

Algorithms Lab.

Álvaro Marco
amarco@inf.ethz.ch

June 7, 2016

Contents

1 Basic	2
1.1 Theory	2
2 Complexities	2
2.1 Boost Graph Library	2
2.1.1 CGAL	2
3 Problems	3
3.1 Greedy & BFS/DFS	3
3.1.1 Aliens	3
3.1.2 Boats	4
3.1.3 Next path	4
3.1.4 Race tracks	5
3.1.5 Snippets	6
3.2 CGAL basic	8
3.2.1 Antenna	8
3.2.2 Almost antenna	8
3.2.3 Hiking maps	9
3.2.4 Hit	10
3.2.5 First hit	10
3.3 Graphs: Dijkstra & MST	12
3.3.1 Ant challenge	12
3.3.2 First steps BGL	13
3.3.3 Important bridges	13
3.3.4 Return of the Jedi	14
3.3.5 Tracking	15
3.4 Dynamic programming	17
3.4.1 Bonus level	17
3.4.2 Burning coins from two sides	18
3.4.3 DHL	18
3.4.4 Poker chips	19
3.4.5 The great game	20
3.5 Network flows	21
3.5.1 Coin tossing	21
3.5.2 Kingdom defence	23
3.5.3 Shopping trip	24
3.5.4 Tetris	25
3.5.5 The phantom menace	26
3.6 Linear programming	27
3.6.1 Diet	27
3.6.2 Inball	28
3.6.3 Maximize it	29
3.6.4 Portfolios	30
3.6.5 Portfolios revisited	31
3.7 CGAL proximity structures	32
3.7.1 Bistro	32
3.7.2 Germs	33
3.7.3 Graypes	33
3.7.4 H1N1	34
3.7.5 Light the stage	35
3.8 Minimum cut, Bipartite matching	37
3.8.1 Algocoon	37
3.8.2 Knights	38
3.8.3 New hope	40
3.8.4 Satellites	41
3.9 Min cost max flow	43
3.9.1 Bonus level	43
3.9.2 Canteen	44
3.9.3 Carsharing	46
3.9.4 Real estate market	48
3.10 Miscellaneous	50
3.10.1 Buddy selection	50
3.10.2 Divisor distance	50
3.10.3 Monkey island	52

3.11	Combinations	53
3.11.1	Clues	53
3.11.2	Sith	54
3.11.3	Stamps	55
3.11.4	The empire strikes back	56

1 Basic

Include the line:

```
std::ios_base::sync_with_stdio(false);
```

at the beginning of each program.

To print doubles specifying the precision use `setprecision`, e.g.:

```
cout << setprecision(7) << sum << endl;
```

For CGAL programs, add:

```
set(CMAKE_CXX_FLAGS '$CMAKE_CXX_FLAGS -std=c++11')
```

somewhere in the `CMakeLists.txt` file.

Whenever sets are needed consider https://judge.inf.ethz.ch/doc/boost/libs/disjoint_sets/disjoint_sets.html.

Figure 1 shows how the triangulation datastructure works with faces in CGAL. Check slides 19 and 20 in https://judge.inf.ethz.ch/doc/course/cgal_proximity.pdf for further information.

1.1 Theory

- A double can store integers until 2^{53}
- Minimum independent set: maximum number of vertices s.t. there is no edge between the vertices (NP-COMPLETE! however OK for bipartite graph)
- Maximum vertex cover: Minimum number of vertices s.t. all the edges of the graph are included. Complementary to Minimum independent set (NP-COMPLETE! however OK for bipartite graph)
- König's theorem: In bipartite graphs, the size of a minimum vertex cover equals the size of a maximum matching. $|MIS| = n - |MVC|$
- $a \leq b \equiv -a \geq -b$
- $\max(f(x)) = -\min(-f(x))$

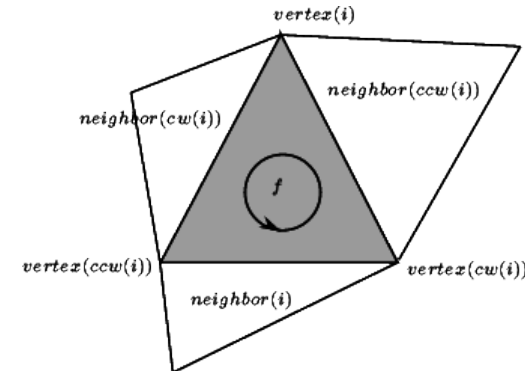


Figure 1: Vertices and neighbor faces in a CGAL triangulation.

2 Complexities

2.1 Boost Graph Library

- Connected components (`connected_components`): $O(V + E)$
- Cycle cancelling (`cycle_canceling`): $O(C \cdot (EV))$ where C is the cost of the initial flow
- Dijkstra (`dijkstra_shortest_paths`): $O(V \log V + E)$
- Kruskal MST (`kruskal_minimum_spanning_tree`): $O(E \log E)$
- Prim MST (`prim_minimum_spanning_tree`): $O(E \log V)$
- Push relabel max flow (`push_relabel_max_flow`): $O(V^3)$
- Strong components (`strong_components`): $O(V + E)$
- Successive shortest path nonneg... (`successive_shortest_path_nonnegative_weights`): $O(|f|(E + V \log V))$

2.1.1 CGAL

Use Kernels in the following order of preference:

1. `CGAL::Exact_predicates_inexact_constructions_kernel` (constructions use double)

2. `CGAL::Exact_predicates_exact_constructions_kernel` (constructions use an exact number type)

3. `CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt`

Try to avoid constructions.

3 Problems

3.1 Greedy & BFS/DFS

3.1.1 Aliens

Keywords— Custom sort

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  struct interval { int left, right; };
8
9  bool equals(const interval &i1, const interval &i2) {
10     return i1.left == i2.left && i1.right == i2.right;
11 }
12
13 bool cmp(const interval &i1, const interval &i2) {
14     return (i1.left != i2.left)
15         ? i1.left < i2.left
16         : i1.right > i2.right;
17 }
18
19 int main() {
20     std::ios_base::sync_with_stdio(false);
21     vector<interval> aliens;
22     int T;
23     cin >> T;
24     while (T--) {
25         int n, m;
26         cin >> n >> m;
27
28         aliens.clear();
29         vector<interval>::iterator it;
30
31         while (n--) {
32             interval in;
33             cin >> in.left >> in.right;
34             if (in.left != 0 || in.right != 0)
35                 aliens.push_back(in);
36         }
```

```

37 sort(alien.begin(), alien.end(), cmp);
38 // check if for each human there exists an alien which
   wound him
39 int end_point = 0;
40 for (it = alien.begin(); it != alien.end(); ++it) {
41     if (it->left > end_point + 1)
42         break;
43     else
44         end_point = max(end_point, it->right);
45 }
46 if (end_point < m) {
47     cout << 0 << endl;
48     continue;
49 }
50 // counting the superiors (intervals not contained in any
   other interval)
51 int superior = 0;
52 int rightmost = -1;
53 for (unsigned int i = 1; i < alien.size(); ++i) {
54     if (equals(alien[i - 1], alien[i])) {
55         if (alien[i - 1].right > rightmost)
56             rightmost = alien[i - 1].right;
57         while (i < alien.size() && equals(alien[i - 1],
   alien[i]))
58             ++i;
59     } else if (alien[i - 1].right > rightmost) {
60         rightmost = alien[i - 1].right;
61         ++superior;
62     }
63 }
64 if (alien.size() == 1 || alien[alien.size() - 1].right >
   rightmost)
65     ++superior;
66 cout << superior << endl;
67 }
68 return 0;
69 }

```

3.1.2 Boats

Keywords— Greedy, Intervals

```

1  #include <iostream>
2  #include <limits>
3  #include <set>
4  #include <vector>
5
6  using namespace std;
7
8  int main() {
9      std::ios_base::sync_with_stdio(false);
10     int T;
11     cin >> T;
12     while (T--) {
13         int N;
14         cin >> N;
15         set< pair<int, int> > boats;
16         while (N--) {
17             int l, p;
18             cin >> l >> p;
19             boats.insert(make_pair(p, l));
20         }
21         int current_start = numeric_limits<int>::min();
22         int best_deadline = numeric_limits<int>::max();
23         int num_boats = 1;
24         for (auto it = boats.begin(); it != boats.end(); ++it) {
25             int p = it->first;
26             int l = it->second;
27             if (p >= best_deadline) {
28                 ++num_boats;
29                 current_start = best_deadline;
30                 best_deadline = numeric_limits<int>::max();
31             }
32             int possible_deadline = max(current_start, p - 1) + 1;
33             best_deadline = min(best_deadline, possible_deadline);
34         }
35         cout << num_boats << endl;
36     }
37     return 0;
38 }

```

3.1.3 Next path

Keywords— BFS

```

1  #include <iostream>
2  #include <queue>
3  #include <vector>
4  #include <algorithm>
5  #include <cstring>
6
7  using namespace std;
8
9  struct state { int n, d; }; // node where we are, depth
10
11 char visited[110];
12
13 int main() {
14     std::ios_base::sync_with_stdio(false);
15     int T;
16     cin >> T;
17     while (T--) {
18         memset(visited, 0, 110);
19         vector<int> empty;
20         vector< vector<int> > graph(110, empty);
21         vector<int>::iterator it;
22
23         int n, m, s, t;
24         cin >> n >> m >> s >> t;
25         while (m--) {
26             int ts, tt;
27             cin >> ts >> tt;
28             graph[ts].push_back(tt);
29         }
30         state is;
31         is.n = s; // set source
32         is.d = 0; // set length of path
33         int times = 0; // t reached 0 times
34         bool path_found = false;
35         queue<state> frontier;
36         frontier.push(is);
37         while (!frontier.empty()) {
38             state current = frontier.front();
39             if (current.n == t) {
40                 ++times;
41                 if (times == 2) {
42                     path_found = true;

```

```

43         cout << current.d << endl;
44         break;
45     }
46 }
47 ++current.d;
48 frontier.pop();
49 for (it = graph[current.n].begin(); it != graph[current.n]
50     ].end(); ++it) {
51     if (visited[*it] < 2) {
52         ++visited[*it];
53         state tmp;
54         tmp.n = *it;
55         tmp.d = current.d;
56         frontier.push(tmp);
57     }
58 }
59 if (!path_found)
60     cout << "no" << endl;
61 }
62 return 0;
63 }

```

3.1.4 Race tracks

Keywords— BFS

```

1  #include <iostream>
2  #include <algorithm>
3  #include <queue>
4  #include <cstring>
5
6  using namespace std;
7
8  struct node {
9      int x, y; // position
10     int sx, sy; // speed
11     int d; // depth
12 };
13
14 int X, Y;
15 int end_x, end_y;
16 bool visited[30][30][7][7];

```

```

17 queue<node> frontier;
18 int hops;
19
20 void clear(queue<node> &q) {
21     queue<node> empty;
22     swap(q, empty);
23 }
24
25 // just to stay inside the bounds of the array
26 int speed(int s) {
27     if (s == -1) { s = 4; }
28     else if (s == -2) { s = 5; }
29     else if (s == -3) { s = 6; }
30     return s;
31 }
32
33 void bfs() {
34     while (!frontier.empty()) {
35         node extracted = frontier.front();
36         frontier.pop();
37         // insert new states here
38         for (int dx = -1; dx != 2; ++dx) {
39             for (int dy = -1; dy != 2; ++dy) {
40                 node tmp = extracted;
41                 tmp.sx += dx; tmp.sy += dy;
42                 int newx = tmp.x + tmp.sx, newy = tmp.y + tmp.sy;
43                 if (tmp.sx >= -3 && tmp.sx <= 3 && tmp.sy >= -3 && tmp.sy <= 3
44                     && newx >= 0 && newx < X && newy >= 0 && newy < Y
45                     && !visited[newx][newy][speed(tmp.sx)][speed(tmp.sy)]) {
46                     visited[newx][newy][speed(tmp.sx)][speed(tmp.sy)] =
47                         true;
48                     ++tmp.d;
49                     if (newx == end_x && newy == end_y) {
50                         hops = tmp.d;
51                         return;
52                     }
53                     tmp.x = newx; tmp.y = newy;
54                     frontier.push(tmp);
55                 }
56             }
57         }
58     }
59 }

```

```

57 }
58 }
59
60 int main() {
61     std::ios_base::sync_with_stdio(false);
62     int T;
63     cin >> T;
64     while (T--) {
65         memset(visited, false, 30 * 30 * 7 * 7);
66         clear(frontier);
67
68         node start;
69         start.sx = start.sy = start.d = 0;
70
71         int P;
72         cin >> X >> Y;
73         cin >> start.x >> start.y >> end_x >> end_y;
74         cin >> P;
75         for (int p = 0; p < P; ++p) {
76             int px1, py1, px2, py2;
77             cin >> px1 >> py1 >> px2 >> py2;
78             for (int i = px1; i <= px2; ++i)
79                 for (int j = py1; j <= py2; ++j)
80                     memset(visited[i][j], 1, 7 * 7);
81         }
82         visited[start.x][start.y][0][0] = true;
83         frontier.push(start);
84         hops = (start.x == end_x && start.y == end_y) ? 0 : dfs();
85         if (hops == -1)
86             cout << "No solution." << endl;
87         else
88             cout << "Optimal solution takes " << hops << " hops." <<
89                 endl;
90     }
91     return 0;
92 }

```

3.1.5 Snippets

Keywords— priority_queue

```

1 #include <iostream>
2 #include <algorithm>

```

```

3 #include <limits>
4 #include <queue>
5 #include <vector>
6
7 using namespace std;
8
9 vector<int> times;
10 vector< vector<int> > positions;
11
12 class CompareDist {
13 public:
14     bool operator()(pair<int, int> n1, pair<int, int> n2) {
15         return n1.first > n2.first;
16     }
17 };
18
19 int main() {
20     std::ios_base::sync_with_stdio(false);
21     int T;
22     cin >> T;
23     while (T--) {
24         int n, m, p;
25         times.clear();
26         positions.clear();
27         cin >> n;
28         for (int i = 0; i < n; ++i) {
29             cin >> m;
30             times.push_back(m);
31         }
32         for (int i = 0; i < n; ++i) {
33             vector<int> newv;
34             positions.push_back(newv);
35             for (int j = 0; j < times[i]; ++j) {
36                 cin >> p;
37                 positions[i].push_back(p);
38             }
39         }
40
41         int a = positions[0][0], b = positions[0][0];
42         // greater<int>
43         priority_queue< pair<int, int>, vector< pair<int, int> >,
44             CompareDist > current_snippets;

```

```

45         for (int i = 0; i < n; ++i) {
46             sort(positions[i].begin(), positions[i].end());
47             iterators.push_back(positions[i].begin());
48             current_snippets.push(make_pair(*iterators[i], i));
49             a = min(a, *iterators[i]);
50             b = max(b, *iterators[i]);
51             ++iterators[i];
52         }
53         int min_snippet = b - a + 1;
54         int pos_first_last = -1; // position of first iterator
55                                 // arriving at last position
56         while (!current_snippets.empty()) {
57             auto top_snippet = current_snippets.top();
58             current_snippets.pop();
59             int pos = top_snippet.first;
60             a = pos;
61             if (pos_first_last != -1)
62                 a = min(a, pos_first_last);
63             min_snippet = min(min_snippet, b - a + 1);
64             //cout << "[" << a << ", " << b << "]" << endl;
65             int owner = top_snippet.second;
66             if (iterators[owner] != positions[owner].end()) {
67                 current_snippets.push(make_pair(*iterators[owner],
68                     owner));
69                 b = max(b, *iterators[owner]);
70                 ++iterators[owner];
71             } else if (pos_first_last == -1) {
72                 pos_first_last = pos;
73             }
74             cout << min_snippet << endl;
75         }
76         return 0;
77     }

```

3.2 CGAL basic

3.2.1 Antenna

Keywords— `ceil_to_double`, `Min_circle_2`

```
1 #include <CGAL/
   Exact_predicates_exact_constructions_kernel_with_sqrt.h>
2 #include <CGAL/Min_circle_2.h>
3 #include <CGAL/Min_circle_2_traits_2.h>
4 #include <iostream>
5 #include <vector>
6
7 typedef CGAL::
   Exact_predicates_exact_constructions_kernel_with_sqrt K;
8 typedef CGAL::Min_circle_2_traits_2<K> Traits;
9 typedef CGAL::Min_circle_2<Traits> Min_circle;
10
11 double ceil_to_double(const K::FT& x) {
12     double a = std::ceil(CGAL::to_double(x));
13     while (a < x) a += 1;
14     while (a-1 >= x) a -= 1;
15     return a;
16 }
17
18 int main() {
19     std::ios_base::sync_with_stdio(false);
20     while (true) {
21         int n;
22         std::cin >> n;
23         if (n == 0)
24             break;
25
26         std::vector<K::Point_2> citizens;
27         while (n-->) {
28             double x, y;
29             std::cin >> x >> y;
30             citizens.push_back(K::Point_2(x, y));
31         }
32         Min_circle mc(citizens.begin(), citizens.end(), true);
33         Traits::Circle c = mc.circle();
34         K::FT r = sqrt(c.squared_radius());
35         std::cout << (long long) ceil_to_double(r) << std::endl;
36     }
```

```
37     return 0;
38 }
```

3.2.2 Almost antenna

Keywords— `Min_circle_2`

```
1 #include <CGAL/
   Exact_predicates_exact_constructions_kernel_with_sqrt.h>
2 #include <CGAL/Min_circle_2.h>
3 #include <CGAL/Min_circle_2_traits_2.h>
4 #include <iostream>
5 #include <algorithm>
6 #include <vector>
7
8 typedef CGAL::
   Exact_predicates_exact_constructions_kernel_with_sqrt K;
9 typedef CGAL::Min_circle_2_traits_2<K> Traits;
10 typedef CGAL::Min_circle_2<Traits> Min_circle;
11
12 double ceil_to_double(const K::FT& x) {
13     double a = std::ceil(CGAL::to_double(x));
14     while (a < x) a += 1;
15     while (a-1 >= x) a -= 1;
16     return a;
17 }
18
19 int main() {
20     std::ios_base::sync_with_stdio(false);
21     while (true) {
22         int n;
23         std::cin >> n;
24         if (n == 0)
25             break;
26
27         std::vector<K::Point_2> citizens;
28         double x, y;
29         while (n-->) {
30             std::cin >> x >> y;
31             citizens.push_back(K::Point_2(x, y));
32         }
33         bool first = true;
34         K::FT min_r;
```



