
        //cout << "edge: " << edge_pair.first.first << "-" << edge_pair.first.second << " weight: " << ↙
             ↳ edge_pair.second << "\n";
        if(edge_pair.second > f) {
            add_edge(edge_pair.first.first, edge_pair.first.second, g);
        }
    }

    vector<Vertex> mateMap (num_vertices(g));
    bool matching_success = checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);

    if(matching_success) {
        if(matching_size(g, &mateMap[0]) < n/2 )  cout << "optimal\n";
        else cout << "not␣optimal\n";
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
```

```
73      return 0;
74  }
```

Satellites

```cpp
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/max_cardinality_matching.hpp>
#include <boost/graph/bipartite.hpp>
#include <boost/graph/depth_first_search.hpp>
using namespace std;
using namespace boost;

#define UNVISITED 0
#define VISITED 1
#define LEFT 0
#define RIGHT 1

typedef vector<int> vi;
typedef property<vertex_index_t, int> VertexProperties;
typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, no_property> Graph;
typedef adjacency_list<vecS, vecS, directedS, VertexProperties, no_property> Digraph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::vertex_iterator VertexIterator;
typedef property_map<Graph, vertex_index_t>::type VIndexMap;

vi visited;

struct mark_visited : public boost::dfs_visitor<> {
    template <class Vertex, class Digraph>
    void finish_vertex(Vertex u, const Digraph& g) {
        visited[u] = VISITED;
        //cout << u << " set to visited. \n";
    }
};

void testcase() {
    int groundstations, satellites, links; cin >> groundstations >> satellites >> links;

    Digraph g_final(groundstations + satellites);
    Graph g(groundstations + satellites);

    vi color(num_vertices(g));
    for(int edge = 0; edge < links; ++edge) {
        int v1, v2; cin >> v1 >> v2;
        v2 = v2 + groundstations;
        add_edge(v1, v2, g);
        add_edge(v1, v2, g_final);
        color[v1] = LEFT;
        color[v2] = RIGHT;
    }

    vector<Vertex> mateMap(num_vertices(g), UNVISITED);
    bool success = checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);


    visited.clear();
    visited.assign(num_vertices(g), UNVISITED);
    for(int matching = 0; matching < mateMap.size(); ++matching) {
        if(color[matching] == RIGHT && mateMap[matching] != graph_traits<Graph>::null_vertex())
            add_edge(matching, mateMap[matching], g_final);  // add an edge from R to L.
        if(mateMap[matching] == graph_traits<Graph>::null_vertex() && color[matching] == LEFT)
            visited[matching] = VISITED;
    }

    mark_visited vis;
    for(int start = 0; start < visited.size(); ++start) {
        if((color[start] == LEFT) && (visited[start] == VISITED)) {
            //cout << "start dfs at " << start << " visited: " << visited[start] << "\n";
            //depth_first_search(g_final, root_vertex(vertex(start, g_final)).visitor(vis));
            vector<default_color_type> colors(num_vertices(g_final));
            depth_first_visit(g_final, vertex(start, g_final), vis, &colors[0]);
        }
    }

```

```
74      vi solution_ground;
75      vi solution_sat;
76      for(int c = 0; c < color.size(); ++c) {
77          if(color[c] == LEFT && visited[c] == UNVISITED) {
78              solution_ground.push_back(c);
79          }
80          if(color[c] == RIGHT && visited[c] == VISITED) {
81              solution_sat.push_back(c-groundstations);
82          }
83      }
84
85      cout << solution_ground.size() << "␣" << solution_sat.size() << "\n";
86      for(int sol = 0; sol < solution_ground.size(); ++sol) cout << solution_ground[sol] << "␣";
87      for(int sol = 0; sol < solution_sat.size(); ++sol) cout << solution_sat[sol] << "␣";
88      cout << "\n";
89  }
90
91  int main() {
92      int TC; cin >> TC;
93      while(TC--) testcase();
94      return 0;
95  }
```

Kingdom Defence

```cpp
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.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 property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
typedef graph_traits<Graph>::edge_descriptor Edge;

void add_edge(int f, int t, int cap, Graph& g) {
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap rev_edge = get(edge_reverse, g);

    Edge edge;
    tie(edge, tuples::ignore) = add_edge(f, t, g);
    Edge reverse_edge;
    tie(reverse_edge, tuples::ignore) = add_edge(t, f, g);
    capacity[edge] = cap;
    rev_edge[edge] = reverse_edge;
    capacity[reverse_edge] = 0;
    rev_edge[reverse_edge] = edge;
}

void testcase() {
    int V, E; cin >> V >> E;
    Graph g(V+2);
    int source = V;
    int sink = V+1;

    vector<int> vertices;
    for(int v = 0; v < V; ++v) {
        int g, d; cin >> g >> d;
        vertices.push_back(d - g);
    }

    for(int e = 0; e < E; ++e) {
        int f, t, lb, ub; cin >> f >> t >> lb >> ub;
        add_edge(f, t, ub-lb, g);
        vertices[f] += lb;
        vertices[t] -= lb;
    }

    int flow_out = 0;
    bool all_pos = true;
    for(int v = 0; v < V; ++v) {
        if(vertices[v] < 0) {
            add_edge(source, v, abs(vertices[v]), g);
        } else if(vertices[v] > 0) {
            all_pos = false;
            add_edge(v, sink, vertices[v], g);
            flow_out += abs(vertices[v]);
        }
    }

    int max_flow = push_relabel_max_flow(g, source, sink);
    (max_flow == flow_out || all_pos) ? cout << "yes\n" : cout << "no\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

Coin Tossing

```cpp
#include <iostream>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>
#include <boost/tuple/tuple.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 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;
typedef graph_traits<Graph>::edge_descriptor Edge;

void add_edge(int from, int to, int c, Graph& g) {
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap reverse = get(edge_reverse, g);
    ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);

    Edge there, back;
    tie(there, tuples::ignore) = add_edge(from, to, g);
    tie(back, tuples::ignore) = add_edge(to, from, g);
    capacity[there] = c;
    capacity[back] = 0;
    reverse[there] = back;
    reverse[back] = there;
}

void testcase() {
    int N, M; cin >> N >> M;
    Graph g(N+M+2);
    int source = N+M+1;
    int sink = source + 1;

    for(int m = N; m < N+M; ++m) {
        int p1, p2, outcome;
        cin >> p1 >> p2 >> outcome;
        add_edge(source, m, 1, g);

        if(outcome == 1) {
            add_edge(m, p1, 1, g);
        }
        if(outcome == 2) {
            add_edge(m, p2, 1, g);
        }
        if(outcome == 0) {
            add_edge(m, p1, 1, g);
            add_edge(m, p2, 1, g);
        }
    }

    int sum = 0;
    for(int p = 0; p < N; ++p) {
        int score;  cin >> score;
        sum += score;
        add_edge(p, sink, score, g);
    }

    int f_max = push_relabel_max_flow(g, source, sink);
    if(M == sum && f_max == sum) cout << "yes\n";
    else cout << "no\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

Antenna

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
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 ceil_to_double(const K::FT& x)
{
  double a = ceil(CGAL::to_double(x));
  while (a < x) a += 1;
  while (a-1 >= x) a -= 1;
  return a;
}

void testcase(int n) {
    vector<K::Point_2> citizens;
    for(int coord = 0; coord < n; ++coord) {
        double x, y; cin >> x >> y;
        K::Point_2 citizen(x, y);
        citizens.push_back(citizen);
    }

    Min_circle mc(citizens.begin(), citizens.end(), true);  // true important for speed.
    Traits::Circle c = mc.circle();
    K::FT radius = sqrt(c.squared_radius());

    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);  // scientific notation will be used otherwise!
    cout << ceil_to_double(radius) << "\n";
}

int main() {
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
    return 0;
}
```

