

## Problems

Checking Change . . . . .	2	Germes . . . . .	44
Dominoes . . . . .	4	Graypes . . . . .	46
Shelves . . . . .	6	H1N1 . . . . .	47
Even Pairs . . . . .	7	HikingMaps . . . . .	49
Aliens . . . . .	8	Maximize It! . . . . .	51
Boats . . . . .	9	Collisions . . . . .	53
False Coin . . . . .	11	Diet . . . . .	54
Formulas . . . . .	14	Porfolios . . . . .	55
Race Tracks . . . . .	16	Inball . . . . .	56
Burning Coins . . . . .	19	Monkey Island . . . . .	57
Jump . . . . .	20	Placing Knights . . . . .	58
Light Pattern . . . . .	21	Shopping Trip . . . . .	59
Longest Path . . . . .	22	TheeV . . . . .	60
Ants . . . . .	24	Poker Chips . . . . .	61
Bridges . . . . .	25	Portfolio Revisited . . . . .	62
Build The Graph . . . . .	27	Stamp Exhibition . . . . .	64
Deleted Entries . . . . .	29	Tetris . . . . .	66
Shy Programmers . . . . .	30	Beach Bar . . . . .	67
Algocoon Group . . . . .	31	Cover . . . . .	68
Buddies . . . . .	32	Divisor Distance . . . . .	70
Satellites . . . . .	34	Tiles . . . . .	71
Kingdom Defence . . . . .	36	Deleted Entries Stike Back . . . . .	72
Coin Tossing . . . . .	37	Light The Stage . . . . .	73
Antenna . . . . .	38	Radiation . . . . .	74
Almost Antenna . . . . .	39	Sweepers . . . . .	75
Hit . . . . .	40	The Bracelet . . . . .	77
First Hit . . . . .	41	Knights . . . . .	79
Search Snippets . . . . .	42	Next Path . . . . .	81
Bistro . . . . .	43	Odd Route . . . . .	83
		Radiation 2 . . . . .	84

# Checking Change

```
1  #include <vector>
2  #include <iostream>
3  #include <algorithm>
4  #include <string>
5  #include <sstream>
6  using namespace std;
7
8  vector<string> answers;
9
10 int main(int argc, char const *argv[])
11 {
12
13     int currencies;
14     cin >> currencies;
15
16     for (int currency = 0; currency < currencies; currency++)
17     {
18
19         int coins_count;
20         int testcases;
21
22         cin >> coins_count >> testcases;
23
24         vector<int> coins;
25         for (int coins_it = 0; coins_it < coins_count; coins_it++)
26         {
27             int coin;
28             cin >> coin;
29             coins.push_back(coin);
30         }
31
32         vector<int> tests;
33         for (int testcase = 0; testcase < testcases; testcase++)
34         {
35             int test;
36             cin >> test;
37             tests.push_back(test);
38         }
39
40         // find maximum of tests
41         vector<int>::iterator max_test_it = max_element(tests.begin(), tests.end());
42         int max_test = *max_test_it;
43         int N = max_test + 1;
44
45         vector<int>::iterator max_coin_it = max_element(coins.begin(), coins.end());
46         int max_coin = *max_coin_it;
47
48         vector<int>::iterator min_coin_it = min_element(coins.begin(), coins.end());
49         int min_coin = *min_coin_it;
50
51         // instantiate array with size max(tests)
52         int arraysize = 2;
53         vector<int> counts(arraysize);
54
55         // fill indices we already know -> coins, set to zero where index smaller than index of smallest coin.
56         for (int i = 0; i < min_coin; i++)
57         {
58             if (min_coin >= arraysize)
59             {
60                 arraysize += min_coin + 10;
61                 counts.resize(arraysize);
62                 //cout << "vector size now " << arraysize;
63             }
64             counts[i] = 0;
65         }
66
67         for (vector<int>::iterator coins_it = coins.begin(); coins_it != coins.end(); coins_it++)
68         {
69             if (*coins_it <= max_coin)
70             {
71                 if (*coins_it >= arraysize)
72                 {
73                     arraysize += *coins_it + 1;
```

```

74         counts.resize(arraysize);
75         //cout << "vector size now " << arraysize;
76     }
77     counts[*coins_it] = 1;
78 }
79 }
80
81 // iterate over counts, combine all minimums.
82 for (int n = min_coin + 1; n < N; n++)
83 {
84     if (arraysize <= n)
85     {
86         arraysize += 1;
87         counts.resize(arraysize);
88         //cout << "vector size now " << arraysize;
89     }
90
91     signed int min = -1;
92     for(int backward = n-1; backward >= min_coin; backward--) {
93
94         if (counts[n] == 1)
95         {
96             min = 1;
97         } else {
98             if(counts[backward] != 0 && counts[n-backward] != 0) {
99                 int new_min = counts[backward] + counts[n-backward];
100                 //cout << n << ": counts[backward]: " << counts[backward] << " counts[n-backward]: " << ↵
101                     ↵ counts[n-backward] << "new_min: " << new_min << "\n";
102                 if (min > new_min || min == -1)
103                 {
104                     min = new_min;
105                 }
106             }
107         }
108     }
109     if (min == -1)
110     {
111         min = 0;
112     }
113     counts[n] = min;
114 }
115
116 /*int i = 0;
117 for (vector<int>::iterator elements = counts.begin(); elements != counts.end(); elements++)
118 {
119     cout << i++ << ": " << *elements << " \n";
120 }*/
121
122 for (vector<int>::iterator test = tests.begin(); test != tests.end(); test++)
123 {
124     int answer = counts[*test];
125
126     stringstream ss;
127     if (answer == 0)
128     {
129         ss << "not_possible";
130     } else {
131         ss << answer;
132     }
133
134     answers.push_back(ss.str());
135 }
136
137 }
138
139 for (vector<string>::iterator answer = answers.begin(); answer != answers.end(); answer++)
140     cout << *answer << "\n";
141
142 return 0;
143 }

```

# Dominoes

```
1  /*
2  * Benjamin Grhbiel
3  * Domino
4  */
5
6  #include <iostream>
7  #include <vector>
8  #include <map>
9  using namespace std;
10
11 int main (int argc, const char *argv[])
12 {
13
14     ios_base::sync_with_stdio(false);
15
16     int testcases;
17     cin >> testcases;
18
19     map< int, vector<int> > index;
20
21     for (int testcase = 0; testcase < testcases; testcase++) {
22
23         long int dominoes;
24         cin >> dominoes;
25
26         for (int dominoPos = 1; dominoPos <= dominoes; dominoPos++) {
27             int height;
28             cin >> height;
29             index[testcase].push_back(height);
30         }
31     }
32
33     for (map<int, vector<int> >::iterator it = index.begin(); it != index.end(); it++) {
34         //cout << "Testcase: " << it->first << " Tiles: " << it->second.size() << "\n";
35
36         vector<int> tiles = it->second;
37
38         if (tiles.size() == 0) {
39             cout << 0;
40         }
41         else
42         {
43             int intervalRight = 0;
44             int iteration = 0;
45             int counter = 0;
46
47             for (vector<int>::iterator tile_it = tiles.begin(); tile_it != tiles.end(); tile_it++) {
48
49                 if (iteration > intervalRight) {
50                     //cout << "Break; iteration > intervalRight \n";
51                     break;
52                 }
53
54                 int h = *tile_it;
55                 int newIntervalRight = h + iteration - 1;
56
57                 if(newIntervalRight > intervalRight) {
58                     intervalRight = newIntervalRight;
59                 }
60
61                 iteration++;
62                 //cout << "intervalRight: " << intervalRight << " iteration: " << iteration << "\n";
63                 counter++;
64             }
65
66             cout << counter << "\n";
67         }
68     }
69 }
70
71 return 0;
72
73
```



# Shelves

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(void) {
6      // speeds up read and write
7      ios_base::sync_with_stdio(false);
8
9      // number of testcases we need to run
10     int nrCases;
11     cin >> nrCases;
12
13     for(int i = 0; i < nrCases; i++) {
14         // read the input for the test case
15         int l, m, n;
16         cin >> l >> m >> n;
17
18         // number of the two shelves and remaining length
19         int cm = 0;
20         int cn = 0;
21         int r = l;
22
23         for(int tmpCn = l/n; tmpCn >= 0 && r != 0; tmpCn--) {
24             // calculate the number of the small shelves
25             int tmpCm = (l - tmpCn * n) / m;
26             if(tmpCm >= n) {
27                 break;
28             }
29
30             // calculate the new remaining space and use it when smaller
31             int tmpR = l - tmpCn * n - tmpCm * m;
32             if(tmpR < r) {
33                 cn = tmpCn;
34                 cm = tmpCm;
35                 r = tmpR;
36             }
37         }
38
39         // output the result
40         cout << cm << " " << cn << " " << r << '\n';
41     }
42
43     return 0;
44 }
```

# Even Pairs

Even Pairs missing

# Aliens

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <climits>
5  using namespace std;
6
7  typedef vector<pair<int, int> > vii;    // sorted by left, right.
8  bool sortDescAsc(const pair<int, int>& lhs, const pair<int, int>& rhs) {
9      if(lhs.first == rhs.first)
10         return (lhs.second > rhs.second);
11     else
12         return lhs.first < rhs.first;
13 }
14
15 void testcase() {
16     int n, m;
17     cin >> n >> m;
18     vii intervals;
19     int superior = n;
20     for(int i = 0; i < n; ++i) {
21         int pi, qi;
22         cin >> pi >> qi;
23         if(pi == 0 && qi == 0) {
24             --superior;
25             continue;
26         }
27         pair<int, int> entry = make_pair(pi, qi);
28         intervals.push_back(entry);
29     }
30
31     sort(intervals.begin(), intervals.end(), sortDescAsc);
32
33     int left = 0;
34     int right = 0;
35     for(int i = 0; i < intervals.size(); ++i) {
36         if(i+1 < intervals.size() && intervals[i+1].first == intervals[i].first && intervals[i+1].second ==
37             ↵ intervals[i].second)
38             --superior;
39         else if(left == intervals[i].first && right == intervals[i].second)
40             --superior;
41         else if(right >= intervals[i].second)
42             --superior;
43
44         if(right < intervals[i].second) {
45             left = intervals[i].first;
46             if(right != 0 && left-right > 1) {
47                 cout << "0\n";
48                 return;
49             }
50             right = intervals[i].second;
51         }
52     }
53
54     cout << superior << "\n";
55 }
56
57 int main() {
58     int TC;
59     cin >> TC;
60     while(TC--) testcase();
61 }
```



# Boats

```
1  #include <vector>
2  #include <iostream>
3  #include <algorithm>
4  using namespace std;
5
6  struct Boat {
7      int ring;
8      int length;
9      bool taken;
10
11 };
12 inline bool operator<( const Boat& lhs, const Boat& rhs ) {
13     return lhs.ring < rhs.ring;
14 }
15 inline bool operator<( int lhs, const Boat& rhs ) {
16     return lhs <= rhs.ring;
17 }
18 inline bool operator<(const Boat& lhs, const int &val) {
19     return (lhs.ring < val);
20 }
21
22 void testcase() {
23     int boats; cin >> boats;
24     vector<Boat> boat_list;
25
26     for (int i = 0; i < boats; ++i)
27     {
28         int length, ring; cin >> length >> ring;
29         Boat boat;
30         boat.length = length;
31         boat.ring = ring;
32         boat.taken = false;
33         boat_list.push_back(boat);
34     }
35
36     std::sort(boat_list.begin(), boat_list.end());
37
38     int counter = 1;
39     int rightmost = boat_list[0].ring;
40     boat_list[0].taken = true;
41
42     // Problem 1: rightmost < boat_list.back().ring ... meaning, we stopped too early, neglecting the last boat.
43     // Problem 2: Endless loop in the scenario of just one boat... as rightmost = boat_list.back().ring.
44     while((rightmost <= boat_list.back().ring) && (boat_list.size() != 1)) {
45
46         vector<Boat>::iterator up = lower_bound(boat_list.begin(), boat_list.end(), rightmost);
47         int index = (up - boat_list.begin());
48         int next = index;
49         //cerr << "next: " << next << "\n";
50
51         // check if already taken, if yes, move pointer to the right.
52         if(boat_list[next].taken == true) next++;
53
54         int local_rightmost;
55         int min_rightmost = -1;
56         int boat_index;
57         do {
58             int ring = boat_list[next].ring;
59             int left = ring - rightmost;
60             int right = boat_list[next].length - left;
61
62             if(right < 0) local_rightmost = ring;
63             else local_rightmost = ring + right;
64
65             //cerr << "local_rightmost: " << local_rightmost << " min_rightmost: " << min_rightmost << "\n";
66             if((local_rightmost < min_rightmost) || (min_rightmost == -1)) {
67                 min_rightmost = local_rightmost;
68                 boat_index = next;
69                 //cerr << "local minimum set: " << local_rightmost << " boat_index: " << boat_index << "\n";
70             }
71             next++;
72         }
73         // 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

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int solve(int numberOfCoins, vector< pair<char, vector<int> > > equations);
6
7  vector<int> answers;
8
9  int main(int argc, char const *argv[])
10 {
11     int datasets;
12     cin >> datasets;
13
14     for (int dataset = 0; dataset < datasets; dataset++) {
15         //cout << "data set: " << dataset << "\n";
16
17         int numberOfCoins, numberOfWeighings;
18         cin >> numberOfCoins >> numberOfWeighings;
19
20         vector< pair<char, vector<int> > > equations;
21         equations.clear();
22         for (int i = 0; i < numberOfWeighings; i++) {
23             //cout << "reading weighing: " << i << " \n";
24             int coinsInPan;
25             cin >> coinsInPan;
26
27             vector<int> coins;
28             coins.clear();
29             for (int j = 0; j < (coinsInPan*2); j++)
30             {
31                 int coin;
32                 cin >> coin;
33                 coins.push_back(coin);
34                 //cout << "reading coin: " << j << "\n";
35             }
36
37             char operatorSymbol;
38             cin >> operatorSymbol;
39
40             equations.push_back(make_pair(operatorSymbol, coins));
41         }
42
43         int result = solve(numberOfCoins, equations);
44         if (result != 0)
45         {
46             answers.push_back(result);
47         } else {
48             answers.push_back(result);
49         }
50     }
51 }
52
53
54 for (vector<int>::iterator answer = answers.begin(); answer != answers.end(); answer++) {
55     cout << *answer << "\n";
56 }
57
58 return 0;
59 }
60
61 int solve(int numberOfCoins, vector< pair<char, vector<int> > > equations) {
62     vector<int> falseCoins;
63
64     for(int n = 1; n <= numberOfCoins; n++) {
65         int coin_id = n;
66
67         bool holding = true;
68         //cout << "coin_id: " << coin_id << "\n";
69
70         vector<int> lightWeightedCoins (numberOfCoins+1, 1);
71         lightWeightedCoins.at(coin_id) = 0;
72         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: " << ↵
103             ↵ rightSum[0] << "\n";
104         //cout << "coin_id: " << coin_id << " leftSum heavy: " << leftSum[1] << " rightSum heavy: " << ↵
105             ↵ rightSum[1] << "\n";
106
107         char symbol = eq_it->first;
108         if (symbol == '<')
109         {
110             bool verdict_light = leftSum[0] < rightSum[0]; // assuming false coin is lighter than others
111             bool verdict_heavy = leftSum[1] < rightSum[1]; // assuming false coin is heavier than others
112
113             if (verdict_light || verdict_heavy)
114             {
115                 // possible
116             } else {
117                 holding = false;
118                 break;
119             }
120         }
121         if (symbol == '>')
122         {
123             bool verdict_light = leftSum[0] > rightSum[0]; // assuming false coin is lighter than others
124             bool verdict_heavy = leftSum[1] > rightSum[1]; // assuming false coin is heavier than others
125
126             if (verdict_light || verdict_heavy)
127             {
128                 // possible
129             } else {
130                 holding = false;
131                 break;
132             }
133         }
134         if (symbol == '=')
135         {
136             bool verdict_light = leftSum[0] == rightSum[0]; // assuming false coin is lighter than others
137             bool verdict_heavy = leftSum[1] == rightSum[1]; // assuming false coin is heavier than others
138
139             if (verdict_light || verdict_heavy)
140             {
141                 // possible
142                 //cout << "checking equation: " << leftSum[0] << "=" << rightSum[0] << " OR " << leftSum[1] << ↵
143                     ↵ "=" << rightSum[1];
144             } else {
145                 //cout << "does not hold...";
146                 holding = false;
147                 break;
148             }
149         }
150     }