```

35 Min_circle mc_all(citizens.begin(), citizens.end(), true);
36 for (auto it = mc_all.support_points_begin();
37      it != mc_all.support_points_end(); ++it) {
38     auto to_remove = std::find(citizens.begin(), citizens.end
39                                (), *it);
40     citizens.erase(to_remove);
41     Min_circle mc(citizens.begin(), citizens.end(), true);
42     Traits::Circle c = mc.circle();
43     auto r = sqrt(c.squared_radius());
44     if (first) {
45         min_r = r;
46         first = false;
47     } else if (r < min_r) {
48         min_r = r;
49     }
50     citizens.push_back(*it);
51 }
52 std::cout << (long long) ceil_to_double(min_r) << std::endl
53 ;
54 }

```

3.2.3 Hiking maps

Keywords— all_of, left_turn

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <iostream>
3  #include <limits>
4  #include <vector>
5
6  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
7
8  K::Point_2 q0, q1, q2, q3, q4, q5;
9
10 bool inside_map(K::Point_2 &p) {
11     return (CGAL::left_turn(q0, q1, p) || CGAL::collinear(q0, q1,
12                                                             p))
13     && (CGAL::left_turn(q2, q3, p) || CGAL::collinear(q2, q3,
14                                                         p))
15     && (CGAL::left_turn(q4, q5, p) || CGAL::collinear(q4, q5,
16                                                         p));

```

```

14 }
15
16 // WARNING!! modifies the first vector
17 void vsum(std::vector<int> &a, std::vector<int> &b) {
18     for (int i = 0; i < a.size(); ++i)
19         a[i] += b[i];
20 }
21
22 void vsub(std::vector<int> &a, std::vector<int> &b) {
23     for (int i = 0; i < a.size(); ++i)
24         a[i] -= b[i];
25 }
26
27 bool all_contained(std::vector<int> &contained) {
28     return std::all_of(contained.begin(), contained.end(), [](int
29                                                                i){return i>0;});
30 }
31
32 int main() {
33     std::ios_base::sync_with_stdio(false);
34     int T;
35     std::cin >> T;
36     while (T--) {
37         int m, n;
38         std::cin >> m >> n;
39         std::vector<K::Point_2> legs;
40         legs.reserve(m);
41         for (int i = 0; i < m; ++i) {
42             int x, y;
43             std::cin >> x >> y;
44             legs.push_back(K::Point_2(x, y));
45         }
46         std::vector< std::vector<int> > cont_leg;
47         for (int i = 0; i < n; ++i) {
48             int x0, y0, x1, y1, x2, y2, x3, y3, x4, y4, x5, y5;
49             std::cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2 >> x3 >> y3
50                 >> x4 >> y4
51                 >> x5 >> y5;
52             q0 = K::Point_2(x0, y0);
53             q1 = K::Point_2(x1, y1);
54             q2 = K::Point_2(x2, y2);
55             q3 = K::Point_2(x3, y3);
56             q4 = K::Point_2(x4, y4);

```

```

55     q5 = K::Point_2(x5, y5);
56
57     // orient the points to test with left_turn after
58     if (CGAL::right_turn(q0, q1, q2))
59         std::swap(q0, q1);
60     if (CGAL::right_turn(q2, q3, q4))
61         std::swap(q2, q3);
62     if (CGAL::right_turn(q4, q5, q0))
63         std::swap(q4, q5);
64
65     std::vector<int> contained(m - 1, 0);
66     for (int li = 0; li < legs.size() - 1; ++li) {
67         if (inside_map(legs[li]) && inside_map(legs[li + 1]))
68             contained[li] = 1;
69     }
70     cont_leg.push_back(contained);
71 }
72 int i = 0;
73 std::vector<int> contained(m - 1, 0);
74 int min_size = std::numeric_limits<int>::max();
75 for (int j = i; j < n; ++j) {
76     vsum(contained, cont_leg[j]);
77     while (all_contained(contained)) {
78         min_size = std::min(min_size, j - i + 1);
79         vsub(contained, cont_leg[i]);
80         ++i;
81     }
82 }
83 std::cout << min_size << std::endl;
84 }
85 return 0;
86 }

```

3.2.4 Hit

Keywords— do_intersect, Ray_2, Segment_2

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <iostream>
3  #include <vector>
4
5  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
6  typedef K::Point_2 P;

```

```

7  typedef K::Segment_2 S;
8
9  int main() {
10     std::ios_base::sync_with_stdio(false);
11     while (true) {
12         int n;
13         std::cin >> n;
14         if (n == 0)
15             break;
16
17         std::vector<S> segments;
18         double x, y, a, b;
19         std::cin >> x >> y >> a >> b;
20         P p = P(x, y);
21         P q = P(a, b);
22         auto phileas_ray = K::Ray_2(p, q);
23         while (n--) {
24             double r, s, t, u;
25             std::cin >> r >> s >> t >> u;
26             p = P(r, s);
27             q = P(t, u);
28             S seg = S(p, q);
29             segments.push_back(seg);
30         }
31         bool intersect = false;
32         for (auto it = segments.begin(); it != segments.end(); ++it)
33             if (CGAL::do_intersect(phileas_ray, *it)) {
34                 intersect = true;
35                 break;
36             }
37     }
38     std::cout << (intersect ? "yes" : "no") << std::endl;
39 }
40 return 0;
41 }

```

3.2.5 First hit

Keywords— floor_to_double, intersection

```

1  #include <CGAL/Exact_predicates_exact_constructions_kernel.h>
2  #include <iostream>

```

```

3 #include <vector>
4
5 typedef CGAL::Exact_predicates_exact_constructions_kernel K;
6 typedef K::Point_2 P;
7 typedef K::Segment_2 S;
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 int main() {
17     std::ios_base::sync_with_stdio(false);
18     while (true) {
19         int n;
20         std::cin >> n;
21         if (n == 0)
22             break;
23
24         std::vector<S> segments;
25         double x, y, a, b;
26         std::cin >> x >> y >> a >> b;
27         P_src = P(x, y);
28         q = P(a, b);
29         auto phileas_ray = K::Ray_2(p_src, q);
30         while (n--) {
31             double r, s, t, u;
32             std::cin >> r >> s >> t >> u;
33             S seg = S(P(r, s), P(t, u));
34             segments.push_back(seg);
35         }
36         bool one_found = false;
37         P min_dist_p;
38         K::FT min_dist;
39         for (auto it = segments.begin(); it != segments.end(); ++it) {
40             if (CGAL::do_intersect(phileas_ray, *it)) {
41                 auto o = CGAL::intersection(phileas_ray, *it);
42                 K::FT dist;
43                 P ipoint;
44                 if (P* op = boost::get<P>(&*o)) {

```

```

45         dist = CGAL::squared_distance(p_src, *op);
46         ipoint = *op;
47     } else if (S* os = boost::get<S>(&*o)) {
48         if (CGAL::squared_distance(p_src, os->source())
49             < CGAL::squared_distance(p_src, os->target())) {
50             dist = CGAL::squared_distance(p_src, os->source());
51             ipoint = os->source();
52         } else {
53             dist = CGAL::squared_distance(p_src, os->target());
54             ipoint = os->target();
55         }
56     }
57     if (!one_found) {
58         min_dist = dist;
59         min_dist_p = ipoint;
60         one_found = true;
61     } else if (dist < min_dist) {
62         min_dist = dist;
63         min_dist_p = ipoint;
64     }
65 }
66 }
67 if (one_found)
68     std::cout << (long long) floor_to_double(min_dist_p.x())
69     << " "
70     << (long long) floor_to_double(min_dist_p.y())
71     << std::endl;
72 else
73     std::cout << "no" << std::endl;
74 }
75 return 0;
76 }

```

3.3 Graphs: Dijkstra & MST

3.3.1 Ant challenge

Keywords— Dijkstra, Kruskal

```
1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4 #include <boost/graph/adjacency_list.hpp>
5 #include <boost/graph/dijkstra_shortest_paths.hpp>
6 #include <boost/graph/kruskal_min_spanning_tree.hpp>
7 #include <boost/tuple/tuple.hpp>
8
9 using namespace boost;
10 using namespace std;
11
12 typedef adjacency_list<vecS, vecS, undirectedS, no_property,
13     property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     int T;
20     cin >> T;
21     while (T--) {
22         vector<Graph> species;
23         vector<WeightMap> wms;
24         vector<int> hives;
25         int t, e, s, a, b;
26         cin >> t >> e >> s >> a >> b;
27         for (int ss = 0; ss < s; ++ss) {
28             Graph newg(t);
29             species.push_back(newg);
30             wms.push_back(get(edge_weight, newg));
31         }
32         while (e--) {
33             int t1, t2;
34             cin >> t1 >> t2;
35             for (int ss = 0; ss < s; ++ss) {
36                 Edge e;
37                 int w;
```

```
38         tie(e, tuples::ignore) = add_edge(t1, t2, species[ss]);
39         wms[ss][e] = w;
40     }
41 }
42 for (int ss = 0; ss < s; ++ss) {
43     int h;
44     cin >> h;
45     hives.push_back(h);
46 }
47 Graph finalg(t);
48 WeightMap finalwm;
49 for (int ss = 0; ss < s; ++ss) {
50     vector<Edge> mst;
51     kruskal_minimum_spanning_tree(species[ss], back_inserter(
52         mst));
53     for (auto ite = mst.begin(); ite != mst.end(); ++ite) {
54         Edge e;
55         tie(e, tuples::ignore) = add_edge(source(*ite, species[
56             ss]),
57             target(*ite, species[
58                 ss]),
59             finalg);
60         finalwm[e] = wms[ss][*ite];
61     }
62 }
63 vector<int> dist(t);
64 vector<int> pred(t);
65 dijkstra_shortest_paths(finalg, a,
66     predecessor_map(
67         make_iterator_property_map(pred
68             .begin(),
69             get(vertex_index, finalg)))
70     .distance_map(
71         make_iterator_property_map(
72             dist.begin(),
73             get(vertex_index, finalg))
74     )
75 );
76 cout << dist[b] << endl;
77 }
78 return 0;
79 }
```

3.3.2 First steps BGL

Keywords— Dijkstra, Kruskal

```
1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4 #include <boost/graph/adjacency_list.hpp>
5 #include <boost/graph/dijkstra_shortest_paths.hpp>
6 #include <boost/graph/kruskal_min_spanning_tree.hpp>
7 #include <boost/tuple/tuple.hpp>
8
9 using namespace boost;
10 using namespace std;
11
12 typedef adjacency_list<vecS, vecS, undirectedS, no_property,
13     property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     int T;
20     cin >> T;
21     while (T--) {
22         int n, m;
23         cin >> n >> m;
24         Graph G(n);
25         WeightMap wm = get(edge_weight, G);
26         while (m--) {
27             int u, v, c;
28             Edge e;
29             cin >> u >> v >> c;
30             tie(e, tuples::ignore) = add_edge(v, u, G);
31             wm[e] = c;
32         }
33         // Sum of weight of MST
34         int w = 0;
35         vector<Edge> mst;
36         kruskal_minimum_spanning_tree(G, back_inserter(mst));
37         for (auto ite = mst.begin(); ite != mst.end(); ++ite)
38             w += wm[*ite];
39         // distance from node 0 to node furthest from it
40         int d = -1;
```

```
40 vector<int> dist(n);
41 vector<int> pred(n);
42 dijkstra_shortest_paths(G, 0, predecessor_map(
43     make_iterator_property_map(pred.begin(),
44         get(vertex_index, G)))
45     .distance_map(
46         make_iterator_property_map(
47             dist.begin(),
48             get(vertex_index, G))
49         );
50 graph_traits<Graph>::vertex_iterator vi, vend;
51 for (tie(vi, vend) = vertices(G); vi != vend; ++vi)
52     d = max(d, dist[*vi]);
53 cout << w << " " << d << endl;
54 }
```

3.3.3 Important bridges

Keywords— Custom sort, Kruskal

```
1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4 #include <boost/graph/adjacency_list.hpp>
5 #include <boost/graph/connected_components.hpp>
6 #include <boost/graph/kruskal_min_spanning_tree.hpp>
7 #include <boost/tuple/tuple.hpp>
8
9 using namespace boost;
10 using namespace std;
11
12 typedef adjacency_list<vecS, vecS, undirectedS, no_property,
13     property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     int T;
20     cin >> T;
```

```

20 while (T--) {
21     int n, m;
22     cin >> n >> m;
23     Graph G(n);
24     WeightMap wm = get(edge_weight, G);
25     while (m--) {
26         int e1, e2;
27         Edge e;
28         cin >> e1 >> e2;
29         tie(e, tuples::ignore) = add_edge(e1, e2, G);
30         wm[e] = 1;
31     }
32     vector<Edge> mst;
33     kruskal_minimum_spanning_tree(G, back_inserter(mst));
34     int num_critical = 0;
35     vector< pair<int, int> > critical_bridges;
36     for (auto ite = mst.begin(); ite != mst.end(); ++ite) {
37         vector<int> component(num_vertices(G));
38         int src = source(*ite, G);
39         int tgt = target(*ite, G);
40         remove_edge(src, tgt, G);
41         int num = connected_components(G, &component[0]);
42         add_edge(src, tgt, G);
43         if (num != 2) { // WARNING this should be 1! It's only 2
                        // because of by default vertex
44             critical_bridges.push_back(make_pair(min(src, tgt), max
                (src, tgt)));
45             ++num_critical;
46         }
47     }
48     cout << num_critical << endl;
49     sort(critical_bridges.begin(), critical_bridges.end(),
50         [](const std::pair<int,int> &left, const std::pair<int
            ,int> &right) {
51             return left.first != right.first ?
52                 left.first < right.first :
53                 left.second < right.second;
54         });
55     for (auto it = critical_bridges.begin(); it !=
        critical_bridges.end(); ++it)
56         cout << it->first << " " << it->second << endl;
57 }
58 return 0;

```

```

59 }

```

3.3.4 Return of the Jedi

Keywords— Kruskal, Second MST

Check https://judge.inf.ethz.ch/doc/course/return_of_the_jedi.pdf

```

1  #include <iostream>
2  #include <algorithm>
3  #include <limits>
4  #include <stack>
5  #include <vector>
6  #include <boost/graph/adjacency_list.hpp>
7  #include <boost/graph/kruskal_min_spanning_tree.hpp>
8  #include <boost/tuple/tuple.hpp>
9
10 using namespace boost;
11 using namespace std;
12
13 typedef adjacency_list<vecS, vecS, undirectedS, no_property,
    property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::vertex_descriptor Vertex;
15 typedef graph_traits<Graph>::edge_descriptor Edge;
16 typedef graph_traits<Graph>::edge_iterator EdgeIt;
17 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
18 typedef property_map<Graph, edge_weight_t>::type WeightMap;
19
20 // WARNING! costs is an in-out parameter
21 void dfs(int src, vector<bool> &visited, Graph &T, WeightMap &
    wm,
22     vector<int> &costs, int max_edge) {
23     costs[src] = max_edge;
24     OutEdgeIt ei, oeiend;
25     for (tie(ei, oeiend) = out_edges(src, T); ei != oeiend; ++ei)
26     {
27         int edge_w = max(max_edge, wm[*ei]);
28         Vertex v = target(*ei, T);
29         if (!visited[v]) {
30             visited[v] = true;
31             dfs(v, visited, T, wm, costs, edge_w);
32         }
33     }

```

```

33 }
34
35 int main() {
36     std::ios_base::sync_with_stdio(false);
37     int T;
38     cin >> T;
39     while (T--) {
40         int n, i_tatoo;
41         cin >> n >> i_tatoo;
42
43         Graph G(n);
44         WeightMap wm = get(edge_weight, G);
45         for (int j = 0; j < n-1; ++j) {
46             for (int k = j+1; k < n; ++k) {
47                 int w;
48                 cin >> w;
49                 Edge e;
50                 tie(e, tuples::ignore) = add_edge(j, k, G);
51                 wm[e] = w;
52             }
53         }
54         // Build MST T
55         vector<Edge> mst;
56         kruskal_minimum_spanning_tree(G, back_inserter(mst));
57         long long mst_cost = 0;
58         Graph T(n);
59         WeightMap wm_T = get(edge_weight, T);
60         for (auto ite = mst.begin(); ite != mst.end(); ++ite) {
61             Edge e;
62             Vertex u = source(*ite, G);
63             Vertex v = target(*ite, G);
64             tie(e, tuples::ignore) = add_edge(u, v, T);
65             int cost = wm[*ite];
66             wm_T[e] = cost;
67             mst_cost += cost;
68         }
69         // DFS on T for each vertex
70         vector< vector<int> > max_edge(n); // max edge in the path
71                                         // from v to each
72                                         // other vertex in the
73                                         // mst
74
75         for (int v = 0; v < n; ++v) {
76             vector<int> costs(n, 0);

```

```

74         vector<bool> visited(n, false);
75         visited[v] = true;
76         dfs(v, visited, T, wm_T, costs, 0);
77         max_edge[v] = costs;
78     }
79     long long second_mst = numeric_limits<long long>::max();
80     EdgeIt ei, eend;
81     for (tie(ei, eend) = edges(G); ei != eend; ++ei) {
82         Vertex u = source(*ei, G);
83         Vertex v = target(*ei, G);
84         // edge(u,v,g) returns pair<edge_descriptor, bool> where
85         // the bool is
86         // whether the edge exists
87         if (edge(u, v, T).second) continue; // if it's in the MST
88         // skip it!
89         int delta = wm[*ei] - max_edge[u][v];
90         second_mst = min(second_mst, mst_cost + delta);
91     }
92     cout << second_mst << endl;
93 }

```

3.3.5 Tracking

Keywords— Dijkstra

```

1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/dijkstra_shortest_paths.hpp>
7  #include <boost/tuple/tuple.hpp>
8
9  using namespace boost;
10 using namespace std;
11
12 typedef adjacency_list<vecS, vecS, directedS, no_property,
13     property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;

```

```

16 // add edge from u to v and from v to u with cost c
17 void add_edges(Graph &G, WeightMap &wm, int u, int v, int c) {
18     Edge e;
19     tie(e, tuples::ignore) = add_edge(u, v, G);
20     wm[e] = c;
21     tie(e, tuples::ignore) = add_edge(v, u, G);
22     wm[e] = c;
23 }
24
25 int main() {
26     std::ios_base::sync_with_stdio(false);
27     int T;
28     cin >> T;
29     while (T--) {
30         int n, m, k, x, y;
31         cin >> n >> m >> k >> x >> y;
32         int nvert = n * (k + 1);
33         Graph G(nvert);
34         WeightMap wm;
35         while (m--) {
36             int a, b, c, d;
37             cin >> a >> b >> c >> d;
38             if (d) { // bridge
39                 for (int i = 0; i < k; ++i) {
40                     Edge e;
41                     tie(e, tuples::ignore) = add_edge(a, b + n, G);
42                     wm[e] = c;
43                     tie(e, tuples::ignore) = add_edge(b, a + n, G);
44                     wm[e] = c;
45                     a += n;
46                     b += n;
47                 }
48                 add_edges(G, wm, a, b, c);
49             } else {
50                 for (int i = 0; i <= k; ++i) { // for k+1 !!!
51                     add_edges(G, wm, a, b, c);
52                     a += n;
53                     b += n;
54                 }
55             }
56         }
57         vector<int> dist(nvert);
58         vector<int> pred(nvert);

```