Almost Antenna

```cpp
#include <set>
#include <iostream>
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
typedef CGAL::Min_circle_2_traits_2<K> MinCircleTraits;
typedef CGAL::Min_circle_2<MinCircleTraits> Min_circle;
typedef Min_circle::Support_point_iterator support_iter;

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

void testcase(int n) {
    vector<K::Point_2> points;
    for(int point = 0; point < n; ++point) {
        double x, y; cin >> x >> y;
        K::Point_2 p(x, y);
        points.push_back(p);
    }

    Min_circle min_circle(points.begin(), points.end(), true);
    MinCircleTraits::Circle c = min_circle.circle();
    K::FT old_radius = c.squared_radius();
    K::FT min_radius; bool min_radius_set = false;

    for(support_iter iter = min_circle.support_points_begin(); iter != min_circle.support_points_end(); ++iter) {
        // find supporting point in set. Delete it temporarily.
        vector<K::Point_2>::iterator temp_it = find(points.begin(), points.end(), *iter);
        K::Point_2 point = *temp_it;
        points.erase(temp_it);

        // create new min_circle, get squared radius.
        Min_circle temp_circle (points.begin(), points.end(), true);
        MinCircleTraits::Circle actual_circle = temp_circle.circle();
        K::FT new_radius = actual_circle.squared_radius();

        // compare radius of old min_circle with new one.
        if(new_radius == old_radius) {
            min_radius = old_radius; break;
        } else if(!min_radius_set || new_radius < min_radius) {
            min_radius = new_radius;
            min_radius_set = true;
        }

        // reinsert the point
        points.push_back(point);
    }

    double result = ceil_to_double(CGAL::sqrt(min_radius));
    cout << result << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
    return 0;
}
```

Hit

```cpp
#include <iostream>
#include <vector>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

void testcase(int n) {
    double x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    K::Point_2 p1(x1, y1);
    K::Point_2 p2(x2, y2);
    K::Ray_2 ray(p1, p2);

    vector<K::Segment_2> obstacles;
    for(int o = 0; o < n; ++o) {
        double r, s, t, u;
        cin >> r >> s >> t >> u;
        K::Point_2 p1(r, s);
        K::Point_2 p2(t, u);
        K::Segment_2 obstacle(p1, p2);
        obstacles.push_back(obstacle);
    }

    bool intersect = false;
    for(int obstacle = 0; obstacle < obstacles.size(); ++obstacle) {
        if(CGAL::do_intersect(obstacles[obstacle], ray)) {
            intersect = true;
            break;
        }
    }

    (intersect) ? cout << "yes\n" : cout << "no\n";
}

int main() {
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
}
```

First Hit

```cpp
#include <iostream>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/enum.h>
#include <climits>
using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel K;

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;
}

void testcase(int n) {
    K::Ray_2 ray;
    double x1, y1, x2, y2; cin >> ray;

    bool min_exists = false;
    K::FT current_dist;
    K::Point_2 current_point;

    for(size_t o = 0; o < n; ++o) {
        double r, s, t, u; cin >> r >> s >> t >> u;
        K::Point_2 p1(r, s);
        K::Point_2 p2(t, u);
        K::Segment_2 obstacle (p1, p2);

        if(CGAL::do_intersect(ray, obstacle)) {
            K::Point_2 intersection_point;
            CGAL::Object o = CGAL::intersection(ray, obstacle);
            if(const K::Point_2* p = CGAL::object_cast<K::Point_2>(&o))
                intersection_point = *p;
            else if (const K::Segment_2* s = CGAL::object_cast<K::Segment_2>(&o))
                intersection_point =
                        CGAL::has_smaller_distance_to_point(ray.source(), s->source(), s->target()) ?
                        s->source() : s->target();
            else throw runtime_error("strange␣segment␣intersection");
            K::FT intersection_dist = CGAL::squared_distance(intersection_point, ray.source());
            if(!min_exists || current_dist > intersection_dist) {
                current_dist = intersection_dist;
                current_point = intersection_point;
                min_exists = true;
            }
        }
    }

    if(min_exists) cout << floor_to_double(current_point.x()) << "␣" << floor_to_double(current_point.y()) << "\n";
    else cout << "no\n";
}

int main() {
    cin.sync_with_stdio(false);
    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
}
```

Search Snippets

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
#include <functional>
#include <cmath>
using namespace std;

typedef vector<int> vi;

void testcase() {
    int n; cin >> n;
    vector<vi> posting_list(n);

    vi Npositions(n);
    for(int i = 0; i < n; ++i) { int m; cin >> m; Npositions[i] = m; }

    for(int word = 0; word < Npositions.size(); ++word) {
        for(int position = 0; position < Npositions[word]; ++position) {
            int input_position; cin >> input_position;
            posting_list[word].push_back(input_position);
        }
    }

    vi pointers(n, 0);
    priority_queue<int> max_heap;
    priority_queue<pair<int, int>, std::vector<pair<int, int> >, greater<pair<int, int> > > min_heap;

    for(int list = 0; list < n; ++list) {
        int value = posting_list[list][pointers[list]];
        max_heap.push(value);
        min_heap.push(make_pair(value, list));
    }

    int min_interval = 1073741825;
    while(true) {
        pair<int, int> min_pair = min_heap.top(); min_heap.pop();
        int min_value = min_pair.first;
        int min_list = min_pair.second;

        int max_value = max_heap.top();
        int min_new = abs(max_value - min_value);
        min_interval = min(min_new, min_interval);

        if(pointers[min_list] == posting_list[min_list].size()-1) { break; }
        int jump = sqrt(posting_list[min_list].size());
        while((pointers[min_list]+jump < posting_list[min_list].size()-1) &&
            (posting_list[min_list][pointers[min_list]+jump] < min_heap.top().first)) {
            pointers[min_list] += jump;
        }
        pointers[min_list]++;

        int new_value = posting_list[min_list][pointers[min_list]];
        max_heap.push(new_value);
        min_heap.push(make_pair(new_value, min_list));
    }

    cout << min_interval+1 << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Bistro

```cpp
#include <vector>
#include <iostream>
#include <cmath>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
typedef Triangulation::Finite_faces_iterator faces_iterator;

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;
}

void testcase(int n) {
    vector<K::Point_2> delaunay_vertices;
    for(int i = 0; i < n; ++i) {
        K::Point_2 p; cin >> p;
        delaunay_vertices.push_back(p);
    }

    Triangulation t;
    t.insert(delaunay_vertices.begin(), delaunay_vertices.end());

    int points; cin >> points;
    for(int i = 0; i < points; ++i) {
        K::Point_2 p; cin >> p;
        Triangulation::Vertex_handle v = t.nearest_vertex(p);
        K::Point_2 vp = v->point();
        K::FT distance = CGAL::squared_distance(p, vp);
        cout << floor_to_double(distance) << "\n";
    }
}

int main() {
    cin.sync_with_stdio(false);
    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
    return 0;
}
```

Germs

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
#include <CGAL/Triangulation_vertex_base_with_info_2.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel     K;
typedef CGAL::Triangulation_vertex_base_with_info_2<int, K>     Vb;
typedef CGAL::Triangulation_data_structure_2<Vb>                Tds;
typedef CGAL::Delaunay_triangulation_2<K, Tds>                  Delaunay;
typedef Delaunay::Finite_edges_iterator                         FEI;
typedef Delaunay::Finite_vertices_iterator                      FVI;

void testcase(int N) {
    double left, bottom, right, top; cin >> left >> bottom >> right >> top;
    vector<K::Segment_2> rectangle;
    rectangle.push_back(K::Segment_2(K::Point_2(left, bottom), K::Point_2(left, top)));
    rectangle.push_back(K::Segment_2(K::Point_2(left, top), K::Point_2(right, top)));
    rectangle.push_back(K::Segment_2(K::Point_2(right, top), K::Point_2(right, bottom)));
    rectangle.push_back(K::Segment_2(K::Point_2(right, bottom), K::Point_2(left, bottom)));

    vector<pair<K::Point_2, int> > points;
    for(int b = 0; b < N; ++b) {
        double x, y; cin >> x >> y;
        points.push_back(make_pair(K::Point_2(x, y), b));
    }