```

```
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

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void init_mergesort(vector<int> &racers, vector<int> &aux, int left, int right);
6  void sort(vector<int> &racers, vector<int> &aux, int left, int right);
7  void merge(vector<int> &racers, vector<int> &aux, int left, int pivot, int right);
8
9  vector<unsigned long> answers;
10
11 unsigned long overpasses;
12
13 int main(int argc, char const *argv[])
14 {
15
16     int testcases;
17     cin >> testcases;
18
19     for (int testcase = 0; testcase < testcases; testcase++)
20     {
21         int size;
22         cin >> size;
23
24         vector<int> racers;
25         vector<int> aux;
26
27         for (int racer = 0; racer < size; racer++)
28         {
29             int pos;
30             cin >> pos;
31             racers.push_back(pos);
32         }
33
34         aux = racers;
35
36         overpasses = 0;
37         init_mergesort(racers, aux, 0, size-1);
38         answers.push_back(overpasses % 10000);
39     }
40
41     for(vector<unsigned long>::iterator iter = answers.begin(); iter != answers.end(); iter++) {
42         cout << *iter << "\n";
43     }
44
45     return 0;
46 }
47
48 void init_mergesort(vector<int> &racers, vector<int> &aux, int left, int right) {
49     int pivot = (left + right) / 2;
50
51     sort(racers, aux, left, pivot);
52     sort(racers, aux, pivot + 1, right);
53     merge(racers, aux, left, pivot, right);
54 }
55
56 void sort(vector<int> &racers, vector<int> &aux, int left, int right) {
57     if (left < right)
58     {
59         int pivot = (left + right) / 2;
60         sort(racers, aux, left, pivot);
61         sort(racers, aux, pivot+1, right);
62         merge(racers, aux, left, pivot, right);
63     }
64 }
65
66 void merge(vector<int> &racers, vector<int> &aux, int left, int pivot, int right) {
67
68     unsigned long local_overpasses = 0;
69     int a = left;
70     int i = left;
71     int j = pivot + 1;
72
73     // 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

```
1  #include <vector>
2  #include <set>
3  #include <queue>
4  #include <sstream>
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  vector<string> answers;
10
11 int main(int argc, char const *argv[])
12 {
13
14     int testsets;
15     cin >> testsets;
16
17     for (int testset = 0; testset < testsets; testset++) {
18
19         int m, n;
20         cin >> m >> n;
21
22         int s1, s2;
23         cin >> s1 >> s2;
24
25         int f1, f2;
26         cin >> f1 >> f2;
27
28         int numberObstacles;
29         cin >> numberObstacles;
30
31         vector< vector<bool> > obstacles (m, vector<bool>(n));
32         for (int o = 0; o < numberObstacles; o++)
33         {
34             int x1, y1, x2, y2;
35             cin >> x1 >> y1 >> x2 >> y2;
36
37             for (int x = x1; x <= x2; x++)
38             {
39                 for (int y = y1; y <= y2; y++) {
40                     obstacles[x][y] = true;
41                 }
42             }
43         }
44
45         if (obstacles[f1][f2] == true)
46         {
47             answers.push_back("No solution.");
48             continue;
49         }
50
51         // visited states
52         vector< vector< set<pair<int,int> > > > visited(m, vector<set<pair<int,int> > >(n) );
53
54         // fifo queue for BFS
55         queue<pair< pair< pair<int, int>, int>, pair<int, int> > > fifo;
56
57         // adding starting point to fifo queue
58         pair<pair< pair<int, int>,int>, pair<int, int> > start_point = make_pair( make_pair( make_pair(s1,s2) , 0),
59                                     ↵ make_pair(0,0));
60         fifo.push(start_point);
61         visited[s1][s2].insert(make_pair(0,0));
62
63         bool success = false;
64
65         while (!fifo.empty()) {
66             pair<
67                 pair<
68                     pair<int, int>, int>,
69                     pair<int,int>
70                 > current_element = fifo.front();
71
72             // remove current element
73             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),
124                             ↵ current_hops + 1);
125                 pair< pair< pair<int, int>, int>, pair<int,int> > fifo_element = ↵
126                             ↵ make_pair(child_position, child_velocity);
127                 fifo.push(fifo_element);
128
129                 // add to visited nodes
130                 visited[new_x][new_y].insert(child_velocity);
131             }
132         }
133     }
134 }
135
136 if(success == true) {
137     loopend:
138     break;
139 }
140
141 if (success == false) answers.push_back("No_solution.");
142 }
143
144 for (vector<string>::iterator iter = answers.begin(); iter != answers.end(); iter++)
145 {
146

```

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

# Burning Coins

```
1  #include <vector>
2  #include <iostream>
3  using namespace std;
4
5  #define UNDEFINED -1
6  typedef vector<int> vi;
7  typedef vector<vi> vii;
8
9  int subsequence(int left, int right, vi& coins, vii& dp_table) {
10     if(dp_table[left][right] != UNDEFINED) return dp_table[left][right];
11
12     if(left > right) left = right;
13     if(left == right) return dp_table[left][right] = coins[left];
14     if(right - left == 1) return dp_table[left][right] = max(coins[left], coins[right]);
15
16     int min_left = min(subsequence(left+2, right, coins, dp_table), subsequence(left+1, right-1, coins, dp_table));
17     int min_right = min(subsequence(left, right-2, coins, dp_table), subsequence(left+1, right-1, coins, dp_table));
18     return dp_table[left][right] = max(coins[left]+min_left, coins[right]+min_right);
19 }
20
21 void testcase() {
22     int n; cin >> n;
23     vi coins(n);
24     for(int i = 0; i < n; ++i) {
25         int input; cin >> input;
26         coins[i] = input;
27     }
28
29     vii dp_table(n, vi(n, UNDEFINED));
30     subsequence(0, n-1, coins, dp_table);
31     cout << dp_table[0][n-1] << "\n";
32 }
33
34 int main() {
35     int TC; cin >> TC;
36     while(TC--) testcase();
37     return 0;
38 }
```

# Jump

```
1  #include <vector>
2  #include <iostream>
3  #include <queue>
4  using namespace std;
5
6  typedef vector<unsigned long int> vi;
7
8  void testcase() {
9      int n, k; cin >> n >> k;
10     int input; cin >> input; // ignore first input.
11
12     priority_queue<pair<long unsigned int, int>, vector<pair<long unsigned int, int> >, greater<pair<long unsigned int, int> > > min_heap;
13
14     vi dp_table;
15     dp_table.push_back(0);
16
17     for(int i = 1; i < n; ++i) {
18         while((!min_heap.empty()) && (min_heap.top().second < max(0, i - k))) min_heap.pop();
19         min_heap.push(make_pair(dp_table[i-1], i-1));
20
21         int input; cin >> input;
22         long unsigned int new_min = input + min_heap.top().first;
23         dp_table.push_back(new_min);
24     }
25     cout << dp_table[n-1] << "\n";
26 }
27
28 int main() {
29     ios_base::sync_with_stdio(false);
30     int TC; cin >> TC;
31     while(TC--) testcase();
32     return 0;
33 }
```

# Light Pattern

```
1  #include <vector>
2  #include <iostream>
3  #include <cmath>
4  using namespace std;
5
6  #define SWAP 1
7  #define NO_SWAP 0
8  typedef vector<int> vi;
9  typedef vector<vi> vii;
10
11 void testcase() {
12     int n, k, x; cin >> n >> k >> x;
13
14     vi pattern;
15     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 {
16         pattern.push_back(0); }
17
18     vii changes(n/k, vi(2));
19     for(int i = 0, p = 0, b = 0; i < n; ++i, ++p) {
20         int input; cin >> input;
21         (pattern[p] == input) ? changes[b][SWAP] += 1 : changes[b][NO_SWAP] += 1;
22         if(p == k-1) { p = -1; ++b; }
23     }
24
25     vii dp_table(n/k, vi(2));
26     dp_table[0][SWAP] = changes[0][SWAP] + 1;
27     dp_table[0][NO_SWAP] = changes[0][NO_SWAP];
28     for(int b = 1; b < (n/k); ++b) {
29         dp_table[b][SWAP] = min(dp_table[b-1][SWAP] + changes[b][SWAP], dp_table[b-1][NO_SWAP] + 2 + changes[b][SWAP]);
30         dp_table[b][NO_SWAP] = min(dp_table[b-1][NO_SWAP] + changes[b][NO_SWAP], dp_table[b-1][SWAP] +
31             changes[b][NO_SWAP]);
32     }
33
34     cout << min(dp_table[(n/k)-1][SWAP], dp_table[(n/k)-1][NO_SWAP]) << "\n";
35 }
36
37 int main() {
38     int TC; cin >> TC;
39     while(TC--) testcase();
40     return 0;
41 }
```

# Longest Path

```
1  #include <vector>
2  #include <queue>
3  #include <iostream>
4  #include <algorithm>
5  using namespace std;
6
7  typedef vector<int> vi;
8  typedef vector<vi> AdjacencyList;
9
10 void drill(int target, int comingFrom, AdjacencyList& adj, vi& max, vector<priority_queue<int> >& incomingPaths, vi& longest, bool start) {
11     if(adj[target].size() == 1 && !start) {
12         max[target] = 0;
13         incomingPaths[comingFrom].push(1);
14         return;
15     }
16
17     for(unsigned int outgoing = 0; outgoing < adj[target].size(); ++outgoing) {
18         if(adj[target][outgoing] != comingFrom)
19             drill(adj[target][outgoing], target, adj, max, incomingPaths, longest, false);
20     }
21
22     int first = incomingPaths[target].top(); incomingPaths[target].pop();
23     int second = 0;
24     if(!incomingPaths[target].empty()) {
25         second = incomingPaths[target].top(); incomingPaths[target].pop();
26     }
27
28     max[target] = first;
29     longest[target] = first + second;
30     incomingPaths[comingFrom].push(first+1);
31 }
32
33 void testcase() {
34     int vertices; cin >> vertices;
35
36     if(vertices == 1) { int v1, v2; cin >> v1 >> v2; cerr << 1 << "\n"; return; }
37
38     AdjacencyList adj(vertices);
39     vi max(vertices, 0);
40     vi longest(vertices, 0);
41     vector<priority_queue<int> > incomingPaths(vertices);
42
43     for(int input = 0; input < vertices-1; ++input) {
44         int v1, v2; cin >> v1 >> v2;
45         adj[v1].push_back(v2);
46         adj[v2].push_back(v1);
47     }
48
49     drill(0, 0, adj, max, incomingPaths, longest, true);
50     cout << *max_element(longest.begin(), longest.end())+1 << "\n";
51 }
52
53 int main() {
54     ios_base::sync_with_stdio(false);
55     int TC; cin >> TC;
56     while(TC--) testcase();
57     return 0;
58 }
```

  

```
1  #include <vector>
2  #include <iostream>
3  #include <queue>
4  #include <algorithm>
5  using namespace std;
6
7  typedef vector<int> vi;
8  typedef vector<vi> vii;
9  int N;
10
11 pair<int, int> DFS(int start, vii& adj, vi& dist, vi& visited) {
12     queue<int> fifo;
13     fifo.push(start);
```

```

14     visited[start] = 1;
15
16     while(!fifo.empty()) {
17         int parent_id = fifo.front(); fifo.pop();
18         for(int child = 0; child < adj[parent_id].size(); ++child) {
19             int child_id = adj[parent_id][child];
20             if(visited[child_id] == 0) {
21                 fifo.push(child_id);
22                 visited[child_id] = 1;
23                 dist[child_id] = dist[parent_id] + 1;
24             }
25         }
26     }
27     vi::iterator it = max_element(dist.begin(), dist.end());
28     pair<int, int> val;
29     val.first = it - dist.begin();
30     val.second = *it;
31     return val;
32 }
33
34 void testcase() {
35     cin >> N; // N vertices, by definition N-1 edges.
36     vii adj(N);
37     vi dist(N, 0);
38     vi visited(N, 0);
39
40     for(int n = 0; n < N-1; ++n) {
41         int v1, v2; cin >> v1 >> v2;
42         adj[v1].push_back(v2);
43         adj[v2].push_back(v1);
44     }
45     if(N == 1) { cout << 0 << "\n"; return; }
46
47     pair<int, int> pass1 = DFS(0, adj, dist, visited);
48     dist.assign(N, 0); visited.assign(N, 0);
49     pair<int, int> pass2 = DFS(pass1.first, adj, dist, visited);
50     cout << pass2.second+1 << "\n";
51 }
52
53 int main() {
54     cin.sync_with_stdio(false);
55     int TC; cin >> TC;
56     while(TC--) testcase();
57     return 0;
58 }

```

# Ants