```

59     dijkstra_shortest_paths(G, x,
60                             predecessor_map(
61                                 make_iterator_property_map(pred
62                                     .begin(),
63                                     get(vertex_index, G)))
64                             .distance_map(
65                                 make_iterator_property_map(
66                                     dist.begin(),
67                                     get(vertex_index, G))
68                             );
69     cout << dist[y + k*n] << endl;
70 }
71 return 0;
72 }

```


3.4 Dynamic programming

3.4.1 Bonus level

Keywords— ATTENTION! this solution gives only 80 points! (100 probably replacing the map by a vector) check 3.9.1 for 100 points

```
1 #include <iostream>
2 #include <algorithm>
3 #include <map>
4 #include <vector>
5
6 using namespace std;
7
8 int n; // size of the board
9 int D; // number of diagonals in the board
10 int board[100][100];
11 map<tuple< int, tuple<int, int> >, int> stored;
12
13 int mario(int d, pair<int, int> pos1, pair<int, int> pos2) {
14     //cout << d << " [" << pos1.first << ", " << pos1.second <<
15     //    "]" [" << pos2.first << ", " << pos2.second << "]" << endl;
16     if (d == D-1)
17         return board[n - 1][n - 2] + board[n - 2][n - 1];
18
19     int max_element = numeric_limits<int>::min();
20     auto np1 = make_pair(pos1.first + 1, pos1.second);
21     auto np2 = make_pair(pos2.first + 1, pos2.second);
22     if (np1.first < n && np2.first < n && np1 < np2) {
23         auto key = make_pair(d + 1, make_pair(np1.first, np2.first));
24         auto it = stored.find(key);
25         int res = it->second;
26         if (it == stored.end())
27             res = mario(d + 1, np1, np2);
28         max_element = max(max_element, res);
29     }
30     np1 = make_pair(pos1.first + 1, pos1.second);
31     np2 = make_pair(pos2.first, pos2.second + 1);
32     if (np1.first < n && np2.second < n && np1 < np2) {
33         auto key = make_pair(d + 1, make_pair(np1.first, np2.first));
34         auto it = stored.find(key);
35         int res = it->second;
```

```
35     if (it == stored.end())
36         res = mario(d + 1, np1, np2);
37     max_element = max(max_element, res);
38 }
39 np1 = make_pair(pos1.first, pos1.second + 1);
40 np2 = make_pair(pos2.first + 1, pos2.second);
41 if (np1.second < n && np2.first < n && np1 < np2) {
42
43     auto key = make_pair(d + 1, make_pair(np1.first, np2.first));
44     auto it = stored.find(key);
45     int res = it->second;
46     if (it == stored.end())
47         res = mario(d + 1, np1, np2);
48     max_element = max(max_element, res);
49 }
50 np1 = make_pair(pos1.first, pos1.second + 1);
51 np2 = make_pair(pos2.first, pos2.second + 1);
52 if (np1.second < n && np2.second < n && np1 < np2) {
53     auto np1 = make_pair(pos1.first, pos1.second + 1);
54     auto np2 = make_pair(pos2.first, pos2.second + 1);
55     auto key = make_pair(d + 1, make_pair(np1.first, np2.first));
56     auto it = stored.find(key);
57     int res = it->second;
58     if (it == stored.end())
59         res = mario(d + 1, np1, np2);
60     max_element = max(max_element, res);
61 }
62
63 int vp1 = board[pos1.first][pos1.second];
64 int vp2 = 0;
65 if (pos1 != pos2)
66     vp2 = board[pos2.first][pos2.second];
67
68 auto key = make_pair(d, make_pair(pos1.first, pos2.first));
69 auto value = max_element + vp1 + vp2;
70 auto it = stored.find(key);
71 if (it == stored.end())
72     stored.insert(make_pair(key, value));
73 return value;
74 }
75
```

```

76 int main() {
77     std::ios_base::sync_with_stdio(false);
78     int T;
79     cin >> T;
80     while (T--) {
81         cin >> n;
82         stored.clear();
83         for (int i = 0; i < n; ++i) {
84             for (int j = 0; j < n; ++j) {
85                 int a;
86                 cin >> a;
87                 board[i][j] = a;
88             }
89         }
90         D = 2*n - 2;
91         cout << mario(1, make_pair(0, 1), make_pair(1, 0)) + board
92             [0][0] + board[n - 1][n - 1] << endl;
93     }
94     return 0;
}

```

3.4.2 Burning coins from two sides

Keywords—Minimax

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <cstring>
5
6  #define S 1000
7
8  using namespace std;
9
10 vector<int> coins;
11 int stored[S][S];
12
13 int max_(int f, int l);
14
15 int min_(int f, int l) {
16     if (f > l)
17         return 0;
18

```

```

19     int v1 = (stored[f + 1][l] == -1) ? max_(f + 1, l) : stored[f
20         + 1][l];
21     int v2 = (stored[f][l - 1] == -1) ? max_(f, l - 1) : stored[f
22         ][l - 1];
23     stored[f][l] = min(v1, v2);
24     return stored[f][l];
25 }
26
27 int max_(int f, int l) {
28     if (f > l)
29         return 0;
30
31     int v1 = (stored[f + 1][l] == -1) ? min_(f + 1, l) : stored[f
32         + 1][l];
33     int v2 = (stored[f][l - 1] == -1) ? min_(f, l - 1) : stored[f
34         ][l - 1];
35     stored[f][l] = max(coins[f] + v1, coins[l] + v2);
36     return stored[f][l];
37 }
38
39 int main() {
40     std::ios_base::sync_with_stdio(false);
41     int T;
42     cin >> T;
43     while (T--) {
44         int n;
45         cin >> n;
46         coins.clear();
47         memset(stored, -1, S * S * sizeof(int));
48         for (int i = 0; i < n; ++i) {
49             int v;
50             cin >> v;
51             coins.push_back(v);
52         }
53         cout << max_(0, coins.size() - 1) << endl;
54     }
55     return 0;
56 }

```

3.4.3 DHL

```

1  #include <iostream>

```

```

2  #include <algorithm>
3  #include <vector>
4  #include <cstring>
5
6  using namespace std;
7
8  vector<int> A;
9  vector<int> B;
10
11 int stored[1005][1005];
12
13 int rec_try(int i, int j) {
14     int asum = 0;
15     for (int a = 0; a < i; ++a)
16         asum += A[a];
17     int bsum = 0;
18     for (int b = 0; b < j; ++b)
19         bsum += B[b];
20
21     if (i == 0)
22         return A[i] * bsum;
23     if (j == 0)
24         return asum * B[j];
25     int best = asum * bsum;
26
27     asum = 0;
28     for (int a = i - 1; a >= 0; --a) {
29         asum += A[a];
30         int cost = asum * B[j - 1];
31         if (stored[a][j - 1] == -1)
32             stored[a][j - 1] = rec_try(a, j - 1);
33         best = min(best, cost + stored[a][j - 1]);
34     }
35     bsum = 0;
36     for (int b = j - 1; b >= 0; --b) {
37         bsum += B[b];
38         int cost = A[i - 1] * bsum;
39         if (stored[i - 1][b] == -1)
40             stored[i - 1][b] = rec_try(i - 1, b);
41         best = min(best, cost + stored[i - 1][b]);
42     }
43     return best;
44 }

```

```

45
46 int main() {
47     std::ios_base::sync_with_stdio(false);
48     int T;
49     cin >> T;
50     while (T--) {
51         memset(stored, -1, sizeof(int) * 1005 * 1005);
52         A.clear();
53         B.clear();
54         int n;
55         cin >> n;
56         int tmp;
57         for (int i = 0; i < n; ++i) {
58             cin >> tmp;
59             A.push_back(tmp - 1);
60         }
61         for (int i = 0; i < n; ++i) {
62             cin >> tmp;
63             B.push_back(tmp - 1);
64         }
65         cout << rec_try(n, n) << endl;
66     }
67     return 0;
68 }

```

3.4.4 Poker chips

Unfortunately only 30 points :(

```

1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4  #include <cstring>
5
6  using namespace std;
7
8  const int SIZE = 1025;
9
10 vector<int> ms;
11 vector< vector<int> > stacks;
12
13 int dp[SIZE][SIZE];
14

```

```

15 int main() {
16     std::ios_base::sync_with_stdio(false);
17     int T;
18     cin >> T;
19     while (T--) {
20         memset(dp, 0, sizeof(short) * SIZE * SIZE);
21         ms.clear();
22         stacks.clear();
23         int n;
24         cin >> n;
25         for (int i = 0; i < n; ++i) {
26             int m;
27             cin >> m;
28             ms.push_back(m);
29             vector<int> newv;
30             stacks.push_back(newv);
31         }
32         for (int i = 0; i < n; ++i) {
33             stacks[i].push_back(-1); // insert unused element just
                                     // for index convenience
34             for (int j = 0; j < ms[i]; ++j) {
35                 int c;
36                 cin >> c;
37                 stacks[i].push_back(c);
38             }
39         }
40         for (int i = 1; i <= ms[0]; ++i) {
41             for (int j = 1; j <= ms[1]; ++j) {
42                 int match = stacks[0][i] == stacks[1][j];
43                 dp[i][j] = max(dp[i-1][j], max(dp[i][j-1], dp[i-1][j-1]
                                     + match));
44             }
45         }
46         cout << dp[ms[0]][ms[1]] << endl;
47     }
48     return 0;
49 }

```

3.4.5 The great game

Keywords— Minimax, max_element, min_element

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <cstring>
5
6 #define S 50001
7
8 using namespace std;
9
10 int minstored[S];
11 int maxstored[S];
12
13 int max_(vector< vector<int> > &graph, int src, int tgt);
14
15 int min_(vector< vector<int> > &graph, int src, int tgt) {
16     if (src == tgt) {
17         minstored[src] = 0;
18         return minstored[src];
19     }
20
21     vector<int> dist;
22     for (int v: graph[src]) {
23         if (maxstored[v] == -1)
24             max_(graph, v, tgt);
25         dist.push_back(maxstored[v]);
26     }
27     minstored[src] = *min_element(begin(dist), end(dist)) + 1;
28     return minstored[src];
29 }
30
31 int max_(vector< vector<int> > &graph, int src, int tgt) {
32     if (src == tgt) {
33         maxstored[src] = 0;
34         return maxstored[src];
35     }
36
37     vector<int> dist;
38     for (int v: graph[src]) {
39         if (minstored[v] == -1)
40             min_(graph, v, tgt);
41         dist.push_back(minstored[v]);
42     }
43     maxstored[src] = *max_element(begin(dist), end(dist)) + 1;

```

```

44     return maxstored[src];
45 }
46
47 int main() {
48     std::ios_base::sync_with_stdio(false);
49     int T;
50     cin >> T;
51     while (T--) {
52         int n, m, r, b;
53         cin >> n >> m >> r >> b;
54         vector< vector<int> > graph(n + 1);
55         for (int i = 0; i < m; ++i) {
56             int u, v;
57             cin >> u >> v;
58             graph[u].push_back(v);
59         }
60         memset(minstored, -1, S * sizeof(int));
61         memset(maxstored, -1, S * sizeof(int));
62         int red_moves = min_(graph, r, n);
63         int black_moves = min_(graph, b, n);
64         cout << (black_moves < red_moves
65             || (black_moves == red_moves && black_moves%2 == 0)) <<
66             endl;
67     }
68     return 0;
69 }

```

3.5 Network flows

3.5.1 Coin tossing

```

1  #include <iostream>
2  #include <algorithm>
3  #include <map>
4  #include <vector>
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/push_relabel_max_flow.hpp>
7  #include <boost/tuple/tuple.hpp>
8
9  using namespace std;
10 using namespace boost;
11
12 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13 typedef adjacency_list<vecS, vecS, directedS, no_property,
14     property<edge_capacity_t, long,
15     property<edge_residual_capacity_t, long,
16     property<edge_reverse_t, Traits::edge_descriptor> > > >
17     Graph;
18 typedef property_map<Graph, edge_capacity_t>::type
19     EdgeCapacityMap;
20 typedef property_map<Graph, edge_residual_capacity_t>::type
21     ResidualCapacityMap;
22 typedef property_map<Graph, edge_reverse_t>::type
23     ReverseEdgeMap;
24
25 typedef graph_traits<Graph>::vertex_descriptor Vertex;
26 typedef graph_traits<Graph>::edge_descriptor Edge;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt;
28
29 struct EdgeAdder {
30     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
31         ReverseEdgeMap &rev_edge)
32         : G(G), capacity(capacity), rev_edge(rev_edge) {}
33
34     void addEdge(int u, int v, long c) {
35         Edge e, reverseE;
36         tie(e, tuples::ignore) = add_edge(u, v, G);
37         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
38         capacity[e] = c;
39         capacity[reverseE] = 0;
40         rev_edge[e] = reverseE;

```

```

35     rev_edge[reverseE] = e;
36 }
37 Graph &G;
38 EdgeCapacityMap &capacity;
39 ReverseEdgeMap &rev_edge;
40 };
41
42 int main() {
43     std::ios_base::sync_with_stdio(false);
44     int T;
45     cin >> T;
46     while (T--) {
47         int n, m;
48         cin >> n >> m;
49         vector<int> final_scores(n, 0);
50         map<pair<int, int>, int> unknown_rounds;
51         for (int mm = 0; mm < m; ++mm) {
52             int a, b, c;
53             cin >> a >> b >> c;
54             switch (c) {
55                 case 0: {
56                     int tmp = max(a, b);
57                     a = min(a, b);
58                     b = tmp;
59                     auto p = make_pair(a, b);
60                     auto it = unknown_rounds.find(p);
61                     if (it != unknown_rounds.end()) {
62                         ++unknown_rounds[p];
63                     } else {
64                         unknown_rounds.insert(pair<pair<int, int>, int> (p,
65                             1));
66                     }
67                     break;
68                 }
69                 case 1:
70                     --final_scores[a];
71                     break;
72                 case 2:
73                     --final_scores[b];
74                     break;
75             }
76             bool impossible = false;

```

```

77     int s_sum = 0;
78     for (int i = 0; i < n; ++i) {
79         int s;
80         cin >> s;
81         final_scores[i] += s;
82         s_sum += s;
83         if (final_scores[i] < 0)
84             impossible = true;
85     }
86     if (impossible || s_sum != m) {
87         cout << "no" << endl;
88         continue;
89     }
90     Graph G(final_scores.size() + unknown_rounds.size());
91     EdgeCapacityMap capacity = get(edge_capacity, G);
92     ReverseEdgeMap rev_edge = get(edge_reverse, G);
93     ResidualCapacityMap res_capacity = get(
94         edge_residual_capacity, G);
95     EdgeAdder ea(G, capacity, rev_edge);
96
97     Vertex _source = add_vertex(G);
98     Vertex sink = add_vertex(G);
99     int i = 0;
100     for (auto it = unknown_rounds.begin(); it != unknown_rounds
101         .end(); ++it) {
102         ea.addEdge(_source, i, it->second);
103         ea.addEdge(i, it->first.first + unknown_rounds.size(), it
104             ->second);
105         ea.addEdge(i, it->first.second + unknown_rounds.size(),
106             it->second);
107         ++i;
108     }
109     i = unknown_rounds.size();
110     for (auto it = final_scores.begin(); it != final_scores.end
111         ()); ++it) {
112         ea.addEdge(i, sink, *it);
113         ++i;
114     }
115     push_relabel_max_flow(G, _source, sink);
116     EdgeIt e, eend;
117     for (tie(e, eend) = edges(G); e != eend; ++e) {
118         if (sink == target(*e, G) && res_capacity[*e] != 0) {
119             break;

```

```

115     }
116 }
117     cout << ((e == eend) ? "yes" : "no") << endl;
118 }
119 return 0;
120 }

```

3.5.2 Kingdom defence

Keywords— Minimum flow per edge

```

1 // the tricky thing about this problem is to add a lower bound
  // for the edges in
2 // the flow algorithm. For this purpose for each edge with a
  // lower bound:
3 // * Add an edge from source to the dest of the original edge
  // (weight=lower
4 //   bound)
5 // * Add an edge from source of original edge to sink (weight=
  // lower bound)
6 // * The original edge now has weight upper_bound -
  // lower_bound
7 #include <iostream>
8 #include <algorithm>
9 #include <map>
10 #include <vector>
11 #include <boost/graph/adjacency_list.hpp>
12 #include <boost/graph/push_relabel_max_flow.hpp>
13 #include <boost/tuple/tuple.hpp>
14
15 using namespace std;
16 using namespace boost;
17
18 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
19 typedef adjacency_list<vecS, vecS, directedS, no_property,
20   property<edge_capacity_t, long,
21   property<edge_residual_capacity_t, long,
22   property<edge_reverse_t, Traits::edge_descriptor> > > >
  Graph;
23 typedef property_map<Graph, edge_capacity_t>::type
  EdgeCapacityMap;
24 typedef property_map<Graph, edge_residual_capacity_t>::type
  ResidualCapacityMap;

```

```

25 typedef property_map<Graph, edge_reverse_t>::type
  ReverseEdgeMap;
26 typedef graph_traits<Graph>::vertex_descriptor Vertex;
27 typedef graph_traits<Graph>::edge_descriptor Edge;
28 typedef graph_traits<Graph>::edge_iterator EdgeIt; // Iterator
29
30 struct EdgeAdder {
31     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
32       ReverseEdgeMap &rev_edge)
33       : G(G), capacity(capacity), rev_edge(rev_edge) {}
34
35     void addEdge(int u, int v, long c) {
36         Edge e, reverseE;
37         tie(e, tuples::ignore) = add_edge(u, v, G);
38         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
39         capacity[e] = c;
40         capacity[reverseE] = 0;
41         rev_edge[e] = reverseE;
42         rev_edge[reverseE] = e;
43     }
44     Graph &G;
45     EdgeCapacityMap &capacity;
46     ReverseEdgeMap &rev_edge;
47 };
48
49 int main() {
50     int T;
51     cin >> T;
52     while (T--) {
53         int l, p; // locations and paths
54         cin >> l >> p;
55
56         Graph G(l);
57         EdgeCapacityMap capacity = get(edge_capacity, G);
58         ReverseEdgeMap rev_edge = get(edge_reverse, G);
59         EdgeAdder ea(G, capacity, rev_edge);
60
61         Vertex _source = add_vertex(G);
62         Vertex sink = add_vertex(G);
63
64         int total_flow = 0;
65         map<pair<int, int>, int> flows;
66         for (int i = 0; i < l; ++i) {

```