    Delaunay t;
    t.insert(points.begin(), points.end());

    vector<pair<double, pair<int, int> > > edges;
    for(FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
        Delaunay::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
        Delaunay::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);
        K::FT edge_length = CGAL::sqrt(CGAL::squared_distance(v1->point(), v2->point()));
        edges.push_back(make_pair(edge_length, make_pair(v1->info(), v2->info())));
    }

    for(FVI p = t.finite_vertices_begin(); p != t.finite_vertices_end(); ++p) {
        Delaunay::Vertex_handle vertex = p;
        K::FT min; bool min_set = false;
        for(int seg = 0; seg < 4; ++seg) {
            K::FT seg_min = CGAL::squared_distance(rectangle[seg], vertex->point());
            if(min_set == false || min > seg_min) { min_set = true; min = seg_min; }
        }
        edges.push_back(make_pair(2*CGAL::sqrt(min), make_pair(p->info(), p->info())));
    }

    sort(edges.begin(), edges.end());

    int dead = 0;
    int pointer = 0;
    int h = 0;
    bool first_time = true;
    vector<int> deadlist(N, 0);

    while(dead != N) {
        double min_length = 2 * (pow(h, 2.0) + 0.5);
        int temp_dead = 0;
        while(edges[pointer].first <= min_length && pointer < edges.size()) {
            int v1 = edges[pointer].second.first;
            int v2 = edges[pointer].second.second;
            if(deadlist[v1] == 0) { ++temp_dead; deadlist[v1] = 1; }
            if(deadlist[v2] == 0) { ++temp_dead; deadlist[v2] = 1; }
            ++pointer;
        }
        if(dead == 0 && temp_dead > 0) cout << h << "␣";
        dead += temp_dead;
        if((N-dead)/(double)N < 0.5 && first_time) {
            cout << h << "␣";
            first_time = false;
```

```
74          }
75          if(N == dead) cout << h << "\n";
76          ++h;
77      }
78  }
79
80  int main() {
81       while(true) {
82          int N; cin >> N;
83          if(N == 0) return 0;
84          testcase(N);
85      }
86  }
```

Graypes

```cpp
#include <vector>
#include <iostream>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h> // use inexact because Input points == output points.
#include <CGAL/Delaunay_triangulation_2.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
typedef Triangulation::Finite_edges_iterator FEI;

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

void testcase(int n) {
    vector<K::Point_2> points;
    for(int i = 0; i < n; ++i) {
        K::Point_2 p; cin >> p;
        points.push_back(p);
    }

    Triangulation t;
    t.insert(points.begin(), points.end());
    K::FT min_length;
    bool min_set = false;
    for (FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); e++) {
        // REMEMBER bad idea: K::Segment_2 seg = t.segment(edge); seg.squared_length().
        Triangulation::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
        Triangulation::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);

        K::FT length = CGAL::squared_distance(v1->point(), v2->point());
        if(!min_set || min_length > length) {
            min_length = length;
            min_set = true;
        }
    }

    double seconds = ceil_to_double(CGAL::sqrt(min_length)*50);
    cout << seconds << "\n";
}

int main() {
    cin.sync_with_stdio(false);
    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
    while(true) {
        int n; cin >> n;
        if(n == 0) return 0;
        testcase(n);
    }
}
```

H1N1

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel     K;
typedef CGAL::Delaunay_triangulation_2<K>                       Delaunay;
typedef Delaunay::All_faces_iterator                            AFI;
typedef map<Delaunay::Face_handle, int>                         StateMap;

int testcase(int N) {
    vector<K::Point_2> points;
    for(int n = 0; n < N; ++n) {
        double x, y; cin >> x >> y;
        points.push_back(K::Point_2(x, y));
    }

    int M; cin >> M;
    vector<pair<K::Point_2, double> > people;
    for(int m = 0; m < M; ++m) {
        double x, y, d; cin >> x >> y >> d;
        people.push_back(make_pair(K::Point_2(x, y), d));
    }

    StateMap state;
    Delaunay t;
    t.insert(points.begin(), points.end());

    for(int p = 0; p < M; ++p) {
        K::Point_2 coord = people[p].first;
        K::FT d = people[p].second;

        if(CGAL::squared_distance(coord, t.nearest_vertex(coord)->point()) < d) {
            cout << "n";
            continue;
        }

        Delaunay::Face_handle start_face = t.locate(coord);
        if(t.is_infinite(start_face)) {
            cout << "y";
            continue;
        }

        bool stop = false;
        queue<Delaunay::Face_handle> fifo;
        fifo.push(start_face);
        int bfs_id = p+1;
        state[start_face] = bfs_id;
        while(!fifo.empty()) {
            Delaunay::Face_handle f = fifo.front(); fifo.pop();
            for(int e = 0; e < 3; ++e) {
                K::Segment_2 seg = t.segment(f, e);
                Delaunay::Face_handle neighbour = f->neighbor(e);

                if((seg.squared_length() >= 4*d) && state[neighbour] != bfs_id){
                    if(t.is_infinite(neighbour)) {
                        cout << "y";
                        stop = true;
                        break;
                    }
                    fifo.push(neighbour);
                    state[neighbour] = bfs_id;
                }
            }
            if(stop) break;

        }
        if(!stop) cout << "n";
    }
    cout << "\n";
}
```

```
74
75  int main() {
76      while(true) {
77          int N; cin >> N;
78          if(N == 0) return 0;
79          testcase(N);
80      }
81  }
```

HikingMaps

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/ch_jarvis.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;      // does not pass last TC with exact constructions.
typedef vector<int> vi;
typedef vector<vi> vii;

void testcase() {
    int M, N; cin >> M >> N; // M-1 legs, N maps.

    vector<pair<K::Point_2, K::Point_2> > legs;     // using a vector a segment, prevents from passing the 4th TC.
    K::Point_2 prev;
    cin >> prev;
    for(int m = 1; m < M; ++m) {
        int x, y; cin >> x >> y;
        K::Point_2 now(x, y);
        legs.push_back(make_pair(prev, now));
        prev = now;
    }

    vii lists(M-1); // storing "leg contained by map" data.
    for(int n = 0; n < N; ++n) {
        vector<K::Point_2> points(6);
        for(int i = 0; i < 6; ++i)
            cin >> points[i];

        vector<K::Point_2> ccw; // store the given vertices in counter-clockwise fashion.
        CGAL::ch_jarvis_march(points.begin(), points.end(), points[0], points[0], back_inserter(ccw));
        if(points[1] != ccw[1]) {   // ugly... making sure two consecutive vertices span a triangle edge.
            ccw.clear();
            CGAL::ch_jarvis_march(points.begin(), points.end(), points[1], points[1], back_inserter(ccw));
        }

        for(int l = 0; l < legs.size(); ++l) {  // iterate over each leg.
            bool isOutside;     // is set if to true, if origin or source is to the right to the edges.
            for(int p = 0; p < ccw.size()-1; p = p+2) {
                isOutside = (CGAL::right_turn(ccw[p], ccw[p+1], legs[l].first) ||
                CGAL::right_turn(ccw[p], ccw[p+1], legs[l].second)) ? true : false;   // if one of the leg points ↙
                    ↳ outside, then set to yes.
                if(isOutside) break;
            }
            if(!isOutside) lists[l].push_back(n);   // both end points of leg are inside.
        }
    }

    vi pointers(M-1, 0);
    priority_queue<int> max_heap;
    priority_queue<pair<int, int>, std::vector<pair<int, int> >, greater<pair<int, int> > > min_heap;
    for(int l = 0; l < lists.size(); ++l) {
        max_heap.push(lists[l][0]);
        min_heap.push(make_pair(lists[l][0], l));
    }

    int min_interval = INT_MAX;
    while(true) {
        pair<int, int> min_pair = min_heap.top(); min_heap.pop();
        int min_value = min_pair.first;
        int min_list = min_pair.second;

        int max_value = max_heap.top();
        int min_new = abs(max_value - min_value);
        min_interval = min(min_new, min_interval);
        if(pointers[min_list] == lists[min_list].size()-1) break;

        pointers[min_list]++;
        int new_value = lists[min_list][pointers[min_list]];
        max_heap.push(new_value);
        min_heap.push(make_pair(new_value, min_list));
```

```
73      }
74
75      cout << min_interval+1 << "\n";
76  }
77
78  int main() {
79      ios_base::sync_with_stdio(false);
80      int TC; cin >> TC;
81      while(TC--) testcase();
82      return 0;
83  }
```

Maximize It!