```
1  #include <vector>
2  #include <iostream>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/graph_traits.hpp>
5  #include <boost/tuple/tuple.hpp>
6  #include <boost/graph/kruskal_min_spanning_tree.hpp>
7  #include <boost/graph/dijkstra_shortest_paths.hpp>
8  using namespace std;
9  using namespace boost;
10
11 typedef property<edge_weight_t, int, property<edge_index_t, int> > EdgeProperties;
12 typedef property<vertex_index_t, int> VertexProperties;
13 typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, EdgeProperties> Graph;
14 typedef graph_traits<Graph>::vertex_descriptor Vertex;
15 typedef graph_traits<Graph>::edge_descriptor Edge;
16 typedef property_map<Graph, edge_weight_t>::type WeightMap;
17 typedef property_map<Graph, edge_index_t>::type EIndexMap;
18 typedef property_map<Graph, vertex_index_t>::type VIndexMap;
19 typedef graph_traits<Graph>::edge_iterator EdgeIterator;
20 typedef vector<int> vi;
21 typedef vector<vi> vii;
22 typedef vector<Edge> ve;
23
24 void testcase() {
25     int N, M, S, a, b; cin >> N >> M >> S >> a >> b;
26
27     Graph g;
28     WeightMap weightMap = get(edge_weight, g);
29     EIndexMap eIndexMap = get(edge_index, g);
30     vii weights(M);
31
32     for(int e = 0; e < M; ++e) {
33         int t1, t2; cin >> t1 >> t2;
34         for(int s = 0; s < S; ++s) {
35             int s_weight; cin >> s_weight;
36             weights[e].push_back(s_weight);
37         }
38
39         Edge edge; bool success;
40         tie(edge, success) = add_edge(t1, t2, g);
41         eIndexMap[edge] = e;
42     }
43
44     Graph final;
45     WeightMap weightMapFinal = get(edge_weight, final);
46
47     for(int s = 0; s < S; ++s) {
48         int hive; cin >> hive;
49
50         EdgeIterator eit, eend;
51         for(tie(eit, eend) = edges(g); eit != eend; ++eit) weightMap[*eit] = weights[eIndexMap[*eit]][s];
52
53         ve mst(num_vertices(g)-1);
54         kruskal_minimum_spanning_tree(g, mst.begin());
55         for(ve::iterator edge = mst.begin(); edge != mst.end(); ++edge) {
56             Edge newEdge; bool success;
57             tie(newEdge, success) = add_edge(source(*edge, g), target(*edge, g), final);
58             weightMapFinal[newEdge] = weightMap[*edge];
59         }
60     }
61
62     vi d(num_vertices(final));
63     dijkstra_shortest_paths(final, vertex(a, final), distance_map(&d[0]));
64     cout << d[b] << "\n";
65 }
66
67 int main() {
68     ios_base::sync_with_stdio(false);
69     int TC; cin >> TC;
70     while(TC--) testcase();
71     return 0;
72 }
```



# Bridges

```
1  #include <vector>
2  #include <iostream>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/biconnected_components.hpp>
6  using namespace std;
7  using namespace boost;
8
9  typedef property<vertex_index_t, int> VertexProperties;
10 typedef adjacency_list< vecS, vecS, undirectedS, VertexProperties, no_property> Graph;
11 typedef property_map<Graph, vertex_index_t>::type VIndexMap;
12 typedef graph_traits<Graph>::vertex_descriptor Vertex;
13 typedef graph_traits<Graph>::edge_descriptor Edge;
14 typedef graph_traits<Graph>::adjacency_iterator Alter;
15 typedef vector<int> vi;
16 typedef vector<Vertex> vv;
17 typedef pair<int, int> pi;
18
19 void testcase() {
20     int N, M; cin >> N >> M;
21
22     if(N == 0 || M == 0) { cout << "0\n"; return; }
23
24     Graph g(N);
25     VIndexMap index = get(vertex_index, g);
26
27     for(int m = 0; m < M; ++m) {
28         int v1, v2; cin >> v1 >> v2;
29         add_edge(v1, v2, g);
30     }
31
32     vv art_points;
33     vi discover_time(num_vertices(g));
34     vi low_point(num_vertices(g));
35     vector<pi> bridges;
36     articulation_points(g,
37                         back_inserter(art_points),
38                         discover_time_map(&discover_time[0]).lowpoint_map(&low_point[0]));
39
40     // workaround for "root not chosen as articulation point if only one child".
41     if(out_degree(vertex(1, g), g) == 1) {
42         Vertex root = vertex(1, g);
43         art_points.insert(art_points.begin(), root);
44     }
45     for(int v = 0; v < art_points.size(); ++v) {
46         Vertex art_point = art_points[v];
47         Alter neighbour, neighbour_end;
48         for(tie(neighbour, neighbour_end) = adjacent_vertices(art_point, g); neighbour != neighbour_end; ++neighbour) {
49             if(low_point[*neighbour] > discover_time[art_point]) {
50                 //cout << "bridge found between: " << index[art_point] << "-" << index[*neighbour] << "\n";
51                 bridges.push_back(make_pair(min(index[art_point], index[*neighbour]), max(index[art_point],
52                 ↵ index[*neighbour])));
53             }
54         }
55     }
56     sort(bridges.begin(), bridges.end());
57     cout << bridges.size() << "\n";
58     for(int b = 0; b < bridges.size(); ++b) {
59         cout << bridges[b].first << "□" << bridges[b].second << "\n";
60     }
61 }
62
63 int main() {
64     int TC; cin >> TC;
65     while(TC--) testcase();
66     return 0;
67 }
68
69 #include <vector>
70 #include <iostream>
71 #include <algorithm>
72 #include <set>
```

```

5  using namespace std;
6
7  #define UNVISITED 0
8  #define VISITED 1
9  #define EXPLORED 2
10
11 typedef vector<int> vi;
12 typedef vector<vi> vii;
13 typedef pair<int, int> pi;
14
15 vi visited;
16 vi dfs_num;
17 vi dfs_low;
18
19 void dfs(int vertex, int parent, vii& adj, int counter) {
20     for(signed int child = 0; child < adj[vertex].size(); ++child) {
21         int child_vertex = adj[vertex][child];
22         if(child_vertex != parent) {
23             if(visited[child_vertex] == EXPLORED) {
24                 dfs_low[vertex] = min(dfs_num[child_vertex], dfs_low[vertex]);
25             }
26
27             if(visited[child_vertex] == UNVISITED) {
28                 visited[child_vertex] = EXPLORED;
29                 dfs_num[child_vertex] = ++counter;
30                 dfs_low[child_vertex] = dfs_num[child_vertex];
31                 dfs(child_vertex, vertex, adj, counter);
32             }
33         }
34     }
35
36     dfs_low[parent] = min(dfs_low[parent], dfs_low[vertex]);
37     visited[vertex] = VISITED;
38 }
39
40 void testcase() {
41     int N, M; cin >> N >> M;
42     visited.clear(); dfs_low.clear(); dfs_num.clear();
43     vii adj(N); visited.assign(N, UNVISITED); dfs_num.assign(N, 0); dfs_low.assign(N, 0);
44
45     if(N == 0 || M == 0) { cout << "0\n"; return; }
46
47     for(int m = 0; m < M; ++m) {
48         int v1, v2; cin >> v1 >> v2;
49         adj[(v1-1)].push_back(v2-1);
50         adj[(v2-1)].push_back(v1-1);
51     }
52
53     dfs_num[0] = 0; dfs_low[0] = 0; visited[0] = EXPLORED;
54     dfs(0, 0, adj, 0);
55
56     vector<pi> bridges;
57     set<int> art_points;
58     for(int u = 0; u < N; ++u) {
59         for(int v = 0; v < adj[u].size(); ++v) {
60             if(dfs_low[adj[u][v]] > dfs_num[u]) {
61                 bridges.push_back(make_pair(min(u, adj[u][v]), max(u, adj[u][v])));
62             }
63             if(dfs_low[adj[u][v]] >= dfs_num[u]) {
64                 // if it is not root, or it is root but has more than 1 child:
65                 art_points.insert(u);
66             }
67         }
68     }
69     sort(bridges.begin(), bridges.end());
70     cout << bridges.size() << "\n";
71     for(signed int b = 0; b < bridges.size(); ++b) {
72         cout << bridges[b].first+1 << " " << bridges[b].second+1 << "\n";
73     }
74 }
75
76 int main() {
77     int TC; cin >> TC;
78     while(TC--) testcase();
79     return 0;
80 }

```

# Build The Graph

```
1  #include <iostream>
2  #include <boost/graph/adjacency_list.hpp>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/kruskal_min_spanning_tree.hpp>
5  #include <boost/graph/dijkstra_shortest_paths.hpp>
6  using namespace std;
7  using namespace boost;
8
9  // create internal properties
10 typedef property<vertex_index_t, int> IndexProperty;
11 typedef property<edge_weight_t, int> WeightProperty;
12
13 // adjacency list with properties
14 typedef adjacency_list<vecS, vecS, undirectedS, no_property, WeightProperty, IndexProperty> Graph;
15
16 // Vertex and edge type
17 typedef graph_traits<Graph>::vertex_descriptor Vertex;
18 typedef graph_traits<Graph>::edge_descriptor Edge;
19 typedef graph_traits<Graph>::edge_iterator EdgeIterator;
20
21 // Property maps for accessing the properties
22 typedef property_map<Graph, edge_weight_t>::type WeightMap;
23 typedef property_map<Graph, vertex_index_t>::type IndexMap;
24
25 int main() {
26     ios_base::sync_with_stdio(false);
27     int t; cin >> t;
28
29     for(int i = 0; i < t; i++) {
30         int m, n; cin >> n >> m;
31
32         Graph G(n);
33         WeightMap weightMap = get(edge_weight, G);
34
35         for(int j = 0; j < m; j++) {
36             int v1, v2, w;
37             cin >> v1 >> v2 >> w;
38             Edge e;
39             tie(e, tuples::ignore) = add_edge(v1, v2, G);
40             weightMap[e] = w;
41         }
42
43         vector<Edge> spanningTree;
44         kruskal_minimum_spanning_tree(G, back_inserter(spanningTree));
45         int sumOfWeights = 0;
46
47         Graph mstGraph(n);
48         WeightMap mstWeightMap = get(edge_weight, mstGraph);
49         for (vector<Edge>::iterator ei = spanningTree.begin(); ei != spanningTree.end(); ++ei) {
50             sumOfWeights += weightMap[*ei];
51         }
52
53         vector<int> distances(n);
54         vector<Vertex> p_map(num_vertices(G));
55
56         Vertex startVertex = vertex(0, G);
57         dijkstra_shortest_paths(G, startVertex, predecessor_map(&p_map[0]).distance_map(&distances[0]));
58
59         int longestDistance = 0;
60         for(int k = 0; k < n; k++) {
61             int distance = distances[k];
62
63             if(distance > longestDistance) {
64                 longestDistance = distance;
65             }
66         }
67
68         cout << sumOfWeights << "\n" << longestDistance << endl;
69
70         /* Playing around with backtracking shortest path.
71         IndexMap index;
72         int target = 3;
73         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

```
1  #include <vector>
2  #include <iostream>
3  #include <queue>
4  #include <algorithm>
5  using namespace std;
6
7  typedef vector<int> vi;
8  typedef vector<vi> vii;
9
10 int k;
11
12 void testcase() {
13     int n, m, k;
14     cin >> n >> m >> k;
15
16     vii adj(n);
17     vii groups(k);
18     vi col(n, -1);
19
20     for(int e = 0; e < m; ++e) {
21         int v1, v2; cin >> v1 >> v2;
22         adj[v1].push_back(v2);
23         adj[v2].push_back(v1);
24     }
25
26     queue<int> q; // lifo
27     int c = 0;
28     q.push(0);
29     col[0] = c;
30     groups[c].push_back(0);
31
32     while(!q.empty()) {
33         const int v = q.front(); q.pop();
34         for(int child = 0; child < adj[v].size(); ++child) {
35             const int u = adj[v][child];
36             if(col[u] != -1) continue;
37
38             c = (c == k-1) ? 0 : ++c;
39             if(col[v] == c) { c = (c == k-1) ? 0 : ++c; }
40             col[u] = c;
41             groups[c].push_back(u);
42             q.push(u);
43         }
44     }
45
46     if(n >= k && find(col.begin(), col.end(), -1) == col.end()) {
47         cout << "yes\n";
48         for(int g = 0; g < k; ++g) {
49             cout << groups[g].size();
50             for(int i = 0; i < groups[g].size(); ++i) {
51                 cout << "□" << groups[g][i];
52             }
53             cout << "\n";
54         }
55     } else {
56         cout << "no\n";
57     }
58     col.clear();
59     adj.clear();
60     groups.clear();
61 }
62
63 int main() {
64     int TC; cin >> TC;
65     while(TC--) testcase();
66 }
```

# Shy Programmers

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/boyer_myrvold_planar_test.hpp>
5  using namespace std;
6  using namespace boost;
7
8  typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;
9
10 void testcase() {
11     int N, M; cin >> N >> M;
12     Graph g(N+1);
13     vector<int> processed(N, 0);
14     for(int m = 0; m < M; ++m) {
15         int a, b; cin >> a >> b;
16         add_edge(a, b, g);
17         if(!processed[a]) { add_edge(a, N+1, g); processed[a] = 1; }
18         if(!processed[b]) { add_edge(b, N+1, g); processed[b] = 1; }
19     }
20
21     if(boyer_myrvold_planarity_test(g))
22         cout << "yes\n";
23     else
24         cout << "no\n";
25 }
26
27 int main() {
28     int TC; cin >> TC;
29     while(TC--) testcase();
30     return 0;
31 }
```

## Algocoon Group

Missing.

# Buddies

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <string>
5  #include <utility>
6  #include <boost/tuple/tuple.hpp>
7  #include <boost/graph/adjacency_list.hpp>
8  #include <boost/graph/max_cardinality_matching.hpp>
9  using namespace std;
10 using namespace boost;
11
12 typedef vector<int> vi;
13 typedef pair<int, int> ii;
14
15 typedef property<vertex_index_t, int> VertexProperties;
16 typedef property<edge_weight_t, int> EdgeProperties;
17 typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, EdgeProperties> Graph;
18 typedef graph_traits<Graph>::vertex_descriptor Vertex;
19 typedef graph_traits<Graph>::edge_descriptor Edge;
20 typedef property_map<Graph, vertex_index_t>::type VIndexMap;
21 typedef graph_traits<Graph>::edge_iterator EdgeIterator;
22
23 void testcase() {
24     int n, c, f; cin >> n >> c >> f;
25     map<string, vi> char_map;
26
27     for(int student = 0; student < n; ++student) {
28         for(int characteric = 0; characteric < c; ++characteric) {
29             string input; cin >> input;
30             if(char_map.count(input) == 0) {
31                 vi students; students.push_back(student);
32                 char_map.insert(make_pair(input, students));
33             }
34             else { char_map[input].push_back(student); }
35         }
36     }
37
38     map<ii, int> edges;
39     for(map<string, vi>::iterator iter = char_map.begin(); iter != char_map.end(); ++iter) {
40         pair<string, vi> value_pair = *iter;
41         vi& values = value_pair.second;
42         for(int s1 = 0; s1 < values.size()-1; ++s1) {
43             for(int s2 = s1+1; s2 < values.size(); ++s2) {
44                 ii edge = make_pair(values[s1], values[s2]);
45                 if(edges.count(edge) == 0) { edges.insert(make_pair(edge, 1)); }
46                 else { edges[edge]++; }
47             }
48         }
49     }
50
51     Graph g(n);
52     for(map<ii, int>::iterator iter = edges.begin(); iter != edges.end(); ++iter) {
53         pair<ii, int> edge_pair = *iter;
54         //cout << "edge: " << edge_pair.first.first << "-" << edge_pair.first.second << " weight: " << ↀ
55             << edge_pair.second << "\n";
56         if(edge_pair.second > f) {
57             add_edge(edge_pair.first.first, edge_pair.first.second, g);
58         }
59     }
60
61     vector<Vertex> mateMap (num_vertices(g));
62     bool matching_success = checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
63
64     if(matching_success) {
65         if(matching_size(g, &mateMap[0]) < n/2 ) cout << "optimal\n";
66         else cout << "not_ↀoptimal\n";
67     }
68
69     int main() {
70         ios_base::sync_with_stdio(false);
71         int TC; cin >> TC;
72         while(TC--) testcase();
73     }
```