```

66     int g, d;
67     cin >> g >> d;
68     total_flow += d;
69     auto epair = make_pair(_source, i);
70     flows.insert(pair<pair<int, int>, int> (epair, g));
71     epair = make_pair(i, sink);
72     flows.insert(pair<pair<int, int>, int> (epair, d));
73 }
74 while (p-->0) {
75     int f, t, min_, max_;
76     cin >> f >> t >> min_ >> max_;
77     total_flow += min_;
78     ea.addEdge(f, t, max_ - min_);
79
80     auto epair = make_pair(_source, t);
81     flows[epair] += min_;
82     epair = make_pair(f, sink);
83     flows[epair] += min_;
84 }
85 for (auto it = flows.begin(); it != flows.end(); ++it)
86     ea.addEdge(it->first.first, it->first.second, it->second)
87         ;
88 cout << (total_flow == push_relabel_max_flow(G, _source,
89         sink) ? "yes" : "no")
90         << endl;
91 }
92 return 0;

```

3.5.3 Shopping trip

Keywords— Edge disjoint paths

```

1 #include <iostream>
2 #include <algorithm>
3 #include <vector>
4
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/push_relabel_max_flow.hpp>
7 #include <boost/graph/edmonds_karp_max_flow.hpp>
8 #include <boost/tuple/tuple.hpp>
9

```

```

10 using namespace std;
11 using namespace boost;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,
17     property<edge_reverse_t, Traits::edge_descriptor>>>>
18     Graph;
19 typedef property_map<Graph, edge_capacity_t>::type
20     EdgeCapacityMap;
21 typedef property_map<Graph, edge_residual_capacity_t>::type
22     ResidualCapacityMap;
23 typedef property_map<Graph, edge_reverse_t>::type
24     ReverseEdgeMap;
25 typedef graph_traits<Graph>::vertex_descriptor Vertex;
26 typedef graph_traits<Graph>::edge_descriptor Edge;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt; // Iterator
28
29 struct EdgeAdder {
30     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
31         ReverseEdgeMap &rev_edge)
32         : G(G), capacity(capacity), rev_edge(rev_edge) {}
33
34     void addEdge(int u, int v, long c) {
35         Edge e, reverseE;
36         tie(e, tuples::ignore) = add_edge(u, v, G);
37         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
38         capacity[e] = c;
39         capacity[reverseE] = 0;
40         rev_edge[e] = reverseE;
41         rev_edge[reverseE] = e;
42     }
43
44     Graph &G;
45     EdgeCapacityMap &capacity;
46     ReverseEdgeMap &rev_edge;
47 };
48
49 int main() {
50     std::ios_base::sync_with_stdio(false);
51     int T;
52     cin >> T;
53     while (T-->0) {

```



```

48     int n, m, s;
49     cin >> n >> m >> s;
50     int tmps = s;
51     Graph G(n);
52     Vertex sink = add_vertex(G);
53     EdgeCapacityMap capacity = get(edge_capacity, G);
54     ReverseEdgeMap rev_edge = get(edge_reverse, G);
55     EdgeAdder ea(G, capacity, rev_edge);
56     while (tmps--> 0) {
57         int store;
58         cin >> store;
59         ea.addEdge(store, sink, 1);
60     }
61     while (m--> 0) {
62         int u, v;
63         cin >> u >> v;
64         ea.addEdge(u, v, 1);
65         ea.addEdge(v, u, 1);
66     }
67
68     long flow = push_relabel_max_flow(G, 0, sink);
69     cout << (flow == s ? "yes" : "no") << endl;
70 }
71 return 0;
72 }

```

3.5.4 Tetris

Keywords— Vertex capacities

```

1  #include <iostream>
2  #include <algorithm>
3  #include <limits>
4  #include <vector>
5  #include <cassert>
6  #include <boost/graph/adjacency_list.hpp>
7  #include <boost/graph/push_relabel_max_flow.hpp>
8  #include <boost/tuple/tuple.hpp>
9
10 using namespace std;
11 using namespace boost;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;

```

```

14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,
17     property<edge_reverse_t, Traits::edge_descriptor> > > >
18     Graph;
19 typedef property_map<Graph, edge_capacity_t>::type
20     EdgeCapacityMap;
21 typedef property_map<Graph, edge_residual_capacity_t>::type
22     ResidualCapacityMap;
23 typedef property_map<Graph, edge_reverse_t>::type
24     ReverseEdgeMap;
25 typedef graph_traits<Graph>::vertex_descriptor Vertex;
26 typedef graph_traits<Graph>::edge_descriptor Edge;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt; // Iterator
28
29 struct EdgeAdder {
30     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
31         ReverseEdgeMap &rev_edge)
32         : G(G), capacity(capacity), rev_edge(rev_edge) {}
33
34     void addEdge(int u, int v, long c) {
35         Edge e, reverseE;
36         tie(e, tuples::ignore) = add_edge(u, v, G);
37         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
38         capacity[e] = c;
39         capacity[reverseE] = 0;
40         rev_edge[e] = reverseE;
41         rev_edge[reverseE] = e;
42     }
43
44     Graph &G;
45     EdgeCapacityMap &capacity;
46     ReverseEdgeMap &rev_edge;
47 };
48
49 int main() {
50     std::ios_base::sync_with_stdio(false);
51     int T;
52     cin >> T;
53     while (T--> 0) {
54         int w, n;
55         cin >> w >> n;
56         set< pair<int, int> > pieces;
57         int full_width_pieces = 0;

```

```

52   for (int i = 0; i < n; ++i) {
53       int a, b;
54       cin >> a >> b;
55       if (a > b)
56           swap(a, b);
57       if (a == 0 && b == w) // special case of full width piece
58           ++full_width_pieces;
59       else
60           pieces.insert(make_pair(a, b));
61   }
62   Graph G(2 * w);
63   EdgeCapacityMap capacity = get(edge_capacity, G);
64   ReverseEdgeMap rev_edge = get(edge_reverse, G);
65   ResidualCapacityMap res_capacity = get(
66       edge_residual_capacity, G);
67   EdgeAdder ea(G, capacity, rev_edge);
68   for (int i = 1; i < w; ++i) // vertex capacities
69       ea.addEdge(i - 1, i + w, 1);
70   for (auto p: pieces)
71       ea.addEdge(p.first + w, p.second - 1, numeric_limits<int>
72           >::max());
73   long flow = push_relabel_max_flow(G, w, w - 1);
74   cout << flow + full_width_pieces << endl;
75   }
76   return 0;
77 }

```

3.5.5 The phantom menace

Keywords— Vertex capacities

```

1  #include <iostream>
2  #include <algorithm>
3  #include <map>
4  #include <queue>
5  #include <vector>
6  #include <boost/graph/adjacency_list.hpp>
7  #include <boost/graph/push_relabel_max_flow.hpp>
8
9  using namespace std;
10 using namespace boost;
11
12 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;

```

```

13 typedef adjacency_list<vecS, vecS, directedS, no_property,
14     property<edge_capacity_t, long,
15     property<edge_residual_capacity_t, long,
16     property<edge_reverse_t, Traits::edge_descriptor> > > >
17     Graph;
18 typedef property_map<Graph, edge_capacity_t>::type
19     EdgeCapacityMap;
20 typedef property_map<Graph, edge_residual_capacity_t>::type
21     ResidualCapacityMap;
22 typedef property_map<Graph, edge_reverse_t>::type
23     ReverseEdgeMap;
24 typedef graph_traits<Graph>::vertex_descriptor Vertex;
25 typedef graph_traits<Graph>::edge_descriptor Edge;
26 typedef graph_traits<Graph>::edge_iterator EdgeIt;
27
28 struct EdgeAdder {
29     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
30         ReverseEdgeMap &rev_edge)
31         : G(G), capacity(capacity), rev_edge(rev_edge) {}
32
33     void addEdge(int u, int v, long c) {
34         Edge e, reverseE;
35         tie(e, tuples::ignore) = add_edge(u, v, G);
36         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
37         capacity[e] = c;
38         capacity[reverseE] = 0;
39         rev_edge[e] = reverseE;
40         rev_edge[reverseE] = e;
41     }
42
43     Graph &G;
44     EdgeCapacityMap &capacity;
45     ReverseEdgeMap &rev_edge;
46 };
47
48 int main() {
49     std::ios_base::sync_with_stdio(false);
50     int T;
51     cin >> T;
52     while (T--) {
53         int n, m, s, d;
54         cin >> n >> m >> s >> d;
55         // Define graph
56         Graph G(2 * n);

```

```

51 EdgeCapacityMap capacity = get(edge_capacity, G);
52 ReverseEdgeMap rev_edge = get(edge_reverse, G);
53 ResidualCapacityMap res_capacity = get(
    edge_residual_capacity, G);
54 EdgeAdder ea(G, capacity, rev_edge);
55 Vertex _source = add_vertex(G);
56 Vertex _sink = add_vertex(G);
57 for (int i = 0; i < n; ++i) // vertex capacity
58     ea.addEdge(i, i + n, 1);
59 for (int i = 0; i < m; ++i) {
60     int u, v;
61     cin >> u >> v;
62     ea.addEdge(u + n, v, 1);
63 }
64 for (int i = 0; i < s; ++i) {
65     int src;
66     cin >> src;
67     ea.addEdge(_source, src, 1);
68 }
69 for (int i = 0; i < d; ++i) {
70     int tgt;
71     cin >> tgt;
72     ea.addEdge(tgt + n, _sink, 1);
73 }
74 int flow = push_relabel_max_flow(G, _source, _sink);
75 cout << flow << endl;
76 }
77 return 0;
78 }

```

3.6 Linear programming

3.6.1 Diet

Keywords— floor_to_double

```

1  #include <iostream>
2  #include <CGAL/basic.h>
3  #include <CGAL/QP_models.h>
4  #include <CGAL/QP_functions.h>
5
6  #ifdef CGAL_USE_GMP
7  #include <CGAL/Gmpz.h>
8  typedef CGAL::Gmpz ET;
9  #else
10 #include <CGAL/MP_Float.h>
11 typedef CGAL::MP_Float ET;
12 #endif
13
14 typedef CGAL::Quadratic_program<int> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16
17 double floor_to_double(const CGAL::Quotient<ET>& x) {
18     double a = std::ceil(CGAL::to_double(x));
19     while (a > x) a -= 1;
20     while (a+1 <= x) a += 1;
21     return a;
22 }
23
24 int main() {
25     std::ios_base::sync_with_stdio(false);
26     while (true) {
27         int n, m;
28         std::cin >> n >> m;
29         if (n == 0 && m == 0)
30             break;
31
32         Program lp(CGAL::SMALLER, true, 0, false, 0); // all foods
33             >= 0
34         // nutrients
35         for (int i = 0; i < n; ++i) {
36             int min_i, max_i;
37             std::cin >> min_i >> max_i;
38             // ax <= b --> -ax >= -b

```

```

38     lp.set_b(2*i, -min_i); // lower bound for nutrient
39     lp.set_b(2*i + 1, max_i); // upper bound for nutrient
40 }
41 // foods
42 for (int i = 0; i < m; ++i) {
43     int p;
44     std::cin >> p;
45     lp.set_c(i, p); // min equation price multiplier (min \
        sum_i p_i*f_i)
46     for (int j = 0; j < n; ++j) {
47         int c;
48         std::cin >> c;
49         lp.set_a(i, 2*j, -c); // factors for lower bound
            nutrient
50         lp.set_a(i, 2*j + 1, c); // factors for upper bound
            nutrient
51     }
52 }
53 Solution s = CGAL::solve_nonnegative_linear_program(lp, ET
    ());
54 if (s.is_optimal()) {
55     std::cout << (int) floor_to_double(s.objective_value())
        << std::endl;
56 } else {
57     std::cout << "No such diet." << std::endl;
58 }
59 }
60 return 0;
61 }

```

3.6.2 Inball

Keywords—

```

1 #include <iostream>
2 #include <CGAL/basic.h>
3 #include <CGAL/QP_models.h>
4 #include <CGAL/QP_functions.h>
5
6 #ifdef CGAL_USE_GMP
7 #include <CGAL/Gmpz.h>
8 typedef CGAL::Gmpz ET;
9 #else

```

```

10 #include <CGAL/MP_Float.h>
11 typedef CGAL::MP_Float ET;
12 #endif
13
14 typedef CGAL::Quadratic_program<int> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     while (true) {
20         int n, d;
21         std::cin >> n;
22         if (n == 0)
23             break;
24         std::cin >> d;
25
26         // Let c be the center of the ball and r the radius. The
            constraints can be
27         // phrased as:
28         //  $a^T (x + r \cdot a / \|a\|_2) \leq b$ 
29         // The above, in words: a is normal to  $a^T$  (i.e.
            perpendicular) and
30         //  $a / \|a\|_2$  is then the normal unit vector. In addition,
31         //  $a^T a = (\|a\|_2)^2$ . So the constraint finally is :
32         //  $a^T x + r \cdot \|a\|_2 \leq b$ 
33         const int R = d;
34         Program lp(CGAL::SMALLER, false, 0, false, 0);
35         lp.set_l(R, true, 0); // the radius cannot be negative
36         lp.set_c(R, -1); // min -R = max R
37         for (int i = 0; i < n; ++i) {
38             int sq_sum = 0; // to compute the norm
39             for (int j = 0; j < d; ++j) {
40                 int a;
41                 std::cin >> a;
42                 lp.set_a(j, i, a);
43                 sq_sum += a*a;
44             }
45             lp.set_a(R, i, sqrt(sq_sum));
46
47             int b;
48             std::cin >> b;
49             lp.set_b(i, b);
50         }

```

```

51 Solution s = CGAL::solve_linear_program(lp, ET());
52 if (s.is_optimal()) {
53     std::cout << (int) CGAL::to_double(-s.objective_value())
54         << std::endl;
55 } else if (s.is_unbounded()) {
56     std::cout << "inf" << std::endl;
57 } else {
58     std::cout << "none" << std::endl;
59 }
60 return 0;
61 }

```

3.6.3 Maximize it

Keywords— Quadratic programming

```

1  #include <iostream>
2  #include <CGAL/basic.h>
3  #include <CGAL/QP_models.h>
4  #include <CGAL/QP_functions.h>
5
6  #ifdef CGAL_USE_GMP
7  #include <CGAL/Gmpz.h>
8  typedef CGAL::Gmpz ET;
9  #else
10 #include <CGAL/MP_Float.h>
11 typedef CGAL::MP_Float ET;
12 #endif
13
14 typedef CGAL::Quadratic_program<int> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16
17 double ceil_to_double(const CGAL::Quotient<ET>& x) {
18     double a = std::ceil(CGAL::to_double(x));
19     while (a < x) a += 1;
20     while (a-1 >= x) a -= 1;
21     return a;
22 }
23
24 double floor_to_double(const CGAL::Quotient<ET>& x) {
25     double a = std::ceil(CGAL::to_double(x));
26     while (a > x) a -= 1;

```

```

27 while (a+1 <= x) a += 1;
28 return a;
29 }
30
31 int main() {
32     std::ios_base::sync_with_stdio(false);
33     while (true) {
34         int p, a, b;
35         std::cin >> p;
36         if (p == 0)
37             break;
38         std::cin >> a >> b;
39
40         const int X = 0;
41         const int Y = 1;
42         const int Z = 2;
43         Program final_qp;
44         if (p == 1) {
45             // CGAL::SMALLER makes the restrictions <=
46             Program qp(CGAL::SMALLER, true, 0, false, 0); // x, y >=
47                 0
48             qp.set_a(X, 0, 1); qp.set_a(Y, 0, 1); qp.set_b(0, 4); //
49                 x+y <= 4
50             qp.set_a(X, 1, 4); qp.set_a(Y, 1, 2); qp.set_b(1, a * b);
51                 // 4x+2y <= ab
52             qp.set_a(X, 2, -1); qp.set_a(Y, 2, 1); qp.set_b(2, 1); //
53                 -x+y <= 1
54             // max f(x) = - min -f(x)
55             // -by + ax^2
56             qp.set_c(Y, -b); qp.set_d(X, X, 2*a);
57             final_qp = qp;
58         } else if (p == 2) {
59             // CGAL::LARGER makes the restrictions >=
60             Program qp(CGAL::LARGER, false, 0, false, 0);
61             qp.set_u(X, true, 0); qp.set_u(Y, true, 0); // x, y <= 0
62             qp.set_a(X, 0, 1); qp.set_a(Y, 0, 1); qp.set_b(0, -4); //
63                 xy >= -4
64             // 4x+2y+z^2 >= -ab
65             qp.set_a(X, 1, 4); qp.set_a(Y, 1, 2); qp.set_a(Z, 1, 1);
66                 qp.set_b(1, -(a*b));
67             qp.set_a(X, 2, -1); qp.set_a(Y, 2, 1); qp.set_b(2, -1);
68                 // -x*y >= -1
69             // min ax^2 + by + z^4

```

```

64     qp.set_d(X, X, 2*a); qp.set_c(Y, b); qp.set_d(Z, Z, 2);
65     final_qp = qp;
66 }
67 Solution s = CGAL::solve_quadratic_program(final_qp, ET());
68 if (s.is_optimal() && p == 1) {
69     std::cout << (int) floor_to_double(-s.objective_value())
70     << std::endl;
71 } else if (s.is_optimal() && p == 2) {
72     std::cout << (int) ceil_to_double(s.objective_value()) <<
73     std::endl;
74 } else if (s.is_unbounded()) {
75     std::cout << "unbounded" << std::endl;
76 } else {
77     std::cout << "no" << std::endl;
78 }
79 return 0;

```

3.6.4 Portfolios

Keywords— Quadratic programming

```

1  #include <iostream>
2  #include <CGAL/basic.h>
3  #include <CGAL/QP_models.h>
4  #include <CGAL/QP_functions.h>
5
6  #ifdef CGAL_USE_GMP
7  #include <CGAL/Gmpz.h>
8  typedef CGAL::Gmpz ET;
9  #else
10 #include <CGAL/MP_Float.h>
11 typedef CGAL::MP_Float ET;
12 #endif
13
14 typedef CGAL::Quadratic_program<int> Program;
15 typedef CGAL::Quadratic_program_solution<ET> Solution;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19
20     while (true) {

```