```
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>
using namespace std;

#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;

void program_1(int a, int b) {
    Program qp (CGAL::SMALLER, true, 0, false, 0);      // use bounds instead of extra constraints.
    const int X = 0;
    const int Y = 1;

    // minimize -b*y + a*x^2
    qp.set_c(Y, -b);
    qp.set_d(X, X, a*2);

    // x + y <= 4
    qp.set_a(X, 0, 1);
    qp.set_a(Y, 0, 1);
    qp.set_b(0, 4);

    // 4x + 2y <= a*b
    qp.set_a(X, 1, 4);
    qp.set_a(Y, 1, 2);
    qp.set_b(1, a*b);

    // -x + y <= 1
    qp.set_a(X, 2, -1);
    qp.set_a(Y, 2, 1);
    qp.set_b(2, 1);

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

    if(s.is_optimal()) {
        int sign;
        (s.objective_value() <= 0) ? sign = -1 : sign = 1;
        cout << floor(to_double(sign*s.objective_value())) << "\n";     // std::ceil?, ceil_to_double fct?
    } else if(s.is_unbounded())
        cout << "unbounded\n";
    else if(s.is_infeasible())
        cout << "no\n";

}

void program_2(int a, int b) {
    Program qp (CGAL::SMALLER, false, 0, true, 0);
    const int X = 0;
    const int Y = 1;
    const int Z = 2;

    qp.set_l(Z, 0);
    qp.set_u(Z, false);

    // minimize a*x^2 + b*y + z^4
    qp.set_d(X, X, 2*a);
    qp.set_d(Z, Z, 2*1);        // by convention: we multiply value by 2.

    qp.set_c(Y, b);

```

```
74      qp.set_a(X, 0, 1);
75      qp.set_a(Y, 0, 1);
76      qp.set_b(0, -4);
77      qp.set_r(0, CGAL::LARGER);
78
79      qp.set_a(X, 1, 4);
80      qp.set_a(Y, 1, 2);
81      qp.set_a(Z, 1, 1);
82      qp.set_b(1, -1*a*b);
83      qp.set_r(1, CGAL::LARGER);
84
85      qp.set_a(X, 2, -1);
86      qp.set_a(Y, 2, 1);
87      qp.set_b(2, -1);
88      qp.set_r(2, CGAL::LARGER);
89
90      qp.set_a(Z, 3, 1);
91      qp.set_b(3, 0);
92      qp.set_r(3, CGAL::LARGER);
93
94      Solution s = CGAL::solve_quadratic_program(qp, ET());
95      assert(s.solves_quadratic_program(qp));
96
97      if(s.is_optimal()) {
98          double result = ceil(CGAL::to_double(s.objective_value()));
99          cout << result << "\n";
100     }
101     else if(s.is_unbounded())
102         cout << "unbounded\n";
103     else if(s.is_infeasible())
104         cout << "no\n";
105 }
106
107 int main() {
108     ios_base::sync_with_stdio(false);
109     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
110     int p, a, b;
111     while(true) {
112         cin >> p;
113         if(p == 0) return 0;
114         cin >> a >> b;
115         if(p == 1) program_1(a, b);
116         if(p == 2) program_2(a, b);
117     }
118 }
```

Collisions

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
#include <CGAL/Triangulation_vertex_base_with_info_2.h>
using namespace std;

typedef CGAL::Exact_predicates_inexact_constructions_kernel     K;
typedef CGAL::Delaunay_triangulation_2<K>                       D_Triangulation;
typedef D_Triangulation::Finite_edges_iterator                  FEI;
typedef set<D_Triangulation::Vertex_handle>                      vertex_set;

void testcase() {
    int n, d; cin >> n >> d;

    vector<K::Point_2> points;
    for(int i = 0; i < n; ++i) {
        int x, y; cin >> x >> y;
        points.push_back(K::Point_2(x, y));
    }

    D_Triangulation t;
    t.insert(points.begin(), points.end());
    vertex_set in_danger;
    for(FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
        D_Triangulation::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
        D_Triangulation::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);
        K::FT squared_d = CGAL::squared_distance(v1->point(), v2->point());
        double distance = CGAL::sqrt(squared_d);

        if(distance < d) {
           in_danger.insert(v1); in_danger.insert(v2);
        }
    }
    cout << in_danger.size() << "\n";

}

int main() {
    int TC; std::cin >> TC;
    while(TC--) testcase();
}
```

Diet

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
using namespace std;

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

typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

// N: nutrients, M: foods
void testcase(int N, int M) {
    Program lp(CGAL::SMALLER, true, 0, false, 0);

    for(int n = 0; n < N; ++n) {
        int min, max; cin >> min >> max;
        lp.set_b(n, min);
        lp.set_r(n, CGAL::LARGER);
        lp.set_b(N+n, max);
    }

    for(int m = 0; m < M; ++m) {
        int p; cin >> p;
        lp.set_c(m, p);

        for(int n = 0; n < N; ++n) {
            int amount; cin >> amount;
            lp.set_a(m, n, amount);
            lp.set_a(m, N+n, amount);
        }
    }

    Solution s = CGAL::solve_linear_program(lp, ET());
    assert (s.solves_linear_program(lp));

    if(s.is_infeasible())
        cout << "No such diet.\n";
    else
        cout << floor(to_double(s.objective_value())) << "\n";
}

int main() {
    while(true) {
        int N, M; cin >> N >> M;
        if(N == 0 && M == 0) return 0;
        testcase(N, M);
    }
}
```

Porfolios

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
using namespace std;

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

typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

// N: assets, M: portfolios
void testcase(int N, int M) {
    Program qp(CGAL::SMALLER, true, 0, false, 0);

    for(int n = 0; n < N; ++n) {
        int c, r; cin >> c >> r;
        qp.set_a(n, 0, c);
        qp.set_a(n, 1, r);
    }

    for(int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            int cij; cin >> cij;
            qp.set_d(i, j, 2*cij);
        }
    }

    for(int m = 0; m < M; ++m) {
        int C, R, V; cin >> C >> R >> V;
        qp.set_b(0, C);
        qp.set_b(1, R);
        qp.set_r(1, CGAL::LARGER);

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

        //cout << s;

        if(s.is_optimal() && (to_double(s.objective_value()) <= V)) {
            cout << "Yes.\n";
        } else {
            cout << "No.\n";
        }
    }
}

int main() {
    while(true) {
        int N, M; cin >> N >> M;
        if(N == 0 && M == 0) return 0;
        testcase(N, M);
    }
}
```

Inball

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
using namespace std;