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

# Satellites

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/max_cardinality_matching.hpp>
6  #include <boost/graph/bipartite.hpp>
7  #include <boost/graph/depth_first_search.hpp>
8  using namespace std;
9  using namespace boost;
10
11 #define UNVISITED 0
12 #define VISITED 1
13 #define LEFT 0
14 #define RIGHT 1
15
16 typedef vector<int> vi;
17 typedef property<vertex_index_t, int> VertexProperties;
18 typedef adjacency_list<vecS, vecS, undirectedS, VertexProperties, no_property> Graph;
19 typedef adjacency_list<vecS, vecS, directedS, VertexProperties, no_property> Digraph;
20 typedef graph_traits<Graph>::vertex_descriptor Vertex;
21 typedef graph_traits<Graph>::edge_descriptor Edge;
22 typedef graph_traits<Graph>::vertex_iterator VertexIterator;
23 typedef property_map<Graph, vertex_index_t>::type VIndexMap;
24
25 vi visited;
26
27 struct mark_visited : public boost::dfs_visitor<> {
28     template <class Vertex, class Digraph>
29     void finish_vertex(Vertex u, const Digraph& g) {
30         visited[u] = VISITED;
31         //cout << u << " set to visited. \n";
32     }
33 };
34
35 void testcase() {
36     int groundstations, satellites, links; cin >> groundstations >> satellites >> links;
37
38     Digraph g_final(groundstations + satellites);
39     Graph g(groundstations + satellites);
40
41     vi color(num_vertices(g));
42     for(int edge = 0; edge < links; ++edge) {
43         int v1, v2; cin >> v1 >> v2;
44         v2 = v2 + groundstations;
45         add_edge(v1, v2, g);
46         add_edge(v1, v2, g_final);
47         color[v1] = LEFT;
48         color[v2] = RIGHT;
49     }
50
51     vector<Vertex> mateMap(num_vertices(g), UNVISITED);
52     bool success = checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
53
54
55     visited.clear();
56     visited.assign(num_vertices(g), UNVISITED);
57     for(int matching = 0; matching < mateMap.size(); ++matching) {
58         if(color[matching] == RIGHT && mateMap[matching] != graph_traits<Graph>::null_vertex())
59             add_edge(matching, mateMap[matching], g_final); // add an edge from R to L.
60         if(mateMap[matching] == graph_traits<Graph>::null_vertex() && color[matching] == LEFT)
61             visited[matching] = VISITED;
62     }
63
64     mark_visited vis;
65     for(int start = 0; start < visited.size(); ++start) {
66         if((color[start] == LEFT) && (visited[start] == VISITED)) {
67             //cout << "start dfs at " << start << " visited: " << visited[start] << "\n";
68             //depth_first_search(g_final, root_vertex(vertex(start, g_final)).visitor(vis));
69             vector<default_color_type> colors(num_vertices(g_final));
70             depth_first_visit(g_final, vertex(start, g_final), vis, &colors[0]);
71         }
72     }
73 }
```

```

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

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/push_relabel_max_flow.hpp>
6  using namespace std;
7  using namespace boost;
8
9  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
10 typedef adjacency_list<vecS, vecS, directedS, no_property,
11     property<edge_capacity_t, long,
12     property<edge_residual_capacity_t, long,
13     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
14 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
15 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
16 typedef graph_traits<Graph>::edge_descriptor Edge;
17
18 void add_edge(int f, int t, int cap, Graph& g) {
19     EdgeCapacityMap capacity = get(edge_capacity, g);
20     ReverseEdgeMap rev_edge = get(edge_reverse, g);
21
22     Edge edge;
23     tie(edge, tuples::ignore) = add_edge(f, t, g);
24     Edge reverse_edge;
25     tie(reverse_edge, tuples::ignore) = add_edge(t, f, g);
26     capacity[edge] = cap;
27     rev_edge[edge] = reverse_edge;
28     capacity[reverse_edge] = 0;
29     rev_edge[reverse_edge] = edge;
30 }
31
32 void testcase() {
33     int V, E; cin >> V >> E;
34     Graph g(V+2);
35     int source = V;
36     int sink = V+1;
37
38     vector<int> vertices;
39     for(int v = 0; v < V; ++v) {
40         int g, d; cin >> g >> d;
41         vertices.push_back(d - g);
42     }
43
44     for(int e = 0; e < E; ++e) {
45         int f, t, lb, ub; cin >> f >> t >> lb >> ub;
46         add_edge(f, t, ub-lb, g);
47         vertices[f] += lb;
48         vertices[t] -= lb;
49     }
50
51     int flow_out = 0;
52     bool all_pos = true;
53     for(int v = 0; v < V; ++v) {
54         if(vertices[v] < 0) {
55             add_edge(source, v, abs(vertices[v]), g);
56         } else if(vertices[v] > 0) {
57             all_pos = false;
58             add_edge(v, sink, vertices[v], g);
59             flow_out += abs(vertices[v]);
60         }
61     }
62
63     int max_flow = push_relabel_max_flow(g, source, sink);
64     (max_flow == flow_out || all_pos) ? cout << "yes\n" : cout << "no\n";
65 }
66
67 int main() {
68     int TC; cin >> TC;
69     while(TC--) testcase();
70 }
```

# Coin Tossing

```
1  #include <iostream>
2  #include <boost/graph/adjacency_list.hpp>
3  #include <boost/graph/push_relabel_max_flow.hpp>
4  #include <boost/tuple/tuple.hpp>
5  using namespace std;
6  using namespace boost;
7
8  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
9  typedef adjacency_list<vecS, vecS, directedS, no_property,
10     property<edge_capacity_t, long,
11     property<edge_residual_capacity_t, long,
12     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
13  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
14  typedef property_map<Graph, edge_residual_capacity_t>::type ResidualCapacityMap;
15  typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
16  typedef graph_traits<Graph>::edge_descriptor Edge;
17
18  void add_edge(int from, int to, int c, Graph& g) {
19     EdgeCapacityMap capacity = get(edge_capacity, g);
20     ReverseEdgeMap reverse = get(edge_reverse, g);
21     ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
22
23     Edge there, back;
24     tie(there, tuples::ignore) = add_edge(from, to, g);
25     tie(back, tuples::ignore) = add_edge(to, from, g);
26     capacity[there] = c;
27     capacity[back] = 0;
28     reverse[there] = back;
29     reverse[back] = there;
30 }
31
32 void testcase() {
33     int N, M; cin >> N >> M;
34     Graph g(N+M+2);
35     int source = N+M+1;
36     int sink = source + 1;
37
38     for(int m = N; m < N+M; ++m) {
39         int p1, p2, outcome;
40         cin >> p1 >> p2 >> outcome;
41         add_edge(source, m, 1, g);
42
43         if(outcome == 1) {
44             add_edge(m, p1, 1, g);
45         }
46         if(outcome == 2) {
47             add_edge(m, p2, 1, g);
48         }
49         if(outcome == 0) {
50             add_edge(m, p1, 1, g);
51             add_edge(m, p2, 1, g);
52         }
53     }
54
55     int sum = 0;
56     for(int p = 0; p < N; ++p) {
57         int score; cin >> score;
58         sum += score;
59         add_edge(p, sink, score, g);
60     }
61
62     int f_max = push_relabel_max_flow(g, source, sink);
63     if(M == sum && f_max == sum) cout << "yes\n";
64     else cout << "no\n";
65 }
66
67 int main() {
68     int TC; cin >> TC;
69     while(TC--) testcase();
70 }
```

# Antenna

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
5  #include <CGAL/Min_circle_2.h>
6  #include <CGAL/Min_circle_2_traits_2.h>
7  using namespace std;
8
9  typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
10 typedef CGAL::Min_circle_2_traits_2<K> Traits;
11 typedef CGAL::Min_circle_2<Traits> Min_circle;
12
13 double ceil_to_double(const K::FT& x)
14 {
15     double a = ceil(CGAL::to_double(x));
16     while (a < x) a += 1;
17     while (a-1 >= x) a -= 1;
18     return a;
19 }
20
21 void testcase(int n) {
22     vector<K::Point_2> citizens;
23     for(int coord = 0; coord < n; ++coord) {
24         double x, y; cin >> x >> y;
25         K::Point_2 citizen(x, y);
26         citizens.push_back(citizen);
27     }
28
29     Min_circle mc(citizens.begin(), citizens.end(), true); // true important for speed.
30     Traits::Circle c = mc.circle();
31     K::FT radius = sqrt(c.squared_radius());
32
33     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0); // scientific notation will be used otherwise!
34     cout << ceil_to_double(radius) << "\n";
35 }
36
37 int main() {
38     while(true) {
39         int n; cin >> n;
40         if(n == 0) return 0;
41         testcase(n);
42     }
43     return 0;
44 }
```

# Almost Antenna

```
1  #include <set>
2  #include <iostream>
3  #include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
4  #include <CGAL/Min_circle_2.h>
5  #include <CGAL/Min_circle_2_traits_2.h>
6  using namespace std;
7
8  typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
9  typedef CGAL::Min_circle_2_traits_2<K> MinCircleTraits;
10 typedef CGAL::Min_circle_2<MinCircleTraits> Min_circle;
11 typedef Min_circle::Support_point_iterator support_iter;
12
13 double ceil_to_double(const K::FT& x)
14 {
15     double a = ceil(CGAL::to_double(x));
16     while (a < x) a += 1;
17     while (a-1 >= x) a -= 1;
18     return a;
19 }
20
21 void testcase(int n) {
22     vector<K::Point_2> points;
23     for(int point = 0; point < n; ++point) {
24         double x, y; cin >> x >> y;
25         K::Point_2 p(x, y);
26         points.push_back(p);
27     }
28
29     Min_circle min_circle(points.begin(), points.end(), true);
30     MinCircleTraits::Circle c = min_circle.circle();
31     K::FT old_radius = c.squared_radius();
32     K::FT min_radius; bool min_radius_set = false;
33
34     for(support_iter iter = min_circle.support_points_begin(); iter != min_circle.support_points_end(); ++iter) {
35         // find supporting point in set. Delete it temporarily.
36         vector<K::Point_2>::iterator temp_it = find(points.begin(), points.end(), *iter);
37         K::Point_2 point = *temp_it;
38         points.erase(temp_it);
39
40         // create new min_circle, get squared radius.
41         Min_circle temp_circle (points.begin(), points.end(), true);
42         MinCircleTraits::Circle actual_circle = temp_circle.circle();
43         K::FT new_radius = actual_circle.squared_radius();
44
45         // compare radius of old min_circle with new one.
46         if(new_radius == old_radius) {
47             min_radius = old_radius; break;
48         } else if(!min_radius_set || new_radius < min_radius) {
49             min_radius = new_radius;
50             min_radius_set = true;
51         }
52
53         // reinsert the point
54         points.push_back(point);
55     }
56
57     double result = ceil_to_double(CGAL::sqrt(min_radius));
58     cout << result << "\n";
59 }
60
61 int main() {
62     ios_base::sync_with_stdio(false);
63     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
64     while(true) {
65         int n; cin >> n;
66         if(n == 0) return 0;
67         testcase(n);
68     }
69     return 0;
70 }
```

# Hit

```
1  #include <iostream>
2  #include <vector>
3  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
4  using namespace std;
5
6  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7
8  void testcase(int n) {
9      double x1, y1, x2, y2;
10     cin >> x1 >> y1 >> x2 >> y2;
11     K::Point_2 p1(x1, y1);
12     K::Point_2 p2(x2, y2);
13     K::Ray_2 ray(p1, p2);
14
15     vector<K::Segment_2> obstacles;
16     for(int o = 0; o < n; ++o) {
17         double r, s, t, u;
18         cin >> r >> s >> t >> u;
19         K::Point_2 p1(r, s);
20         K::Point_2 p2(t, u);
21         K::Segment_2 obstacle(p1, p2);
22         obstacles.push_back(obstacle);
23     }
24
25     bool intersect = false;
26     for(int obstacle = 0; obstacle < obstacles.size(); ++obstacle) {
27         if(CGAL::do_intersect(obstacles[obstacle], ray)) {
28             intersect = true;
29             break;
30         }
31     }
32
33     (intersect) ? cout << "yes\n" : cout << "no\n";
34 }
35
36 int main() {
37     while(true) {
38         int n; cin >> n;
39         if(n == 0) return 0;
40         testcase(n);
41     }
42 }
```



# First Hit

```
1  #include <iostream>
2  #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
3  #include <CGAL/enum.h>
4  #include <climits>
5  using namespace std;
6
7  typedef CGAL::Exact_predicates_exact_constructions_kernel K;
8
9  double floor_to_double(const K::FT& x) {
10     double a = std::floor(CGAL::to_double(x));
11     while (a > x) a -= 1;
12     while (a+1 <= x) a += 1;
13     return a;
14 }
15
16 void testcase(int n) {
17     K::Ray_2 ray;
18     double x1, y1, x2, y2; cin >> ray;
19
20     bool min_exists = false;
21     K::FT current_dist;
22     K::Point_2 current_point;
23
24     for(size_t o = 0; o < n; ++o) {
25         double r, s, t, u; cin >> r >> s >> t >> u;
26         K::Point_2 p1(r, s);
27         K::Point_2 p2(t, u);
28         K::Segment_2 obstacle (p1, p2);
29
30         if(CGAL::do_intersect(ray, obstacle)) {
31             K::Point_2 intersection_point;
32             CGAL::Object o = CGAL::intersection(ray, obstacle);
33             if(const K::Point_2* p = CGAL::object_cast<K::Point_2>(&o))
34                 intersection_point = *p;
35             else if (const K::Segment_2* s = CGAL::object_cast<K::Segment_2>(&o))
36                 intersection_point =
37                     CGAL::has_smaller_distance_to_point(ray.source(), s->source(), s->target()) ?
38                     s->source() : s->target();
39             else throw runtime_error("strange segment intersection");
40             K::FT intersection_dist = CGAL::squared_distance(intersection_point, ray.source());
41             if(!min_exists || current_dist > intersection_dist) {
42                 current_dist = intersection_dist;
43                 current_point = intersection_point;
44                 min_exists = true;
45             }
46         }
47     }
48
49     if(min_exists) cout << floor_to_double(current_point.x()) << "␣" << floor_to_double(current_point.y()) << "\n";
50     else cout << "no\n";
51 }
52
53 int main() {
54     cin.sync_with_stdio(false);
55     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
56     while(true) {
57         int n; cin >> n;
58         if(n == 0) return 0;
59         testcase(n);
60     }
61 }
```