```

21     int n, m;
22     std::cin >> n >> m;
23     if (n == 0 && m == 0)
24         break;
25
26     const int ret_const = 0; // expected return constraint
27     const int cos_const = 1; // cost constraint
28     Program qp(CGAL::SMALLER, true, 0, false, 0); // all alphas
29         >= 0
30     for (int i = 0; i < n; ++i) {
31         int c, r;
32         std::cin >> c >> r;
33         qp.set_a(i, ret_const, r); // component for return lower
34             bound inequation
35         qp.set_a(i, cos_const, c); // component for cost upper
36             bound inequation
37     }
38     qp.set_r(ret_const, CGAL::LARGER);
39     for (int i = 0; i < n; ++i) {
40         for (int j = 0; j < n; ++j) {
41             int v;
42             std::cin >> v;
43             if (j <= i)
44                 qp.set_d(i, j, 2*v);
45         }
46     }
47     for (int i = 0; i < m; ++i) {
48         int C, R, V;
49         std::cin >> C >> R >> V;
50         qp.set_b(ret_const, R);
51         qp.set_b(cos_const, C);
52
53         Solution s = CGAL::solve_nonnegative_quadratic_program(qp,
54             ET());
55         std::cout << (s.is_optimal() && CGAL::to_double(s.
56             objective_value()) <= V
57             ? "Yes." : "No.") << std::endl;
58     }
59 }
60 return 0;

```

3.6.5 Portfolios revisited

Keywords— Binary search, Quadratic programming

```
1 #include <iostream>
2 #include <limits>
3 #include <CGAL/basic.h>
4 #include <CGAL/QP_models.h>
5 #include <CGAL/QP_functions.h>
6
7 #ifdef CGAL_USE_GMP
8 #include <CGAL/Gmpz.h>
9 typedef CGAL::Gmpz ET;
10 #else
11 #include <CGAL/MP_Float.h>
12 typedef CGAL::MP_Float ET;
13 #endif
14
15 typedef CGAL::Quadratic_program<int> Program;
16 typedef CGAL::Quadratic_program_solution<ET> Solution;
17
18 double find_result(Solution &s, std::vector<int> &returns) {
19     double result = 0;
20     int idx = 0;
21     for (auto it = s.variable_values_begin(); it !=
22          s.variable_values_end(); ++it) {
23         result += returns[idx] * CGAL::to_double(*it);
24         ++idx;
25     }
26     return result;
27 }
28
29 int main() {
30     std::ios_base::sync_with_stdio(false);
31     while (true) {
32         int n, m;
33         std::cin >> n >> m;
34         if (n == 0 && m == 0)
35             break;
36
37         std::vector<int> returns(n);
38         const int ret_const = 0; // expected return constraint
39         const int cos_const = 1; // cost constraint
40         Program qp(CGAL::SMALLER, true, 0, false, 0); // all alphas
```

```

41         >= 0
42         for (int i = 0; i < n; ++i) {
43             int c, r;
44             std::cin >> c >> r;
45             returns[i] = r;
46             qp.set_a(i, ret_const, r); // component for return lower
47                                     bound inequation
48             qp.set_a(i, cos_const, c); // component for cost upper
49                                     bound inequation
50         }
51         qp.set_r(ret_const, CGAL::LARGER);
52         for (int i = 0; i < n; ++i) {
53             for (int j = 0; j < n; ++j) {
54                 int v;
55                 std::cin >> v;
56                 if (j <= i)
57                     qp.set_d(i, j, 2*v);
58             }
59         }
60         for (int i = 0; i < m; ++i) {
61             int C, V;
62             std::cin >> C >> V;
63             qp.set_b(cos_const, C);
64
65             double best_result = std::numeric_limits<double>::min();
66             int lower = 0;
67             int upper = 40 * 1000000;
68             while (lower + 1 != upper) {
69                 int middle = lower + (upper - lower) / 2;
70                 //std::cout << "[" << lower << ", " << upper << "]" <<
71                 //std::endl;
72                 //std::cout << " " << middle << std::endl;
73                 qp.set_b(ret_const, middle);
74
75                 Solution s = CGAL::solve_nonnegative_quadratic_program(
76                     qp, ET());
77                 if (s.is_optimal() && CGAL::to_double(s.objective_value
78                     ()) <= V) {
79                     lower = middle;
80                     best_result = std::max(best_result, find_result(s,
81                         returns));
82                 } else {
83                     upper = middle;
84                 }
85             }
86         }
87     }
88 }
```

```

77     }
78     }
79     std::cout << best_result << std::endl;
80 }
81 }
82 return 0;
83 }

```

3.7 CGAL proximity structures

3.7.1 Bistro

Keywords— Triangulation, nearest_vertex Distance to closest point in triangulation.

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <CGAL/Delaunay_triangulation_2.h>
3  #include <iostream>
4
5  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
6  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
7
8  int main() {
9      std::ios_base::sync_with_stdio(false);
10     while (true) {
11         int n, m;
12         std::cin >> n;
13         if (n == 0)
14             break;
15
16         std::vector<K::Point_2> restaurants;
17         restaurants.reserve(n);
18         while (n--) {
19             int x, y;
20             std::cin >> x >> y;
21             restaurants.push_back(K::Point_2(x, y));
22         }
23         Triangulation t;
24         t.insert(restaurants.begin(), restaurants.end());
25         std::cin >> m;
26         while (m--) {
27             int x, y;
28             std::cin >> x >> y;
29             auto proposed_loc = K::Point_2(x, y);
30             auto nearest_rest = t.nearest_vertex(proposed_loc)->point
31                 ();
32             std::cout << (unsigned long long)
33                 CGAL::to_double(CGAL::squared_distance(nearest_rest,
34                 proposed_loc))
35                 << std::endl;
36         }
37     }
38     return 0;

```



```
37 }
```

3.7.2 Germs

Keywords—

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <CGAL/Delaunay_triangulation_2.h>
3 #include <iostream>
4 #include <map>
5
6 typedef unsigned int uint;
7 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
9
10 double ceil_to_double(const K::FT& x) {
11     double a = std::ceil(CGAL::to_double(x));
12     while (a < x) a += 1;
13     while (a-1 >= x) a -= 1;
14     return a;
15 }
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     while (true) {
20         int n;
21         std::cin >> n;
22         if (n == 0)
23             break;
24
25         // rectangle
26         int l, b, r, t;
27         std::cin >> l >> b >> r >> t;
28         auto top = K::Segment_2(K::Point_2(l, t), K::Point_2(r, t))
29             ;
30         auto right = K::Segment_2(K::Point_2(r, t), K::Point_2(r, b
31             ));
32         auto bottom = K::Segment_2(K::Point_2(l, b), K::Point_2(r,
33             b));
34         auto left = K::Segment_2(K::Point_2(l, t), K::Point_2(l, b
35             ));
36         // bacteria
37         std::vector<K::Point_2> bacteria;
```

```
34 bacteria.reserve(n);
35 std::map<K::Point_2, double> min_time;
36 while (n--> 0) {
37     int x, y;
38     std::cin >> x >> y;
39     auto p = K::Point_2(x, y);
40     bacteria.push_back(p);
41     auto dist = CGAL::min(squared_distance(top, p),
42         CGAL::min(squared_distance(right, p),
43             CGAL::min(squared_distance(bottom, p),
44                 squared_distance(left, p))));
45     double time_to_border = ceil_to_double(sqrt(sqrt(dist) -
46         0.5));
47     min_time[p] = time_to_border;
48 }
49 Triangulation tri;
50 tri.insert(bacteria.begin(), bacteria.end());
51 for (auto e = tri.finite_edges_begin(); e != tri.
52     finite_edges_end(); ++e) {
53     auto dist = tri.segment(e).squared_length();
54     double time = ceil_to_double(sqrt((sqrt(dist) - 1) / 2));
55     auto p1 = tri.segment(e).point(0);
56     auto p2 = tri.segment(e).point(1);
57     min_time[p1] = std::min(min_time[p1], time);
58     min_time[p2] = std::min(min_time[p2], time);
59 }
60 std::vector<double> times;
61 times.reserve(bacteria.size());
62 for (auto it = min_time.begin(); it != min_time.end(); ++it
63     )
64     times.push_back(it->second);
65 std::sort(times.begin(), times.end());
66 std::cout << (uint) times[0] << " "
67     << (uint) times[times.size() / 2] << " "
68     << times[times.size() - 1] << std::endl;
69 }
70 return 0;
71 }
```

3.7.3 Graypes

Keywords— Minimum distance between all possible pairs of elements in a triangu-

lation.

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <CGAL/Delaunay_triangulation_2.h>
3 #include <iostream>
4
5 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
6 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
7 typedef Triangulation::Edge_iterator Edge_iterator;
8
9 double ceil_to_double(const K::FT& x) {
10     double a = std::ceil(CGAL::to_double(x));
11     while (a < x) a += 1;
12     while (a-1 >= x) a -= 1;
13     return a;
14 }
15
16 int main() {
17     std::ios_base::sync_with_stdio(false);
18     while (true) {
19         int n;
20         std::cin >> n;
21         if (n == 0)
22             break;
23
24         std::vector<K::Point_2> graypes;
25         graypes.reserve(n);
26         while (n-->) {
27             int x, y;
28             std::cin >> x >> y;
29             graypes.push_back(K::Point_2(x, y));
30         }
31         Triangulation t;
32         t.insert(graypes.begin(), graypes.end());
33         bool first = true;
34         K::FT min_dist;
35         for (auto e = t.finite_edges_begin(); e != t.
            finite_edges_end(); ++e) {
36             auto dist = t.segment(e).squared_length();
37             if (dist < min_dist || first) {
38                 first = false;
39                 min_dist = dist;
40             }
41         }
```

```
42         std::cout << (int) ceil_to_double(sqrt(min_dist) / 2 * 100)
43             << std::endl;
44     }
45     return 0;
46 }
```

3.7.4 H1N1

Keywords— Motion planning, Triangulation DFS

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <CGAL/Delaunay_triangulation_2.h>
3 #include <iostream>
4 #include <map>
5 #include <queue>
6
7 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
9 typedef Triangulation::Face_handle Face_handle;
10
11 int main() {
12     std::ios_base::sync_with_stdio(false);
13     while (true) {
14         int n, m;
15         std::cin >> n;
16         if (n == 0)
17             break;
18
19         std::vector<K::Point_2> infected;
20         infected.reserve(n);
21         while (n-->) {
22             int x, y;
23             std::cin >> x >> y;
24             infected.push_back(K::Point_2(x, y));
25         }
26         Triangulation t;
27         t.insert(infected.begin(), infected.end());
28         std::cin >> m;
29         while (m-->) {
30             int x, y;
31             double d;
32             std::cin >> x >> y >> d;
33             auto healthy = K::Point_2(x, y);
```

```

34     auto nearest_infected = t.nearest_vertex(healthy)->point
      ();
35     // already inside the infection area
36     if (CGAL::to_double(CGAL::squared_distance(
      nearest_infected, healthy)) < d) {
37         std::cout << "n";
38         continue;
39     }
40     auto face = t.locate(healthy);
41     // already outside the infection area
42     if (t.is_infinite(face->vertex(0)) || t.is_infinite(face
      ->vertex(1))
43         || t.is_infinite(face->vertex(2))) {
44         std::cout << "y";
45         continue;
46     }
47     // BFS on the faces of the triangulation starting at the
      one containing
48     // the query point. Add neighbors iff the edge between it
      and the current
49     // face has squared length >= 4d [(2*sqrt(d))^2]
50     std::map<Face_handle, bool> visited;
51     bool path_found = false;
52     std::queue<Face_handle> q;
53     q.push(face);
54     while (!q.empty()) {
55         auto f = q.front();
56         q.pop();
57         for (int i = 0; i < 3; ++i) {
58             // see slide 19 of CGAL proximity structures
59             int i_v1 = i % 3;
60             int i_v2 = (i + 1) % 3;
61             int i_neighbor = (i + 2) % 3;
62
63             auto p1 = f->vertex(i_v1)->point();
64             auto p2 = f->vertex(i_v2)->point();
65             if (CGAL::to_double(CGAL::squared_distance(p1, p2))
      >= 4*d) {
66                 auto neighbor = f->neighbor(i_neighbor);
67                 if (t.is_infinite(neighbor->vertex(0))
68                     || t.is_infinite(neighbor->vertex(1))
69                     || t.is_infinite(neighbor->vertex(2))) {
70                     path_found = true;

```

```

71             break;
72         }
73         if (!visited[neighbor]) {
74             visited[neighbor] = true;
75             q.push(neighbor);
76         }
77     }
78     }
79     if (path_found)
80         break;
81     }
82     if (path_found)
83         std::cout << "y";
84     else
85         std::cout << "n";
86     }
87     std::cout << std::endl;
88 }
89 return 0;
90 }

```

3.7.5 Light the stage

Keywords— Triangulation

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <CGAL/Delaunay_triangulation_2.h>
3  #include <iostream>
4  #include <limits>
5  #include <vector>
6
7  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
8  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
9
10 int main() {
11     std::ios_base::sync_with_stdio(false);
12     int T;
13     std::cin >> T;
14     while (T-->0) {
15         int m, n;
16         std::cin >> m >> n;
17
18         std::vector<K::Point_2> participants;

```

<pre> 19 std::vector<int> participant_radius; 20 std::vector<K::Point_2> lamps; 21 std::vector<int> time_dead(m, std::numeric_limits<int>::max ()); 22 for (int i = 0; i < m; ++i) { 23 int x, y, r; 24 std::cin >> x >> y >> r; 25 participants.push_back(K::Point_2(x, y)); 26 participant_radius.push_back(r); 27 } 28 int h; 29 std::cin >> h; 30 for (int i = 0; i < n; ++i) { 31 int x, y; 32 std::cin >> x >> y; 33 lamps.push_back(K::Point_2(x, y)); 34 } 35 Triangulation t; 36 t.insert(lamps.begin(), lamps.end()); 37 for (int i = 0; i < m; ++i) { 38 // check if intersection area > 0 39 K::Point_2 nearest_lamp = t.nearest_vertex(participants[i]]->point()); 40 double closest_dist = CGAL::squared_distance(nearest_lamp , 41 participants [i]); 42 double radius_intersect = h + participant_radius[i]; 43 double radius_intersect2 = radius_intersect * radius_intersect; 44 if (radius_intersect2 <= closest_dist) // the closest is already too far 45 continue; 46 for (int j = 0; j < n; ++j) { 47 double dist = CGAL::squared_distance(participants[i], lamps[j]); 48 if (radius_intersect2 > dist) { 49 time_dead[i] = j; 50 break; 51 } 52 } 53 } 54 int winner_time = *std::max_element(time_dead.begin(), </pre>	<pre> time_dead.end()); 55 for (int i = 0; i < m; ++i) 56 if (time_dead[i] == winner_time) std::cout << i << " "; 57 std::cout << std::endl; 58 } 59 return 0; 60 } </pre>
---	---

3.8 Minimum cut, Bipartite matching

3.8.1 Algocoon

Keywords— Minimum cut

Check https://judge.inf.ethz.ch/doc/course/algocoon_group.pdf

```
1 #include <iostream>
2 #include <algorithm>
3 #include <queue>
4 #include <set>
5 #include <vector>
6 #include <boost/graph/adjacency_list.hpp>
7 #include <boost/graph/push_relabel_max_flow.hpp>
8 #include <boost/graph/edmonds_karp_max_flow.hpp>
9 #include <boost/tuple/tuple.hpp>
10
11 using namespace std;
12 using namespace boost;
13
14 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
15 typedef adjacency_list<vecS, vecS, directedS, no_property,
16     property<edge_capacity_t, long,
17     property<edge_residual_capacity_t, long,
18     property<edge_reverse_t, Traits::edge_descriptor> > > >
19     Graph;
20
21 typedef property_map<Graph, edge_capacity_t>::type
22     EdgeCapacityMap;
23 typedef property_map<Graph, edge_residual_capacity_t>::type
24     ResidualCapacityMap;
25 typedef property_map<Graph, edge_reverse_t>::type
26     ReverseEdgeMap;
27
28 typedef graph_traits<Graph>::vertex_descriptor Vertex;
29 typedef graph_traits<Graph>::edge_descriptor Edge;
30 typedef graph_traits<Graph>::edge_iterator EdgeIt;
31 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
32
33 struct EdgeAdder {
34     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
35         ReverseEdgeMap &rev_edge)
36         : G(G), capacity(capacity), rev_edge(rev_edge) {}
37
38     void addEdge(int u, int v, long c) {
39         Edge e, reverseE;
```

```
33 tie(e, tuples::ignore) = add_edge(u, v, G);
34 tie(reverseE, tuples::ignore) = add_edge(v, u, G);
35 capacity[e] = c;
36 capacity[reverseE] = 0;
37 rev_edge[e] = reverseE;
38 rev_edge[reverseE] = e;
39 }
40 Graph &G;
41 EdgeCapacityMap &capacity;
42 ReverseEdgeMap &rev_edge;
43 };
44
45 int main() {
46     std::ios_base::sync_with_stdio(false);
47     int T;
48     cin >> T;
49     while (T--) {
50         int n, m;
51         cin >> n >> m;
52         Graph G(n);
53         EdgeCapacityMap capacity = get(edge_capacity, G);
54         ReverseEdgeMap rev_edge = get(edge_reverse, G);
55         ResidualCapacityMap res_capacity = get(
56             edge_residual_capacity, G);
57         EdgeAdder ea(G, capacity, rev_edge);
58         while (m--) {
59             int a, b, c;
60             cin >> a >> b >> c;
61             ea.addEdge(a, b, c);
62         }
63
64         int best_source = -1;
65         int best_sink = -1;
66         int min_flow = numeric_limits<int>::max();
67         set<pair<int, int> > explored_pairs;
68         for (int i = 1; i < n; ++i) {
69             int flow = push_relabel_max_flow(G, 0, i);
70             if (flow < min_flow) {
71                 min_flow = flow;
72                 best_source = 0;
73                 best_sink = i;
74             }
75         }
76         flow = push_relabel_max_flow(G, i, 0);
```

```

75     if (flow < min_flow) {
76         min_flow = flow;
77         best_source = i;
78         best_sink = 0;
79     }
80 }
81
82 push_relabel_max_flow(G, best_source, best_sink);
83 vector<int> vis(n, false);
84 vis[best_source] = true;
85 std::queue<int> Q;
86 Q.push(best_source);
87 while (!Q.empty()) {
88     const int u = Q.front();
89     Q.pop();
90     OutEdgeIt ebegin, eend;
91     for (tie(ebegin, eend) = out_edges(u, G); ebegin != eend; ++
        ebegin) {
92         const int v = target(*ebegin, G);
93         if (res_capacity[*ebegin] == 0 || vis[v]) continue;
94         vis[v] = true;
95         Q.push(v);
96     }
97 }
98
99 cout << min_flow << endl;
100 cout << count(vis.begin(), vis.end(), true);
101 for (int i = 0; i < n; ++i) {
102     if (vis[i]) cout << " " << i;
103 }
104 cout << endl;
105 }
106 return 0;
107 }