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

typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

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

    while(n > 0) {
        int d; cin >> d;
        Program lp(CGAL::SMALLER, false, 0, false, 0);
        lp.set_c(d, -1);
        lp.set_l(d, true, 0);

        for(int i = 0; i < n; ++i) {
            int l2 = 0;
            for(int j = 0; j < d; ++j) {
                int a; cin >> a;
                lp.set_a(j, i, a);
                l2 += a*a;
            }
            l2 = sqrt(l2);
            lp.set_a(d, i, l2);

            int b; cin >> b;
            lp.set_b(i, b);
        }

        Solution s = CGAL::solve_linear_program(lp, ET());
        if(s.is_infeasible()) {
            cout << "none\n";
        } else if(s.is_unbounded()) {
            cout << "inf\n";
        } else {
            cout << floor(-CGAL::to_double(s.objective_value())) << "\n";
        }

        cin >> n;
    }
}
```

Monkey Island

```cpp
#include <vector>
#include <iostream>
#include <climits>
#include <boost/graph/strong_components.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
using namespace std;
using namespace boost;

typedef vector<int> vi;
typedef adjacency_list<vecS, vecS, directedS, no_property, no_property> Graph;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::edge_iterator EdgeIterator;

void testcase() {
    int N, M; cin >> N >> M;

    Graph g(N);
    for(int e = 0; e < M; ++e) {
        int v1, v2;
        cin >> v1 >> v2;
        add_edge(v1-1, v2-1, g);
    }

    vi costs(N);
    for(int n = 0; n < N; ++n) {
        int cost; cin >> cost;
        costs[n] = cost;
    }

    vector<int> scc(N);
    int nscc = strong_components(g, &scc[0]);

    vi incoming_comp(nscc, 0);
    EdgeIterator ebeg, eend;
    for(tie(ebeg, eend) = edges(g); ebeg != eend; ++ebeg) {
        int u = source(*ebeg, g);
        int v = target(*ebeg, g);
        if(scc[u] != scc[v]) incoming_comp[scc[v]] = 1;
    }

    int total = 0;
    for(int comp = 0; comp < nscc; ++comp) {
        if(incoming_comp[comp] == 1) continue;
        int min_cost = INT_MAX;
        for(int v = 0; v < N; ++v) {
            if(scc[v] == comp) min_cost = min(min_cost, costs[v]);
        }
        total += min_cost;
    }

    cout << total << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Placing Knights

```cpp
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/max_cardinality_matching.hpp>
using namespace std;
using namespace boost;

typedef vector<int> vi;
typedef vector<vi> vii;
typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;

int N;

int co_to_index(int i, int j) {
    return i*N + j;
}

void add_valid_edges(int i, int j, vii& holes, Graph& g) {
    int y = 1;
    for(int x = -2; x <= 2; x = x + 4) {
        if(i+y >= 0 && i+y < N && j+x >= 0 && j+x <= N && holes[i+y][j+x] == 1) {
            add_edge(co_to_index(i, j), co_to_index(i+y, j+x), g);
        }
    }
    y = 2;
    for(int x = -1; x <= 1; x = x + 2) {
        if(i+y >= 0 && i+y < N && j+x >= 0 && j+x <= N && holes[i+y][j+x] == 1) {
            add_edge(co_to_index(i, j), co_to_index(i+y, j+x), g);
        }
    }
}

void testcase() {
    cin >> N;
    Graph g(N*N);
    vii holes(N, vi(N));
    int sum_holes = 0;

    for(int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            int hole; cin >> hole;
            holes[i][j] = hole;
            if(holes[i][j] == 0) ++sum_holes;
        }
    }

    for(int i = 0; i < N-1; ++i) {
        for(int j = 0; j < N; ++j) {
            if(holes[i][j] == 1) add_valid_edges(i, j, holes, g);
        }
    }

    vector<Vertex> mateMap(num_vertices(g), 0);
    checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
    // mistake: forgot to substract the holes.
    cout << num_vertices(g)- sum_holes - matching_size(g, &mateMap[0]) << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Shopping Trip

```cpp
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.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 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;
typedef graph_traits<Graph>::edge_descriptor Edge;

void testcase() {
    int n, m, s; cin >> n >> m >> s;
    Graph g(n);
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap rev_edge = get(edge_reverse, g);
    ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);

    for(int store = 0; store < s; ++store){
        int store_vertex; cin >> store_vertex;
        Edge edge;
        tie(edge, tuples::ignore) = add_edge(store_vertex, n, g);
        Edge reverse_edge;
        tie(reverse_edge, tuples::ignore) = add_edge(n, store_vertex, g);
        capacity[edge] = 1;
        rev_edge[edge] = reverse_edge;
        capacity[reverse_edge] = 0;
        rev_edge[reverse_edge] = edge;
    }

    for(int e = 0; e < m; ++e) {
        int v1, v2; cin >> v1 >> v2;
        Edge edge;
        tie(edge, tuples::ignore) = add_edge(v1, v2, g);
        Edge reverse_edge;
        tie(reverse_edge, tuples::ignore) = add_edge(v2, v1, g);
        capacity[edge] = 1;
        rev_edge[edge] = reverse_edge;
        capacity[reverse_edge] = 0;
        rev_edge[reverse_edge] = edge;
        Edge edge2;
        tie(edge2, tuples::ignore) = add_edge(v2, v1, g);
        Edge reverse_edge2;
        tie(reverse_edge2, tuples::ignore) = add_edge(v1, v2, g);
        capacity[edge2] = 1;
        rev_edge[edge2] = reverse_edge2;
        capacity[reverse_edge2] = 0;
        rev_edge[reverse_edge2] = edge2;
    }

    long max_flow = push_relabel_max_flow(g, 0, n);
    if(max_flow == s) cout << "yes\n"; else cout << "no\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

TheeV

```cpp
#include <iostream>
#include <vector>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel K;
typedef CGAL::Min_circle_2_traits_2<K> MinCircleTraits;
typedef CGAL::Min_circle_2<MinCircleTraits> Min_circle;
typedef vector<pair<K::FT, K::Point_2> > dp;

bool pairCompare(const pair<K::FT, K::Point_2>& lhs, const pair<K::FT, K::Point_2>& rhs) {
    return lhs.first > rhs.first;
}

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

void testcase() {
    int N; cin >> N;
    dp cities;

    int x, y; cin >> x >> y;
    K::Point_2 capitol(x, y);
    cities.push_back(make_pair(0, capitol));

    for(int n = 1; n < N; ++n) {
        int x, y; cin >> x >> y;
        K::Point_2 p(x, y);
        K::FT dist = CGAL::squared_distance(capitol, p);
        cities.push_back(make_pair(dist, p));
    }
    sort(cities.begin(), cities.end(), pairCompare);

    int i = 0;
    K::FT r1 = cities[0].first, r2 = 0;
    K::FT t = r1;
    Min_circle mc;
    while(r1 > r2 && i < N-1) {
        r1 = cities[i+1].first;

        //cout << "insert in mincircle: " << cities[i].second << "\n";
        mc.insert(cities[i].second);
        MinCircleTraits::Circle c = mc.circle();
        r2 = c.squared_radius();
        //cout << "r1: " << r1 << "\n" << "r2: " << r2 << "\n";
        //cout << "diff: " << abs(r1 - r2) << " r1: " << r1 << " r2:" << r2 << "\n";
        ++i;
    }

    if(r1 == r2)
        t = r1;
    if(r2 > r1)
        t = min(r2, cities[i-1].first);

    cout << ceil_to_double(t) << "\n";
}

int main() {
    cin.sync_with_stdio(false);
    cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

Poker Chips

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <cmath>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vii;
typedef map<vector<int>, int> vector_int;

vi M;
int N;
vector_int dp_table;
vii chips;

int find_max(vi& state) {
    if(dp_table.count(state) == 1)
        return dp_table[state];

    for(int n = 1; n < pow(2.0, N); ++n) {
        vi new_state = state;
        int T = 0;
        int prev = -1;

        for(int k = 0; k < N; ++k) {
            if((n & (1 << k)) && (state[k] != 0)) {
                int color = chips[k][state[k]-1];
                if(prev == color || prev == -1) {
                    --new_state[k];
                    prev = color;
                    ++T;
                } else {
                    T = 0;  // !important to avoids wasted loops and computing invalid states.
                    break;
                }
            }
        }

        if(T != 0) { // if T=0, then invalid subset.
            int K = (T <= 1) ? 0 : pow(2.0, T-2);
            dp_table[state] = max(find_max(new_state) + K, dp_table[state]);
        }
    }

    return dp_table[state];
}

void testcase() {
    cin >> N;
    M = vi(N);
    for(int n = 0; n < N; ++n)
        cin >> M[n];

    chips = vii(N);
    for(int n = 0; n < N; ++n) {
        for(int m = 0; m < M[n]; ++m) {
            int col; cin >> col;
            chips[n].push_back(col);
        }
    }

    dp_table = vector_int();
    cout << find_max(M) << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Portfolio Revisited