## Search Snippets

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <queue>
5  #include <functional>
6  #include <cmath>
7  using namespace std;
8
9  typedef vector<int> vi;
10
11 void testcase() {
12     int n; cin >> n;
13     vector<vi> posting_list(n);
14
15     vi Npositions(n);
16     for(int i = 0; i < n; ++i) { int m; cin >> m; Npositions[i] = m; }
17
18     for(int word = 0; word < Npositions.size(); ++word) {
19         for(int position = 0; position < Npositions[word]; ++position) {
20             int input_position; cin >> input_position;
21             posting_list[word].push_back(input_position);
22         }
23     }
24
25     vi pointers(n, 0);
26     priority_queue<int> max_heap;
27     priority_queue<pair<int, int>, std::vector<pair<int, int> >, greater<pair<int, int> > > min_heap;
28
29     for(int list = 0; list < n; ++list) {
30         int value = posting_list[list][pointers[list]];
31         max_heap.push(value);
32         min_heap.push(make_pair(value, list));
33     }
34
35     int min_interval = 1073741825;
36     while(true) {
37         pair<int, int> min_pair = min_heap.top(); min_heap.pop();
38         int min_value = min_pair.first;
39         int min_list = min_pair.second;
40
41         int max_value = max_heap.top();
42         int min_new = abs(max_value - min_value);
43         min_interval = min(min_new, min_interval);
44
45         if(pointers[min_list] == posting_list[min_list].size()-1) { break; }
46         int jump = sqrt(posting_list[min_list].size());
47         while((pointers[min_list]+jump < posting_list[min_list].size()-1) &&
48             (posting_list[min_list][pointers[min_list]+jump] < min_heap.top().first)) {
49             pointers[min_list] += jump;
50         }
51         pointers[min_list]++;
52
53         int new_value = posting_list[min_list][pointers[min_list]];
54         max_heap.push(new_value);
55         min_heap.push(make_pair(new_value, min_list));
56     }
57
58     cout << min_interval+1 << "\n";
59 }
60
61 int main() {
62     ios_base::sync_with_stdio(false);
63     int TC; cin >> TC;
64     while(TC--) testcase();
65     return 0;
66 }
```

# Bistro

```
1  #include <vector>
2  #include <iostream>
3  #include <cmath>
4  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
5  #include <CGAL/Delaunay_triangulation_2.h>
6  using namespace std;
7
8  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
9  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
10 typedef Triangulation::Finite_faces_iterator faces_iterator;
11
12 double floor_to_double(const K::FT& x)
13 {
14     double a = std::floor(CGAL::to_double(x));
15     while (a > x) a -= 1;
16     while (a+1 <= x) a += 1;
17     return a;
18 }
19
20 void testcase(int n) {
21     vector<K::Point_2> delaunay_vertices;
22     for(int i = 0; i < n; ++i) {
23         K::Point_2 p; cin >> p;
24         delaunay_vertices.push_back(p);
25     }
26
27     Triangulation t;
28     t.insert(delaunay_vertices.begin(), delaunay_vertices.end());
29
30     int points; cin >> points;
31     for(int i = 0; i < points; ++i) {
32         K::Point_2 p; cin >> p;
33         Triangulation::Vertex_handle v = t.nearest_vertex(p);
34         K::Point_2 vp = v->point();
35         K::FT distance = CGAL::squared_distance(p, vp);
36         cout << floor_to_double(distance) << "\n";
37     }
38 }
39
40 int main() {
41     cin.sync_with_stdio(false);
42     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
43     while(true) {
44         int n; cin >> n;
45         if(n == 0) return 0;
46         testcase(n);
47     }
48     return 0;
49 }
```

# Germes

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
5  #include <CGAL/Delaunay_triangulation_2.h>
6  #include <CGAL/Triangulation_vertex_base_with_info_2.h>
7  using namespace std;
8
9  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
10 typedef CGAL::Triangulation_vertex_base_with_info_2<int, K> Vb;
11 typedef CGAL::Triangulation_data_structure_2<Vb> Tds;
12 typedef CGAL::Delaunay_triangulation_2<K, Tds> Delaunay;
13 typedef Delaunay::Finite_edges_iterator FEI;
14 typedef Delaunay::Finite_vertices_iterator FVI;
15
16 void testcase(int N) {
17     double left, bottom, right, top; cin >> left >> bottom >> right >> top;
18     vector<K::Segment_2> rectangle;
19     rectangle.push_back(K::Segment_2(K::Point_2(left, bottom), K::Point_2(left, top)));
20     rectangle.push_back(K::Segment_2(K::Point_2(left, top), K::Point_2(right, top)));
21     rectangle.push_back(K::Segment_2(K::Point_2(right, top), K::Point_2(right, bottom)));
22     rectangle.push_back(K::Segment_2(K::Point_2(right, bottom), K::Point_2(left, bottom)));
23
24     vector<pair<K::Point_2, int> > points;
25     for(int b = 0; b < N; ++b) {
26         double x, y; cin >> x >> y;
27         points.push_back(make_pair(K::Point_2(x, y), b));
28     }
29
30     Delaunay t;
31     t.insert(points.begin(), points.end());
32
33     vector<pair<double, pair<int, int> > > edges;
34     for(FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
35         Delaunay::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
36         Delaunay::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);
37         K::FT edge_length = CGAL::sqrt(CGAL::squared_distance(v1->point(), v2->point()));
38         edges.push_back(make_pair(edge_length, make_pair(v1->info(), v2->info())));
39     }
40
41     for(FVI p = t.finite_vertices_begin(); p != t.finite_vertices_end(); ++p) {
42         Delaunay::Vertex_handle vertex = p;
43         K::FT min; bool min_set = false;
44         for(int seg = 0; seg < 4; ++seg) {
45             K::FT seg_min = CGAL::squared_distance(rectangle[seg], vertex->point());
46             if(min_set == false || min > seg_min) { min_set = true; min = seg_min; }
47         }
48         edges.push_back(make_pair(2*CGAL::sqrt(min), make_pair(p->info(), p->info())));
49     }
50
51     sort(edges.begin(), edges.end());
52
53     int dead = 0;
54     int pointer = 0;
55     int h = 0;
56     bool first_time = true;
57     vector<int> deadlist(N, 0);
58
59     while(dead != N) {
60         double min_length = 2 * (pow(h, 2.0) + 0.5);
61         int temp_dead = 0;
62         while(edges[pointer].first <= min_length && pointer < edges.size()) {
63             int v1 = edges[pointer].second.first;
64             int v2 = edges[pointer].second.second;
65             if(deadlist[v1] == 0) { ++temp_dead; deadlist[v1] = 1; }
66             if(deadlist[v2] == 0) { ++temp_dead; deadlist[v2] = 1; }
67             ++pointer;
68         }
69         if(dead == 0 && temp_dead > 0) cout << h << "┐";
70         dead += temp_dead;
71         if((N-dead)/(double)N < 0.5 && first_time) {
72             cout << h << "┐";
73             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

```
1  #include <vector>
2  #include <iostream>
3  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h> // use inexact because Input points == output points.
4  #include <CGAL/Delaunay_triangulation_2.h>
5  using namespace std;
6
7  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
9  typedef Triangulation::Finite_edges_iterator FEI;
10
11 double ceil_to_double(const K::FT& x)
12 {
13     double a = ceil(CGAL::to_double(x));
14     while (a < x) a += 1;
15     while (a-1 >= x) a -= 1;
16     return a;
17 }
18
19 void testcase(int n) {
20     vector<K::Point_2> points;
21     for(int i = 0; i < n; ++i) {
22         K::Point_2 p; cin >> p;
23         points.push_back(p);
24     }
25
26     Triangulation t;
27     t.insert(points.begin(), points.end());
28     K::FT min_length;
29     bool min_set = false;
30     for (FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); e++) {
31         // REMEMBER bad idea: K::Segment_2 seg = t.segment(edge); seg.squared_length().
32         Triangulation::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
33         Triangulation::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);
34
35         K::FT length = CGAL::squared_distance(v1->point(), v2->point());
36         if(!min_set || min_length > length) {
37             min_length = length;
38             min_set = true;
39         }
40     }
41
42     double seconds = ceil_to_double(CGAL::sqrt(min_length)*50);
43     cout << seconds << "\n";
44 }
45
46 int main() {
47     cin.sync_with_stdio(false);
48     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
49     while(true) {
50         int n; cin >> n;
51         if(n == 0) return 0;
52         testcase(n);
53     }
54 }
```

# H1N1

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
5  #include <CGAL/Delaunay_triangulation_2.h>
6  using namespace std;
7
8  typedef CGAL::Exact_predicates_inexact_constructions_kernel    K;
9  typedef CGAL::Delaunay_triangulation_2<K>                     Delaunay;
10 typedef Delaunay::All_faces_iterator                            AFI;
11 typedef map<Delaunay::Face_handle, int>                         StateMap;
12
13 int testcase(int N) {
14     vector<K::Point_2> points;
15     for(int n = 0; n < N; ++n) {
16         double x, y; cin >> x >> y;
17         points.push_back(K::Point_2(x, y));
18     }
19
20     int M; cin >> M;
21     vector<pair<K::Point_2, double> > people;
22     for(int m = 0; m < M; ++m) {
23         double x, y, d; cin >> x >> y >> d;
24         people.push_back(make_pair(K::Point_2(x, y), d));
25     }
26
27     StateMap state;
28     Delaunay t;
29     t.insert(points.begin(), points.end());
30
31     for(int p = 0; p < M; ++p) {
32         K::Point_2 coord = people[p].first;
33         K::FT d = people[p].second;
34
35         if(CGAL::squared_distance(coord, t.nearest_vertex(coord)->point()) < d) {
36             cout << "n";
37             continue;
38         }
39
40         Delaunay::Face_handle start_face = t.locate(coord);
41         if(t.is_infinite(start_face)) {
42             cout << "y";
43             continue;
44         }
45
46         bool stop = false;
47         queue<Delaunay::Face_handle> fifo;
48         fifo.push(start_face);
49         int bfs_id = p+1;
50         state[start_face] = bfs_id;
51         while(!fifo.empty()) {
52             Delaunay::Face_handle f = fifo.front(); fifo.pop();
53             for(int e = 0; e < 3; ++e) {
54                 K::Segment_2 seg = t.segment(f, e);
55                 Delaunay::Face_handle neighbour = f->neighbor(e);
56
57                 if((seg.squared_length() >= 4*d) && state[neighbour] != bfs_id){
58                     if(t.is_infinite(neighbour)) {
59                         cout << "y";
60                         stop = true;
61                         break;
62                     }
63                     fifo.push(neighbour);
64                     state[neighbour] = bfs_id;
65                 }
66             }
67             if(stop) break;
68
69         }
70         if(!stop) cout << "n";
71     }
72     cout << "\n";
73 }
```

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



# HikingMaps

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <climits>
5  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
6  #include <CGAL/ch_jarvis.h>
7  using namespace std;
8
9  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;          // does not pass last TC with exact constructions.
10 typedef vector<int> vi;
11 typedef vector<vi> vii;
12
13 void testcase() {
14     int M, N; cin >> M >> N; // M-1 legs, N maps.
15
16     vector<pair<K::Point_2, K::Point_2> > legs;    // using a vector a segment, prevents from passing the 4th TC.
17     K::Point_2 prev;
18     cin >> prev;
19     for(int m = 1; m < M; ++m) {
20         int x, y; cin >> x >> y;
21         K::Point_2 now(x, y);
22         legs.push_back(make_pair(prev, now));
23         prev = now;
24     }
25
26     vii lists(M-1); // storing "leg contained by map" data.
27     for(int n = 0; n < N; ++n) {
28         vector<K::Point_2> points(6);
29         for(int i = 0; i < 6; ++i)
30             cin >> points[i];
31
32         vector<K::Point_2> ccw; // store the given vertices in counter-clockwise fashion.
33         CGAL::ch_jarvis_march(points.begin(), points.end(), points[0], points[0], back_inserter(ccw));
34         if(points[1] != ccw[1]) { // ugly... making sure two consecutive vertices span a triangle edge.
35             ccw.clear();
36             CGAL::ch_jarvis_march(points.begin(), points.end(), points[1], points[1], back_inserter(ccw));
37         }
38
39         for(int l = 0; l < legs.size(); ++l) { // iterate over each leg.
40             bool isOutside; // is set if to true, if origin or source is to the right to the edges.
41             for(int p = 0; p < ccw.size()-1; p = p+2) {
42                 isOutside = (CGAL::right_turn(ccw[p], ccw[p+1], legs[l].first) ||
43                             CGAL::right_turn(ccw[p], ccw[p+1], legs[l].second)) ? true : false; // if one of the leg points ↗
44                                     ↘ outside, then set to yes.
45             }
46             if(isOutside) break;
47             if(!isOutside) lists[l].push_back(n); // both end points of leg are inside.
48         }
49     }
50
51     vi pointers(M-1, 0);
52     priority_queue<int> max_heap;
53     priority_queue<pair<int, int>, std::vector<pair<int, int> >, greater<pair<int, int> > > min_heap;
54     for(int l = 0; l < lists.size(); ++l) {
55         max_heap.push(lists[l][0]);
56         min_heap.push(make_pair(lists[l][0], 1));
57     }
58
59     int min_interval = INT_MAX;
60     while(true) {
61         pair<int, int> min_pair = min_heap.top(); min_heap.pop();
62         int min_value = min_pair.first;
63         int min_list = min_pair.second;
64
65         int max_value = max_heap.top();
66         int min_new = abs(max_value - min_value);
67         min_interval = min(min_new, min_interval);
68         if(pointers[min_list] == lists[min_list].size()-1) break;
69
70         pointers[min_list]++;
71         int new_value = lists[min_list][pointers[min_list]];
72         max_heap.push(new_value);
73         min_heap.push(make_pair(new_value, min_list));
74     }
```

```
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!