```

3.8.2 Knights

Keywords— Minimum vertex cover

```

1 #include <iostream>
2 #include <algorithm>
3 #include <limits>
4 #include <map>

```

```

5 #include <queue>
6 #include <vector>
7 #include <boost/graph/adjacency_list.hpp>
8 #include <boost/graph/push_relabel_max_flow.hpp>
9
10 using namespace std;
11 using namespace boost;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,
17     property<edge_reverse_t, Traits::edge_descriptor> > > >
18     Graph;
19 typedef property_map<Graph, edge_capacity_t>::type
20     EdgeCapacityMap;
21 typedef property_map<Graph, edge_residual_capacity_t>::type
22     ResidualCapacityMap;
23 typedef property_map<Graph, edge_reverse_t>::type
24     ReverseEdgeMap;
25
26 typedef graph_traits<Graph>::vertex_descriptor Vertex;
27 typedef graph_traits<Graph>::edge_descriptor Edge;
28 typedef graph_traits<Graph>::edge_iterator EdgeIt;
29 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
30
31 struct EdgeAdder {
32     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
33         ReverseEdgeMap &rev_edge)
34         : G(G), capacity(capacity), rev_edge(rev_edge) {}
35
36     void addEdge(int u, int v, long c) {
37         Edge e, reverseE;
38         tie(e, tuples::ignore) = add_edge(u, v, G);
39         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
40         capacity[e] = c;
41         capacity[reverseE] = 0;
42         rev_edge[e] = reverseE;
43         rev_edge[reverseE] = e;
44     }
45 }
46
47 Graph &G;
48 EdgeCapacityMap &capacity;
49 ReverseEdgeMap &rev_edge;
50 };

```

```

43
44 void connect(map<pair<int, int>, Vertex> &mapping, EdgeAdder &
    ea, int u, int i, int j) {
45     auto it = mapping.find(make_pair(i, j));
46     if (it != mapping.end()) {
47         Vertex v = it->second;
48         ea.addEdge(u, v, 1);
49     }
50 }
51
52 int main() {
53     int T;
54     cin >> T;
55     while (T--) {
56         int n;
57         cin >> n;
58         map<pair<int, int>, Vertex> mapping;
59         int n_fields = 0;
60         for (int i = 0; i < n; ++i) {
61             for (int j = 0; j < n; ++j) {
62                 bool field;
63                 cin >> field;
64                 if (field) {
65                     mapping.insert(make_pair(make_pair(i, j), n_fields));
66                     ++n_fields;
67                 }
68             }
69         }
70         vector<Vertex> black; // black vertices
71         vector<Vertex> white; // white vertices
72         Graph G(n_fields);
73         EdgeCapacityMap capacity = get(edge_capacity, G);
74         ReverseEdgeMap rev_edge = get(edge_reverse, G);
75         ResidualCapacityMap res_capacity = get(
            edge_residual_capacity, G);
76         EdgeAdder ea(G, capacity, rev_edge);
77         Vertex _source = add_vertex(G);
78         Vertex _sink = add_vertex(G);
79         for (auto it = mapping.begin(); it != mapping.end(); ++it)
80             {
81                 auto pos = it->first;
82                 int i = pos.first;
83                 int j = pos.second;
84
85                 Vertex u = it->second;
86                 if ((i + j)%2 == 0) { // black
87                     black.push_back(u);
88                     ea.addEdge(_source, u, 1);
89
90                     connect(mapping, ea, u, i - 1, j - 2);
91                     connect(mapping, ea, u, i - 2, j - 1);
92                     connect(mapping, ea, u, i + 1, j - 2);
93                     connect(mapping, ea, u, i + 2, j - 1);
94                     connect(mapping, ea, u, i - 1, j + 2);
95                     connect(mapping, ea, u, i - 2, j + 1);
96                     connect(mapping, ea, u, i + 1, j + 2);
97                     connect(mapping, ea, u, i + 2, j + 1);
98                 } else { // white
99                     white.push_back(u);
100                     ea.addEdge(u, _sink, 1);
101                 }
102             }
103         push_relabel_max_flow(G, _source, _sink);
104         vector<int> vis(n_fields + 2, false);
105         vis[_source] = true;
106         std::queue<int> Q;
107         Q.push(_source);
108         while (!Q.empty()) {
109             const int u = Q.front();
110             Q.pop();
111             OutEdgeIt ebeg, eend;
112             for (tie(ebeg, eend) = out_edges(u, G); ebeg != eend; ++
                ebeg) {
113                 const int v = target(*ebeg, G);
114                 if (res_capacity[*ebeg] == 0 || vis[v]) continue;
115                 vis[v] = true;
116                 Q.push(v);
117             }
118         }
119         int vertex_cover = 0;
120         for (auto it = black.begin(); it != black.end(); ++it)
121             if (!vis[*it]) ++vertex_cover;
122         for (auto it = white.begin(); it != white.end(); ++it)
123             if (vis[*it]) ++vertex_cover;
124         cout << n_fields - vertex_cover << endl;
125     }
126     return 0;

```

125 }

3.8.3 New hope

Keywords— Bipartite graph, is_bipartite, Minimum vertex cover
50 points. It only works when there is a unique stormtrooper per command center.

```

1  #include <iostream>
2  #include <algorithm>
3  #include <queue>
4  #include <vector>
5  #include <boost/graph/adjacency_list.hpp>
6  #include <boost/graph/bipartite.hpp>
7  #include <boost/graph/push_relabel_max_flow.hpp>
8
9  using namespace std;
10 using namespace boost;
11
12 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13 typedef adjacency_list<vecS, vecS, directedS, no_property,
14     property<edge_capacity_t, long,
15     property<edge_residual_capacity_t, long,
16     property<edge_reverse_t, Traits::edge_descriptor> > > >
17     Graph;
18 typedef property_map<Graph, edge_capacity_t>::type
19     EdgeCapacityMap;
20 typedef property_map<Graph, edge_residual_capacity_t>::type
21     ResidualCapacityMap;
22 typedef property_map<Graph, edge_reverse_t>::type
23     ReverseEdgeMap;
24 typedef graph_traits<Graph>::vertex_descriptor Vertex;
25 typedef graph_traits<Graph>::vertex_iterator VertexIt;
26 typedef graph_traits<Graph>::edge_descriptor Edge;
27 typedef graph_traits<Graph>::edge_iterator EdgeIt;
28 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
29 // for bipartition
30 typedef std::vector<default_color_type> partition_t;
31 typedef typename property_map<Graph, vertex_index_t>::type
32     index_map_t;
33 typedef iterator_property_map<partition_t::iterator,
34     index_map_t> partition_map_t;

```

```

30 struct EdgeAdder {
31     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
32         ReverseEdgeMap &rev_edge)
33         : G(G), capacity(capacity), rev_edge(rev_edge) {}
34
35     void addEdge(int u, int v, long c) {
36         Edge e, reverseE;
37         tie(e, tuples::ignore) = add_edge(u, v, G);
38         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
39         capacity[e] = c;
40         capacity[reverseE] = 0;
41         rev_edge[e] = reverseE;
42         rev_edge[reverseE] = e;
43     }
44     Graph &G;
45     EdgeCapacityMap &capacity;
46     ReverseEdgeMap &rev_edge;
47 };
48
49 int main() {
50     int T;
51     cin >> T;
52     while (T--) {
53         int k, s, m;
54         cin >> k >> s >> m;
55         int num_vert = k * s;
56         Graph G(num_vert);
57         for (int i = 0; i < m; ++i) {
58             int u, v, h;
59             cin >> u >> v >> h;
60             for (int j = 0; j < h; ++j) {
61                 int x, y;
62                 cin >> x >> y;
63                 add_edge(u * s + x, v * s + y, G);
64             }
65         }
66         // build bipartite graph
67         partition_t partition(num_vertices(G));
68         partition_map_t partition_map(partition.begin(), get(
69             vertex_index, G));
70         bool is_bip = is_bipartite(G, get(vertex_index, G),
71             partition_map);
72         Graph bipG(num_vert);

```



```

70 EdgeCapacityMap capacity = get(edge_capacity, bipG);
71 ReverseEdgeMap rev_edge = get(edge_reverse, bipG);
72 ResidualCapacityMap res_capacity = get(
    edge_residual_capacity, bipG);
73 EdgeAdder ea(bipG, capacity, rev_edge);
74 Vertex _source = add_vertex(bipG);
75 Vertex _sink = add_vertex(bipG);
76 if (!is_bip) continue;
77 vector<Vertex> white;
78 vector<Vertex> black;
79 VertexIt vit, vend;
80 for (boost::tie(vit, vend) = vertices(G); vit != vend; ++
    vit) {
81     auto color = get(partition_map, *vit);
82     if (color == color_traits<default_color_type>::white()) {
83         ea.addEdge(_source, *vit, 1);
84         white.push_back(*vit);
85     } else {
86         ea.addEdge(*vit, _sink, 1);
87         black.push_back(*vit);
88     }
89 }
90 for (auto wv: white) {
91     OutEdgeIt ebegin, eend;
92     for (tie(ebegin, eend) = out_edges(wv, G); ebegin != eend; ++
        ebegin) {
93         const int v = target(*ebegin, G);
94         ea.addEdge(wv, v, 1);
95     }
96 }
97 for (auto bv: black) {
98     OutEdgeIt ebegin, eend;
99     for (tie(ebegin, eend) = out_edges(bv, G); ebegin != eend; ++
        ebegin) {
100         const int v = target(*ebegin, bipG);
101         ea.addEdge(v, bv, 1);
102     }
103 }
104 push_relabel_max_flow(bipG, _source, _sink);
105 vector<int> vis(num_vert + 2, false);
106 vis[_source] = true;
107 std::queue<int> Q;
108 Q.push(_source);

```

```

109 while (!Q.empty()) {
110     const int u = Q.front();
111     Q.pop();
112     OutEdgeIt ebegin, eend;
113     for (tie(ebegin, eend) = out_edges(u, bipG); ebegin != eend;
        ++ebegin) {
114         const int v = target(*ebegin, bipG);
115         if (res_capacity[*ebegin] == 0 || vis[v]) continue;
116         vis[v] = true;
117         Q.push(v);
118     }
119 }
120 int vertex_cover = 0;
121 for (auto it = white.begin(); it != white.end(); ++it)
122     if (!vis[*it]) ++vertex_cover;
123 for (auto it = black.begin(); it != black.end(); ++it)
124     if (vis[*it]) ++vertex_cover;
125 cout << num_vert - vertex_cover << endl;
126 }
127 return 0;
128 }

```

3.8.4 Satellites

Keywords— Minimum vertex cover, No-source no-sink

```

1 #include <iostream>
2 #include <algorithm>
3 #include <queue>
4 #include <vector>
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/push_relabel_max_flow.hpp>
7 #include <boost/graph/edmonds_karp_max_flow.hpp>
8 #include <boost/tuple/tuple.hpp>
9
10 using namespace std;
11 using namespace boost;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,

```

```

17     property<edge_reverse_t, Traits::edge_descriptor> > > >
        Graph;
18 typedef property_map<Graph, edge_capacity_t>::type
    EdgeCapacityMap;
19 typedef property_map<Graph, edge_residual_capacity_t>::type
    ResidualCapacityMap;
20 typedef property_map<Graph, edge_reverse_t>::type
    ReverseEdgeMap;
21 typedef graph_traits<Graph>::vertex_descriptor Vertex;
22 typedef graph_traits<Graph>::edge_descriptor Edge;
23 typedef graph_traits<Graph>::edge_iterator EdgeIt;
24 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
25
26 struct EdgeAdder {
27     EdgeAdder(Graph & G, EdgeCapacityMap &capacity,
        ReverseEdgeMap &rev_edge)
28         : G(G), capacity(capacity), rev_edge(rev_edge) {}
29
30     void addEdge(int u, int v, long c) {
31         Edge e, reverseE;
32         tie(e, tuples::ignore) = add_edge(u, v, G);
33         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
34         capacity[e] = c;
35         capacity[reverseE] = 0;
36         rev_edge[e] = reverseE;
37         rev_edge[reverseE] = e;
38     }
39     Graph &G;
40     EdgeCapacityMap &capacity;
41     ReverseEdgeMap &rev_edge;
42 };
43
44 int main() {
45     int T;
46     cin >> T;
47     while (T--) {
48         int g, s, l;
49         cin >> g >> s >> l;
50         Graph G(g + s);
51         EdgeCapacityMap capacity = get(edge_capacity, G);
52         ReverseEdgeMap rev_edge = get(edge_reverse, G);
53         ResidualCapacityMap res_capacity = get(
            edge_residual_capacity, G);

```

```

54     EdgeAdder ea(G, capacity, rev_edge);
55     Vertex _source = add_vertex(G);
56     Vertex _sink = add_vertex(G);
57     while (l--) {
58         int ground, satellite;
59         cin >> ground >> satellite;
60         ea.addEdge(ground, g + satellite, 1);
61     }
62     for (int i = 0; i < g; ++i) {
63         ea.addEdge(_source, i, 1);
64     }
65     for (int i = g; i < g + s; ++i) {
66         ea.addEdge(i, _sink, 1);
67     }
68
69     push_relabel_max_flow(G, _source, _sink);
70     vector<int> vis(g+s+2, false);
71     vis[_source] = true;
72     std::queue<int> Q;
73     Q.push(_source);
74     while (!Q.empty()) {
75         const int u = Q.front();
76         Q.pop();
77         OutEdgeIt ebegin, eend;
78         for (tie(ebegin, eend) = out_edges(u, G); ebegin != eend; ++
            ebegin) {
79             const int v = target(*ebegin, G);
80             if (res_capacity[*ebegin] == 0 || vis[v]) continue;
81             vis[v] = true;
82             Q.push(v);
83         }
84     }
85     // minimum vertex cover is formed by the non visited
        vertices of ground
86     // stations (left side of bipartite graph) and visited
        vertices of
87     // (right side of bipartite graph) satellites
88     int g_prime = 0, s_prime = 0;
89     vector<int> result;
90     for (int i = 0; i < g; ++i) {
91         if (!vis[i]) {
92             ++g_prime;
93             result.push_back(i);

```

```

94     }
95 }
96 for (int i = g; i < g + s; ++i) {
97     if (vis[i]) {
98         ++s_prime;
99         result.push_back(i - g);
100     }
101 }
102 cout << g_prime << " " << s_prime << endl;
103 bool first = true;
104 for (auto it = result.begin(); it != result.end(); ++it) {
105     if (!first)
106         cout << " ";
107     cout << *it;
108     first = false;
109 }
110 cout << endl;
111 }
112 return 0;
113 }

```

3.9 Min cost max flow

3.9.1 Bonus level

Keywords—

Check https://judge.inf.ethz.ch/doc/course/solution_bonus_level.pdf

```

1  #include <iostream>
2  #include <cstdlib>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/
    successive_shortest_path_nonnegative_weights.hpp>
5  #include <boost/graph/find_flow_cost.hpp>
6
7  using namespace boost;
8  using namespace std;
9
10 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
11 typedef adjacency_list<vecS, vecS, directedS, no_property,
12     property<edge_capacity_t, long,
13     property<edge_residual_capacity_t, long,
14     property<edge_reverse_t, Traits::edge_descriptor,
15     property<edge_weight_t, long>>>> Graph;
16
17 typedef property_map<Graph, edge_capacity_t>::type
    EdgeCapacityMap;
18 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
19
20 typedef property_map<Graph, edge_residual_capacity_t>::type
    ResCapacityMap;
21 typedef property_map<Graph, edge_reverse_t>::type
    ReverseEdgeMap;
22
23 typedef graph_traits<Graph>::vertex_descriptor Vertex;
24 typedef graph_traits<Graph>::edge_descriptor Edge;
25 typedef graph_traits<Graph>::edge_iterator EdgeIt;
26 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
27
28 struct EdgeAdder {
29     EdgeAdder(Graph & G, EdgeCapacityMap &capacity, EdgeWeightMap
        &weight, ReverseEdgeMap &rev_edge)
30         : G(G), capacity(capacity), weight(weight), rev_edge(
        rev_edge) {}
31
32     void addEdge(int u, int v, long c, long w) {

```

```

31     Edge e, reverseE;
32     tie(e, tuples::ignore) = add_edge(u, v, G);
33     tie(reverseE, tuples::ignore) = add_edge(v, u, G);
34     capacity[e] = c;
35     weight[e] = w;
36     capacity[reverseE] = 0;
37     weight[reverseE] = -w;
38     rev_edge[e] = reverseE;
39     rev_edge[reverseE] = e;
40 }
41 Graph &G;
42 EdgeCapacityMap &capacity;
43 EdgeWeightMap &weight;
44 ReverseEdgeMap &rev_edge;
45 };
46
47 int main() {
48     std::ios_base::sync_with_stdio(false);
49     int T;
50     cin >> T;
51     while (T--) {
52         int n;
53         cin >> n;
54
55         const int EXTRA_COST = 100;
56         Graph G(3 * n * n);
57         EdgeCapacityMap capacity = get(edge_capacity, G);
58         EdgeWeightMap weight = get(edge_weight, G);
59         ReverseEdgeMap rev_edge = get(edge_reverse, G);
60         ResCapacityMap res_capacity = get(edge_residual_capacity, G);
61
62         EdgeAdder ea(G, capacity, weight, rev_edge);
63         Vertex _source = add_vertex(G);
64         Vertex _sink = add_vertex(G);
65
66         for (int i = 0; i < n; ++i) {
67             for (int j = 0; j < n; ++j) {
68                 int a;
69                 cin >> a;
70                 int current = 3 * j + 3 * n * i;
71                 ea.addEdge(current, current + 1, 1, EXTRA_COST - a);
72                 ea.addEdge(current, current + 2, 1, EXTRA_COST);
73                 if (j != n-1) {