```cpp
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
using namespace std;

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

typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

void testcase(int N, int M) {
    Program qp (CGAL::SMALLER, true, 0, false, 0);

    for(int n = 0; n < N; ++n) {
        int c, r; cin >> c >> r;
        qp.set_a(n, 0, c);
        qp.set_a(n, 1, r);
    }
    qp.set_r(1, CGAL::LARGER);

    for(int i = 0; i < N; ++i) {
        for(int j = 0; j < N; ++j) {
            int vij; cin >> vij;
            qp.set_d(i, j, 2*vij);
        }
    }

    for(int m = 0; m < M; ++m) {
        int C, V; cin >> C >> V;
        int R = 0;
        qp.set_b(0, C);
        qp.set_b(1, R);

        int lo = 0;
        int hi = 100;
        bool fixed = false;
        while(lo <= hi) {
            int mid = (fixed) ? (lo + (hi-lo+1)/2) : hi;

            qp.set_b(1, mid);
            Solution s = CGAL::solve_quadratic_program(qp, ET());
            assert(s.solves_quadratic_program(qp));

            if(s.is_optimal() && s.objective_value() <= V) {
                R = mid;
                if(!fixed) {
                    lo = hi+1;
                    hi = 2*hi;
                } else {
                    lo = mid+1;
                }
            } else {
                fixed = true;
                hi = mid-1;
            }
        }
        cout << R << "\n";
    }
}

int main() {
    while(true) {
        int N, M; cin >> N >> M;
        if(N == 0 && M == 0) return 0;
        testcase(N, M);
```

```
74          }
75      }
```

Stamp Exhibition

```cpp
#include <iostream>
#include <cassert>
#include <cmath>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
using namespace std;

#ifdef CGAL_USE_GMP
#include <CGAL/Gmpq.h>
typedef CGAL::Gmpq ET;
#else
#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;
#endif

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Quadratic_program<double> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

void testcase() {
    int L, S, W; cin >> L >> S >> W;

    vector<K::Point_2> lamps;
    for(int l = 0; l < L; ++l) {
        int x, y; cin >> x >> y;
        lamps.push_back(K::Point_2(x, y));
    }

    vector<pair<K::Point_2, double> > stamps;
    for(int s = 0; s < S; ++s) {
        int x, y; double m; cin >> x >> y >> m;
        stamps.push_back(make_pair(K::Point_2(x, y), m));
    }

    vector<K::Segment_2> walls;
    for(int w = 0; w < W; ++w) {
        int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
        walls.push_back(K::Segment_2(K::Point_2(x1, y1), K::Point_2(x2, y2)));
    }

    if(S == 0) { cout << "yes\n"; return; }
    if(L == 0) { cout << "no\n"; return; }

    Program lp (CGAL::SMALLER, true, 1, true, pow(2.0, 12));
    for(int l = 0; l < L; ++l) {
        for(int s = 0; s < S; ++s) {
            bool intersect = false;
            for(int w = 0; w < W; ++w) {
                K::Segment_2 stamp_lamp(stamps[s].first, lamps[l]);
                if(CGAL::do_intersect(stamp_lamp, walls[w])) {
                    intersect = true;
                    break;
                }
            }

            double param = 0;
            if(!intersect)
                param = 1.0/CGAL::squared_distance(stamps[s].first, lamps[l]);
            lp.set_a(l, s, param);
            lp.set_a(l, S+s, param);
            lp.set_b(s, stamps[s].second);
            lp.set_b(S+s, 1.0);
            lp.set_r(S+s, CGAL::LARGER);
        }
    }

    Solution s = CGAL::solve_linear_program(lp, ET());
    assert (s.solves_linear_program(lp));
    (!s.is_infeasible()) ? cout << "yes\n" : cout << "no\n";
}

```

```cpp
int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Tetris

```cpp
#include <iostream>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>
#include <boost/tuple/tuple.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 property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
typedef graph_traits<Graph>::edge_descriptor Edge;

void add_edge(int from, int to, int cap, Graph& g) {
    //cout << "adding edge: " << from << " " << to << " " << cap << "\n";
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap reverse = get(edge_reverse, g);

    Edge there, back;
    tie(there, tuples::ignore) = add_edge(from, to, g);
    tie(back, tuples::ignore) = add_edge(to, from, g);
    capacity[there] = cap;
    capacity[back] = 0;
    reverse[there] = back;
    reverse[back] = there;
}

void testcase() {
    int W, N; cin >> W >> N;

    int source = 0;
    int sink = W;
    Graph g(2*W);

    for(int v = 1; v < W; ++v) {
        add_edge(v, W+v, 1, g);
    }

    for(int n = 0; n < N; ++n) {
        int v1, v2; cin >> v1 >> v2;
        int from = (min(v1, v2) == 0) ? 0 : min(v1, v2) + W;
        int to = max(v1, v2);
        add_edge(from, to, 1, g);
    }

    int maxflow = push_relabel_max_flow(g, source, sink);
    cout << maxflow << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

Beach Bar

```cpp
#include <vector>
#include <iostream>
#include <climits>
#include <algorithm>
using namespace std;

typedef vector<int> vi;
const int normalize = 1000000;

void testcase() {
    int N; cin >> N;
    vi points;
    for(int n = 0; n < N; ++n) {
        int x; cin >> x;
        points.push_back(x + normalize);
    }
    sort(points.begin(), points.end());

    int g_counter = INT_MIN;
    int g_length = INT_MIN;
    vi solution;
    for(int n = 0; n < N; ++n) {
        int start_interval = points[n];
        int end_interval = start_interval + 200;
        int k = n;
        int counter = 0;
        while(points[k] <= end_interval && k < N) {
            ++counter;
            ++k;
        }
        int length = (points[k-1] - start_interval);

        if(counter > g_counter || (counter == g_counter && length < g_length)) {
            g_counter = counter;
            g_length = length;
            solution.clear();
        }

        if(g_counter == counter && g_length == length) {
            int output =  start_interval + length/2 - normalize;
            solution.push_back(output);
            if(length % 2 != 0) {
                solution.push_back(output+1);
            }
        }
    }

    g_length = (g_length % 2 == 0) ? g_length/2 : g_length/2+1;
    cout << g_counter << "␣" << g_length <<"\n";
    for(int s = 0; s < solution.size(); ++s) {
        cout << solution[s];
        if(s != solution.size() - 1) cout << "␣";
    }
    cout << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Cover

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
using namespace std;

typedef CGAL::Exact_predicates_exact_constructions_kernel           K;
typedef CGAL::Delaunay_triangulation_2<K>                           Delaunay;
typedef Delaunay::Finite_faces_iterator                             FFI;
typedef Delaunay::Finite_edges_iterator                             FEI;