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6  #include <CGAL/Gmpz.h>
7  using namespace std;
8
9  #ifdef CGAL_USE_GMP
10 #include <CGAL/Gmpz.h>
11 typedef CGAL::Gmpz ET;
12 #else
13 #include <CGAL/MP_Float.h>
14 typedef CGAL::MP_Float ET;
15 #endif
16
17 // program and solution types
18 typedef CGAL::Quadratic_program<int> Program;
19 typedef CGAL::Quadratic_program_solution<ET> Solution;
20
21 void program_1(int a, int b) {
22     Program qp (CGAL::SMALLER, true, 0, false, 0);    // use bounds instead of extra constraints.
23     const int X = 0;
24     const int Y = 1;
25
26     // minimize -b*y + a*x^2
27     qp.set_c(Y, -b);
28     qp.set_d(X, X, a*2);
29
30     // x + y <= 4
31     qp.set_a(X, 0, 1);
32     qp.set_a(Y, 0, 1);
33     qp.set_b(0, 4);
34
35     // 4x + 2y <= a*b
36     qp.set_a(X, 1, 4);
37     qp.set_a(Y, 1, 2);
38     qp.set_b(1, a*b);
39
40     // -x + y <= 1
41     qp.set_a(X, 2, -1);
42     qp.set_a(Y, 2, 1);
43     qp.set_b(2, 1);
44
45     Solution s = CGAL::solve_quadratic_program(qp, ET());
46     assert(s.solves_quadratic_program(qp));
47
48     if(s.is_optimal()) {
49         int sign;
50         (s.objective_value() <= 0) ? sign = -1 : sign = 1;
51         cout << floor(to_double(sign*s.objective_value())) << "\n";    // std::ceil?, ceil_to_double fct?
52     } else if(s.is_unbounded())
53         cout << "unbounded\n";
54     else if(s.is_infeasible())
55         cout << "no\n";
56
57 }
58
59 void program_2(int a, int b) {
60     Program qp (CGAL::SMALLER, false, 0, true, 0);
61     const int X = 0;
62     const int Y = 1;
63     const int Z = 2;
64
65     qp.set_l(Z, 0);
66     qp.set_u(Z, false);
67
68     // minimize a*x^2 + b*y + z^4
69     qp.set_d(X, X, 2*a);
70     qp.set_d(Z, Z, 2*1);    // by convention: we multiply value by 2.
71
72     qp.set_c(Y, b);
73 }
```

```

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

```
1  #include <iostream>
2  #include <vector>
3  #include <set>
4  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
5  #include <CGAL/Delaunay_triangulation_2.h>
6  #include <CGAL/Triangulation_vertex_base_with_info_2.h>
7  using namespace std;
8
9  typedef CGAL::Exact_predicates_inexact_constructions_kernel    K;
10 typedef CGAL::Delaunay_triangulation_2<K>                      D_Triangulation;
11 typedef D_Triangulation::Finite_edges_iterator                  FEI;
12 typedef set<D_Triangulation::Vertex_handle>                     vertex_set;
13
14 void testcase() {
15     int n, d; cin >> n >> d;
16
17     vector<K::Point_2> points;
18     for(int i = 0; i < n; ++i) {
19         int x, y; cin >> x >> y;
20         points.push_back(K::Point_2(x, y));
21     }
22
23     D_Triangulation t;
24     t.insert(points.begin(), points.end());
25     vertex_set in_danger;
26     for(FEI e = t.finite_edges_begin(); e != t.finite_edges_end(); ++e) {
27         D_Triangulation::Vertex_handle v1 = e->first->vertex((e->second + 1) % 3);
28         D_Triangulation::Vertex_handle v2 = e->first->vertex((e->second + 2) % 3);
29         K::FT squared_d = CGAL::squared_distance(v1->point(), v2->point());
30         double distance = CGAL::sqrt(squared_d);
31
32         if(distance < d) {
33             in_danger.insert(v1); in_danger.insert(v2);
34         }
35     }
36     cout << in_danger.size() << "\n";
37 }
38
39
40 int main() {
41     int TC; std::cin >> TC;
42     while(TC--> 0) testcase();
43 }
```

# Diet

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6  using namespace std;
7
8  #ifdef CGAL_USE_GMP
9  #include <CGAL/Gmpz.h>
10 typedef CGAL::Gmpz ET;
11 #else
12 #include <CGAL/MP_Float.h>
13 typedef CGAL::MP_Float ET;
14 #endif
15
16 typedef CGAL::Quadratic_program<int> Program;
17 typedef CGAL::Quadratic_program_solution<ET> Solution;
18
19 // N: nutrients, M: foods
20 void testcase(int N, int M) {
21     Program lp(CGAL::SMALLER, true, 0, false, 0);
22
23     for(int n = 0; n < N; ++n) {
24         int min, max; cin >> min >> max;
25         lp.set_b(n, min);
26         lp.set_r(n, CGAL::LARGER);
27         lp.set_b(N+n, max);
28     }
29
30     for(int m = 0; m < M; ++m) {
31         int p; cin >> p;
32         lp.set_c(m, p);
33
34         for(int n = 0; n < N; ++n) {
35             int amount; cin >> amount;
36             lp.set_a(m, n, amount);
37             lp.set_a(m, N+n, amount);
38         }
39     }
40
41     Solution s = CGAL::solve_linear_program(lp, ET());
42     assert (s.solves_linear_program(lp));
43
44     if(s.is_infeasible())
45         cout << "No such diet.\n";
46     else
47         cout << floor(to_double(s.objective_value())) << "\n";
48 }
49
50 int main() {
51     while(true) {
52         int N, M; cin >> N >> M;
53         if(N == 0 && M == 0) return 0;
54         testcase(N, M);
55     }
56 }
```

# Porfolios

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6  using namespace std;
7
8  #ifdef CGAL_USE_GMP
9  #include <CGAL/Gmpz.h>
10 typedef CGAL::Gmpz ET;
11 #else
12 #include <CGAL/MP_Float.h>
13 typedef CGAL::MP_Float ET;
14 #endif
15
16 typedef CGAL::Quadratic_program<int> Program;
17 typedef CGAL::Quadratic_program_solution<ET> Solution;
18
19 // N: assets, M: portfolios
20 void testcase(int N, int M) {
21     Program qp(CGAL::SMALLER, true, 0, false, 0);
22
23     for(int n = 0; n < N; ++n) {
24         int c, r; cin >> c >> r;
25         qp.set_a(n, 0, c);
26         qp.set_a(n, 1, r);
27     }
28
29     for(int i = 0; i < N; ++i) {
30         for(int j = 0; j < N; ++j) {
31             int cij; cin >> cij;
32             qp.set_d(i, j, 2*cij);
33         }
34     }
35
36     for(int m = 0; m < M; ++m) {
37         int C, R, V; cin >> C >> R >> V;
38         qp.set_b(0, C);
39         qp.set_b(1, R);
40         qp.set_r(1, CGAL::LARGER);
41
42         Solution s = CGAL::solve_quadratic_program(qp, ET());
43         assert(s.solves_quadratic_program(qp));
44
45         //cout << s;
46
47         if(s.is_optimal() && (to_double(s.objective_value()) <= V)) {
48             cout << "Yes.\n";
49         } else {
50             cout << "No.\n";
51         }
52     }
53 }
54
55 int main() {
56     while(true) {
57         int N, M; cin >> N >> M;
58         if(N == 0 && M == 0) return 0;
59         testcase(N, M);
60     }
61 }
```

# Inball

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6  using namespace std;
7
8  #ifdef CGAL_USE_GMP
9  #include <CGAL/Gmpz.h>
10 typedef CGAL::Gmpz ET;
11 #else
12 #include <CGAL/MP_Float.h>
13 typedef CGAL::MP_Float ET;
14 #endif
15
16 typedef CGAL::Quadratic_program<int> Program;
17 typedef CGAL::Quadratic_program_solution<ET> Solution;
18
19 int main() {
20     ios_base::sync_with_stdio(false);
21     int n; cin >> n;
22
23     while(n > 0) {
24         int d; cin >> d;
25         Program lp(CGAL::SMALLER, false, 0, false, 0);
26         lp.set_c(d, -1);
27         lp.set_l(d, true, 0);
28
29         for(int i = 0; i < n; ++i) {
30             int l2 = 0;
31             for(int j = 0; j < d; ++j) {
32                 int a; cin >> a;
33                 lp.set_a(j, i, a);
34                 l2 += a*a;
35             }
36             l2 = sqrt(l2);
37             lp.set_a(d, i, l2);
38
39             int b; cin >> b;
40             lp.set_b(i, b);
41         }
42
43         Solution s = CGAL::solve_linear_program(lp, ET());
44         if(s.is_infeasible()) {
45             cout << "none\n";
46         } else if(s.is_unbounded()) {
47             cout << "inf\n";
48         } else {
49             cout << floor(-CGAL::to_double(s.objective_value())) << "\n";
50         }
51
52         cin >> n;
53     }
54 }
```



# Monkey Island

```
1  #include <vector>
2  #include <iostream>
3  #include <climits>
4  #include <boost/graph/strong_components.hpp>
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/tuple/tuple.hpp>
7  using namespace std;
8  using namespace boost;
9
10 typedef vector<int> vi;
11 typedef adjacency_list<vecS, vecS, directedS, no_property, no_property> Graph;
12 typedef graph_traits<Graph>::edge_descriptor Edge;
13 typedef graph_traits<Graph>::edge_iterator EdgeIterator;
14
15 void testcase() {
16     int N, M; cin >> N >> M;
17
18     Graph g(N);
19     for(int e = 0; e < M; ++e) {
20         int v1, v2;
21         cin >> v1 >> v2;
22         add_edge(v1-1, v2-1, g);
23     }
24
25     vi costs(N);
26     for(int n = 0; n < N; ++n) {
27         int cost; cin >> cost;
28         costs[n] = cost;
29     }
30
31     vector<int> scc(N);
32     int nsc = strong_components(g, &scc[0]);
33
34     vi incoming_comp(nsc, 0);
35     EdgeIterator ebeg, eend;
36     for(tie(ebeg, eend) = edges(g); ebeg != eend; ++ebeg) {
37         int u = source(*ebeg, g);
38         int v = target(*ebeg, g);
39         if(scc[u] != scc[v]) incoming_comp[scc[v]] = 1;
40     }
41
42     int total = 0;
43     for(int comp = 0; comp < nsc; ++comp) {
44         if(incoming_comp[comp] == 1) continue;
45         int min_cost = INT_MAX;
46         for(int v = 0; v < N; ++v) {
47             if(scc[v] == comp) min_cost = min(min_cost, costs[v]);
48         }
49         total += min_cost;
50     }
51
52     cout << total << "\n";
53 }
54
55 int main() {
56     int TC; cin >> TC;
57     while(TC--) testcase();
58     return 0;
59 }
```

# Placing Knights

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/max_cardinality_matching.hpp>
6  using namespace std;
7  using namespace boost;
8
9  typedef vector<int> vi;
10 typedef vector<vi> vii;
11 typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;
12 typedef graph_traits<Graph>::vertex_descriptor Vertex;
13
14 int N;
15
16 int co_to_index(int i, int j) {
17     return i*N + j;
18 }
19
20 void add_valid_edges(int i, int j, vii& holes, Graph& g) {
21     int y = 1;
22     for(int x = -2; x <= 2; x = x + 4) {
23         if(i+y >= 0 && i+y < N && j+x >= 0 && j+x <= N && holes[i+y][j+x] == 1) {
24             add_edge(co_to_index(i, j), co_to_index(i+y, j+x), g);
25         }
26     }
27     y = 2;
28     for(int x = -1; x <= 1; x = x + 2) {
29         if(i+y >= 0 && i+y < N && j+x >= 0 && j+x <= N && holes[i+y][j+x] == 1) {
30             add_edge(co_to_index(i, j), co_to_index(i+y, j+x), g);
31         }
32     }
33 }
34
35 void testcase() {
36     cin >> N;
37     Graph g(N*N);
38     vii holes(N, vi(N));
39     int sum_holes = 0;
40
41     for(int i = 0; i < N; ++i) {
42         for(int j = 0; j < N; ++j) {
43             int hole; cin >> hole;
44             holes[i][j] = hole;
45             if(holes[i][j] == 0) ++sum_holes;
46         }
47     }
48
49     for(int i = 0; i < N-1; ++i) {
50         for(int j = 0; j < N; ++j) {
51             if(holes[i][j] == 1) add_valid_edges(i, j, holes, g);
52         }
53     }
54
55     vector<Vertex> mateMap(num_vertices(g), 0);
56     checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
57     // mistake: forgot to subtract the holes.
58     cout << num_vertices(g)- sum_holes - matching_size(g, &mateMap[0]) << "\n";
59 }
60
61 int main() {
62     int TC; cin >> TC;
63     while(TC-->0) testcase();
64     return 0;
65 }
```

# Shopping Trip

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/push_relabel_max_flow.hpp>
6  using namespace std;
7  using namespace boost;
8
9  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
10 typedef adjacency_list<vecS, vecS, directedS, no_property,
11     property<edge_capacity_t, long,
12     property<edge_residual_capacity_t, long,
13     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
14 typedef property_map<Graph, edge_capacity_t::type EdgeCapacityMap;
15 typedef property_map<Graph, edge_residual_capacity_t::type ResidualCapacityMap;
16 typedef property_map<Graph, edge_reverse_t::type ReverseEdgeMap;
17 typedef graph_traits<Graph>::edge_descriptor Edge;
18
19 void testcase() {
20     int n, m, s; cin >> n >> m >> s;
21     Graph g(n);
22     EdgeCapacityMap capacity = get(edge_capacity, g);
23     ReverseEdgeMap rev_edge = get(edge_reverse, g);
24     ResidualCapacityMap res_capacity = get(edge_residual_capacity, g);
25
26     for(int store = 0; store < s; ++store){
27         int store_vertex; cin >> store_vertex;
28         Edge edge;
29         tie(edge, tuples::ignore) = add_edge(store_vertex, n, g);
30         Edge reverse_edge;
31         tie(reverse_edge, tuples::ignore) = add_edge(n, store_vertex, g);
32         capacity[edge] = 1;
33         rev_edge[edge] = reverse_edge;
34         capacity[reverse_edge] = 0;
35         rev_edge[reverse_edge] = edge;
36     }
37
38     for(int e = 0; e < m; ++e) {
39         int v1, v2; cin >> v1 >> v2;
40         Edge edge;
41         tie(edge, tuples::ignore) = add_edge(v1, v2, g);
42         Edge reverse_edge;
43         tie(reverse_edge, tuples::ignore) = add_edge(v2, v1, g);
44         capacity[edge] = 1;
45         rev_edge[edge] = reverse_edge;
46         capacity[reverse_edge] = 0;
47         rev_edge[reverse_edge] = edge;
48         Edge edge2;
49         tie(edge2, tuples::ignore) = add_edge(v2, v1, g);
50         Edge reverse_edge2;
51         tie(reverse_edge2, tuples::ignore) = add_edge(v1, v2, g);
52         capacity[edge2] = 1;
53         rev_edge[edge2] = reverse_edge2;
54         capacity[reverse_edge2] = 0;
55         rev_edge[reverse_edge2] = edge2;
56     }
57
58     long max_flow = push_relabel_max_flow(g, 0, n);
59     if(max_flow == s) cout << "yes\n"; else cout << "no\n";
60 }
61
62 int main() {
63     int TC; cin >> TC;
64     while(TC--) testcase();
65     return 0;
66 }
```

# TheeV