```

```

73                     ea.addEdge(current + 1, current + 3, 1, 0);
74                     ea.addEdge(current + 2, current + 3, 1, 0);
75                 }
76                 if (i != n-1) {
77                     ea.addEdge(current + 1, current + 3 * n, 1, 0);
78                     ea.addEdge(current + 2, current + 3 * n, 1, 0);
79                 }
80             }
81         }
82         ea.addEdge(_source, 0, 2, 0);
83         ea.addEdge(n * n * 3 - 2, _sink, 1, 0);
84         ea.addEdge(n * n * 3 - 1, _sink, 1, 0);
85
86         successive_shortest_path_nonnegative_weights(G, _source,
87             _sink);
88         int cost = find_flow_cost(G);
89         cout << -cost + 2 * (2 * n - 1) * EXTRA_COST << endl;
90     }
91 }

```

3.9.2 Canteen

Keywords—

Check <https://judge.inf.ethz.ch/doc/course/canteen.pdf>

```

1  #include <iostream>
2  #include <cstdlib>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/cycle_canceling.hpp>
5  #include <boost/graph/push_relabel_max_flow.hpp>
6  #include <boost/graph/
7      successive_shortest_path_nonnegative_weights.hpp>
8  #include <boost/graph/find_flow_cost.hpp>
9
10 using namespace boost;
11 using namespace std;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,
17     property<edge_reverse_t, Traits::edge_descriptor,

```

```

17         property <edge_weight_t, long> > > > Graph;
18
19     typedef property_map<Graph, edge_capacity_t>::type
        EdgeCapacityMap;
20     typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap
        ;
21     typedef property_map<Graph, edge_residual_capacity_t>::type
        ResCapacityMap;
22     typedef property_map<Graph, edge_reverse_t>::type
        ReverseEdgeMap;
23     typedef graph_traits<Graph>::vertex_descriptor Vertex;
24     typedef graph_traits<Graph>::edge_descriptor Edge;
25     typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
26
27     struct EdgeAdder {
28         EdgeAdder(Graph & G, EdgeCapacityMap &capacity, EdgeWeightMap
            &weight, ReverseEdgeMap &rev_edge)
29             : G(G), capacity(capacity), weight(weight), rev_edge(
                rev_edge) {}
30
31         void addEdge(int u, int v, long c, long w) {
32             Edge e, reverseE;
33             tie(e, tuples::ignore) = add_edge(u, v, G);
34             tie(reverseE, tuples::ignore) = add_edge(v, u, G);
35             capacity[e] = c;
36             weight[e] = w;
37             capacity[reverseE] = 0;
38             weight[reverseE] = -w;
39             rev_edge[e] = reverseE;
40             rev_edge[reverseE] = e;
41         }
42         Graph &G;
43         EdgeCapacityMap &capacity;
44         EdgeWeightMap &weight;
45         ReverseEdgeMap &rev_edge;
46     };
47
48     int main() {
49         std::ios_base::sync_with_stdio(false);
50
51         int T;
52         cin >> T;
53         while (T--) {
54             int n;
55             cin >> n;
56
57             const int EXTRA_COST = 20;
58             Graph G(n);
59             EdgeCapacityMap capacity = get(edge_capacity, G);
60             EdgeWeightMap weight = get(edge_weight, G);
61             ReverseEdgeMap rev_edge = get(edge_reverse, G);
62             ResCapacityMap res_capacity = get(edge_residual_capacity, G
                );
63             EdgeAdder ea(G, capacity, weight, rev_edge);
64             Vertex _source = add_vertex(G);
65             Vertex _sink = add_vertex(G);
66
67             for (int i = 0; i < n; ++i) {
68                 int a, c;
69                 cin >> a >> c;
70                 ea.addEdge(_source, i, a, c + EXTRA_COST);
71             }
72             int total_students = 0;
73             for (int i = 0; i < n; ++i) {
74                 int s, p;
75                 cin >> s >> p;
76                 total_students += s;
77                 ea.addEdge(i, _sink, s, -p + EXTRA_COST);
78             }
79             for (int i = 0; i < n-1; ++i) {
80                 int v, e;
81                 cin >> v >> e;
82                 ea.addEdge(i, i + 1, v, e);
83             }
84
85             //int flow = push_relabel_max_flow(G, _source, _sink);
86             //cycle_canceling(G);
87             successive_shortest_path_nonnegative_weights(G, _source,
                _sink);
88             int flow = 0;
89             OutEdgeIt e, eend;
90             for (tie(e, eend) = out_edges(vertex(_source, G), G); e !=
                eend; ++e) {
91                 flow += capacity[*e] - res_capacity[*e];
92             }
93

```

```

94     int cost = find_flow_cost(G);
95     cout << (flow == total_students ? "possible" : "impossible"
96           ) << " " << flow
97           << " " << -cost + flow * 2 * EXTRA_COST << endl;
98 }

```

3.9.3 Carsharing

Keywords— equal_range

Check <https://judge.inf.ethz.ch/doc/course/carsharing.pdf>

```

1  #include <iostream>
2  #include <limits>
3  #include <cstdlib>
4  #include <boost/graph/adjacency_list.hpp>
5  #include <boost/graph/cycle_canceling.hpp>
6  #include <boost/graph/push_relabel_max_flow.hpp>
7  #include <boost/graph/
    successive_shortest_path_nonnegative_weights.hpp>
8  #include <boost/graph/find_flow_cost.hpp>
9
10 using namespace boost;
11 using namespace std;
12
13 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
14 typedef adjacency_list<vecS, vecS, directedS, no_property,
15     property<edge_capacity_t, long,
16     property<edge_residual_capacity_t, long,
17     property<edge_reverse_t, Traits::edge_descriptor,
18     property<edge_weight_t, long>>>> Graph;
19
20 typedef property_map<Graph, edge_capacity_t>::type
    EdgeCapacityMap;
21 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
22
23 typedef property_map<Graph, edge_residual_capacity_t>::type
    ResCapacityMap;
24 typedef property_map<Graph, edge_reverse_t>::type
    ReverseEdgeMap;
25 typedef graph_traits<Graph>::vertex_descriptor Vertex;
26 typedef graph_traits<Graph>::edge_descriptor Edge;

```

```

26 typedef graph_traits<Graph>::edge_iterator EdgeIt;
27 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
28
29 struct EdgeAdder {
30     EdgeAdder(Graph & G, EdgeCapacityMap &capacity, EdgeWeightMap
31         &weight, ReverseEdgeMap &rev_edge)
32         : G(G), capacity(capacity), weight(weight), rev_edge(
33             rev_edge) {}
34
35     void addEdge(int u, int v, long c, long w) {
36         Edge e, reverseE;
37         tie(e, tuples::ignore) = add_edge(u, v, G);
38         tie(reverseE, tuples::ignore) = add_edge(v, u, G);
39         capacity[e] = c;
40         weight[e] = w;
41         capacity[reverseE] = 0;
42         weight[reverseE] = -w;
43         rev_edge[e] = reverseE;
44         rev_edge[reverseE] = e;
45     }
46     Graph &G;
47     EdgeCapacityMap &capacity;
48     EdgeWeightMap &weight;
49     ReverseEdgeMap &rev_edge;
50 };
51
52 struct Request {
53     int s, t; // source, target
54     int d, a; // departure, arrival
55     int p;    // profit
56     Request(int ps, int pt, int pd, int pa, int pp)
57         : s(ps), t(pt), d(pd), a(pa), p(pp) {}
58 };
59
60 int vertices_between(int dep_t, int arr_t, vector<int> &times)
61 {
62     int pos1 = distance(times.begin(), equal_range(times.begin(),
63         times.end(), dep_t).first);
64     int pos2 = distance(times.begin(), equal_range(times.begin(),
65         times.end(), arr_t).first);
66     return pos2 - pos1;
67 }

```

```

66 int main() {
67     std::ios_base::sync_with_stdio(false);
68     int T;
69     cin >> T;
70     while (T--) {
71         int N, S;
72         cin >> N >> S;
73         vector<int> sources(S);
74         for (int i = 0; i < S; ++i)
75             cin >> sources[i];
76         vector<Request> requests;
77         requests.reserve(N);
78         vector< set<int> > station_times(S);
79         set<int> set_times;
80         set< pair<int, int> > vertices;
81         int t_0 = numeric_limits<int>::max();
82         int t_max = 0;
83         for (int i = 0; i < N; ++i) {
84             int s, t, d, a, p;
85             cin >> s >> t >> d >> a >> p;
86             t_0 = min(t_0, d);
87             t_max = max(t_max, a);
88             --s; --t;
89             requests.push_back(Request(s, t, d, a, p));
90             station_times[s].insert(d);
91             station_times[t].insert(a);
92             vertices.insert(make_pair(s, d));
93             vertices.insert(make_pair(t, a));
94             set_times.insert(d);
95             set_times.insert(a);
96         }
97         vector<int> all_times(set_times.begin(), set_times.end());
98         vector< vector<int> > station_times_v(S);
99         for (int i = 0; i < S; ++i)
100             station_times_v[i] = vector<int> (station_times[i].begin
101                 (), station_times[i].end());
102         // define the graph
103         const int MAX_PROFIT = 100;
104         Graph G(vertices.size());
105         EdgeCapacityMap capacity = get(edge_capacity, G);
106         EdgeWeightMap weight = get(edge_weight, G);
107         ReverseEdgeMap rev_edge = get(edge_reverse, G);
108         ResCapacityMap res_capacity = get(edge_residual_capacity, G
109
110             );
111         EdgeAdder ea(G, capacity, weight, rev_edge);
112         Vertex _source = add_vertex(G);
113         Vertex _sink = add_vertex(G);
114         // offsets
115         vector<int> offsets(S);
116         int offset = 0;
117         for (int i = 0; i < S; ++i) {
118             offsets[i] = offset;
119             offset += station_times[i].size();
120         }
121         // source -> first time (or source -> sink if no times)
122         for (int i = 0; i < S; ++i) {
123             if (station_times_v[i].size() == 0) continue;
124             int between = vertices_between(t_0, station_times_v[i].
125                 front(), all_times);
126             ea.addEdge(_source, offsets[i], sources[i], MAX_PROFIT *
127                 between);
128         }
129         // connection between times
130         for (int i = 0; i < S; ++i) {
131             if (station_times_v[i].size() == 0) continue;
132             int between = vertices_between(station_times_v[i].back(),
133                 t_max, all_times);
134             ea.addEdge(offsets[i] + station_times[i].size() - 1,
135                 _sink,
136                 numeric_limits<int>::max(), MAX_PROFIT *
137                 between);
138             for (int j = 0; j < station_times[i].size() - 1; ++j) {
139                 between = vertices_between(station_times_v[i][j],
140                     station_times_v[i][j + 1],
141                     all_times);
142                 ea.addEdge(offsets[i] + j, offsets[i] + j + 1,
143                     numeric_limits<int>::max(),
144                     MAX_PROFIT * between);
145             }
146         }
147         // request connections
148         for (auto req: requests) {
149             auto src_times = station_times_v[req.s];
150             int pos1 = distance(src_times.begin(), equal_range(
151                 src_times.begin(),
152                 src_times.end(), req.d).first);

```



```

142     int u = offsets[req.s] + pos1;
143     auto tgt_times = station_times_v[req.t];
144     int pos2 = distance(tgt_times.begin(), equal_range(
145         tgt_times.begin(),
146         tgt_times.end(), req.a).first);
147     int v = offsets[req.t] + pos2;
148     int between = vertices_between(req.d, req.a, all_times);
149     ea.addEdge(u, v, 1, MAX_PROFIT * between - req.p);
150 }
151 //int flow = push_relabel_max_flow(G, _source, _sink);
152 //cycle_canceling(G);
153 successive_shortest_path_nonnegative_weights(G, _source,
154     _sink);
155 int flow = 0;
156 OutEdgeIt e, eend;
157 for (tie(e, eend) = out_edges(vertex(_source, G), G); e !=
158     eend; ++e)
159     flow += capacity[*e] - res_capacity[*e];
160 int cost = find_flow_cost(G);
161 int max_dist = vertices_between(t_0, t_max, all_times);
162 cout << flow * max_dist * MAX_PROFIT - cost << endl;
163 }
164 return 0;
165 }

```

3.9.4 Real estate market

Keywords— Integer programming

Check https://judge.inf.ethz.ch/doc/course/real_estate_handout.pdf

```

1 #include <iostream>
2 #include <cstdlib>
3 #include <boost/graph/adjacency_list.hpp>
4 #include <boost/graph/cycle_canceling.hpp>
5 #include <boost/graph/push_relabel_max_flow.hpp>
6 #include <boost/graph/
7     successive_shortest_path_nonnegative_weights.hpp>
8 #include <boost/graph/find_flow_cost.hpp>
9
10 using namespace boost;
11 using namespace std;

```

```

12 typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
13 typedef adjacency_list<vecS, vecS, directedS, no_property,
14     property<edge_capacity_t, long,
15     property<edge_residual_capacity_t, long,
16     property<edge_reverse_t, Traits::edge_descriptor,
17     property<edge_weight_t, long>>>> Graph;
18
19 typedef property_map<Graph, edge_capacity_t>::type
20     EdgeCapacityMap;
21 typedef property_map<Graph, edge_weight_t>::type EdgeWeightMap;
22 ;
23 typedef property_map<Graph, edge_residual_capacity_t>::type
24     ResCapacityMap;
25 typedef property_map<Graph, edge_reverse_t>::type
26     ReverseEdgeMap;
27
28 typedef graph_traits<Graph>::vertex_descriptor Vertex;
29 typedef graph_traits<Graph>::edge_descriptor Edge;
30 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
31
32 struct EdgeAdder {
33     EdgeAdder(Graph & G, EdgeCapacityMap &capacity, EdgeWeightMap
34         &weight, ReverseEdgeMap &rev_edge)
35         : G(G), capacity(capacity), weight(weight), rev_edge(
36             rev_edge) {}
37
38 void addEdge(int u, int v, long c, long w) {
39     Edge e, reverseE;
40     tie(e, tuples::ignore) = add_edge(u, v, G);
41     tie(reverseE, tuples::ignore) = add_edge(v, u, G);
42     capacity[e] = c;
43     weight[e] = w;
44     capacity[reverseE] = 0;
45     weight[reverseE] = -w;
46     rev_edge[e] = reverseE;
47     rev_edge[reverseE] = e;
48 }
49
50 Graph &G;
51 EdgeCapacityMap &capacity;
52 EdgeWeightMap &weight;
53 ReverseEdgeMap &rev_edge;
54 };
55
56 int main() {

```



```

49  std::ios_base::sync_with_stdio(false);
50
51  int T;
52  cin >> T;
53  while (T--) {
54      int N, M, S;
55      cin >> N >> M >> S;
56
57      const int BUYER = 0;
58      const int LAND = N;
59      const int STATE = N + M;
60      const int DELTA = 100;
61      Graph G(N + M + S);
62      EdgeCapacityMap capacity = get(edge_capacity, G);
63      EdgeWeightMap weight = get(edge_weight, G);
64      ReverseEdgeMap rev_edge = get(edge_reverse, G);
65      ResCapacityMap res_capacity = get(edge_residual_capacity, G
        );
66      EdgeAdder ea(G, capacity, weight, rev_edge);
67      Vertex _source = add_vertex(G);
68      Vertex _sink = add_vertex(G);
69
70      for (int i = 0; i < S; ++i) {
71          int l;
72          cin >> l;
73          ea.addEdge(STATE + i, _sink, l, 0);
74      }
75      for (int i = 0; i < M; ++i) {
76          int s;
77          cin >> s;
78          ea.addEdge(LAND + i, STATE + s - 1, 1, 0);
79      }
80      for (int i = 0; i < N; ++i) {
81          ea.addEdge(_source, BUYER + i, 1, 0);
82          for (int j = 0; j < M; ++j) {
83              int b;
84              cin >> b;
85              ea.addEdge(BUYER + i, LAND + j, 1, DELTA - b);
86          }
87      }
88
89      successive_shortest_path_nonnegative_weights(G, _source,
        _sink);

```

```

90      int flow = 0;
91      OutEdgeIt e, eend;
92      for (tie(e, eend) = out_edges(vertex(_source, G), G); e !=
        eend; ++e) {
93          flow += capacity[*e] - res_capacity[*e];
94      }
95      int cost = find_flow_cost(G);
96
97      cout << flow << " " << flow * DELTA - cost << endl;
98  }
99 }

```

3.10 Miscellaneous

3.10.1 Buddy selection

Keywords— Max cardinality matching, Set intersection

```
1 #include <iostream>
2 #include <algorithm>
3 #include <set>
4 #include <vector>
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/max_cardinality_matching.hpp>
7 #include <boost/tuple/tuple.hpp>
8
9 using namespace boost;
10 using namespace std;
11
12 typedef adjacency_list<vecS, vecS, undirectedS, no_property,
13     property<edge_weight_t, int> > Graph;
14 typedef graph_traits<Graph>::edge_descriptor Edge;
15 typedef property_map<Graph, edge_weight_t>::type WeightMap;
16
17 int main() {
18     std::ios_base::sync_with_stdio(false);
19     int T;
20     cin >> T;
21     while (T--) {
22         int n, f, c;
23         cin >> n >> c >> f;
24         Graph G(n);
25         vector< set<string> > characteristics;
26         WeightMap wm = get(edge_weight, G);
27         for (int u = 0; u < n; ++u) {
28             string characteristic;
29             set<string> newset;
30             characteristics.push_back(newset);
31             for (int i = 0; i < c; ++i) {
32                 cin >> characteristic;
33                 characteristics[u].insert(characteristic);
34             }
35             for (int v = 0; v < (int) characteristics.size(); ++v) {
36                 if (u == v) continue;
37
38                 set<string> intersect;
```

```
38         set_intersection(characteristics[u].begin(),
39             characteristics[u].end(),
40             characteristics[v].begin(),
41             characteristics[v].end(),
42             inserter(intersect, intersect.begin()));
43         if ((int) intersect.size() > f) {
44             Edge e;
45             tie(e, tuples::ignore) = add_edge(u, v, G);
46             wm[e] = (int) intersect.size();
47         }
48     }
49     vector<graph_traits<Graph>::vertex_descriptor> mate(n);
50     edmonds_maximum_cardinality_matching(G, &mate[0]);
51     graph_traits<Graph>::vertex_iterator vi, vi_end;
52     int connections = 0;
53     for(tie(vi, vi_end) = vertices(G); vi != vi_end; ++vi)
54         if (mate[*vi] != graph_traits<Graph>::null_vertex() && *
55             vi < mate[*vi])
56             ++connections;
57
58     cout << ((connections == n/2) ? "not optimal" : "optimal")
59         << endl;
60 }
61 return 0;
62 }
```