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

template<typename T>
K::FT check_intersection(const T* obj, const K::Point_2 p1, const vector<K::Segment_2>& rectangle) {
    for (int i = 0; i < 4; ++i) {
        if(!do_intersect(rectangle[i], *obj)) continue;
        CGAL::Object o = intersection(rectangle[i], *obj);
        const K::Point_2* p2 = CGAL::object_cast<K::Point_2>(&o);
        K::FT sqrd = CGAL::squared_distance(p1, *p2);
        return sqrd;
    }
    return 0;
}

void testcase(int N) {
    vector<K::Point_2> points;
    vector<K::Segment_2> rectangle;

    double x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    K::Point_2 sw (x1, y1);
    K::Point_2 nw(x1, y2);
    K::Point_2 se(x2, y1);
    K::Point_2 ne(x2, y2);
    rectangle.push_back(K::Segment_2(sw, nw));
    rectangle.push_back(K::Segment_2(se, ne));
    rectangle.push_back(K::Segment_2(sw, se));
    rectangle.push_back(K::Segment_2(nw, ne));

    for(int n = 0; n < N; ++n) {
        double x, y; cin >> x >> y;
        points.push_back(K::Point_2(x, y));
    }

    // O(n log n)
    Delaunay t;
    t.insert(points.begin(), points.end());
    K::FT min_rad;

    // check corners
    min_rad = CGAL::squared_distance(sw, t.nearest_vertex(sw)->point());
    min_rad = max(min_rad, CGAL::squared_distance(se, t.nearest_vertex(se)->point()));
    min_rad = max(min_rad, CGAL::squared_distance(nw, t.nearest_vertex(nw)->point()));
    min_rad = max(min_rad, CGAL::squared_distance(ne, t.nearest_vertex(ne)->point()));

    // iterate over all faces to find largest circle - O(N)
    for(FFI f = t.finite_faces_begin(); f != t.finite_faces_end(); ++f) {
        K::Point_2 cc = t.circumcenter(f);
        if(cc.x() >= x1 && cc.x() <= x2 && cc.y() >= y1 && cc.y() <= y2) {
            K::Point_2 point = f->vertex(1)->point();
            K::FT dist = CGAL::squared_distance(point, cc);
            min_rad = max(min_rad, dist);
        }
    }

    // check for intersection with rectangle boundary - O(n*4)
```

```
74      for(FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
75          CGAL::Object o = t.dual(e);
76          if(const K::Ray_2* r = CGAL::object_cast<K::Ray_2>(&o))
77              min_rad = max(min_rad, check_intersection(r, t.segment(e).source(), rectangle));
78          else if(const K::Segment_2* s = CGAL::object_cast<K::Segment_2>(&o))
79              min_rad = max(min_rad, check_intersection(s, t.segment(e).source(), rectangle));
80      }
81
82      cout << ceil(CGAL::sqrt(to_double(min_rad))) << "\n";
83  }
84
85  int main() {
86      cin.sync_with_stdio(false);
87      cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
88      while(true) {
89          int N; cin >> N;
90          if(N == 0) return 0;
91          testcase(N);
92      }
93  }
```

Divisor Distance

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

int ancestor(int v) {
    for(int k = 2; k <= ceil(sqrt(v)); ++k) {
        if(v % k == 0) return (v/k);
    }
    return 1;
}

void testcase() {
    int N, C; cin >> N >> C;
    for(int c = 0; c < C; ++c) {
        int v1, v2; cin >> v1 >> v2;
        int counter = 0;
        while(v1 != v2) {
            ++counter;
            if(v1 < v2) {
                v2 = ancestor(v2);
            } else {
                v1 = ancestor(v1);
            }
        }
        cout << counter << "\n";
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Tiles

```cpp
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/max_cardinality_matching.hpp>
using namespace std;
using namespace boost;

typedef vector<int> vi;
typedef vector<vi> vii;
typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;
typedef graph_traits<Graph>::vertex_descriptor Vertex;

void testcase() {
    int W, H; cin >> W >> H;

    vii matrix(H);
    int blocked = 0;
    int vcounter = 0;
    for(int h = 0; h < H; ++h) {
        for(int w = 0; w < W; ++w) {
            char input; cin >> input;
            blocked += (input == 'x');
            matrix[h].push_back((input == '.') ? vcounter++ : -1);
        }
    }

    int V = (W*H - blocked);
    if(V % 2 == 1) {
        cout << "no\n";
        return;
    }

    Graph g(V);
    for(int h = 0; h < H; ++h) {
        for(int w = 0; w < W; ++w) {
            if(matrix[h][w] == -1) continue;
            if(w+1 < W && matrix[h][w+1] != -1) add_edge(matrix[h][w], matrix[h][w+1], g);
            if(h+1 < H && matrix[h+1][w] != -1) add_edge(matrix[h][w], matrix[h+1][w], g);
        }
    }

    vector<Vertex> mateMap(num_vertices(g), 0);
    checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
    int matching = matching_size(g, &mateMap[0]);

    if(matching * 2 == V) cout << "yes\n";
    else cout << "no\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Deleted Entries Stike Back

Missing.

Light The Stage

Missing.

Radiation

Missing.

Sweepers

```
#include <iostream>
#include <vector>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>
#include <boost/graph/strong_components.hpp>
using namespace std;
using namespace boost;

typedef vector<int> vi;
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 property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::vertex_descriptor Vertex;

int N, M, S;

void add_edge(int from, int to, int cap, Graph& g) {
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap reverse = get(edge_reverse, g);

    Edge there, back;
    tie(there, tuples::ignore) = add_edge(from, to, g);
    tie(back, tuples::ignore) = add_edge(to, from, g);
    capacity[there] = cap;
    capacity[back] = 0;
    reverse[there] = back;
    reverse[back] = there;
}

void testcase() {
    cin >> N >> M >> S;
    int source = N;
    int sink = N+1;
    Graph g(N+2);
    vi starts(N, 0), exits(N, 0);

    for(int s = 0; s < S; ++s) {
        int room; cin >> room;
        ++starts[room];
    }

    for(int s = 0; s < S; ++s) {
        int room; cin >> room;
        ++exits[room];
    }

    for(int m = 0; m < M; ++m) {
        int v1, v2; cin >> v1 >> v2;
        add_edge(v1, v2, 1, g);
        add_edge(v2, v1, 1, g);
    }

    for(int n = 0; n < N; ++n) {
        if(starts[n] > 0) add_edge(source, n, starts[n], g);
        if(exits[n] > 0) add_edge(n, sink, exits[n], g);
    }

    bool isEulerian = true;
    bool isConnected = false;
    graph_traits<Graph>::vertex_iterator viter, vend;
    for (tie(viter, vend) = vertices(g); viter != vend; ++viter) {
        if(*viter == source || *viter == sink) continue;
        int count = out_degree(*viter, g);
        if(starts[*viter] > 0) ++count;
        if(exits[*viter] > 0) ++count;
        count = count/2;
        if(count % 2 == 1) {
```

```
            isEulerian = false;
            break;
        }
    }

    if(!isEulerian) {
        cout << "no\n";
        return;
    }

    int maxflow = push_relabel_max_flow(g, source, sink);
    if(maxflow != S)
        cout << "no\n";
    else
        cout << "yes\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
}
```