```
1  #include <iostream>
2  #include <vector>
3  #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
4  #include <CGAL/Min_circle_2.h>
5  #include <CGAL/Min_circle_2_traits_2.h>
6  using namespace std;
7
8  typedef CGAL::Exact_predicates_exact_constructions_kernel K;
9  typedef CGAL::Min_circle_2_traits_2<K> MinCircleTraits;
10 typedef CGAL::Min_circle_2<MinCircleTraits> Min_circle;
11 typedef vector<pair<K::FT, K::Point_2> > dp;
12
13 bool pairCompare(const pair<K::FT, K::Point_2>& lhs, const pair<K::FT, K::Point_2>& rhs) {
14     return lhs.first > rhs.first;
15 }
16
17 double ceil_to_double(const K::FT& x) {
18     double a = std::ceil(CGAL::to_double(x));
19     while (a < x) a += 1;
20     while (a >= x+1) a -= 1;
21     return a;
22 }
23
24 void testcase() {
25     int N; cin >> N;
26     dp cities;
27
28     int x, y; cin >> x >> y;
29     K::Point_2 capitol(x, y);
30     cities.push_back(make_pair(0, capitol));
31
32     for(int n = 1; n < N; ++n) {
33         int x, y; cin >> x >> y;
34         K::Point_2 p(x, y);
35         K::FT dist = CGAL::squared_distance(capitol, p);
36         cities.push_back(make_pair(dist, p));
37     }
38     sort(cities.begin(), cities.end(), pairCompare);
39
40     int i = 0;
41     K::FT r1 = cities[0].first, r2 = 0;
42     K::FT t = r1;
43     Min_circle mc;
44     while(r1 > r2 && i < N-1) {
45         r1 = cities[i+1].first;
46
47         //cout << "insert in mincircle: " << cities[i].second << "\n";
48         mc.insert(cities[i].second);
49         MinCircleTraits::Circle c = mc.circle();
50         r2 = c.squared_radius();
51         //cout << "r1: " << r1 << "\n" << "r2: " << r2 << "\n";
52         //cout << "diff: " << abs(r1 - r2) << " r1: " << r1 << " r2:" << r2 << "\n";
53         ++i;
54     }
55
56     if(r1 == r2)
57         t = r1;
58     if(r2 > r1)
59         t = min(r2, cities[i-1].first);
60
61     cout << ceil_to_double(t) << "\n";
62 }
63
64 int main() {
65     cin.sync_with_stdio(false);
66     cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
67     int TC; cin >> TC;
68     while(TC--) testcase();
69 }
```

# Poker Chips

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <cmath>
5  using namespace std;
6
7  typedef vector<int> vi;
8  typedef vector<vi> vii;
9  typedef map<vector<int>, int> vector_int;
10
11 vi M;
12 int N;
13 vector_int dp_table;
14 vii chips;
15
16 int find_max(vi& state) {
17     if(dp_table.count(state) == 1)
18         return dp_table[state];
19
20     for(int n = 1; n < pow(2.0, N); ++n) {
21         vi new_state = state;
22         int T = 0;
23         int prev = -1;
24
25         for(int k = 0; k < N; ++k) {
26             if((n & (1 << k)) && (state[k] != 0)) {
27                 int color = chips[k][state[k]-1];
28                 if(prev == color || prev == -1) {
29                     --new_state[k];
30                     prev = color;
31                     ++T;
32                 } else {
33                     T = 0; // !important to avoids wasted loops and computing invalid states.
34                     break;
35                 }
36             }
37         }
38
39         if(T != 0) { // if T=0, then invalid subset.
40             int K = (T <= 1) ? 0 : pow(2.0, T-2);
41             dp_table[state] = max(find_max(new_state) + K, dp_table[state]);
42         }
43     }
44
45     return dp_table[state];
46 }
47
48 void testcase() {
49     cin >> N;
50     M = vi(N);
51     for(int n = 0; n < N; ++n)
52         cin >> M[n];
53
54     chips = vii(N);
55     for(int n = 0; n < N; ++n) {
56         for(int m = 0; m < M[n]; ++m) {
57             int col; cin >> col;
58             chips[n].push_back(col);
59         }
60     }
61
62     dp_table = vector_int();
63     cout << find_max(M) << "\n";
64 }
65
66 int main() {
67     ios_base::sync_with_stdio(false);
68     int TC; cin >> TC;
69     while(TC--) testcase();
70     return 0;
71 }
```

# Portfolio Revisited

```
1  #include <iostream>
2  #include <cassert>
3  #include <CGAL/basic.h>
4  #include <CGAL/QP_models.h>
5  #include <CGAL/QP_functions.h>
6  using namespace std;
7
8  #ifdef CGAL_USE_GMP
9  #include <CGAL/Gmpz.h>
10 typedef CGAL::Gmpz ET;
11 #else
12 #include <CGAL/MP_Float.h>
13 typedef CGAL::MP_Float ET;
14 #endif
15
16 typedef CGAL::Quadratic_program<int> Program;
17 typedef CGAL::Quadratic_program_solution<ET> Solution;
18
19 void testcase(int N, int M) {
20     Program qp (CGAL::SMALLER, true, 0, false, 0);
21
22     for(int n = 0; n < N; ++n) {
23         int c, r; cin >> c >> r;
24         qp.set_a(n, 0, c);
25         qp.set_a(n, 1, r);
26     }
27     qp.set_r(1, CGAL::LARGER);
28
29     for(int i = 0; i < N; ++i) {
30         for(int j = 0; j < N; ++j) {
31             int vij; cin >> vij;
32             qp.set_d(i, j, 2*vij);
33         }
34     }
35
36     for(int m = 0; m < M; ++m) {
37         int C, V; cin >> C >> V;
38         int R = 0;
39         qp.set_b(0, C);
40         qp.set_b(1, R);
41
42         int lo = 0;
43         int hi = 100;
44         bool fixed = false;
45         while(lo <= hi) {
46             int mid = (fixed) ? (lo + (hi-lo+1)/2) : hi;
47
48             qp.set_b(1, mid);
49             Solution s = CGAL::solve_quadratic_program(qp, ET());
50             assert(s.solves_quadratic_program(qp));
51
52             if(s.is_optimal() && s.objective_value() <= V) {
53                 R = mid;
54                 if(!fixed) {
55                     lo = hi+1;
56                     hi = 2*hi;
57                 } else {
58                     lo = mid+1;
59                 }
60             } else {
61                 fixed = true;
62                 hi = mid-1;
63             }
64         }
65         cout << R << "\n";
66     }
67 }
68
69 int main() {
70     while(true) {
71         int N, M; cin >> N >> M;
72         if(N == 0 && M == 0) return 0;
73         testcase(N, M);
74     }
```

74 }  
75 }

# Stamp Exhibition

```
1  #include <iostream>
2  #include <cassert>
3  #include <cmath>
4  #include <CGAL/basic.h>
5  #include <CGAL/QP_models.h>
6  #include <CGAL/QP_functions.h>
7  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
8  using namespace std;
9
10 #ifdef CGAL_USE_GMP
11 #include <CGAL/Gmpq.h>
12 typedef CGAL::Gmpq ET;
13 #else
14 #include <CGAL/MP_Float.h>
15 typedef CGAL::MP_Float ET;
16 #endif
17
18 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
19 typedef CGAL::Quadratic_program<double> Program;
20 typedef CGAL::Quadratic_program_solution<ET> Solution;
21
22 void testcase() {
23     int L, S, W; cin >> L >> S >> W;
24
25     vector<K::Point_2> lamps;
26     for(int l = 0; l < L; ++l) {
27         int x, y; cin >> x >> y;
28         lamps.push_back(K::Point_2(x, y));
29     }
30
31     vector<pair<K::Point_2, double> > stamps;
32     for(int s = 0; s < S; ++s) {
33         int x, y; double m; cin >> x >> y >> m;
34         stamps.push_back(make_pair(K::Point_2(x, y), m));
35     }
36
37     vector<K::Segment_2> walls;
38     for(int w = 0; w < W; ++w) {
39         int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
40         walls.push_back(K::Segment_2(K::Point_2(x1, y1), K::Point_2(x2, y2)));
41     }
42
43     if(S == 0) { cout << "yes\n"; return; }
44     if(L == 0) { cout << "no\n"; return; }
45
46     Program lp (CGAL::SMALLER, true, 1, true, pow(2.0, 12));
47     for(int l = 0; l < L; ++l) {
48         for(int s = 0; s < S; ++s) {
49             bool intersect = false;
50             for(int w = 0; w < W; ++w) {
51                 K::Segment_2 stamp_lamp(stamps[s].first, lamps[l]);
52                 if(CGAL::do_intersect(stamp_lamp, walls[w])) {
53                     intersect = true;
54                     break;
55                 }
56             }
57
58             double param = 0;
59             if(!intersect)
60                 param = 1.0/CGAL::squared_distance(stamps[s].first, lamps[l]);
61             lp.set_a(l, s, param);
62             lp.set_a(l, S+s, param);
63             lp.set_b(s, stamps[s].second);
64             lp.set_b(S+s, 1.0);
65             lp.set_r(S+s, CGAL::LARGER);
66         }
67     }
68
69     Solution s = CGAL::solve_linear_program(lp, ET());
70     assert (s.solves_linear_program(lp));
71     (!s.is_infeasible()) ? cout << "yes\n" : cout << "no\n";
72 }
73
```



```
74  int main() {  
75      int TC; cin >> TC;  
76      while(TC--) testcase();  
77      return 0;  
78  }
```

# Tetris

```
1  #include <iostream>
2  #include <boost/graph/adjacency_list.hpp>
3  #include <boost/graph/push_relabel_max_flow.hpp>
4  #include <boost/tuple/tuple.hpp>
5  using namespace std;
6  using namespace boost;
7
8  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
9  typedef adjacency_list<vecS, vecS, directedS, no_property,
10     property<edge_capacity_t, long,
11     property<edge_residual_capacity_t, long,
12     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
13  typedef property_map<Graph, edge_capacity_t::type> EdgeCapacityMap;
14  typedef property_map<Graph, edge_reverse_t::type> ReverseEdgeMap;
15  typedef graph_traits<Graph>::edge_descriptor Edge;
16
17  void add_edge(int from, int to, int cap, Graph& g) {
18     //cout << "adding edge: " << from << " " << to << " " << cap << "\n";
19     EdgeCapacityMap capacity = get(edge_capacity, g);
20     ReverseEdgeMap reverse = get(edge_reverse, g);
21
22     Edge there, back;
23     tie(there, tuples::ignore) = add_edge(from, to, g);
24     tie(back, tuples::ignore) = add_edge(to, from, g);
25     capacity[there] = cap;
26     capacity[back] = 0;
27     reverse[there] = back;
28     reverse[back] = there;
29 }
30
31 void testcase() {
32     int W, N; cin >> W >> N;
33
34     int source = 0;
35     int sink = W;
36     Graph g(2*W);
37
38     for(int v = 1; v < W; ++v) {
39         add_edge(v, W+v, 1, g);
40     }
41
42     for(int n = 0; n < N; ++n) {
43         int v1, v2; cin >> v1 >> v2;
44         int from = (min(v1, v2) == 0) ? 0 : min(v1, v2) + W;
45         int to = max(v1, v2);
46         add_edge(from, to, 1, g);
47     }
48
49     int maxflow = push_relabel_max_flow(g, source, sink);
50     cout << maxflow << "\n";
51 }
52
53 int main() {
54     int TC; cin >> TC;
55     while(TC--) testcase();
56 }
```

# Beach Bar

```
1  #include <vector>
2  #include <iostream>
3  #include <climits>
4  #include <algorithm>
5  using namespace std;
6
7  typedef vector<int> vi;
8  const int normalize = 1000000;
9
10 void testcase() {
11     int N; cin >> N;
12     vi points;
13     for(int n = 0; n < N; ++n) {
14         int x; cin >> x;
15         points.push_back(x + normalize);
16     }
17     sort(points.begin(), points.end());
18
19     int g_counter = INT_MIN;
20     int g_length = INT_MIN;
21     vi solution;
22     for(int n = 0; n < N; ++n) {
23         int start_interval = points[n];
24         int end_interval = start_interval + 200;
25         int k = n;
26         int counter = 0;
27         while(points[k] <= end_interval && k < N) {
28             ++counter;
29             ++k;
30         }
31         int length = (points[k-1] - start_interval);
32
33         if(counter > g_counter || (counter == g_counter && length < g_length)) {
34             g_counter = counter;
35             g_length = length;
36             solution.clear();
37         }
38
39         if(g_counter == counter && g_length == length) {
40             int output = start_interval + length/2 - normalize;
41             solution.push_back(output);
42             if(length % 2 != 0) {
43                 solution.push_back(output+1);
44             }
45         }
46     }
47
48     g_length = (g_length % 2 == 0) ? g_length/2 : g_length/2+1;
49     cout << g_counter << "\n" << g_length << "\n";
50     for(int s = 0; s < solution.size(); ++s) {
51         cout << solution[s];
52         if(s != solution.size() - 1) cout << "\n";
53     }
54     cout << "\n";
55 }
56
57 int main() {
58     int TC; cin >> TC;
59     while(TC--) testcase();
60     return 0;
61 }
```

# Cover

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
5  #include <CGAL/Delaunay_triangulation_2.h>
6  using namespace std;
7
8  typedef CGAL::Exact_predicates_exact_constructions_kernel      K;
9  typedef CGAL::Delaunay_triangulation_2<K>                     Delaunay;
10 typedef Delaunay::Finite_faces_iterator                        FFI;
11 typedef Delaunay::Finite_edges_iterator                       FEI;
12
13 double ceil_to_double(const K::FT& x) {
14     double a = ceil(CGAL::to_double(x));
15     while (a < x) a += 1;
16     while (a-1 >= x) a -= 1;
17     return a;
18 }
19
20 template<typename T>
21 K::FT check_intersection(const T* obj, const K::Point_2 p1, const vector<K::Segment_2>& rectangle) {
22     for (int i = 0; i < 4; ++i) {
23         if(!do_intersect(rectangle[i], *obj)) continue;
24         CGAL::Object o = intersection(rectangle[i], *obj);
25         const K::Point_2* p2 = CGAL::object_cast<K::Point_2>(&o);
26         K::FT sqrd = CGAL::squared_distance(p1, *p2);
27         return sqrd;
28     }
29     return 0;
30 }
31
32 void testcase(int N) {
33     vector<K::Point_2> points;
34     vector<K::Segment_2> rectangle;
35
36     double x1, y1, x2, y2;
37     cin >> x1 >> y1 >> x2 >> y2;
38     K::Point_2 sw(x1, y1);
39     K::Point_2 nw(x1, y2);
40     K::Point_2 se(x2, y1);
41     K::Point_2 ne(x2, y2);
42     rectangle.push_back(K::Segment_2(sw, nw));
43     rectangle.push_back(K::Segment_2(se, ne));
44     rectangle.push_back(K::Segment_2(sw, se));
45     rectangle.push_back(K::Segment_2(nw, ne));
46
47     for(int n = 0; n < N; ++n) {
48         double x, y; cin >> x >> y;
49         points.push_back(K::Point_2(x, y));
50     }
51
52     // O(n log n)
53     Delaunay t;
54     t.insert(points.begin(), points.end());
55     K::FT min_rad;
56
57     // check corners
58     min_rad = CGAL::squared_distance(sw, t.nearest_vertex(sw)->point());
59     min_rad = max(min_rad, CGAL::squared_distance(se, t.nearest_vertex(se)->point()));
60     min_rad = max(min_rad, CGAL::squared_distance(nw, t.nearest_vertex(nw)->point()));
61     min_rad = max(min_rad, CGAL::squared_distance(ne, t.nearest_vertex(ne)->point()));
62
63     // iterate over all faces to find largest circle - O(N)
64     for(FFI f = t.finite_faces_begin(); f != t.finite_faces_end(); ++f) {
65         K::Point_2 cc = t.circumcenter(f);
66         if(cc.x() >= x1 && cc.x() <= x2 && cc.y() >= y1 && cc.y() <= y2) {
67             K::Point_2 point = f->vertex(1)->point();
68             K::FT dist = CGAL::squared_distance(point, cc);
69             min_rad = max(min_rad, dist);
70         }
71     }
72
73     // 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

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int ancestor(int v) {
6      for(int k = 2; k <= ceil(sqrt(v)); ++k) {
7          if(v % k == 0) return (v/k);
8      }
9      return 1;
10 }
11
12 void testcase() {
13     int N, C; cin >> N >> C;
14     for(int c = 0; c < C; ++c) {
15         int v1, v2; cin >> v1 >> v2;
16         int counter = 0;
17         while(v1 != v2) {
18             ++counter;
19             if(v1 < v2) {
20                 v2 = ancestor(v2);
21             } else {
22                 v1 = ancestor(v1);
23             }
24         }
25         cout << counter << "\n";
26     }
27 }
28
29 int main() {
30     ios_base::sync_with_stdio(false);
31     int TC; cin >> TC;
32     while(TC--) testcase();
33     return 0;
34 }
```

# Tiles

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/max_cardinality_matching.hpp>
6  using namespace std;
7  using namespace boost;
8
9  typedef vector<int> vi;
10 typedef vector<vi> vii;
11 typedef adjacency_list<vecS, vecS, undirectedS, no_property, no_property> Graph;
12 typedef graph_traits<Graph>::vertex_descriptor Vertex;
13
14 void testcase() {
15     int W, H; cin >> W >> H;
16
17     vii matrix(H);
18     int blocked = 0;
19     int vcounter = 0;
20     for(int h = 0; h < H; ++h) {
21         for(int w = 0; w < W; ++w) {
22             char input; cin >> input;
23             blocked += (input == 'x');
24             matrix[h].push_back((input == '.' ? vcounter++ : -1));
25         }
26     }
27
28     int V = (W*H - blocked);
29     if(V % 2 == 1) {
30         cout << "no\n";
31         return;
32     }
33
34     Graph g(V);
35     for(int h = 0; h < H; ++h) {
36         for(int w = 0; w < W; ++w) {
37             if(matrix[h][w] == -1) continue;
38             if(w+1 < W && matrix[h][w+1] != -1) add_edge(matrix[h][w], matrix[h][w+1], g);
39             if(h+1 < H && matrix[h+1][w] != -1) add_edge(matrix[h][w], matrix[h+1][w], g);
40         }
41     }
42
43     vector<Vertex> mateMap(num_vertices(g), 0);
44     checked_edmonds_maximum_cardinality_matching(g, &mateMap[0]);
45     int matching = matching_size(g, &mateMap[0]);
46
47     if(matching * 2 == V) cout << "yes\n";
48     else cout << "no\n";
49 }
50
51 int main() {
52     int TC; cin >> TC;
53     while(TC--) testcase();
54     return 0;
55 }
```

# Deleted Entries Strike Back

Missing.