3.10.2 Divisor distance

Keywords— Prime numbers

```
1 #include <iostream>
2 #include <algorithm>
3 #include <bitset>
4 #include <vector>
5
6 using namespace std;
7
8 #define HIGHER_BOUND 10000010
9 bitset<HIGHER_BOUND> is_prime;
10
11 // NOT USED to solve the problem!! just here for reference :)
```

```

12 bool is_prime(int n) {
13     long long i = 2;
14     // 1 is not prime
15     if (n <= 1) return false;
16     // If the number is even
17     if (n == 2) return true;
18     else if (n%i == 0) return false;
19     // If the number is odd
20     ++i;
21     long long root = sqrt(n);
22     while (root >= i) {
23         if (n%i == 0)
24             return false;
25         i += 2;
26     }
27     return true;
28 }
29
30 void prime_sieve() {
31     is_prime.set(); // all true;
32     is_prime.reset(0); // 0 not prime
33     is_prime.reset(1); // 1 not prime
34     for (int prime = 2; prime * prime < HIGHER_BOUND; ++prime) {
35         if (is_prime[prime]) {
36             // all multiples of the prime number are not prime
37             for (int multiple = prime + prime; multiple <
38                 HIGHER_BOUND; multiple += prime)
39                 is_prime.reset(multiple);
40         }
41     }
42
43     // return next smallest prime that divides n
44     int next_smallest_prime(int n, int current) {
45         if (is_prime[n])
46             return 1;
47         for (int prime = current; prime < n; ++prime)
48             if (is_prime[prime] && n%prime == 0)
49                 return prime;
50         return 1;
51     }
52
53     // returns a vector with all the divisors of v

```

```

54 vector<int> divisors(int v) {
55     vector<int> dv;
56     dv.push_back(v);
57     if (v == 1)
58         return dv;
59     int prime = 2;
60     while (!is_prime[v]) {
61         prime = next_smallest_prime(v, prime);
62         int mdiv = v / prime;
63         dv.push_back(mdiv);
64         v = mdiv;
65     }
66     dv.push_back(1);
67     return dv;
68 }
69
70 int main() {
71     std::ios_base::sync_with_stdio(false);
72
73     prime_sieve();
74     int T;
75     cin >> T;
76     while (T--) {
77         int n, c;
78         cin >> n >> c;
79         while (c--) {
80             int v1, v2;
81             cin >> v1 >> v2;
82             vector<int> dv1 = divisors(v1);
83             vector<int> dv2 = divisors(v2);
84             int i = 0;
85             int j = 0;
86             while (dv1[i] != dv2[j]) {
87                 if (dv1[i] > dv2[j])
88                     ++i;
89                 else
90                     ++j;
91             }
92             cout << i + j << endl;
93         }
94     }
95     return 0;
96 }

```

3.10.3 Monkey island

Keywords— Strongly connected components

```
1 #include <iostream>
2 #include <algorithm>
3 #include <limits>
4 #include <vector>
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/strong_components.hpp>
7
8 using namespace boost;
9 using namespace std;
10
11 typedef adjacency_list<vecS, vecS, directedS> Graph;
12 typedef graph_traits<Graph>::edge_descriptor Edge;
13 typedef graph_traits<Graph>::out_edge_iterator
14     out_edge_iterator;
15
16 int which_component(vector< set<int> > &components, int elem) {
17     for (int i = 0; i < components.size(); ++i) {
18         if (components[i].find(elem) != components[i].end())
19             return i;
20     }
21     return -1;
22 }
23
24 int main() {
25     std::ios_base::sync_with_stdio(false);
26     int T;
27     cin >> T;
28     while (T--) {
29         int n, m;
30         cin >> n >> m;
31         Graph G(n);
32         vector<int> costs;
33         for (int i = 0; i < m; ++i) {
34             int u, v;
35             cin >> u >> v;
36             add_edge(u-1, v-1, G);
37         }
```

```
37
38     int c;
39     cin >> c;
40     costs.push_back(c);
41 }
42 vector<int> scc(n);
43 int nscc = strong_components(G,
44     make_iterator_property_map(scc.begin(), get(
45         vertex_index, G)));
46 vector< set<int> > components(nscc);
47 for (int i = 0; i < scc.size(); ++i) {
48     components[scc[i]].insert(i);
49 }
50 vector<int> parent_scc(nscc, -1);
51 for (int i = 0; i < nscc; ++i) {
52     for (auto it = components[i].begin(); it != components[i]
53         .end(); ++it) {
54         out_edge_iterator oei, oeie;
55         for (tie(oei, oeie) = out_edges(*it, G); oei !=
56             oeie; ++oei) {
57             int t = target(*oei, G);
58             if (components[i].find(t) == components[i].end())
59                 parent_scc[which_component(components, t)] = i;
60         }
61     }
62 }
63 // for the SCC with no parents (parent=-1) sum the cost
64 int final_cost = 0;
65 for (int i = 0; i < nscc; ++i) {
66     if (parent_scc[i] == -1) {
67         int cheapest = numeric_limits<int>::max();
68         for (auto it = components[i].begin(); it != components[i]
69             .end(); ++it)
70             cheapest = min(cheapest, costs[*it]);
71         final_cost += cheapest;
72     }
73 }
74 cout << final_cost << endl;
75 }
76 return 0;
77 }
```

3.11 Combinations

3.11.1 Clues

Keywords— connected_components, Edge_circulator, is_bipartite, Vertex_circulator

```
1 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2 #include <CGAL/Delaunay_triangulation_2.h>
3 #include <iostream>
4 #include <vector>
5 #include <boost/graph/adjacency_list.hpp>
6 #include <boost/graph/bipartite.hpp>
7 #include <boost/graph/connected_components.hpp>
8
9 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
10 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
11 typedef Triangulation::Face_handle Face_handle;
12 typedef Triangulation::Vertex_handle Vertex_handle;
13
14 typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::undirectedS> Graph;
15
16 int main() {
17     std::ios_base::sync_with_stdio(false);
18     int T;
19     std::cin >> T;
20     while (T--) {
21         int n, m, r;
22         std::cin >> n >> m >> r;
23         double r2 = ((double) r) * ((double) r);
24         std::vector<K::Point_2> stations;
25         stations.reserve(n);
26         std::map<K::Point_2, int> vertex_ids;
27         for (int i = 0; i < n; ++i) {
28             int x, y;
29             std::cin >> x >> y;
30             auto p = K::Point_2(x, y);
31             stations.push_back(p);
32             vertex_ids.insert(std::make_pair(p, i));
33         }
34         // Define triangulation
35         Triangulation t;
36         t.insert(stations.begin(), stations.end());
37         // Define graph
```

```
38 Graph G(n);
39 for (auto vit = t.finite_vertices_begin(); vit != t.finite_vertices_end(); ++vit) {
40     auto vc = t.incident_vertices(vit); // Vertex_circulator
41     do {
42         if (CGAL::squared_distance(vit->point(), vc->point())
43             <= r2) {
44             int u = vertex_ids.find(vit->point())->second;
45             int v = vertex_ids.find(vc->point())->second;
46             if (u > v)
47                 add_edge(u, v, G);
48         }
49     } while (++vc != t.incident_vertices(vit));
50 }
51 bool interferences = !boost::is_bipartite(G);
52 if (!interferences) { // maybe not all needed edges were included in the graph
53     for (auto vit = t.finite_vertices_begin(); vit != t.finite_vertices_end(); ++vit) {
54         auto ec = t.incident_edges(vit); // Edge_circulator
55         std::vector<K::Point_2> edge_points;
56         do {
57             if (t.is_infinite(ec)) continue;
58             auto f = ec->first;
59             auto p = f->vertex(f->ccw(ec->second))->point();
60             for (auto ep: edge_points) {
61                 if (CGAL::squared_distance(p, ep) <= r2) {
62                     int u = vertex_ids.find(p)->second;
63                     int v = vertex_ids.find(ep)->second;
64                     add_edge(u, v, G);
65                 }
66             }
67             edge_points.push_back(p);
68         } while (++ec != t.incident_edges(vit));
69     }
70     interferences = !boost::is_bipartite(G);
71 }
72 std::vector<int> component(boost::num_vertices(G));
73 connected_components(G, &component[0]);
74 for (int i = 0; i < m; ++i) {
75     int ax, ay, bx, by;
76     std::cin >> ax >> ay >> bx >> by;
```

```

77     if (interferences) {
78         std::cout << "n";
79         continue;
80     }
81     auto holmes = K::Point_2(ax, ay);
82     auto watson = K::Point_2(bx, by);
83     if (CGAL::squared_distance(holmes, watson) <= r2) {
84         std::cout << "y";
85         continue;
86     }
87     auto nearest_h = t.nearest_vertex(holmes)->point();
88     int comp_h = component[vertex_ids.find(nearest_h)->second
89         ];
89     auto nearest_w = t.nearest_vertex(watson)->point();
90     int comp_w = component[vertex_ids.find(nearest_w)->second
91         ];
91     if (CGAL::squared_distance(holmes, nearest_h) <= r2
92         && CGAL::squared_distance(watson, nearest_w) <= r2
93         && comp_h == comp_w) {
94         std::cout << "y";
95         continue;
96     }
97     std::cout << "n";
98 }
99 std::cout << std::endl;
100 }
101 return 0;
102 }

```

3.11.2 Sith

Keywords— Binary search, connected_components, max_element, Triangulation

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <CGAL/Delaunay_triangulation_2.h>
3  #include <boost/graph/adjacency_list.hpp>
4  #include <boost/graph/connected_components.hpp>
5  #include <iostream>
6  #include <map>
7
8  typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
9  typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
10

```

```

11 typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::
12     undirectedS> Graph;
13
14 bool conquer(std::vector<K::Point_2> &planets,
15     std::map<K::Point_2, int> &vertex_ids, int k, double r2) {
16     Graph G;
17     Triangulation t;
18     t.insert(planets.begin() + k, planets.end());
19     for (auto eit = t.finite_edges_begin(); eit != t.
20         finite_edges_end(); ++eit) {
21         auto f = eit->first;
22         auto p1 = f->vertex(f->cw(eit->second))->point();
23         auto p2 = f->vertex(f->ccw(eit->second))->point();
24         if (CGAL::squared_distance(p1, p2) <= r2)
25             add_edge(vertex_ids[p1], vertex_ids[p2], G);
26     }
27     if (boost::num_vertices(G) == 0) return false;
28     std::vector<int> component(boost::num_vertices(G));
29     int ncomp = connected_components(G, &component[0]);
30     std::vector<int> n_elements(ncomp, 0);
31     for (int i = 0; i < component.size(); ++i)
32         ++n_elements[component[i]];
33     return *max_element(n_elements.begin(), n_elements.end()) >=
34         k;
35 }
36
37 int main() {
38     std::ios_base::sync_with_stdio(false);
39     int T;
40     std::cin >> T;
41     while (T-->0) {
42         int n, r;
43         std::cin >> n >> r;
44         double r2 = ((double) r) * ((double) r);
45         std::vector<K::Point_2> planets;
46         planets.reserve(n);
47         std::map<K::Point_2, int> vertex_ids;
48         for (int i = 0; i < n; ++i) {
49             int x, y;
50             std::cin >> x >> y;
51             auto p = K::Point_2(x, y);
52             planets.push_back(p);
53             vertex_ids.insert(std::make_pair(p, i));
54         }
55     }
56 }

```

```

51     }
52     int upper = n / 2;
53     int lower = 1;
54     int k = 1;
55     while (lower <= upper) {
56         int middle = (upper - lower) / 2 + lower;
57         //std::cout << "[" << lower << ", " << upper << "]" << middle
58         << std::endl;
59         if (conquer(planets, vertex_ids, middle, r2)) {
60             k = middle;
61             lower = middle + 1;
62         } else {
63             upper = middle - 1;
64         }
65         std::cout << k << std::endl;
66     }
67     return 0;
68 }

```

3.11.3 Stamps

Keywords— Linear programming, do_intersect

Check <https://judge.inf.ethz.ch/doc/course/stamps.pdf>

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <CGAL/basic.h>
3  #include <CGAL/QP_models.h>
4  #include <CGAL/QP_functions.h>
5  #include <iostream>
6  #include <vector>
7
8  typedef CGAL::Gmpq ET;
9  typedef CGAL::Quadratic_program<double> Program;
10 typedef CGAL::Quadratic_program_solution<ET> Solution;
11 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
12
13 int main() {
14     std::ios_base::sync_with_stdio(false);
15     int T;
16     std::cin >> T;
17     while (T--) {

```

```

18     int l, s, w;
19     std::cin >> l >> s >> w;
20
21     Program lp(CGAL::SMALLER, true, 1, true, 4096); // 2^12 =
22     4096
23     std::vector<K::Point_2> lamps;
24     lamps.reserve(l);
25     std::vector<K::Point_2> stamps;
26     stamps.reserve(s);
27     std::vector<K::Segment_2> walls;
28     walls.reserve(w);
29     for (int i = 0; i < l; ++i) {
30         int x, y;
31         std::cin >> x >> y;
32         lamps.push_back(K::Point_2(x, y));
33     }
34     for (int j = 0; j < s; ++j) {
35         int x, y, M;
36         std::cin >> x >> y >> M;
37         stamps.push_back(K::Point_2(x, y));
38         lp.set_b(2 * j, -1); // \sum{ p_i * r } >= 1
39         lp.set_b(2 * j + 1, M); // \sum{ p_i * r } <= M
40     }
41     for (int k = 0; k < w; ++k) {
42         int a, b, c, d;
43         std::cin >> a >> b >> c >> d;
44         walls.push_back(K::Segment_2(K::Point_2(a, b), K::Point_2(
45             c, d)));
46     }
47     for (int j = 0; j < s; ++j) {
48         for (int i = 0; i < l; ++i) {
49             auto tmp_seg = K::Segment_2(lamps[i], stamps[j]);
50             bool no_wall_between = true;
51             for (int k = 0; k < w; ++k) {
52                 if (CGAL::do_intersect(tmp_seg, walls[k])) {
53                     no_wall_between = false;
54                     break;
55                 }
56             }
57             if (no_wall_between) {
58                 double dist = CGAL::squared_distance(lamps[i], stamps
59                     [j]);
60                 lp.set_a(i, 2 * j, -1 / dist);

```

```

58         lp.set_a(i, 2 * j + 1, 1 / dist);
59     }
60 }
61 }
62 Solution sol = CGAL::solve_linear_program(lp, ET());
63 std::cout << (sol.is_infeasible() ? "no" : "yes") << std::
    endl;
64 }
65 return 0;
66 }

```

3.11.4 The empire strikes back

Keywords— Linear programming, Triangulation

```

1  #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
2  #include <CGAL/Delaunay_triangulation_2.h>
3  #include <iostream>
4  #include <vector>
5  #include <cassert>
6  #include <CGAL/basic.h>
7  #include <CGAL/QP_models.h>
8  #include <CGAL/QP_functions.h>
9
10 typedef CGAL::Gmpq ET;
11 typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
12 typedef CGAL::Delaunay_triangulation_2<K> Triangulation;
13 typedef CGAL::Quadratic_program<ET> Program;
14 typedef CGAL::Quadratic_program_solution<ET> Solution;
15
16 int main() {
17     std::ios_base::sync_with_stdio(false);
18     int T;
19     std::cin >> T;
20     while (T--) {
21         int a, s, b, e;
22         std::cin >> a >> s >> b >> e;
23         Program lp(CGAL::LARGER, true, 0, false, 0); // all
            energies >= 0
24         // asteroids
25         std::vector<K::Point_2> asteroids(a);
26         for (int i = 0; i < a; ++i) {
27             int x, y, d;

```

```

28         std::cin >> x >> y >> d;
29         lp.set_b(i, d);
30         asteroids[i] = K::Point_2(x, y);
31     }
32     // laser shots
33     std::vector<K::Point_2> shots(s);
34     for (int var = 0; var < s; ++var) {
35         int x, y;
36         std::cin >> x >> y;
37         lp.set_c(var, 1);
38         shots[var] = K::Point_2(x, y);
39     }
40     // bounty hunters
41     std::vector<K::Point_2> hunters(b);
42     for (int i = 0; i < b; ++i) {
43         int x, y;
44         std::cin >> x >> y;
45         hunters[i] = K::Point_2(x, y);
46     }
47     Triangulation t;
48     t.insert(hunters.begin(), hunters.end());
49     for (int var = 0; var < s; ++var) {
50         ET r2 = -1;
51         if (b > 0) { // if there are hunters, find the closest
            one!
52             auto nearest_hunter = t.nearest_vertex(shots[var])->
                point();
53             r2 = CGAL::squared_distance(shots[var], nearest_hunter)
                ;
54         }
55         for (int i = 0; i < a; ++i) {
56             ET dist = CGAL::squared_distance(shots[var], asteroids[
                i]);
57             if (r2 == -1 || dist <= r2)
58                 lp.set_a(var, i, ET(1) / CGAL::max(ET(1), dist));
59         }
60     }
61     Solution sol = CGAL::solve_nonnegative_linear_program(lp,
        ET());
62     std::cout << (sol.is_optimal() && sol.objective_value() <=
        e ? "y" : "n")
        << std::endl;
63 }
64 }

```



```
65     return 0;  
66 }
```

Index

A

all_of, 9

B

BFS, 4, 5

Binary search, 31, 32, 56

Bipartite graph, 42

C

ceil_to_double, 8

connected_components, 55, 56

Custom sort, 3, 13

D

Dijkstra, 12, 13, 15

do_intersect, 10, 57

E

Edge disjoint paths, 24

Edge_circulator, 55

equal_range, 48

F

floor_to_double, 10, 27

G

Greedy, 4

I

Integer programming, 50

intersection, 10

Intervals, 4

is_bipartite, 42, 55

K

Kruskal MST, 12–14

L

left_turn, 9

Linear programming, 57, 58

M

Max cardinality matching, 52

max_element, 20, 56

Min_circle_2, 8

min_element, 20

Minimax, 18, 20

Minimum cut, 39

Minimum flow per edge, 23

Minimum vertex cover, 40, 42, 44

Motion planning, 37

N

nearest_vertex, 35

No-source no-sink, 44

P

Prime numbers, 53

priority_queue, 6

Q

Quadratic programming, 29–31

R

Ray_2, 10

S

Second MST, 14

Segment_2, 10

Set intersection, 52

Strictly greater inequality, 32

Strongly connected components, 54

T

Triangulation, 35, 38, 56, 58

Triangulation DFS, 37

V

Vertex capacities, 25, 26

Vertex_circulator, 55