The Bracelet

```cpp
#include <iostream>
#include <stack>
#include <set>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/connected_components.hpp>
using namespace std;
using namespace boost;

typedef vector<pair<int, int> > vpi;
typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
typedef graph_traits<Graph>::vertex_iterator VI;
typedef graph_traits<Graph>::out_edge_iterator EI;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef property_map<Graph, edge_weight_t>::type WeightMap;

void printEulerGraph(int v, Graph& g) {
    WeightMap weight = get(edge_weight, g);
    stack<int> fifo;
    fifo.push(v);
    vector<int> sol;
    while(!fifo.empty()) {
        int v = fifo.top();
        EI ebegin, eend;
        bool hasFreeEdge = false;
        for(tie(ebegin, eend) = out_edges(v, g); ebegin != eend; ++ebegin) {
            if(weight[*ebegin] == 0) {
                hasFreeEdge = true;
                weight[*ebegin] = 1;
                fifo.push(boost::target(*ebegin, g));
                break;
            }
        }
        if(!hasFreeEdge) {
            sol.push_back(v);
            fifo.pop();
        }
    }
    for(int s = 0; s < sol.size()-1; ++s) {
        cout << sol[s] << " " << sol[s+1] << "\n";
    }
    cout << "\n";
}

void testcase(int TC) {
    cout << "Case #" << ++TC << "\n";
    int N; cin >> N;

    Graph g(50);
    WeightMap weight = get(edge_weight, g);
    set<int> colors;

    for(int n = 0; n < N; ++n) {
        int v1, v2; cin >> v1 >> v2;
        colors.insert(v1); colors.insert(v2);
        Edge e;
        tie(e, tuples::ignore) = add_edge(v1, v2, g);
        weight[e] = 0;
    }

    vector<int> component(num_vertices(g));
    int num = connected_components(g, &component[0]) - (51 - colors.size());
    int start = -1;
    VI vbegin, vend;
    for(tie(vbegin, vend) = vertices(g); vbegin != vend; ++vbegin) {
        int deg = out_degree(*vbegin, g);
        if(deg % 2 == 1 || num > 1) {
            cout << "some beads may be lost\n\n";
            return;
        }
        if(deg > 0) start = *vbegin;
    }

```

```
74      printEulerGraph(start, g);
75  }
76
77  int main() {
78      int TC; cin >> TC;
79      for(int t = 0; t < TC; ++t) testcase(t);
80  }
```

Knights

```
#include <iostream>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/adjacency_list.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 property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
typedef graph_traits<Graph>::edge_descriptor Edge;

int M;
int N;
int K;

int index(int x, int y) {
    return y*M + x;
}

void add_edges(int from, int to, Graph& g) {
    EdgeCapacityMap capacity = get(edge_capacity, g);
    ReverseEdgeMap reverse = get(edge_reverse, g);

    Edge there, back;
    tie(there, tuples::ignore) = add_edge(from, to, g);
    tie(back, tuples::ignore) = add_edge(to, from, g);
    capacity[there] = 1;
    capacity[back] = 0;
    reverse[there] = back;
    reverse[back] = there;
}

void testcase() {
    cin >> M >> N >> K;    // M: cols, N: rows, K: #knights

    int graph_size = 2*(M*N)+2;     // M*N for each coordinate, 2*(M*N) because we need vertex-disjoint paths only.
    Graph g(graph_size);
    int source = graph_size-2;
    int sink = graph_size-1;
    for(int y = 0; y < N; ++y) {
        for(int x = 0; x < M; ++x) {
            int v_in = index(x, y);
            int v_out = index(x, y) + M*N;

            add_edges(v_in, v_out, g);

            if(x+1 < M) {
                add_edges(v_out, index(x+1, y), g);
                add_edges(index(x+1, y)+(M*N), v_in, g);
            }
            if(y+1 < N) {
                add_edges(v_out, index(x, y+1), g);
                add_edges(index(x, y+1)+(M*N), v_in, g);
            }
            if(x-1 < 0 || x+1 >= M || y-1 < 0 || y+1 >= N) {
                add_edges(v_out, sink, g);
            }
        }
    }

    for(int k = 0; k < K; ++k) {
        int x, y; cin >> x >> y;
        add_edges(source, index(x, y), g);
    }

    int maxflow = push_relabel_max_flow(g, source, sink);
    cout << maxflow << "\n";
}
```

```
74
75  int main() {
76      int TC; cin >> TC;
77      while(TC--) testcase();
78  }
```

Next Path

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <boost/graph/adjacency_list.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
using namespace std;
using namespace boost;

const int MAX_LENGTH = 100000000;   // do not pick INT_MAX otherwise overflow resulting in -INT_MAX confusing min.

typedef vector<int> vi;
typedef adjacency_list<vecS, vecS, directedS, no_property, property<edge_weight_t, int> > Graph;
typedef graph_traits<Graph>::edge_descriptor Edge;
typedef graph_traits<Graph>::vertex_descriptor Vertex;
typedef property_map<Graph, edge_weight_t>::type WeightMap;
typedef graph_traits<Graph>::out_edge_iterator OutEdgeIterator;

int BFS(int start, int end, Graph& g) {
    if(start == end) return 0;
    vi distances(num_vertices(g), -1);
    std::queue<int> fifo;
    fifo.push(start);
    distances[start] = 0;
    while(!fifo.empty()) {
        int v = fifo.front(); fifo.pop();
        OutEdgeIterator ebegin, eend;
        for(tie(ebegin, eend) = out_edges(v, g); ebegin != eend; ++ebegin) {
            int u = target(*ebegin, g);
            if(distances[u] == -1) {
                distances[u] = distances[v] + 1;
                fifo.push(u);
                if(u == end) return distances[u];
            }
        }
    }
    return MAX_LENGTH;
}

void testcase() {
    int N, M, s, t; cin >> N >> M >> s >> t;
    --t; --s;

    Graph g(N);
    WeightMap weights = get(edge_weight, g);

    for(int m = 0; m < M; ++m) {
        int v1, v2; cin >> v1 >> v2;
        Edge edge;
        tie(edge, tuples::ignore) = add_edge(v1-1, v2-1, g);
        weights[edge] = 1;
    }

    vi d(N);
    vector<Vertex> p(N);
    dijkstra_shortest_paths(g, s, predecessor_map(&p[0]).distance_map(&d[0]));

    if(d[t] == INT_MAX) { cout << "no\n"; return; }     // there is no path from s to t.

    int sp = MAX_LENGTH;
    int b = t;
    int prev = t;
    while(true) {
        OutEdgeIterator ebegin, eend;
        for(tie(ebegin, eend) = out_edges(b, g); ebegin != eend; ++ebegin) {
            if(target(*ebegin, g) == prev && prev != s && b != t) continue; // do not pick the edge in P, start end ↙
                ↳ end path are special states.
            sp = min(sp, d[source(*ebegin, g)] + 1 + BFS(target(*ebegin, g), t, g));
        }
        if(b == s || sp == d[t]) break;
        prev = b;
        b = p[b];
    }
```

```
73      (sp == MAX_LENGTH) ? cout << "no\n" : cout << sp << "\n";
74  }
75
76  int main() {
77      ios_base::sync_with_stdio(false);
78      int TC; cin >> TC;
79      while(TC--) testcase();
80  }
```

Odd Route

```cpp
#include <iostream>
#include <vector>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
#include <boost/tuple/tuple.hpp>
#include <climits>
using namespace std;
using namespace boost;

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

void add_edges(Graph& g, int u, int v, int w) {
    int uee = u*4;      int vee = v*4;
    int ueo = uee+1;     int veo = vee+1;
    int uoe = uee+2;     int voe = vee+2;
    int uoo = uee+3;     int voo = vee+3;

    EdgeWeightMap weights = get(edge_weight, g);

    Edge edge;
    if(w % 2 == 0) {
        tie(edge, tuples::ignore) = add_edge(uee, voe, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(ueo, voo, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(uoe, vee, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(uoo, veo, g); weights[edge] = w;
    } else {
        tie(edge, tuples::ignore) = add_edge(uee, voo, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(ueo, voe, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(uoe, veo, g); weights[edge] = w;
        tie(edge, tuples::ignore) = add_edge(uoo, vee, g); weights[edge] = w;
    }
}

void testcase() {
    int N, M, s, t; cin >> N >> M >> s >> t;
    Graph g(N*4);

    for(int m = 0; m < M; ++m) {
        int u, v, w; cin >> u >> v >> w;
        add_edges(g, u, v, w);
    }

    vector<int> d(num_vertices(g), -1);
    dijkstra_shortest_paths(g, s*4, distance_map(&d[0]));
    (d[4*t+3] < INT_MAX) ? cout << d[4*t+3] : cout << "no";
    cout << "\n";
}

int main() {
    int TC; cin >> TC;
    while(TC--) testcase();
    return 0;
}
```

Radiation 2

Missing.