# Light The Stage

Missing.

## Radiation

Missing.

# Sweepers

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/tuple/tuple.hpp>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/push_relabel_max_flow.hpp>
6  #include <boost/graph/strong_components.hpp>
7  using namespace std;
8  using namespace boost;
9
10 typedef vector<int> vi;
11 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
12 typedef adjacency_list<vecS, vecS, directedS, no_property,
13     property<edge_capacity_t, long,
14     property<edge_residual_capacity_t, long,
15     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
16 typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
17 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
18 typedef graph_traits<Graph>::edge_descriptor Edge;
19 typedef graph_traits<Graph>::vertex_descriptor Vertex;
20
21 int N, M, S;
22
23 void add_edge(int from, int to, int cap, Graph& g) {
24     EdgeCapacityMap capacity = get(edge_capacity, g);
25     ReverseEdgeMap reverse = get(edge_reverse, g);
26
27     Edge there, back;
28     tie(there, tuples::ignore) = add_edge(from, to, g);
29     tie(back, tuples::ignore) = add_edge(to, from, g);
30     capacity[there] = cap;
31     capacity[back] = 0;
32     reverse[there] = back;
33     reverse[back] = there;
34 }
35
36 void testcase() {
37     cin >> N >> M >> S;
38     int source = N;
39     int sink = N+1;
40     Graph g(N+2);
41     vi starts(N, 0), exits(N, 0);
42
43     for(int s = 0; s < S; ++s) {
44         int room; cin >> room;
45         ++starts[room];
46     }
47
48     for(int s = 0; s < S; ++s) {
49         int room; cin >> room;
50         ++exits[room];
51     }
52
53     for(int m = 0; m < M; ++m) {
54         int v1, v2; cin >> v1 >> v2;
55         add_edge(v1, v2, 1, g);
56         add_edge(v2, v1, 1, g);
57     }
58
59     for(int n = 0; n < N; ++n) {
60         if(starts[n] > 0) add_edge(source, n, starts[n], g);
61         if(exits[n] > 0) add_edge(n, sink, exits[n], g);
62     }
63
64     bool isEulerian = true;
65     bool isConnected = false;
66     graph_traits<Graph>::vertex_iterator viter, vend;
67     for (tie(viter, vend) = vertices(g); viter != vend; ++viter) {
68         if(*viter == source || *viter == sink) continue;
69         int count = out_degree(*viter, g);
70         if(starts[*viter] > 0) ++count;
71         if(exits[*viter] > 0) ++count;
72         count = count/2;
73         if(count % 2 == 1) {
```

```

74         isEulerian = false;
75         break;
76     }
77 }
78
79 if(!isEulerian) {
80     cout << "no\n";
81     return;
82 }
83
84 int maxflow = push_relabel_max_flow(g, source, sink);
85 if(maxflow != S)
86     cout << "no\n";
87 else
88     cout << "yes\n";
89 }
90
91 int main() {
92     int TC; cin >> TC;
93     while(TC--) testcase();
94 }

```

# The Bracelet

```
1  #include <iostream>
2  #include <stack>
3  #include <set>
4  #include <boost/tuple/tuple.hpp>
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/connected_components.hpp>
7  using namespace std;
8  using namespace boost;
9
10 typedef vector<pair<int, int> > vpi;
11 typedef adjacency_list<vecS, vecS, undirectedS, no_property, property<edge_weight_t, int> > Graph;
12 typedef graph_traits<Graph>::vertex_iterator VI;
13 typedef graph_traits<Graph>::out_edge_iterator EI;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;
16
17 void printEulerGraph(int v, Graph& g) {
18     WeightMap weight = get(edge_weight, g);
19     stack<int> fifo;
20     fifo.push(v);
21     vector<int> sol;
22     while(!fifo.empty()) {
23         int v = fifo.top();
24         EI ebegin, eend;
25         bool hasFreeEdge = false;
26         for(tie(ebegin, eend) = out_edges(v, g); ebegin != eend; ++ebegin) {
27             if(weight[*ebegin] == 0) {
28                 hasFreeEdge = true;
29                 weight[*ebegin] = 1;
30                 fifo.push(boost::target(*ebegin, g));
31                 break;
32             }
33         }
34         if(!hasFreeEdge) {
35             sol.push_back(v);
36             fifo.pop();
37         }
38     }
39     for(int s = 0; s < sol.size()-1; ++s) {
40         cout << sol[s] << "□" << sol[s+1] << "\n";
41     }
42     cout << "\n";
43 }
44
45 void testcase(int TC) {
46     cout << "Case_□#" << ++TC << "\n";
47     int N; cin >> N;
48
49     Graph g(50);
50     WeightMap weight = get(edge_weight, g);
51     set<int> colors;
52
53     for(int n = 0; n < N; ++n) {
54         int v1, v2; cin >> v1 >> v2;
55         colors.insert(v1); colors.insert(v2);
56         Edge e;
57         tie(e, tuples::ignore) = add_edge(v1, v2, g);
58         weight[e] = 0;
59     }
60
61     vector<int> component(num_vertices(g));
62     int num = connected_components(g, &component[0]) - (51 - colors.size());
63     int start = -1;
64     VI vbegin, vend;
65     for(tie(vbegin, vend) = vertices(g); vbegin != vend; ++vbegin) {
66         int deg = out_degree(*vbegin, g);
67         if(deg % 2 == 1 || num > 1) {
68             cout << "some_□beads_□may_□be_□lost\n\n";
69             return;
70         }
71         if(deg > 0) start = *vbegin;
72     }
73 }
```

```
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

```
1  #include <iostream>
2  #include <boost/tuple/tuple.hpp>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/push_relabel_max_flow.hpp>
5  using namespace std;
6  using namespace boost;
7
8  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
9  typedef adjacency_list<vecS, vecS, directedS, no_property,
10     property<edge_capacity_t, long,
11     property<edge_residual_capacity_t, long,
12     property<edge_reverse_t, Traits::edge_descriptor> > > > Graph;
13  typedef property_map<Graph, edge_capacity_t::type EdgeCapacityMap;
14  typedef property_map<Graph, edge_reverse_t::type ReverseEdgeMap;
15  typedef graph_traits<Graph>::edge_descriptor Edge;
16
17  int M;
18  int N;
19  int K;
20
21  int index(int x, int y) {
22     return y*M + x;
23 }
24
25  void add_edges(int from, int to, Graph& g) {
26     EdgeCapacityMap capacity = get(edge_capacity, g);
27     ReverseEdgeMap reverse = get(edge_reverse, g);
28
29     Edge there, back;
30     tie(there, tuples::ignore) = add_edge(from, to, g);
31     tie(back, tuples::ignore) = add_edge(to, from, g);
32     capacity[there] = 1;
33     capacity[back] = 0;
34     reverse[there] = back;
35     reverse[back] = there;
36 }
37
38  void testcase() {
39     cin >> M >> N >> K;    // M: cols, N: rows, K: #knights
40
41     int graph_size = 2*(M*N)+2;    // M*N for each coordinate, 2*(M*N) because we need vertex-disjoint paths only.
42     Graph g(graph_size);
43     int source = graph_size-2;
44     int sink = graph_size-1;
45     for(int y = 0; y < N; ++y) {
46         for(int x = 0; x < M; ++x) {
47             int v_in = index(x, y);
48             int v_out = index(x, y) + M*N;
49
50             add_edges(v_in, v_out, g);
51
52             if(x+1 < M) {
53                 add_edges(v_out, index(x+1, y), g);
54                 add_edges(index(x+1, y)+M*N, v_in, g);
55             }
56             if(y+1 < N) {
57                 add_edges(v_out, index(x, y+1), g);
58                 add_edges(index(x, y+1)+M*N, v_in, g);
59             }
60             if(x-1 < 0 || x+1 >= M || y-1 < 0 || y+1 >= N) {
61                 add_edges(v_out, sink, g);
62             }
63         }
64     }
65
66     for(int k = 0; k < K; ++k) {
67         int x, y; cin >> x >> y;
68         add_edges(source, index(x, y), g);
69     }
70
71     int maxflow = push_relabel_max_flow(g, source, sink);
72     cout << maxflow << "\n";
73 }
```

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



## Next Path

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/tuple/tuple.hpp>
6  #include <boost/graph/dijkstra_shortest_paths.hpp>
7  using namespace std;
8  using namespace boost;
9
10 const int MAX_LENGTH = 100000000;    // do not pick INT_MAX otherwise overflow resulting in -INT_MAX confusing min.
11
12 typedef vector<int> vi;
13 typedef adjacency_list<vecS, vecS, directedS, no_property, property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef graph_traits<Graph>::vertex_descriptor Vertex;
16 typedef property_map<Graph, edge_weight_t>::type WeightMap;
17 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIterator;
18
19 int BFS(int start, int end, Graph& g) {
20     if(start == end) return 0;
21     vi distances(num_vertices(g), -1);
22     std::queue<int> fifo;
23     fifo.push(start);
24     distances[start] = 0;
25     while(!fifo.empty()) {
26         int v = fifo.front(); fifo.pop();
27         OutEdgeIterator ebegin, eend;
28         for(tie(ebegin, eend) = out_edges(v, g); ebegin != eend; ++ebegin) {
29             int u = target(*ebegin, g);
30             if(distances[u] == -1) {
31                 distances[u] = distances[v] + 1;
32                 fifo.push(u);
33                 if(u == end) return distances[u];
34             }
35         }
36     }
37     return MAX_LENGTH;
38 }
39
40 void testcase() {
41     int N, M, s, t; cin >> N >> M >> s >> t;
42     --t; --s;
43
44     Graph g(N);
45     WeightMap weights = get(edge_weight, g);
46
47     for(int m = 0; m < M; ++m) {
48         int v1, v2; cin >> v1 >> v2;
49         Edge edge;
50         tie(edge, tuples::ignore) = add_edge(v1-1, v2-1, g);
51         weights[edge] = 1;
52     }
53
54     vi d(N);
55     vector<Vertex> p(N);
56     dijkstra_shortest_paths(g, s, predecessor_map(&p[0]).distance_map(&d[0]));
57
58     if(d[t] == INT_MAX) { cout << "no\n"; return; }    // there is no path from s to t.
59
60     int sp = MAX_LENGTH;
61     int b = t;
62     int prev = t;
63     while(true) {
64         OutEdgeIterator ebegin, eend;
65         for(tie(ebegin, eend) = out_edges(b, g); ebegin != eend; ++ebegin) {
66             if(target(*ebegin, g) == prev && prev != s && b != t) continue; // do not pick the edge in P, start end ↯
67                                     ↵ end path are special states.
68             sp = min(sp, d[source(*ebegin, g)] + 1 + BFS(target(*ebegin, g), t, g));
69         }
70         if(b == s || sp == d[t]) break;
71         prev = b;
72         b = p[b];
73     }
```

```
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

```
1  #include <iostream>
2  #include <vector>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/dijkstra_shortest_paths.hpp>
5  #include <boost/tuple/tuple.hpp>
6  #include <climits>
7  using namespace std;
8  using namespace boost;
9
10 typedef adjacency_list<vecS, vecS, directedS, no_property, property<edge_weight_t, int> > Graph;
11 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
12 typedef graph_traits<Graph>::edge_descriptor Edge;
13 typedef graph_traits<Graph>::vertex_descriptor Vertex;
14
15 void add_edges(Graph& g, int u, int v, int w) {
16     int uee = u*4;    int vee = v*4;
17     int ueo = uee+1;   int veo = vee+1;
18     int uoe = uee+2;   int voe = vee+2;
19     int uoo = uee+3;   int voo = vee+3;
20
21     EdgeWeightMap weights = get(edge_weight, g);
22
23     Edge edge;
24     if(w % 2 == 0) {
25         tie(edge, tuples::ignore) = add_edge(uee, voe, g); weights[edge] = w;
26         tie(edge, tuples::ignore) = add_edge(ueo, voo, g); weights[edge] = w;
27         tie(edge, tuples::ignore) = add_edge(uoe, vee, g); weights[edge] = w;
28         tie(edge, tuples::ignore) = add_edge(uoo, veo, g); weights[edge] = w;
29     } else {
30         tie(edge, tuples::ignore) = add_edge(uee, voo, g); weights[edge] = w;
31         tie(edge, tuples::ignore) = add_edge(ueo, voe, g); weights[edge] = w;
32         tie(edge, tuples::ignore) = add_edge(uoe, veo, g); weights[edge] = w;
33         tie(edge, tuples::ignore) = add_edge(uoo, vee, g); weights[edge] = w;
34     }
35 }
36
37 void testcase() {
38     int N, M, s, t; cin >> N >> M >> s >> t;
39     Graph g(N*4);
40
41     for(int m = 0; m < M; ++m) {
42         int u, v, w; cin >> u >> v >> w;
43         add_edges(g, u, v, w);
44     }
45
46     vector<int> d(num_vertices(g), -1);
47     dijkstra_shortest_paths(g, s*4, distance_map(&d[0]));
48     (d[4*t+3] < INT_MAX) ? cout << d[4*t+3] : cout << "no";
49     cout << "\n";
50 }
51
52 int main() {
53     int TC; cin >> TC;
54     while(TC--) testcase();
55     return 0;
56 }
```

# Radiation 2

Missing.